

Securing Debian Manual

Javier Fernández-Sanguino Peña <jfs@computer.org>

2.6 10 octubre 2002 Wed, 18 Sep 2002 14:09:35 +0200

Abstract

This document describes the process of securing and hardening the default Debian installation. It covers some of the common tasks to set up a secure network environment using Debian GNU/Linux. It also gives additional information on the security tools available as well as the work done by the Debian security team.

Copyright Notice

Copyright © 2002 Javier Fernández-Sanguino Peña

Copyright © 2001 Alexander Reelsen, Javier Fernández-Sanguino Peña

Copyright © 2000 Alexander Reelsen

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 (<http://www.fsf.org/copyleft/fdl.html>) or any later version published by the Free Software Foundation. It is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY.

Contents

1	Introduction	1
1.1	Download the manual	1
1.2	Organizational Notes/Feedback	2
1.3	Prior knowledge	2
1.4	Things that need to be written (FIXME/TODO)	2
1.5	Changelog/History	5
1.5.1	Version 2.6 (september 2002)	5
1.5.2	Version 2.5 (september 2002)	5
1.5.3	Version 2.5 (august 2002)	5
1.5.4	Version 2.4	9
1.5.5	Version 2.3	9
1.5.6	Version 2.3	9
1.5.7	Version 2.2	10
1.5.8	Version 2.1	10
1.5.9	Version 2.0	10
1.5.10	Version 1.99	12
1.5.11	Version 1.98	12
1.5.12	Version 1.97	12
1.5.13	Version 1.96	12
1.5.14	Version 1.95	13
1.5.15	Version 1.94	13
1.5.16	Version 1.93	13
1.5.17	Version 1.92	13

1.5.18	Version 1.91	13
1.5.19	Version 1.9	14
1.5.20	Version 1.8	14
1.5.21	Version 1.7	14
1.5.22	Version 1.6	15
1.5.23	Version 1.5	15
1.5.24	Version 1.4	15
1.5.25	Version 1.3	16
1.5.26	Version 1.2	16
1.5.27	Version 1.1	16
1.5.28	Version 1.0	16
1.6	Credits and Thanks!	16
2	Before you begin	19
2.1	What do you want this system for?	19
2.2	Be aware of general security problems	19
2.3	How does Debian handle security?	21
3	Before and during the installation	23
3.1	Choose a BIOS password	23
3.2	Partitioning the system	23
3.2.1	Choose an intelligent partition scheme	23
3.3	Do not plug to the Internet until ready	25
3.4	Set a root password	25
3.5	Activate shadow passwords and MD5 passwords	25
3.6	Run the minimum number of services required	26
3.6.1	Disabling daemon services	27
3.6.2	Disabling inetd services	28
3.7	Install the minimum amount of software required	28
3.7.1	Removing Perl	29
3.8	Read the debian security mailing lists	31

4	After Installation	33
4.1	Change the BIOS (again)	33
4.2	Set a LILO or GRUB password	33
4.3	Remove root prompt on the kernel	34
4.4	Disallow floppy booting	35
4.5	Restricting console login access	36
4.6	Restricting system reboots through the console	36
4.7	Mounting partitions the right way	36
4.7.1	Setting <code>/tmp noexec</code>	37
4.7.2	Setting <code>/usr read-only</code>	38
4.8	Execute a security update	38
4.9	Subscribe to the Debian Security Announce mailing List	39
4.10	Providing secure user access	39
4.10.1	User authentication: PAM	39
4.10.2	Limiting resource usage: the <code>limits.conf</code> file	42
4.10.3	User Login actions: edit <code>/etc/login.defs</code>	42
4.10.4	Restricting ftp: editing <code>/etc/ftpusers</code>	44
4.10.5	Using <code>su</code>	44
4.10.6	Using <code>sudo</code>	44
4.10.7	Disallow remote administrative access	44
4.10.8	Restricting users's access	44
4.10.9	Hand-made user auditing	45
4.10.10	Complete user audit	46
4.10.11	Reviewing user profiles	46
4.10.12	Setting users <code>umasks</code>	46
4.10.13	Limiting what users can see/access	47
4.10.14	Generating user passwords	48
4.10.15	Checking user passwords	48
4.10.16	Logging off idle users	49
4.11	Using <code>tcpwrappers</code>	49
4.12	The importance of logs and alerts	50

4.12.1	Using and customising logcheck	51
4.12.2	Configuring where alerts are sent	52
4.12.3	Using a loghost	52
4.12.4	Log file permissions	53
4.13	Using chroot	54
4.14	Adding kernel patches	54
4.15	Protecting against buffer overflows	55
4.16	Secure file transfers	56
4.17	File System limits and control	56
4.17.1	Using quotas	56
4.17.2	chattr/lattr	57
4.17.3	Checking file system integrity	58
4.17.4	Setting up setuid check	59
4.18	Securing network access	59
4.18.1	Configuring kernel network features	59
4.18.2	Securing the network on boot-time	60
4.18.3	Configuring firewall features	62
4.18.4	Disabling weak-end hosts issues	62
4.18.5	Protecting against ARP attacks	63
4.19	Taking a snapshot of the system	64
4.20	Other recommendations	65
4.20.1	Do not use software depending on svgalib	65
5	Securing services running on your system	67
5.1	Securing ssh	68
5.1.1	Chrooting ssh	69
5.1.2	Ssh clients	70
5.1.3	Disallowing file transfers	70
5.2	Securing Squid	70
5.3	Securing FTP	71
5.4	Securing access to the X Window System	71

5.4.1	Check your display manager	73
5.5	Securing printing access (The lpd and lprng issue)	73
5.6	Securing the mail service	74
5.6.1	Configuring a Nullmailer	74
5.6.2	Providing secure access to mailboxes	76
5.6.3	Receiving mail securely	76
5.7	Securing BIND	77
5.7.1	Changing BIND's user	79
5.7.2	Chrooting the name server	81
5.8	Securing Apache	83
5.8.1	Disabling users from publishing web contents	83
5.8.2	Logfiles permissions	84
5.8.3	Published web files	84
5.9	Securing finger	84
5.10	General chroot and suid paranoia	85
5.10.1	Automaking chrooting programs	85
5.11	General cleartext password paranoia	85
5.12	Disabling NIS	86
5.13	Disabling RPC services	86
5.14	Adding firewall capabilities	87
5.14.1	Firewalling the local system	87
5.14.2	Using a firewall to protect other systems	88
5.14.3	Configuring the firewall	88
6	Automatic hardening of Debian systems	91
6.1	Harden	91
6.2	Bastille Linux	92
7	Debian Security Infrastructure	95
7.1	The Debian Security Team	95
7.2	Debian Security Advisories	96
7.3	Debian Security Build Infrastructure	97

7.3.1	Developer's guide to security updates	98
7.4	Package signing in Debian	100
7.4.1	The proposed scheme for package signature checks	101
7.4.2	Alternative per-package signing scheme	101
7.4.3	Checking distribution releases	102
8	Security tools in Debian	109
8.1	Remote vulnerability assesment tools	109
8.2	Network scanner tools	110
8.3	Internal audits	111
8.4	Auditing source code	111
8.5	Virtual Private Networks	111
8.5.1	Point to Point tunneling	112
8.6	Public Key Infrastructure (PKI)	113
8.7	SSL Infrastructure	113
8.8	Anti-virus tools	114
8.9	GPG agent	115
9	Before the compromise	117
9.1	Continuously update the system	117
9.1.1	Using Tiger to check for security updates	117
9.1.2	Avoid using the unstable branch	118
9.1.3	Avoid using the testing branch	119
9.1.4	Automatic updates in a Debian GNU/Linux system	119
9.2	Set up Intrusion Detection	120
9.2.1	Network based intrusion detection	121
9.2.2	Host based intrusion detection	121
9.3	Useful kernel patches	122
9.4	Avoiding root-kits	123
9.4.1	Loadable Kernel Modules (LKM)	123
9.4.2	Detecting root-kits	123
9.5	Genius/Paranoia Ideas — what you could do	124
9.5.1	Building a honeypot	126

10 After the compromise	127
10.1 General behavior	127
10.2 Backing up the system	128
10.3 Forensic analysis	128
11 Frequently asked Questions (FAQ)	131
11.1 Security in the Debian operating system	131
11.1.1 Is Debian more secure than X?	131
11.1.2 There are many Debian bugs in Bugtraq. Does this mean that it is very vulnerable?	132
11.1.3 Does Debian have any certification related to security?	132
11.1.4 Are there any hardening programs for Debian?	133
11.1.5 I want to run XYZ service, which one should I choose?	133
11.1.6 How can I make service XYZ more secure in Debian?	133
11.1.7 How can I remove all the banners for services?	134
11.1.8 Are all Debian packages safe?	134
11.1.9 Why are some log files/configuration files world-readable, isn't this insecure?	134
11.1.10 Why does /root/ (or UserX) have 755 permissions?	135
11.1.11 After installing a grsec/firewall, I started receiving many console messages! How do I remove them?	135
11.1.12 Operating system users and groups	136
11.1.13 Why is there a new group when I add a new user? (or Why does Debian give each user one group?)	139
11.1.14 Question regarding services and open ports	139
11.1.15 Common security issues	141
11.1.16 How do I accomplish setting up a service for my users without giving out shell accounts?	142
11.2 My system is vulnerable! (Are you sure?)	143
11.2.1 Vulnerability assessment scanner X says my Debian system is vulnerable!	143
11.2.2 I've seen an attack in my system's logs. Is my system compromised?	143
11.2.3 I have found strange 'MARK' lines in my logs: Am I compromised?	143
11.2.4 I found users using 'su' in my logs: Am I compromised?	144

11.2.5	I have found possible 'SYN flooding' in my logs: Am I under attack? . . .	144
11.2.6	I have found strange root sessions in my logs: Am I compromised?	145
11.2.7	I have suffered a break-in, what do I do?	145
11.2.8	How can I trace an attack?	146
11.2.9	Program X in Debian is vulnerable, what do I do?	146
11.2.10	The version number for a package indicates that I am still running a vul- nerable version!	146
11.2.11	Specific software	146
11.3	Questions regarding the Debian security team	147
11.3.1	What is a Debian Security Advisory (DSA)?	147
11.3.2	The signature on Debian advisories does not verify correctly!	147
11.3.3	How are security incidents handled in Debian?	147
11.3.4	How much time will it take Debian to fix vulnerability XXXX?	147
11.3.5	How is security handled for <code>testing</code> and <code>unstable</code> ?	148
11.3.6	I use an older version of Debian, is it supported by the Debian Security Team?	148
11.3.7	Why are there no official mirrors for <code>security.debian.org</code> ?	148
11.3.8	I've seen DSA 100 and DSA 102, what happened to DSA 101?	149
11.3.9	How can I reach the security team?	149
11.3.10	What difference is there between <code>security@debian.org</code> and <code>debian-security@lists.debian.org</code> ? 149	
11.3.11	How can I contribute to the Debian security team?	149
11.3.12	Who is the Security Team composed of?	150
11.3.13	Does the Debian Security team check every new package in Debian? . . .	150
A	The hardening process step by step	151
B	Configuration checklist	155
C	Setting up a stand-alone IDS	159
D	Setting up a bridge firewall	163
D.1	A bridge providing NAT and firewall capabilities	164
D.2	A bridge providing firewall capabilities	165
D.3	Basic IPtables rules	166

E	Sample script to change the default Bind installation.	169
F	Security update protected by a firewall	175
G	Chroot environment for SSH	177
G.1	Automatically making the environment (the easy way)	177
G.2	Patching SSH to enable chroot functionality	182
G.3	Handmade environment (the hard way)	184

Chapter 1

Introduction

One of the hardest things about writing security documents is that every case is unique. Two things you have to pay attention to are the threat environment and the security needs of the individual site, host, or network. For instance, the security needs of a home user are completely different from a network in a bank. While the primary threat a home user needs to face is the script kiddie type of cracker, a bank network has to worry about directed attacks. Additionally, the bank has to protect their customer's data with arithmetic precision. In short, every user has to consider the tradeoff between usability and security/paranoia.

Note that this manual only covers issues relating to software. The best software in the world can't protect you if someone can physically access the machine. You can place it under your desk, or you can place it in a hardened bunker with an army in front of it. Nevertheless the desktop computer can be much more secure (from a software point of view) than a physically protected one if the desktop is configured properly and the software on the protected machine is full of security holes. Obviously, you must consider both issues.

This document just gives an overview of what you can do to increase the security of your Debian GNU/Linux system. If you have read other documents regarding Linux security, you will find that there are common issues which might overlap with this document. However, this document does not try to be the ultimate source of information you will be using, it only tries to adapt this same information so that it is meaningful to a Debian GNU/Linux system. Different distributions do some things in different ways (startup of daemons is one example); here, you will find material which is appropriate for Debian's procedures and tools.

If you have comments, additions or suggestions, please mail them to Javier Fernández-Sanguino (<mailto:jfs@computer.org>) (alternate address: jfs@debian.org) and they will be incorporated into this manual.

1.1 Download the manual

You can download or view the newest version of the Securing Debian Manual from the Debian Documentation Project (<http://www.debian.org/doc/manuals/securing-debian-howto/>).

Feel free to check out the version control system through its CVS server (<http://cvs.debian.org/ddp/manuals.sgml/securing-howto/?cvsroot=debian-doc>).

You can download also a text version (<http://www.debian.org/doc/manuals/securing-debian-howto/securing-debian-howto.txt>) from the Debian Documentation's Project site. Other formats, like PDF, are not (yet) provided. However, you can download or install the `harden-doc` (<http://packages.debian.org/harden-doc>) package which provides this same document in HTML, txt and PDF formats.

1.2 Organizational Notes/Feedback

Now to the official part. At the moment I (Alexander Reelsen) wrote most paragraphs of this manual, but in my opinion this should not stay the case. I grew up and live with free software, it is part of my everyday use and I guess yours, too. I encourage everybody to send me feedback, hints additions or any other suggestions, you might have.

If you think, you can maintain a certain section or paragraph better, then write to the document maintainer and you are welcome to do it. Especially if you find a section marked as `FIXME`, that means the authors did not have the time yet or the needed knowledge about the topic, drop them a mail immediately.

The topic of this manual makes it quite clear that it is important to keep it up to date, and you can do your part. Please contribute.

1.3 Prior knowledge

The installation of Debian GNU/Linux is not very difficult and you should have been able to install it. If you already have some knowledge about Linux or other Unices and you are a bit familiar with basic security, it will be easier to understand this manual, as this document cannot explain every little detail of a feature (otherwise this would have been a book instead of a manual). If you are not that familiar, however, you might want to take a look at 'Be aware of general security problems' on page 19 for where to find more in-depth information.

1.4 Things that need to be written (FIXME/TODO)

- Write about remote monitoring tools (to check for system availability) such as `monit`, `daemontools` and `mon`. See <http://linux.oreillynet.com/pub/a/linux/2002/05/09/sysadminguide.html>.
- Consider writing a section on how to build Debian-based network appliances (with information such as the base system, `equivs` and `FAI`).
- Check if <http://rr.sans.org/linux/hardening.php> has relevant info not yet covered here.

- Add Information on how to set up a laptop with Debian http://rr.sans.org/linux/debian_laptop.php.
- Add information on how to set up a firewall using Debian GNU/Linux. The section regarding firewalling is oriented currently towards a single system (not protecting others...) also talk on how to test the setup.
- Add information on setting up a proxy firewall with Debian GNU/Linux stating specifically which packages provide proxy services (like `xwp`, `xproxy`, `ftp-proxy`, `redir`, `smtpd`, `nntp-cache`, `dnrd`, `jftpgw`, `oops`, `pnsd`, `perdition`, `transproxy`, `tsocks`). Should point to the manual for any other info. Also note that `zorp` is not (yet) available as a Debian package but *is* a proxy firewall (they provide Debian packages upstream).
- Information on service configuration with `file-rc`
- Check all the reference URLs and remove/fix those no longer available.
- Add information on available replacements (in Debian) for common servers which are useful for limited functionality. Examples:
 - local `lpr` with `cups` (package)?
 - remote `lpr` with `lpr`
 - `bind` with `dnrd`/`maradns`
 - `apache` with `dhttpd`/`thttpd`/`wn` (tux?)
 - `exim`/`sendmail` with `ssmtpd`/`smtpd`/`postfix`
 - `squid` with `tinyproxy`
 - `ftpd` with `oftpd`/`vsftp`
 - ...
- More information regarding security-related kernel patches in Debian, including the ones shown above and specific information on how to enable these patches in a Debian system.
 - Linux Intrusion Detection (`lids-2.2.19`)
 - Linux Trustees (in package `trustees`)
 - NSA Enhanced Linux (<http://www.coker.com.au/selinux/>)
 - `kernel-patch-2.2.18-openwall` (<http://packages.debian.org/kernel-patch-2.2.18-openwall>)
 - `kernel-patch-2.2.19-harden`
 - Linux capabilities (in package `lcap`)
 - `kernel-patch-freeswan`, `kernel-patch-int`
- Details of turning off unnecessary network services (besides `inetd`), it is partly in the hardening procedure but could be broadened a bit.
- Information regarding password rotation which is closely related to policy.

- Policy, and educating users about policy.
- More about tcpwrappers, and wrappers in general?
- `hosts.equiv` and other major security holes.
- Issues with file sharing servers such as Samba and NFS?
- `suidmanager/dpkg-statoverrides`.
- `lpr` and `lprng`.
- Switching off the gnome IP things.
- Talk about `pam_chroot` (see <http://lists.debian.org/debian-security/2002/debian-security-200205/msg00011.html>) and its usefulness to limit users. Introduce information related to <http://online.securityfocus.com/infocus/1575>. `Pdmenu`, for example is available in Debian (while `as` flash is not).
- Talk about chrooting services, some more info on <http://www.linuxfocus.org/English/January2002/article225.shtml>, <http://www.networkdweebs.com/chroot.html> and http://www.linuxsecurity.com/feature_stories/feature_story-99.html
- Talk about programs to make chroot jails. `Compartment` and `chrootuid` are waiting in incoming. Some others (`makejail`, `jailer`) could also be introduced.
- Add information provided by Karl Hegbloom regarding chrooting Bind 9, see http://people.pdxlinux.org/~karlheg/Secure_Bind9_uHOWTO/Secure_Bind_9_uHOWTO.xhtml.
- Add information provided by Pedro Zornenon to chrooting Bind 8 only for potato though :(, see <http://people.debian.org/~pzn/howto/chroot-bind.sh.txt> (include the whole script?).
- More information regarding log analysis software (i.e. `logcheck` and `logcolorise`).
- 'advanced' routing (traffic policing is security related)
- limiting `ssh` access to running certain commands.
- using `dpkg-statoverride`.
- secure ways to share a CD burner among users.
- secure ways of providing networked sound in addition to network display capabilities (so that X clients' sounds are played on the X server's sound hardware)
- securing web browsers.
- setting up `ftp` over `ssh`.
- using crypto loopback file systems.

- encrypting the entire file system.
- steganographic tools.
- setting up a PKA for an organization.
- using LDAP to manage users. There is a HOWTO of ldap+kerberos for Debian at www.bayour.com written by Turbo Fredrikson.
- How to remove information of reduced utility in production systems such as `/usr/share/doc`, `/usr/share/man` (yes, security by obscurity).

1.5 Changelog/History

1.5.1 Version 2.6 (september 2002)

Changes by Chris Tillman, tillman@voicetrak.com.

- Changed around to improve grammar/spelling.
- `s/host.deny/hosts.deny/` (1 place)
- Applied Larry Holish's patch (quite big, fixes a lot of FIXMEs)

1.5.2 Version 2.5 (september 2002)

Changes by Javier Fernández-Sanguino Peña (me).

- Fixed minor typos submitted by Thiemo Nagel.
- Added a footnote suggested by Thiemo Nagel.
- Fixed an URL link.

1.5.3 Version 2.5 (august 2002)

Changes by Javier Fernández-Sanguino Peña (me). There were many things waiting on my inbox (as far back as February) to be included, so I'm going to tag this the *back from honeymoon* release :)

- Applied a patch contributed by Philippe Gaspar regarding the Squid which also kills a FIXME.
- Yet another FAQ item regarding service banners taken from the debian-security mailing list (thread "Telnet information" started 26th July 2002).

- Added a note regarding use of CVE cross references in the *How much time does the Debian security team...* FAQ item.
- Added a new section regarding ARP attacks contributed by Arnaud “Arhuman” Assad.
- New FAQ item regarding dmesg and console login by the kernel.
- Small tidbits of information to the signature-checking issues in packages (it seems to not have gotten past beta release).
- New FAQ item regarding vulnerability assessment tools false positives.
- Added new sections to the chapter that contains information on package signatures and reorganised it as a new *Debian Security Infrastructure* chapter.
- New FAQ item regarding Debian vs. other Linux distributions.
- New section on mail user agents with GPG/PGP functionality in the security tools chapter.
- Clarified how to enable MD5 passwords in woody, added a pointer to PAM as well as a note regarding the max definition in PAM.
- Added a new appendix on how to create chroot environments (after fiddling a bit with makejail and fixing, as well, some of its bugs), integrated duplicate information in all the appendix.
- Added some more information regarding SSH chrooting and its impact on secure file transfers. Some information has been retrieved from the debian-security mailing list (June 2002 thread: *secure file transfers*).
- New sections on how to do automatic updates on Debian systems as well as the caveats of using testing or unstable regarding security updates.
- New section regarding keeping up to date with security patches in the *Before compromise* section as well as a new section about the debian-security-announce mailing list.
- Added information on how to automatically generate strong passwords.
- New section regarding login of idle users.
- Reorganised the securing mail server section based on the *Secure/hardened/minimal Debian* (or “*Why is the base system the way it is?*”) thread on the debian-security mailing list (May 2002).
- Reorganised the section on kernel network parameters, with information provided in the debian-security mailing list (May 2002, *syn flood attacked?* thread) and added a new FAQ item as well.
- New section on how to check users passwords and which packages to install for this.
- New section on PPTP encryption with Microsoft clients discussed in the debian-security mailing list (April 2002).

- Added a new section describing what problems are there when binding any given service to a specific IP address, this information was written based on the bugtraq mailing list in the thread: *Linux kernel 2.4 "weak end host" issue (previously discussed on debian-security as "arp problem")* (started on May 9th 2002 by Felix von Leitner).
- Added information on ssh protocol version 2.
- Added two subsections related to Apache secure configuration (the things specific to Debian, that is).
- Added a new FAQ related to raw sockets, one related to /root, an item related to users' groups and another one related to log and configuration files permissions.
- Added a pointer to a bug in libpam-cracklib that might still be open... (need to check)
- Added more information regarding forensics analysis (pending more information on packet inspection tools such as tcpflow).
- Changed the "what should I do regarding compromise" into a bullet list and included some more stuff.
- Added some information on how to set up the Xscreensaver to lock the screen automatically after the configured timeout.
- Added a note related to the utilities you should not install in the system. Included a note regarding Perl and why it cannot be easily removed in Debian. The idea came after reading Intersect's documents regarding Linux hardening.
- Added information on lvm and journalling file systems, ext3 recommended. The information there might be too generic, however.
- Added a link to the online text version (check).
- Added some more stuff to the information on firewalling the local system, triggered by a comment made by Hubert Chan in the mailing list.
- Added more information on PAM limits and pointers to Kurt Seifried's documents (related to a post by him to bugtraq on April 4th 2002 answering a person that had "discovered" a vulnerability in Debian GNU/Linux related to resource starvation).
- As suggested by Julián Muñoz, provided more information on the default Debian umask and what a user can access if he has been given a shell in the system (scary, huh?)
- Included a note in the BIOS password section due to a comment from Andreas Wohlfeld.
- Included patches provided by Alfred E. Heggstad fixing many of the typos still present in the document.
- Added a pointer to the changelog in the Credits section since most people who contribute are listed here (and not there).

- Added a few more notes to the `chattr` section and a new section after installation talking about system snapshots. Both ideas were contributed by Kurt Pomeroy.
- Added a new section after installation just to remind users to change the boot-up sequence.
- Added some more TODO items provided by Korn Andras.
- Added a pointer to the NIST's guidelines on how to secure DNS provided by Daniel Quinlan.
- Added a small paragraph regarding Debian's SSL certificates infrastructure.
- Added Daniel Quinlan's suggestions regarding `ssh` authentication and `exim`'s relay configuration.
- Added more information regarding securing `bind` including changes suggested by Daniel Quinlan and an appendix with a script to make some of the changes commented on in that section.
- Added a pointer to another item regarding `Bind` chrooting (needs to be merged).
- Added a one liner contributed by Cristian Ionescu-Ildbohrn to retrieve packages with `tcpwrappers` support.
- Added a little bit more info on Debian's default PAM setup.
- Included a FAQ question about using PAM to provide services without shell accounts.
- Moved two FAQ items to another section and added a new FAQ regarding attack detection (and compromised systems).
- Included information on how to set up a bridge firewall (including a sample Appendix). Thanks go to Francois Bayar who sent this to me in March.
- Added a FAQ regarding the `syslogd`'s *MARK heartbeat* from a question answered by Noah Meyerhans and Alain Tesio in December 2001.
- Included information on buffer overflow protection as well as some information on kernel patches.
- Added more information (and reorganised) the firewall section. Updated the information regarding the `iptables` package and the firewall generators available.
- Reorganized the information regarding log checking, moved `logcheck` information from host intrusion detection to that section.
- Added some information on how to prepare a static package for `bind` for chrooting (untested).
- Added a FAQ item regarding some specific servers/services (could be expanded with some of the recommendations from the `debian-security` list).

- Added some information on RPC services (and when it's necessary).
- Added some more information on capabilities (and what lcap does). Is there any good documentation on this? I haven't found any documentation on my 2.4 kernel.
- Fixed some typos.

1.5.4 Version 2.4

Changes by Javier Fernández-Sanguino Peña.

- Rewritten part of the BIOS section.

1.5.5 Version 2.3

Changes by Javier Fernández-Sanguino Peña.

- Wrapped most file locations with the file tag.
- Fixed typo noticed by Edi Stojicevi.
- Slightly changed the remote audit tools section.
- Added some todo items.
- Added more information regarding printers and cups config file (taken from a thread on debian-security).
- Added a patch submitted by Jesus Climent regarding access of valid system users to Proftpd when configured as anonymous server.
- Small change on partition schemes for the special case of mail servers.
- Added Hacking Linux Exposed to the books section.
- Fixed directory typo noticed by Eduardo Pérez Ureta.
- Fixed /etc/ssh typo in checklist noticed by Edi Stojicevi.

1.5.6 Version 2.3

Changes by Javier Fernández-Sanguino Peña.

- Fixed location of dpkg conffile.
- Remove Alexander from contact information.

- Added alternate mail address.
- Fixed Alexander mail address (even if commented out).
- Fixed location of release keys (thanks to Pedro Zorzenon for pointing this out).

1.5.7 Version 2.2

Changes by Javier Fernández-Sanguino Peña.

- Fixed typos, thanks to Jamin W. Collins.
- Added a reference to apt-extracttemplate manpage (documents the APT::ExtractTemplate config).
- Added section about restricted SSH. Information based on that posted by Mark Janssen, Christian G. Warden and Emmanuel Lacour on the debian-security mailing list.
- Added information on anti-virus software.
- Added a FAQ: su logs due to the cron running as root.

1.5.8 Version 2.1

Changes by Javier Fernández-Sanguino Peña.

- Changed FIXME from lshell thanks to Oohara Yuuma.
- Added package to sXid and removed comment since it **is** available.
- Fixed a number of typos discovered by Oohara Yuuma.
- ACID is now available in Debian (in the acidlab package) thanks to Oohara Yuuma for noticing.
- Fixed LinuxSecurity links (thanks to Dave Wreski for telling).

1.5.9 Version 2.0

Changes by Javier Fernández-Sanguino Peña. I wanted to change to 2.0 when all the FIXMEs were, er, fixed but I ran out of 1.9X numbers :(

- Converted the HOWTO into a Manual (now I can properly say RTFM)
- Added more information regarding tcp wrappers and Debian (now many services are compiled with support for them so it's no longer an inetd issue).

- Clarified the information on disabling services to make it more consistent (rpc info still referred to update-rc.d)
- Added small note on lprng.
- Added some more info on compromised servers (still very rough)
- Fixed typos reported by Mark Bucciarelli.
- Added some more steps in password recovery to cover the cases when the admin has set paranoid-mode=on.
- Added some information to set paranoid-mode=on when login in console.
- New paragraph to introduce service configuration.
- Reorganised the *After installation* section so it is more broken up into several issues and it's easier to read.
- Wrote information on how to set up firewalls with the standard Debian 3.0 setup (iptables package).
- Small paragraph explaining why installing connected to the Internet is not a good idea and how to avoid this using Debian tools.
- Small paragraph on timely patching referencing to IEEE paper.
- Appendix on how to set up a Debian snort box, based on what Vladimir sent to the debian-security mailing list (September 3rd 2001)
- Information on how logcheck is set up in Debian and how it can be used to set up HIDS.
- Information on user accounting and profile analysis.
- Included apt.conf configuration for read-only /usr copied from Olaf Meeuwissen's post to the debian-security mailing list
- New section on VPN with some pointers and the packages available in Debian (needs content on how to set up the VPNs and Debian-specific issues), based on Jaroslav Tabor's and Samuli Suonpaa's post to debian-security.
- Small note regarding some programs to automatically build chroot jails
- New FAQ item regarding identd based on a discussion in the debian-security mailing list (February 2002, started by Johannes Weiss).
- New FAQ item regarding inetd based on a discussion in the debian-security mailing list (February 2002).
- Introduced note on rconf in the "disabling services" section.
- Varied the approach regarding LKM, thanks to Philippe Gaspar
- Added pointers to CERT documents and Counterpane resources

1.5.10 Version 1.99

Changes by Javier Fernández-Sanguino Peña.

- Added a new FAQ item regarding time to fix security vulnerabilities.
- Reorganised FAQ sections.
- Started writing a section regarding firewalling in Debian GNU/Linux (could be broadened a bit)
- Fixed typos sent by Matt Kraai
- Fixed DNS information
- Added information on whisker and nbtscan to the auditing section.
- Fixed some wrong URLs

1.5.11 Version 1.98

Changes by Javier Fernández-Sanguino Peña.

- Added a new section regarding auditing using Debian GNU/Linux.
- Added info regarding finger daemon taken from the security mailing list.

1.5.12 Version 1.97

Changes by Javier Fernández-Sanguino Peña.

- Fixed link for Linux Trustees
- Fixed typos (patches from Oohara Yuuma and Pedro Zorzenon)

1.5.13 Version 1.96

Changes by Javier Fernández-Sanguino Peña.

- Reorganized service installation and removal and added some new notes.
- Added some notes regarding using integrity checkers as intrusion detection tools.
- Added a chapter regarding package signatures.

1.5.14 Version 1.95

Changes by Javier Fernández-Sanguino Peña.

- Added notes regarding Squid security sent by Philippe Gaspar.
- Fixed rootkit links thanks to Philippe Gaspar.

1.5.15 Version 1.94

Changes by Javier Fernández-Sanguino Peña.

- Added some notes regarding Apache and Lpr/lpng.
- Added some information regarding noexec and read-only partitions.
- Rewrote how users can help in Debian security issues (FAQ item).

1.5.16 Version 1.93

Changes by Javier Fernández-Sanguino Peña.

- Fixed location of mail program.
- Added some new items to the FAQ.

1.5.17 Version 1.92

Changes by Javier Fernández-Sanguino Peña.

- Added a small section on how Debian handles security
- Clarified MD5 passwords (thanks to 'rocky')
- Added some more information regarding harden-X from Stephen van Egmond
- Added some new items to the FAQ

1.5.18 Version 1.91

Changes by Javier Fernández-Sanguino Peña.

- Added some forensics information sent by Yotam Rubin.
- Added information on how to build a honeynet using Debian GNU/Linux.
- Added some more TODOS.
- Fixed more typos (thanks Yotam!)

1.5.19 Version 1.9

Changes by Javier Fernández-Sanguino Peña.

- Added patch to fix misspellings and some new information (contributed by Yotam Rubin)
- Added references to other online (and offline) documentation both in a section (see ‘Be aware of general security problems’ on page 19) by itself and inline in some sections.
- Added some information on configuring Bind options to restrict access to the DNS server.
- Added information on how to automatically harden a Debian system (regarding the harden package and bastille).
- Removed some done TODOs and added some new ones.

1.5.20 Version 1.8

Changes by Javier Fernández-Sanguino Peña.

- Added the default user/group list provided by Joey Hess to the debian-security mailing list.
- Added information on LKM root-kits (‘Loadable Kernel Modules (LKM)’ on page 123) contributed by Philippe Gaspar.
- Added information on Proftpd contributed by Emmanuel Lacour.
- Recovered the checklist Appendix from Era Eriksson.
- Added some new TODO items and removed other fixed ones.
- Manually included Era’s patches since they were not all included in the previous version.

1.5.21 Version 1.7

Changes by Era Eriksson.

- Typo fixes and wording changes

Changes by Javier Fernández-Sanguino Peña.

- Minor changes to tags in order to keep on removing the tt tags and substitute prgn/package tags for them.

1.5.22 Version 1.6

Changes by Javier Fernández-Sanguino Peña.

- Added pointer to document as published in the DDP (should supersede the original in the near future)
- Started a mini-FAQ (should be expanded) with some questions recovered from my mailbox.
- Added general information to consider while securing.
- Added a paragraph regarding local (incoming) mail delivery.
- Added some pointers to more information.
- Added information regarding the printing service.
- Added a security hardening checklist.
- Reorganized NIS and RPC information.
- Added some notes taken while reading this document on my new Visor :)
- Fixed some badly formatted lines.
- Fixed some typos.
- Added a Genius/Paranoia idea contributed by Gaby Schilders.

1.5.23 Version 1.5

Changes by Josip Rodin and Javier Fernández-Sanguino Peña.

- Added paragraphs related to BIND and some FIXMEs.

1.5.24 Version 1.4

- Small setuid check paragraph
- Various minor cleanups
- Found out how to use `sgml2txt -f` for the txt version

1.5.25 Version 1.3

- Added a security update after installation paragraph
- Added a proftpd paragraph
- This time really wrote something about XDM, sorry for last time

1.5.26 Version 1.2

- Lots of grammar corrections by James Treacy, new XDM paragraph

1.5.27 Version 1.1

- Typo fixes, miscellaneous additions

1.5.28 Version 1.0

- Initial release

1.6 Credits and Thanks!

- Alexander Reelsen wrote the original document.
- Javier Fernández-Sanguino added more info to the original doc.
- Robert van der Meulen provided the quota paragraphs and many good ideas.
- Ethan Benson corrected the PAM paragraph and had some good ideas.
- Dariusz Puchalak contributed some information to several chapters.
- Gaby Schilders contributed a nice Genius/Paranoia idea.
- Era Eriksson smoothed out the language in a lot of places and contributed the checklist appendix.
- Philippe Gaspar wrote the LKM information.
- Yotam Rubin contributed fixes for many typos as well as information regarding bind versions and md5 passwords.
- All the people who made suggestions for improvement that (eventually) got included here (see 'Changelog/History' on page 5)

- (Alexander) All the folks who encouraged me to write this HOWTO (which was later turned into a Manual).
- The whole Debian project.

Chapter 2

Before you begin

2.1 What do you want this system for?

Securing Debian is not very different from securing any other system; in order to do it properly, you must first decide what you intend to do with it. After this, you will have to consider that the following tasks need to be taken care of if you want a really secure system.

You will find that this manual is written from the bottom up, that is, you will read some information on tasks to do before, during and after you install your Debian system. The tasks can also be thought of as:

- Decide which services you need and limit your system to those. This includes deactivating/uninstalling unneeded services, and adding firewall-like filters, or tcpwrappers.
- Limit users and permissions in your system.
- Harden offered services so that, in the event of a service compromise, the impact to your system is minimized.
- Use appropriate tools to guarantee that unauthorized use is detected so that you can take appropriate measures.

2.2 Be aware of general security problems

The following manual does not (usually) go into the details on why some issues are considered security risks. However, you might want to have a better background regarding general UNIX and (specific) Linux security. Take some time to read over security related documents in order to make informed decisions when you are encountered with different choices. Debian GNU/Linux is based on the Linux kernel, so much of the information regarding Linux, as well as from other distributions and general UNIX security also apply to it (even if the tools used, or the programs available, differ).

Some useful documents include:

- The Linux Security HOWTO (<http://www.linuxdoc.org/HOWTO/Security-HOWTO.html>) (also available at LinuxSecurity (<http://www.linuxsecurity.com/docs/LDP/Security-HOWTO.html>)) is one of the best references regarding general Linux Security.
- The Security Quick-Start HOWTO for Linux (<http://www.linuxsecurity.com/docs/LDP/Security-Quickstart-HOWTO/>) is also a very good starting point for novice users (both to Linux and security).
- The Linux Security Administrator's Guide (<http://seifried.org/lasg/>) (provided in Debian through the `lasg` package) is a complete guide that touches all the issues related to security in Linux, from kernel security to VPNs. It is somewhat obsolete (not updated since 1999) and has been superseded by the Linux Security Knowledge Base (<http://seifried.org/lskb>). This documentation is also provided in Debian through the `lskb` package.
- Kurt Seifried's Securing Linux Step by Step (<http://seifried.org/security/os/linux/20020324-securing-linux-step-by-step.html>).
- In *Securing and Optimizing Linux: RedHat Edition* (http://www.linuxdoc.org/links/p_books.html#securing_linux) you can find a similar document to this manual but related to RedHat, some of the issues are not distribution-specific and also apply to Debian.
- IntersectAlliance has published some documents that can be used as reference cards on how to harden linux servers (and their services), the documents are available at their site (<http://www.intersectalliance.com/projects/index.html>).
- For network administrators, a good reference for building a secure network is the *Securing your Domain HOWTO* (<http://www.linuxsecurity.com/docs/LDP/Securing-Domain-HOWTO.html>).
- If you want to evaluate the programs you are going to use (or want to build up some new ones) you should read the *Secure Programs HOWTO* (<http://www.linuxdoc.org/HOWTO/Secure-Programs-HOWTO.html>).
- If you are considering installing Firewall capabilities, you should read the *Firewall HOWTO* (<http://www.linuxdoc.org/HOWTO/Firewall-HOWTO.html>) and the *IPCHAINS HOWTO* (<http://www.linuxdoc.org/HOWTO/IPCHAINS-HOWTO.html>) (for kernels previous to 2.4).
- Finally, a good card to keep handy is the *Linux Security ReferenceCard* (<http://www.linuxsecurity.com/docs/QuickRefCard.pdf>)

In any case, there is more information regarding the services explained here (NFS, NIS, SMB. . .) in many of the HOWTOs of the Linuxdoc Project (<http://www.linuxdoc.org/>). Some of these documents speak on the security side of a given service, so be sure to take a look there too.

The HOWTO documents from the Linux Documentation Project are available in Debian GNU/Linux through the installation of the `doc-linux-text` (text version) or `doc-linux-html` (html

version). After installation these documents will be available at the `/usr/share/doc/HOWTO/en-txt` and `/usr/share/doc/HOWTO/en-html` directories, respectively.

Other recommended Linux books:

- Maximum Linux Security : A Hacker's Guide to Protecting Your Linux Server and Network. Anonymous. Paperback - 829 pages. Sams Publishing. ISBN: 0672313413. July 1999.
- Linux Security By John S. Flowers. New Riders; ISBN: 0735700354. March 1999
- Hacking Linux Exposed (http://www.linux.org/books/ISBN_0072127732.html) By Brian Hatch. McGraw-Hill Higher Education. ISBN 0072127732. April, 2001

Other books (which might be related to general issues regarding UNIX and security and not Linux specific):

- Practical Unix and Internet Security (2nd Edition) (<http://www.ora.com/catalog/puis/noframes.html>) Garfinkel, Simpson, and Spafford, Gene; O'Reilly Associates; ISBN 0-56592-148-8; 1004pp; 1996.
- Firewalls and Internet Security Cheswick, William R. and Bellovin, Steven M.; Addison-Wesley; 1994; ISBN 0-201-63357-4; 320pp.

Some useful Web sites to keep up to date regarding security:

- NIST Security Guidelines (<http://csrc.nist.gov/fasp/index.html>).
- Security Focus (<http://www.securityfocus.com>) the server that hosts the Bugtraq vulnerability database and list, and provides general security information, news and reports.
- Linux Security (<http://www.linuxsecurity.com/>). General information regarding Linux security (tools, news...). Most useful is the main documentation (<http://www.linuxsecurity.com/resources/documentation-1.html>) page.
- Linux firewall and security site (<http://www.linux-firewall-tools.com/linux/>). General information regarding Linux firewalls and tools to control and administrate them.

2.3 How does Debian handle security?

Just so you have a general overview of security in Debian GNU/Linux you should take note of the different issues that Debian tackles in order to provide an overall secure system:

- Debian problems are always handled openly, even security related. Security issues are discussed openly on the debian-security mailing list. Debian Security Advisories are sent to public mailing lists (both internal and external) and are published on the public server. As the Debian Social Contract (http://www.debian.org/social_contract) states:
We Won't Hide Problems
We will keep our entire bug-report database open for public view at all times. Reports that users file on-line will immediately become visible to others.
- Debian follows security issues closely. The security team checks many security related sources, the most important being Bugtraq (<http://www.securityfocus.com/cgi-bin/vulns.pl>), on the lookout for packages with security issues that might be included in Debian.
- Security updates are the first priority. When a security problem arises in a Debian package, the security update is prepared as fast as possible and distributed for our stable and unstable releases, including all architectures.
- Information regarding security is centralized in a single point, <http://security.debian.org/>.
- Debian is always trying to improve the overall security of the distribution for starting new projects, like automatic package signature verification mechanisms.
- Debian provides a number of useful security related tools for system administration and monitoring. Developers try to tightly integrate these tools with the distribution in order to make them a better suite to enforce local security policies. Tools include: integrity checkers, auditing tools, hardening tools, firewall tools, intrusion detection tools, etc.
- Package maintainers are aware of security issues. This leads to many “secure by default” service installations which might put some limits, sometimes, on its normal use. However, Debian does try to balance security issues and ease of administration, systems are not installed de-activated, for example, like the BSD family distributions. In any case, some special security issues, like `setuid` programs, are part of the Debian Policy (<http://www.debian.org/doc/debian-policy/>).

This document as well, tries to enforce a better distribution security-wise, by publishing security information specific to Debian which complements other information-security documents related to the tools used by Debian or the operating system itself (see ‘Be aware of general security problems’ on page 19).

Chapter 3

Before and during the installation

3.1 Choose a BIOS password

Before you install any operating system on your computer, set up a BIOS password. After installation (once you have enabled bootup from the hard disk) you should go back to the BIOS and change the boot sequence to disable booting from floppy, cdrom and other devices that shouldn't boot. Otherwise a cracker only needs physical access and a boot disk to access your entire system.

Disabling booting unless a password is supplied is even better. This can be very effective if you run a server, because it is not rebooted very often. The downside to this tactic is that rebooting requires human intervention which can cause problems if the machine is not easily accessible.

Note: many BIOSes have well known default master passwords, and applications also exist to retrieve the passwords from the BIOS. Corollary: don't depend on this measure to secure console access to system.

3.2 Partitioning the system

3.2.1 Choose an intelligent partition scheme

An intelligent partition scheme depends on the how the machine is used. A good rule of thumb is to be fairly liberal with your partitions and to pay attention to the following factors:

- Any directory tree which a user has write permissions to, such as e.g. `/home` and `/tmp`, should be on a separate partition. This reduces the risk of a user DoS by filling up your `/` mount point and rendering the system unusable. (Note: this is not strictly true, since there is always some space reserved for root which a normal user cannot fill.)
- Any partition which can fluctuate, e.g. `/var` (especially `/var/log`) should also be on a separate partition. On a Debian system, you should create `/var` a little bit bigger than on

other systems, because downloaded packages (the apt cache) are stored in `/var/cache/apt/archives`.

- Any partition where you want to install non-distribution software should be on a separate partition. According to the File Hierarchy Standard, this is `/opt` or `/usr/local`. If these are separate partitions, they will not be erased if you (have to) reinstall Debian itself.
- From a security point of view, it makes sense to try to move static data to its own partition, and then mount that partition read-only. Better yet, put the data on read-only media. See below for more details.

In the case of a mail server it is important to have a separate partition for the mail spool. Remote users (either knowingly or unknowingly) can fill the mail spool (`/var/mail` and/or `/var/spool/mail`). If the spool is on a separate partition, this situation will not render the system unusable. Otherwise (if the spool directory is on the same partition as `/var`) the system might have important problems: log entries will not be created, packages can not be installed, and some programs might even have problems starting up (if they use `/var/run`).

Also, for partitions in which you cannot be sure of the needed space, installing Logical Volume Manager (`lvm-common` and the needed binaries for your kernel, this might be either `lvm10`, `lvm6`, or `lvm5`). Using `lvm`, you can create volume groups that expand multiple physical volumes.

Selecting the appropriate file systems

During the system partitioning you also have to decide which file system you want to use. The default file system selected in the Debian installation for Linux partitions is `ext2`. However, it is recommended you switch to a journalling file system, such as `ext3`, `reiserfs`, `jfs` or `xfs`, to minimize the problems derived from a system crash in the following cases:

- for laptops in all the file systems installed. That way if you run out of battery unexpectedly or the system freezes due to a hardware issue (such as X configuration which is somewhat common) you will be less likely to lose data during a hardware reboot.
- for production systems which store large amounts of data (like mail servers, ftp servers, network file systems...) it is recommended on these partitions. That way, in the event of a system crash, the server will take less time to recover and check the file systems, and data loss will be less likely.

Leaving aside the performance issues regarding journalling file systems (since this sometimes can turn into a religious war), it is usually better to use the `ext3` file system. The reason for this is that it is backwards compatible with `ext2`, so if there are any issues with the journalling you can disable it and still have a working file system. Also, if you need to recover the system with a bootdisk (or CDROM) you do not need a custom kernel. If the kernel is 2.4 `ext3` support is already available, if it is a 2.2 kernel you will be able to boot the file system even if you lose

journaling capabilities. If you are using other journaling file systems you will find that you might not be able to recover unless you have a 2.4 kernel with the needed modules built-in. If you are stuck with a 2.2 kernel in the rescue disk it might even be more difficult to have it access `reiserfs` or `xf`s.

In any case, data integrity might be better under `ext3` since it does file-data journaling while others do only meta-data journaling, see <http://lwn.net/2001/0802/a/ext3-modes.php3>.

3.3 Do not plug to the Internet until ready

The system should not be immediately connected to the Internet during installation. This could sound stupid but network installation is a common method. Since the system will install and activate services immediately, if the system is connected to the Internet and the services are not properly configured you are opening it to attack.

Also note that some services might have security vulnerabilities not fixed in the packages you are using for installation. This is usually true if you are installing from old media (like CD-ROMs). In this case, the system could even be compromised before you finish installation!

Since Debian installation and upgrades can be done over the Internet you might think it is a good idea to use this feature on installation. If the system is going to be directly connected to the Internet (and not protected by a firewall or NAT), it is best to install without connection to the Internet, using a local packages mirror for both the Debian package sources and the security updates. You can set up package mirrors by using another system connected to the Internet with Debian-specific tools (if it's a Debian system) like `apt-move` or `apt-proxy`, or other common mirroring tools, to provide the archive to the installed system. If you cannot do this, you can set up firewall rules to limit access to the system while doing the update (see 'Security update protected by a firewall' on page 175).

3.4 Set a root password

Setting a good root password is the most basic requirement for having a secure system. See `passwd(1)` for some hints on how to create good passwords. You can also use an automatic password generation program to do this for you (see 'Generating user passwords' on page 48).

FIXME: Add pointers to information about good passwords.

3.5 Activate shadow passwords and MD5 passwords

At the end of the installation, you will be asked if shadow passwords should be enabled. Answer yes to this question, so passwords will be kept in the file `/etc/shadow`. Only the root user and the group shadow have read access to this file, so no users will be able to grab a

copy of this file in order to run a password cracker against it. You can switch between shadow passwords and normal passwords at any time by using `shadowconfig`.

Read more on Shadow passwords in Shadow Password (<http://www.linuxdoc.org/HOWTO/Shadow-Password-HOWTO.html>) (`/usr/share/doc/HOWTO/en-txt/Shadow-Password.txt.gz`).

Furthermore, you are queried during installation whether you want to use MD5 hashed passwords. This is generally a very good idea since it allows longer passwords and better encryption. MD5 allows for passwords longer than 8 characters. This, if used wisely, can make it more difficult for attackers to brute-force the system's passwords. Regarding MD5 passwords, this is the default option when installing the latest `passwd` package. You can change this anytime after installation by doing `dpkg-reconfigure -p low passwd`. You can recognize md5 passwords in the `/etc/shadow` file by their `1` prefix.

This, as a matter of fact, modifies all files under `/etc/pam.d` by substituting the password line and include md5 in it:

```
password required pam_unix.so md5 nullok obscure min=6 max=16
```

If `max` is not set over 8 the change will not be useful at all. For more information on this read 'User authentication: PAM' on page 39.

Note: the default configuration in Debian, even when activating MD5 passwords, does not modify the previously set `max` value.

3.6 Run the minimum number of services required

Services are programmes such as ftp servers and web servers. Since they have to be *listening* for incoming connections that request the service, external computers can connect to yours. Services are sometimes vulnerable (i.e. can be compromised under a given attack) and are hence a security risk.

You should not install services which are not needed on your machine. Every installed service might introduce new, perhaps not obvious (or known), security holes on your computer.

As you may already know, when you install a given service the default behavior is to activate it. In a default Debian installation, with no services installed, the footprint of running services is quite low and it's even lower when talking about services offered to the network. The footprint in Debian 2.1 wasn't as tight as in Debian 2.2 (some `inetd` services were enabled by default) and in Debian 2.2 the `rpc portmapper` is enabled upon installation. `Rpc` is installed by default because it is needed for many services, for example NFS, to run on a given system. It can be easily removed, however, see 'Disabling daemon services' on the facing page on how to disable it.

When you install a new network-related service (daemon) in your Debian GNU/Linux system it can be enabled in two ways: through the `inetd` superdaemon (i.e. a line will be added to `/etc/inetd.conf`) or through a standalone program that binds itself to your network interfaces. Standalone programs are controlled through the `/etc/init.d` files, which are called at

boot time through the SysV mechanism (or an alternative one) by using symlinks in `/etc/rc?.d/*` (for more information on how this is done read `/usr/share/doc/sysvinit/README.runlevels.gz`).

If you want to keep some services but use them rarely, use the update-commands, e.g. `update-inetd` and `update-rc.d` to remove them from the startup process.

3.6.1 Disabling daemon services

Disabling a daemon service is quite simple. There are different methods:

- remove links from `/etc/rc${runlevel}.d/` or rename the links (so that they do not begin with 'S')
- move the script file (`/etc/init.d/_service_name_`) to another name (for example `/etc/init.d/OFF._service_name_`)
- remove the execute permission from the `/etc/init.d/_service_name_` file.
- edit the `/etc/init.d/_service_name_` script to have it stop immediately.

You can remove the links from `/etc/rc${runlevel}.d/` manually or using `update-rc.d` (see `update-rc.d(8)`). For example, you can disable a service from executing in the multi-user runlevels by doing:

```
update-rc.d stop XX 2 3 4 5 .
```

Please note that, if you are *not* using `file-rc`, `update-rc.d -f _service_ remove` will not work properly, since *all* links are removed, upon re-installation or upgrade of the package these links will be re-generated (probably not what you wanted). If you think this is not intuitive you are probably right (see Bug 67095 (<http://bugs.debian.org/67095>)). From the manpage:

```
If any files /etc/rcrunlevel.d/[SK]??name already exist then
update-rc.d does nothing. This is so that the system administrator
can rearrange the links, provided that they leave at least one
link remaining, without having their configuration overwritten.
```

If you are using `file-rc` all the information regarding services bootup is handled by a common configuration file and is maintained even if packages are removed from the system.

You can use the TUI (Text User Interface) provided by `rcconf` to do all these changes easily (`rcconf` works both for `file-rc` and normal System V runlevels).

Other (not recommended) methods of disabling services are: `chmod 644 /etc/init.d/daemon` (but that gives an error message when booting), or modifying the `/etc/init.d/daemon`

script (by adding an `exit 0` line at the beginning or commenting out the `start-stop-daemon` part in it). Since `init.d` files are config files, they will not get overwritten upon upgrade.

Unfortunately, unlike other (UNIX) operating systems, services in Debian cannot be disabled by modifying files in `/etc/default/_servicename_`.

FIXME: Add more information on handling daemons using `file-rc`

3.6.2 Disabling `inetd` services

You should stop all unneeded services on your system, like `echo`, `chargen`, `discard`, `daytime`, `time`, `talk`, `ntalk` and `r-services` (`rsh`, `rlogin` and `rcp`) which are considered HIGHLY insecure (use `ssh` instead). After disabling those, you should check if you really need the `inetd` daemon. Many people prefer to use daemons instead of calling services via `inetd`. Denial of Service possibilities exist against `inetd`, which can increase the machine's load tremendously. If you still want to run some kind of `inetd` service, switch to a more configurable `inet` daemon like `xinetd` or `rloginetd`.

You can disable services by editing `/etc/inetd.conf` directly, but Debian provides a better alternative: `update-inetd` (which comments the services in a way that it can easily be turned on again). You could remove the `telnet` daemon by executing this commands to change the config file and to restart the daemon (in this case the `telnet` service is disabled):

```
/usr/sbin/update-inetd --disable telnet
```

If you do want services listening, but do not want to have them listen on all IP addresses of your host, you might want to use an undocumented feature on `inetd`. . Or use an alternate `inetd` daemon like `xinetd`.

3.7 Install the minimum amount of software required

Debian comes with *a lot* of software, for example the Debian 3.0 *woody* release includes almost 6 CD-ROMs of software and thousands of packages. With so much software, and even if the base system installation is quite reduced ¹ you might get carried away and install more than is really needed for your system.

Since you already know what the system is for (don't you?) you should only install software that is really needed for it to work. Any unnecessary tool that is installed might be used by a user that wants to compromise the system or by an external intruder that has gotten shell access (or remote code execution through an exploitable service).

¹For example, in Debian *woody* it is around 40Mbs, try this:

```
$ size=0 $ for i in `grep -A 1 -B 1 "^Section: base" /var/lib/dpkg/available |
grep -A 2 "^Priority: required" |grep "^Installed-Size" |cut -d : -f 2 `; do
size=$((size+$i)); done $ echo $size 34234
```

The presence, for example, of development utilities (a C compiler) or interpreted languages (such as `perl` - but see below -, `python`, `tcl`...) may help an attacker compromise the system even further:

- allowing him to do privilege escalation. It's easier, for example, to run local exploits in the system if there is a debugger and compiler ready to compile and test them!
- providing tools that could help the attacker to use the compromised system as a *base of attack* against other systems²

Of course, an intruder with local shell access can download his own set of tools and execute them, and even the shell itself can be used to make complex programs. Removing unnecessary software will not help *prevent* the problem but will make it slightly more difficult for an attacker to proceed (and some might give up in this situation looking for easier targets). So, if you leave tools in a production system that could be used to remotely attack systems (see 'Remote vulnerability assesment tools' on page 109) you can expect an intruder to use them too if available.

3.7.1 Removing Perl

You must take into account that removing `perl` might not be too easy (as a matter of fact it can be quite difficult) in a Debian system since it is used by many system utilities. Also, the `perl-base` is *Priority: required* (that about says it all). It's still doable, but you will not be able to run any `perl` application in the system; you will also have to fool the package management system to think that the `perl-base` is installed even if it's not.³

Which utilities use `perl`? You can see for yourself:

```
$ for i in /bin/* /sbin/* /usr/bin/* /usr/sbin/*; do [ -f $i ] && {
  type='file $i | grep -il perl'; [ -n "$type" ] && echo $i; }; done
```

These include the following utilities in packages with priority *required* or *important*:

- `/usr/bin/chkdupexe` of package `util-linux`.
- `/usr/bin/replay` of package `bsdutils`.
- `/usr/sbin/cleanup-info` of package `dpkg`.
- `/usr/sbin/dpkg-divert` of package `dpkg`.
- `/usr/sbin/dpkg-statoverride` of package `dpkg`.

²Many intrusions are made just to get access to resources to do illegitimate activity (denial of service attacks, spam, rogue ftp servers, dns pollution...) rather than to obtain confidential data from the compromised system.

³You can make (on another system) a dummy package with `equivs`

- `/usr/sbin/install-info` of package `dpkg`.
- `/usr/sbin/update-alternatives` of package `dpkg`.
- `/usr/sbin/update-rc.d` of package `sysvinit`.
- `/usr/bin/grog` of package `groff-base`.
- `/usr/sbin/adduser` of package `adduser`.
- `/usr/sbin/debconf-show` of package `debconf`.
- `/usr/sbin/deluser` of package `adduser`.
- `/usr/sbin/dpkg-preconfigure` of package `debconf`.
- `/usr/sbin/dpkg-reconfigure` of package `debconf`.
- `/usr/sbin/exigrep` of package `exim`.
- `/usr/sbin/eximconfig` of package `exim`.
- `/usr/sbin/eximstats` of package `exim`.
- `/usr/sbin/exim-upgrade-to-r3` of package `exim`.
- `/usr/sbin/exiqsumm` of package `exim`.
- `/usr/sbin/keytab-lilo` of package `lilo`.
- `/usr/sbin/liloconfig` of package `lilo`.
- `/usr/sbin/lilo_find_mbr` of package `lilo`.
- `/usr/sbin/syslogd-listfiles` of package `sysklogd`.
- `/usr/sbin/syslog-facility` of package `sysklogd`.
- `/usr/sbin/update-inetd` of package `netbase`.

So, without Perl and, unless you remake these utilities in shell script, you will probably not be able to manage any packages (so you will not be able to upgrade the system, which is *not a Good Thing*).

If you are determined to remove Perl from the Debian base system, and you have spare time, submit bug reports to the previous packages including (as a patch) replacements for the utilities above written in shell script.

3.8 Read the debian security mailing lists

It is never wrong to take a look at either the `debian-security-announce` mailing list, where advisories and fixes to released packages are announced by the Debian security team, or at <mailto:debian-security@lists.debian.org>, where you can participate in discussions about things related to Debian security.

In order to receive important security update alerts, send an email to `debian-security-announce-request@lists.debian.org` (<mailto:debian-security-announce-request@lists.debian.org>) with the word “subscribe” in the subject line. You can also subscribe to this moderated email list via the web page at <http://www.debian.org/MailingLists/subscribe>

This mailing list has very low volume, and by subscribing to it you will be immediately alerted of security updates for the Debian distribution. This allows you to quickly download new packages with security bug fixes, which is very important in maintaining a secure system. (See ‘Execute a security update’ on page 38 for details on how to do this.)

Chapter 4

After Installation

Once the system is installed you can still do more to secure the system; some of the steps described in this chapter can be taken. Of course this really depends on your setup but for physical access prevention you should read 'Change the BIOS (again)' on the current page, 'Set a LILO or GRUB password' on this page, 'Remove root prompt on the kernel' on the next page, 'Disallow floppy booting' on page 35, 'Restricting console login access' on page 36, and 'Restricting system reboots through the console' on page 36.

Before connecting to any network, especially if it's a public one you should, at the very least, execute a security update (see 'Execute a security update' on page 38). Optionally, you could take a snapshot of your system (see 'Taking a snapshot of the system' on page 64).

4.1 Change the BIOS (again)

Remember 'Choose a BIOS password' on page 23? Well, then you should now, once you do not need to boot from removable media, to change the default BIOS setup so that it *only* boots from the hard drive. Make sure you will not lose the BIOS password, otherwise, in the event of a hard disk failure you will not be able to return to the BIOS and change the setup so you can recover it using, for example, a CD-ROM.

Another less secure but more convenient way is to change the setup to have the system boot up from the hard disk and, if it fails, try removable media. By the way, this is often done because most people don't use the BIOS password that often; it's easily forgotten.

4.2 Set a LILO or GRUB password

Anybody can easily get a root-shell and change your passwords by entering `<name-of-your-bootimage> init=/bin/sh` at the boot prompt. After changing the passwords and rebooting the system, the person has unlimited root-access and can do anything he/she wants to the system. After this procedure you will not have root access to your system, as you do not know the root password.

To make sure that this cannot happen, you should set a password for the boot loader. You can choose between a global password or a password for a certain image.

For LILO you need to edit the config file `/etc/lilo.conf` and add a password and restricted line as in the example below.

```
image=/boot/2.2.14-vmlinuz
  label=Linux
  read-only
  password=hackme
  restricted
```

When done, rerun `lilo`. Omitting the `restricted` line causes `lilo` to always prompt for a password, regardless of whether LILO was passed parameters. The default permissions for `/etc/lilo.conf` grant read and write permissions to root, and enable read-only access for `lilo.conf`'s group, root.

If you use GRUB instead of LILO, edit `/boot/grub/menu.lst` and add the following two lines at the top (substituting, of course `hackme` with the desired password). This prevents users from editing the boot items. `timeout 3` specifies a 3 second delay before `grub` boots the default item.

```
timeout 3
password hackme
```

To further harden the integrity of the password, you may store the password in an encrypted form. The utility `grub-md5-crypt` generates a hashed password which is compatible with `grub`'s encrypted password algorithm (md5). To specify in `grub` that an md5 format password will be used, use the following directive:

```
timeout 3
password --md5 $1$bw0ez$t1jnxxKLFmZmnDVaQWgjP0
```

The `--md5` parameter was added to instruct `grub` to perform the md5 authentication process. The provided password is the md5 encrypted version of `hackme`. Using the md5 password method is preferable to choosing its cleartext counterpart. More information about `grub` passwords may be found in the `grub-doc` package.

4.3 Remove root prompt on the kernel

Linux 2.4 kernels provide a way to access a root shell while booting which will be presented just after loading the `cramfs` file system. A message will appear to permit the administrator to enter an executable shell with root permissions, this shell can be used to manually load modules when autodetection fails. This behavior is the default for `initrd`'s `linuxrc`. The following message will appear:

```
Press ENTER to obtain a shell (waits 5 seconds)
```

In order to remove this behavior you need to change `/etc/mkinitrd/mkinitrd.conf` and set:

```
# DELAY The number of seconds the linuxrc script should wait to
# allow the user to interrupt it before the system is brought up
DELAY=0
```

Then regenerate your ramdisk image. You can do this for example with:

```
# cd /boot
# mkinitrd -o initrd.img-2.4.18-k7 /lib/modules/2.4.18-k7
```

or (preferred):

```
# dpkg-reconfigure -p low kernel-image-2.4.x-yz
```

Note that Debian 3.0 woody allows users to install 2.4 kernels (selecting *flavors*), *however* the default kernel is 2.2 (save for some architectures for which kernel 2.2 was not ported). If you consider this a bug consider Bug 145244 (<http://bugs.debian.org/145244>) before sending it.

4.4 Disallow floppy booting

The default MBR in Debian before version 2.2 did not act as a usual master boot record and left open a method to easily break into a system:

- Press shift at boot time, and an MBR prompt appears
- Then press F, and your system will boot from floppy disk. This can be used to get root access to the system.

This behavior can be changed by entering:

```
lilo -b /dev/hda
```

Now LILO is put into the MBR. This can also be achieved by adding `boot=/dev/hda` to `lilo.conf`. There is another solution which will disable the MBR prompt completely:

```
install-mbr -i n /dev/hda
```

On the other hand, this “back door”, of which many people are just not aware, may save your skin as well if you run into deep trouble with your installation for whatever reasons.

FIXME check whether this really is true as of 2.2 or was it 2.1? INFO: The bootdisks as of Debian 2.2 do NOT install the mbr, but only LILO.

4.5 Restricting console login access

Some security policies might force administrators to log in to the system through the console with their user/password and then become superuser (with `su` or `sudo`). This policy is implemented in Debian by editing the `/etc/login.defs` file or `/etc/securetty` when using PAM. In:

- `login.defs`, editing the `CONSOLE` variable which defines a file or list of terminals on which root logins are allowed
- `securetty` by adding/removing the terminals to which root access will be allowed.

When using PAM, other changes to the login process, which might include restrictions to users and groups at given times, can be configured in `/etc/pam.d/login`. An interesting feature that can be disabled is the possibility to login with null (blank) passwords. This feature can be limited by removing `nullok` from the line:

```
auth      required  pam_unix.so nullok
```

4.6 Restricting system reboots through the console

If your system has a keyboard attached to it anyone (yes *anyone*) can reboot the system through it without login to the system. This might, or might not, adhere to your security policy. If you want to restrict this, you must check the `/etc/inittab` so that the line that includes `ctrlaltdel` calls `shutdown` with the `-a` switch (remember to run `init q` after making any changes to this file). The default in Debian includes this switch:

```
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
```

Now, in order to allow *some* users to shutdown the system, as the manpage `shutdown(8)` describes, you must create the file `/etc/shutdown.allow` and include there the name of users which can boot the system. When the *three finger salute* (a.k.a. `ctrl+alt+del`) is given the program will check if any of the users listed in the file are logged in. If none of them is, `shutdown` will *not* reboot the system.

4.7 Mounting partitions the right way

When mounting an `ext2` partition, there are several additional options you can apply to the mount call or to `/etc/fstab`. For instance, this is my `fstab` entry for the `/tmp` partition:

```
/dev/hda7    /tmp    ext2    defaults,nosuid,noexec,nodev    0    2
```

You see the difference in the options sections. The option `nosuid` ignores the `setuid` and `setgid` bits completely, while `noexec` forbids execution of any program on that mount point, and `nodev`, ignores devices. This sounds great, but it

- only applies to ext2 file systems
- can be circumvented easily

The `noexec` option prevents binaries from being executed directly, but is easily circumvented:

```
alex@joker:/tmp# mount | grep tmp
/dev/hda7 on /tmp type ext2 (rw,noexec,nosuid,nodev)
alex@joker:/tmp# ./date
bash: ./date: Permission denied
alex@joker:/tmp# /lib/ld-linux.so.2 ./date
Sun Dec  3 17:49:23 CET 2000
```

However, many script kiddies have exploits which try to create and execute files in `/tmp`. If they do not have a clue, they will fall into this pit. In other words, a user cannot be tricked into executing a trojanized binary in `/tmp` e.g. when he incidentally adds `/tmp` into his `PATH`.

Also be forewarned, some script might depend on `/tmp` being executable. Most notably, `Debcnf` has (had?) some issues regarding this, for more information see Bug 116448 (<http://bugs.debian.org/116448>).

The following is a more thorough example. A note, though: `/var` could be set `noexec`, but some software like `Smartlist` keeps its programs in `/var`. The same applies to the `nosuid` option.

<code>/dev/sda6</code>	<code>/usr</code>	<code>ext2</code>	<code>defaults,ro,nodev</code>	<code>0</code>	<code>2</code>
<code>/dev/sda12</code>	<code>/usr/share</code>	<code>ext2</code>	<code>defaults,ro,nodev,nosuid</code>		<code>0</code>
<code>/dev/sda7</code>	<code>/var</code>	<code>ext2</code>	<code>defaults,nodev,usrquota,grpquota</code>		
<code>/dev/sda8</code>	<code>/tmp</code>	<code>ext2</code>	<code>defaults,nodev,nosuid,noexec,usrquota</code>		
<code>/dev/sda9</code>	<code>/var/tmp</code>	<code>ext2</code>	<code>defaults,nodev,nosuid,noexec,usrquota</code>		
<code>/dev/sda10</code>	<code>/var/log</code>	<code>ext2</code>	<code>defaults,nodev,nosuid,noexec</code>	<code>0</code>	
<code>/dev/sda11</code>	<code>/var/account</code>	<code>ext2</code>	<code>defaults,nodev,nosuid,noexec</code>	<code>0</code>	
<code>/dev/sda13</code>	<code>/home</code>	<code>ext2</code>	<code>rw,nosuid,nodev,exec,auto,nouser,asyn</code>		
<code>/dev/fd0</code>	<code>/mnt/fd0</code>	<code>ext2</code>	<code>defaults,users,nodev,nosuid,noexec</code>		
<code>/dev/fd0</code>	<code>/mnt/floppy</code>	<code>vfat</code>	<code>defaults,users,nodev,nosuid,noexec</code>		
<code>/dev/hda</code>	<code>/mnt/cdrom</code>	<code>iso9660</code>	<code>ro,users,nodev,nosuid,noexec</code>		

4.7.1 Setting `/tmp noexec`

Be careful if setting `/tmp noexec` when you want to install new software, since some programs might use it for installation. `Apt` is one such program (see <http://bugs.debian.org/116448>) if not configured properly `APT::ExtractTemplates::TempDir` (see `apt-extracttemplates`

You can set this variable in `/etc/apt/apt.conf` to another directory with `exec` privileges other than `/tmp`.

Regarding `noexec`, please be aware that it might not offer you that much security. Consider this:

```
$ cp /bin/date /tmp
$ /tmp/date
(does not execute due to noexec)
$/lib/ld-linux.so.2 /tmp/date
(works since date is not executed directly)
```

4.7.2 Setting `/usr` read-only

If you set `/usr` read-only you will not be able to install new packages on your Debian GNU/Linux system. You will have to first remount it read-write, install the packages and then remount it read-only. The latest `apt` version (in Debian 3.0 'woody') can be configured to run commands before and after installing packages, so you might want to configure it properly.

To do this modify `/etc/apt/apt.conf` and add:

```
DPkg
{
    Pre-Invoke { "mount /usr -o remount,rw" };
    Post-Invoke { "mount /usr -o remount,ro" };
};
```

Note that the `Post-Invoke` may fail with a `"/usr busy"` error message. This happens mainly when you are using files during the update that got updated. Annoying but not really a big deal. Just make sure these are no longer used and run the `Post-Invoke` manually.

4.8 Execute a security update

As soon as new security bugs are detected in packages, Debian maintainers and upstream authors generally patch them within days or even hours. After the bug is fixed, a new package is provided on <http://security.debian.org>.

If you are installing a Debian release you must take into account that since the release was made there might have been security updates after it has been determined that a given package is vulnerable. Also, there might have been minor releases (there were seven in Debian 2.2 *potato* release) which include these package updates.

You need to note down the date the removable media (if you are using it) was made and check the security site in order to see if there are security updates. If there are and you cannot download the packages from the security site on another system (you are not connected to the

Internet yet? are you?) before connecting to the network you could consider (if not protected by a firewall for example) adding firewall rules so that your system could only connect to security.debian.org and then run the update. A sample configuration is shown in ‘Security update protected by a firewall’ on page 175.

To update the system, put the following line in your `sources.list` and you will get security updates automatically, whenever you update your system.

```
deb http://security.debian.org/ stable/updates main contrib non-free
```

Most people who don't live in a country which prohibits importing or using strong cryptography, should add this line as well:

```
deb http://security.debian.org/debian-non-US stable/non-US main contrib non
```

If you like, you can add the `deb-src` lines to `/etc/apt/sources.list` as well. See `apt(8)` for further details.

FIXME: Add info on how the signature of packages is done so that this can be done automatically through a cron job (big warning: DNS spoofing).

4.9 Subscribe to the Debian Security Announce mailing List

In order to receive information on available security updates you should subscribe yourself to the `debian-security-announce` mailing list in order to receive the Debian Security Advisories (DSAs). See ‘The Debian Security Team’ on page 95 for more information on how the Debian security team works. For information on how to subscribe to the Debian mailing lists read <http://lists.debian.org>.

DSAs are signed with the Debian Security Team's signature which can be retrieved from <http://security.debian.org>.

You should consider, also, subscribing to the `debian-security` mailing list for general discussion on security issues in the Debian operating system.

FIXME: add the key here too?

4.10 Providing secure user access

4.10.1 User authentication: PAM

PAM (Pluggable Authentication Modules) allows system administrators to choose how applications authenticate users. Note that PAM can do nothing unless an application is compiled with support for PAM. Most of the applications that are shipped with Debian 2.2 have

this support built in. Furthermore, Debian did not have PAM support before 2.2. The current default configuration for any PAM-enabled service is to emulate UNIX authentication (read `/usr/share/doc/libpam0g/Debian-PAM-MiniPolicy.gz` for more information on how PAM services *should* work in Debian).

Each application with PAM support provides a configuration file in `/etc/pam.d/` which can be used to modify its behavior:

- what backend is used for authentication.
- what backend is used for sessions.
- how do password checks behave.

The following description is far from complete, for more information you might want to read the The Linux-PAM System Administrator's Guide (<http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam.html>) (at the primary PAM distribution site (<http://www.kernel.org/pub/linux/libs/pam/>)), this document is also provided in the `libpam-doc`.

PAM offers you the possibility to go through several authentication steps at once, without the user's knowledge. You could authenticate against a Berkeley database and against the normal `passwd` file, and the user only logs in if he authenticates correct in both. You can restrict a lot with PAM, just as you can open your system doors very wide. So be careful. A typical configuration line has a control field as its second element. Generally it should be set to `requisite`, which returns a login failure if one module fails.

The first thing I like to do, is to add MD5 support to PAM applications, since this helps protect against dictionary cracks (passwords can be longer if using MD5). The following two lines should be added to all files in `/etc/pam.d/` that grant access to the machine, like `login` and `ssh`.

```
# Be sure to install libpam-cracklib first or you will not be able to log i
password    required    pam_cracklib.so retry=3 minlen=12 difok=3
password    required    pam_unix.so use_authtok nullok md5
```

So, what does this incantation do? The first line loads the `cracklib` PAM module, which provides password strength-checking, prompts for a new password with a minimum length of 12 characters, a difference of at least 3 characters from the old password, and allows 3 retries. The second line introduces the standard authentication module with MD5 passwords and allows a zero length password. The `use_authtok` directive is necessary to hand over the password from the previous module.

FIXME: Check if Bug #112965 (<http://bugs.debian.org/112965>) still stands (i.e. you can run into problems if you do not have a wordlist (such as `wenglish`, `wspanish`, `wbritish`...)). It does not seem to have a dependency to `wordlist` in `woody`.

To make sure that the user `root` can only log into the system from local terminals, the following line should be enabled in `/etc/pam.d/login`:

```
auth    requisite pam_securetty.so
```

Then you should add the terminals from which the user root can log into the system, in `/etc/security/access.conf`. Last but not least the following line should be enabled if you want to set up user limits.

```
session required pam_limits.so
```

This restricts the system resources that users are allowed (see below in ‘Limiting resource usage: the `limits.conf` file’ on the following page). For example, you could restrict the number of concurrent logins (of a given group of users, or system-wide) you may have, the number of processes, the memory size...

Now edit `/etc/pam.d/passwd` and change the first line. You should add the option “md5” to use MD5 passwords, change the minimum length of password from 4 to 6 (or more) and set a maximum length, if you desire. The resulting line will look something like:

```
password required pam_unix.so nullok obscure min=6 max=11 md5
```

If you want to protect `su`, so that only some people can use it to become root on your system, you need to add a new group “wheel” to your system (that is the cleanest way, since no file has such a group permission yet). Add root and the other users that should be able to `su` to the root user to this group. Then add the following line to `/etc/pam.d/su`:

```
auth    requisite pam_wheel.so group=wheel debug
```

This makes sure that only people from the group “wheel” can use `su` to become root. Other users will not be able to become root. In fact they will get a denied message if they try to become root.

If you want only certain users to authenticate at a PAM service, this is quite easy to achieve by using files where the users who are allowed to login (or not) are stored. Imagine you only want to allow user ‘ref’ to log in via `ssh`. So you put him into `/etc/sshusers-allowed` and write the following into `/etc/pam.d/ssh`:

```
auth    required pam_listfile.so item=user sense=allow file=/etc/ssh
```

Last, but not least, create `/etc/pam.d/other` and enter the following lines:

```
auth    required    pam_securetty.so
auth    required    pam_unix_auth.so
auth    required    pam_warn.so
auth    required    pam_deny.so
account required    pam_unix_acct.so
```

```

account required pam_warn.so
account required pam_deny.so
password required pam_unix_passwd.so
password required pam_warn.so
password required pam_deny.so
session required pam_unix_session.so
session required pam_warn.so
session required pam_deny.so

```

These lines will provide a good default configuration for all applications that support PAM (access is denied by default).

4.10.2 Limiting resource usage: the `limits.conf` file

You should really take a serious look into this file. Here you can define user resource limits. If you use PAM, the file `/etc/limits.conf` is ignored and you should use `/etc/security/limits.conf` instead.

If you do not restrict resource usage, *any* user with a valid shell in your system (or even an intruder who compromised the system through a service) can use up as much CPU, memory, stack, etc. as the system can provide. This *resource exhaustion* problem can only be fixed by the use of PAM. Note that there is a way to add resource limits to some shells (for example, bash has `ulimit`, see `bash(1)`), but since not all of them provide the same limits and since the user can change shells (see `chsh(1)`) it is better to place the limits on the PAM modules.

For more information read:

- PAM configuration article (<http://www.samag.com/documents/s=1161/sam0009a/0009a.htm>).
- Seifried's Securing Linux Step by Step (<http://seifried.org/security/os/linux/20020324-securing-linux-step-by-step.html>) on the *Limiting users overview* section.
- LASG (<http://seifried.org/lasg/users/>) in the *Limiting and monitoring users* section.

FIXME: Get a good `limits.conf` up here

4.10.3 User Login actions: edit `/etc/login.defs`

The next step is to edit the basic configuration and action upon user login.

This variable should be set to a higher value to make it harder to use the terminal to log in using brute force. If a wrong password is typed in, the possible attacker (or normal user!) has to wait for 10 seconds to get a new login prompt, which is quite time consuming when you test passwords (manually). Pay attention to the fact that this setting is useless if using a program other than `getty`, such as `mingetty` for example.

```
FAILLOG_ENAB      yes
```

If you enable this variable, failed logins will be logged. It is important to keep track of them to catch someone who tries a brute force attack.

```
LOG_UNKFAIL_ENAB  yes
```

If you set the variable `FAILLOG_ENAB` to `yes`, then you should also set this variable to `yes`. This will record unknown usernames if the login failed. If you do this, make sure the logs have the proper permissions (640 for example, with an appropriate group setting such as `adm`), because users often accidentally enter their password as the username and you do not want others to see it.

```
SYSLOG_SU_ENAB    yes
```

This one enables logging of `su` attempts to `syslog`. Quite important on serious machines but note that this can create privacy issues as well.

```
SYSLOG_SG_ENAB    yes
```

The same as `SYSLOG_SU_ENAB` but applies to the `sg` program.

```
MD5_CRYPT_ENAB   yes
```

As stated above, MD5 sum passwords greatly reduce the problem of dictionary attacks, since you can use longer passwords. If you are using `slink`, read the docs about MD5 before enabling this option. Otherwise this is set in PAM.

```
PASS_MAX_LEN      50
```

If MD5 passwords are activated in your PAM configuration, then this variable should be set to the same value as used there.

4.10.4 Restricting ftp: editing `/etc/ftpusers`

The `/etc/ftpusers` file contains a list of users who are not allowed to log into the host using ftp. Only use this file if you really want to allow ftp (which is not recommended in general, because it uses cleartext passwords). If your daemon supports PAM, you can also use that to allow and deny users for certain services.

FIXME (BUG): Is it a bug that the default `ftpusers` in Debian does *not* include all the administrative users (in `base-passwd`).

4.10.5 Using `su`

If you really need users to become the super user on your system, e.g. for installing packages or adding users, you can use the command `su` to change your identity. You should try to avoid any login as user root and instead use `su`. Actually, the best solution is to remove `su` and switch to `sudo`, as it has more features than `su`. However, `su` is more common as it is used on many other Unices.

4.10.6 Using `sudo`

`sudo` allows the user to execute defined commands under another user's identity, even as root. If the user is added to `/etc/sudoers` and authenticates himself correctly, he is able to run commands which have been defined in `/etc/sudoers`. Violations, such as incorrect passwords or trying to run a program you don't have permission for, are logged and mailed to root.

4.10.7 Disallow remote administrative access

You should modify `/etc/security/access.conf` also so that remote administrative login is disallowed. This way the users need to use `su` (or `sudo`) so that there is always an audit trace whenever a local user wants to use administrative powers.

You need to add the following line to `/etc/security/access.conf`, the default Debian configuration file has a sample line commented out:

```
- :wheel:ALL EXCEPT LOCAL
```

4.10.8 Restricting users's access

Sometimes you might think you need to have users created in your local system in order to provide a given service (pop3 mail service or ftp). Before doing so, first remember that the PAM implementation in Debian GNU/Linux allows you to validate users with a wide variety of external directory services (radius, ldap, etc.) provided by the `libpam` packages.

If users need to be created and the system can be accessed remotely take into account that users will be able to log in to the system. You can fix this by giving users a null (`/dev/null`) shell (it would need to be listed in `/etc/shells`). If you want to allow users to access the system but limit their movements, you can use the `/bin/rbash`, equivalent to adding the `-r` option in `bash` (*RESTRICTED SHELL* see `bash(1)`). Please note that even with restricted shell, a user that access an interactive program (that might allow execution of a subshell) could be able to bypass the limits of the shell.

Debian currently provides in the unstable release (and might be included in the next stable releases) the `pam_chroot` module (in the `libpam-chroot`). An alternative to it is to `chroot` the service that provides remote logging (`ssh`, `telnet`).¹

If you wish to restrict *when* users can access the system you will have to customize `/etc/security/access.conf` for your needs.

Information on how to `chroot ssh` for users is described in ‘Chroot environment for SSH’ on page 177.

4.10.9 Hand-made user auditing

If you are paranoid you might want to add a defined `.profile` in users’ home directories that sets the environment in a way such that they cannot remove audit capabilities from the shell (commands are dumped to `$HISTFILE`). The `.profile` could be set as follows:

```
HISTFILE=/home/_user_/ .bash_history
HISTSIZE=1000000000000000000
HISTFILESIZE=1000000000000000000
set -o HISTFILE
set -o HISTSIZE
set -o HISTFILESIZE
export HISTFILE HISTSIZE HISTFILESIZE
```

Note: the `-o` attribute sets a variable read-only in `bash`.

For this to work the user cannot modify the `.profile` or `.bash_history` but must be able to read the first one and write in the second one. You can do this easily by changing these files and the directory where they reside to be owned by another user (`root`), and give write permissions to the user’s group to the history file. Another option is through the use of the `chattr` program.

If you are completely paranoid and want to audit every user’s command, you could take `bash` source code, edit it and have it send all that the user typed into another file. Or have `ttysnoop` constantly monitor any new `ttys` and dump the output into a file. Other useful program is `Snoopy` (http://sourceforge.net/project/?group_id=2091) which is a user-transparent program that hooks in as a library providing a wrapper around `execve()` calls,

¹`Libpam-chroot` has not been yet thoroughly tested, it does work for `login` but it might not be easy to set up the environment for other programs

any command executed is logged to `syslogd` using the `authpriv` facility (usually stored at `/var/log/auth.log`).

Note that you cannot use the `script` command for this since it will not work as a shell (even if you add it to `/etc/shells`).

4.10.10 Complete user audit

The previous example is a simple way to configure user auditing which might be not useful for complex systems. If this is your case, you need to look at `acct`, the accounting utilities. These will log all the commands run by users or processes in the system, at the expense of disk space.

When activating accounting, all the information on processes and users is kept under `/var/account/`, more specifically in the `pacct`. The accounting package includes some tools (`sa` and `ac`) to analyse this data.

4.10.11 Reviewing user profiles

If you want to *see* what users are usually doing, when they are connecting you can use the `wtmp` database that includes all login information. This file can be processed with several utilities, amongst them `sac` which can output a profile on each user showing in which timeframe they usually log on to the system.

In case you have accounting activated, you can also use the tools provided by it in order to determine when the users access the system and what they execute.

4.10.12 Setting users umasks

Depending on your user policy you might want to change how information is shared between users, that is, what the default permissions of new files created by users are. This change is set by defining a proper `umask` setting for all users. You can change the `UMASK` setting in `/etc/limits.conf`, `/etc/profile`, `/etc/csh.cshrc`, `/etc/csh.login`, `/etc/zshrc` and probably some others (depending on the shells you have installed on your system). Of all of these the last one that gets loaded takes precedence. The order is: PAM's `limits.conf`, the default system configuration for the user's shell, the user's shell (his `~/.profile`, `~/.bash_profile`...)

Debian's default `umask` setting is `022` this means that files (and directories) can be read and accessed by the user's group and by any other users in the system. If this is too permissive for your system you will have to change the `umask` setting for all the shells (and for PAM). Don't forget to modify the files under `/etc/skel/` since these will be new user's defaults when created with the `adduser` command.

Note, however that users can modify their own `umask` setting if they want too, making it more permissive or more restricted.

4.10.13 Limiting what users can see/access

FIXME: Content needed. Tell of consequences of changing packages permissions when upgrading (and admin this paranoid should `chroot` his users BTW).

If you need to grant users access to the system with a shell think about it very carefully. A user can, by default unless in a severely restricted environment (like a `chroot` jail), retrieve quite a lot of information from your system including:

- some configuration files in `/etc`. However, Debian's default permissions for some sensitive files (which might, for example, contain passwords), will prevent access to critical information. To see which files are only accessible by the root user for example `find /etc -type f -a -perm 600 -a -uid 0` as superuser.
- your installed packages, either by looking at the package database, at the `/usr/share/doc` directory or by guessing by looking at the binaries and libraries installed in your system.
- some log files at `/var/log`. Note also that some log files are only accessible to root and the `adm` group (try `find /var/log -type f -a -perm 640`) and some are even only available to the root user (try `find /var/log -type f -a -perm 600 -a -uid 0`).

What can a user see in your system? Probably quite a lot of things, try this (take a deep breath):

```
find / -type f -a -perm +006 2>/dev/null
find / -type d -a -perm +007 2>/dev/null
```

The output is the list of files that a user can *see* and the directories to which he has access.

Limiting access to other user's information

If you still grant shell access to users you might want to limit what information they can view from other users. Users with shell access have a tendency to create quite a number of files under their `$HOME`s: mailboxes, personal documents, configuration of X/GNOME/KDE applications...

In Debian each user is created with one associated group, and no two users belong to the same group. This is the default behavior: when the `userX` is created a group with name `userX` is created and the user is assigned to it. This avoids the concept of a *users* group which might make it more difficult for users to hide information from other users.

However, users' `$HOME` directories are created with `0755` permissions (group-readable and world-readable). The group permissions is not an issue since only the user belongs to the group, however the world permissions might (or might not) be an issue depending on your local policy.

You can change this behaviour so that user creation provides different `$HOME` permissions. To change the behaviour for *new* users when they get created, change `DIR_MODE` in the configuration file `/etc/adduser.conf` to `0750` (no world-readable access).

Users can still share information, but not directly in their `$HOME` directories unless they change its permissions.

Note that this will prevent users from being able to set up personal pages (`~userX`) if a web server is present, since the web server will not be able to read the `$HOME` directory and thus, the `public_html` directory under it.

4.10.14 Generating user passwords

There are many cases when an administrator needs to create many user accounts and provide passwords for all of them. Of course, the administrator could easily just set the password to be the same as the user's account name, but that would not be very sensitive security-wise. A better approach is to use a password generating program. Debian provides `makepasswd`, `apg` and `pwgen` packages which provide programs (the name is the same as the package) that can be used for this purpose. `Makepasswd` will generate true random passwords with an emphasis on security over pronounceability while `pwgen` will try to make meaningless but pronounceable passwords (of course this might depend on your mother language). `ApG` has algorithms to provide for both (there is a client/server version for this program but it is not included in the Debian package).

`passwd` does not allow non-interactive assignment of passwords (since it uses direct tty access). If you want to change passwords when creating a large number of users you can create them using `adduser` with the `--disabled-login` option and then use `chpasswd`² (in the `passwd` package so you already have it installed). If you want to use a file with all the information to make users as a batch process you might be better off using `newusers`.

4.10.15 Checking user passwords

User passwords can sometimes become the *weakest link* in the security of a given system. This is due to some users choosing weak passwords for their accounts (and the more of them that have access to it the greater the chances of this happening). Even if you established checks with the `cracklib` PAM module and password limits as described in 'User authentication: PAM' on page 39 users will still be able to use weak passwords. Since user access might include remote shell access (over `ssh`, hopefully) it's important that a remote attacker is not able to guess user passwords (after he has been able to do user enumeration by other means).

A system administrator must, given a big number of users, check if the passwords they have are consistent with the local security policy. How to check? Try to crack them as an attacker would if he had access to the hashed passwords (the `/etc/shadow` file).

²`chpasswd` cannot handle MD5 password generation so it needs to be given the password in encrypted form before using it, with the `-e` option.

An administrator can use `john` together with an appropriate wordlist³ to check users' passwords and take appropriate action when a weak password is detected.

4.10.16 Logging off idle users

Idle users are usually a security problem, a user might be idle maybe because he's out to lunch or because a remote connection was broken and not re-established. For whatever the reason, idle users might lead to a compromise:

- because the user's console might not be locked and can be accessed by an intruder.
- because an attacker might be able to re-attach himself to a closed network connection and send commands to the remote shell (this is fairly easy if the remote shell is not encrypted as in the case of `telnet`).

Some remote systems have even been compromised through an idle (detached) `screen`.

Automatic disconnection of idle users is usually a part of the local security policy that must be enforced. There are several ways to do this:

- If `bash` is the user shell, a system administrator can set a default `TMOUT` value (see `bash(1)`) which will make the shell automatically remote idle users. Note that it must be set with the `-o` option or users will be able to change (or unset) it.
- Install `timeoutd` and configure `/etc/timeoutd` according to your local security policy. The daemon will watch for idle users and time out their shells accordingly.
- Install `autolog` and configure it to remove idle users.

The `timeoutd` or `autolog` daemons are the preferred method since, after all, users can change their default shell or can, after running their default shell, switch to another (uncontrolled) shell.

4.11 Using `tcpwrappers`

TCP wrappers were developed when there were no real packet filters available and access control was needed. The TCP wrappers allow you to allow or deny a service for a host or a domain and define a default allow or deny rule. If you want more information take a look at `hosts_access(5)`.

Many services installed in Debian are either:

³Try `apt-cache search wordlist` for a list of available packages which might provide wordlists. You can also retrieve wordlists from many ftp sites over the Internet. FIXME: add links

- launched through the `tcpwrapper` service (`tcpd`)
- compiled with `libwrapper` support built-in.

On the one hand, for services configured in `/etc/inetd.conf` (this includes `telnet`, `ftp`, `netbios`, `swat` and `finger`) you will see that the configuration file executes `/usr/sbin/tcpd` first. On the other hand, even if a service is not launched by the `inetd` superdaemon, support for the `tcp` wrappers rules can be compiled into it. Services compiled with `tcp` wrappers in Debian include `ssh`, `portmap`, `in.talk`, `rpc.statd`, `rpc.mountd`, `gdm`, `oaf` (the GNOME activator daemon), `nessus` and many others.

To see which packages use `tcpwrappers` try:

```
$ apt-cache showpkg libwrap0 | egrep '^[[:space:]]' | sort -u | \
  sed 's/,libwrap0$//;s/^[[:space:]]\+//'
```

Take this into account when running `tcpchk`. You can add services that are linked to the wrapper library into the `hosts.deny` and `hosts.allow` files but `tcpchk` will warn that it is not able to find those services since it looks for them in `/etc/inetd.conf` (the manpage is not totally accurate here).

Now, here comes a small trick, and probably the smallest intrusion detection system available. In general, you should have a decent firewall policy as a first line, and `tcp` wrappers as the second line of defense. One little trick is to set up a `SPAWN`⁴ command in `/etc/hosts.deny` that sends mail to root whenever a denied service triggers wrappers:

```
ALL: ALL: SPAWN ( \
  echo -e "\n\
  TCP Wrappers\ : Connection refused\n\
  By\ : $(uname -n)\n\
  Process\ : %d (pid %p)\n\
  User\ : %u\n\
  Host\ : %c\n\
  Date\ : $(date)\n\
  " | /usr/bin/mail -s "Connection to %d blocked" root) &
```

Beware: The above printed example is open to a DoS attack by making many connections in a short period of time. Many emails mean a lot of file I/O by sending only a few packets.

4.12 The importance of logs and alerts

It is easy to see that the treatment of logs and is an important issue in a secure system. Suppose a system is perfectly configured and 99% secure. If the 1% attack occurs, and there are no

⁴be sure to use uppercase here since `spawn` will not work

security measures in place to, first, detect this and, second, raise alarms, the system is not secure at all.

Debian GNU/Linux provides some tools to perform log analysis, most notably `logcheck` or `loganalysis` (both will need some customisation to remove unnecessary things from the report). It might also be useful, if the system is nearby, to have the system logs printed on a virtual console. This is useful since you can (from a distance) see if the system is behaving properly. Debian's `/etc/syslog.conf` comes with a commented default configuration; to enable it uncomment the lines and restart `syslogd` (`/etc/init.d/syslogd restart`):

```
daemon,mail.*;\
    news.=crit;news.=err;news.=notice;\
    *.*=debug;*.=info;\
    *.*=notice;*.=warn          /dev/tty8
```

There is a lot regarding log analysis that cannot be fully covered here, a good resource for information is Counterpane's Log Analysis Resources (<http://www.counterpane.com/log-analysis.html>). In any case, even automated tools are no match for the best analysis tool: your brain.

4.12.1 Using and customising logcheck

The `logcheck` package in Debian is divided into two packages `logcheck` (the main program) and `logcheck-database` (a database of regular expressions for the program). The Debian default (in `/etc/cron.d/logcheck`) is that `logcheck` is run daily at 2 AM and once after each reboot.

This tool can be quite useful if properly customised to alert the administrator to unusual events in the system. `Logcheck` can be fully customised so that it can send mails from events recovered from the logs that are worthy of attention. The default installation includes profiles for ignored events and policy violations for three different setups (workstation, server and paranoid). The Debian package includes a configuration file `/etc/logcheck/logcheck.conf`, sourced by the program, that defines which user the checks are sent to. It also provides a way for packages that provide services to implement new policies in the directories: `/etc/logcheck/hacking.d/_packagename_`, `/etc/logcheck/violations.d/_packagename_`, `/etc/logcheck/violations.ignore.d/_packagename_`, `/etc/logcheck/ignore.d.paranoid/_packagename_`, `/etc/logcheck/ignore.d.server/_packagename_`, and `/etc/logcheck/ignore.d.workstation/_packagename_`. However, not many packages currently do so. If you have a policy that can be useful for other users, please send it as a bug report for the appropriate package (as a *wishlist* bug). For more information read `/usr/share/doc/logcheck/README.Debian`

The best way to configure `logcheck` is to install it (it will ask for the user to which reports should be mailed and generate `/etc/logcheck/logcheck.logfiles` from `syslog` entries). If you wish to add new log files just add them to `/etc/logcheck/logcheck.logfiles`.

The package dependency will also force the installation of `logcheck-database`; during installation it will ask which security level is desired: `workstation`, `server` or `paranoid`. This will make `/etc/logcheck/ignore.d` point to the appropriate directories (through symbolic links). To change this run `dpkg-reconfigure -p low logcheck-database`. Then create the `/etc/ignore.d/local`, this file will hold all the rules to exclude messages that should not be reported. Leave it empty for the moment (a simple `cp /dev/null /etc/ignore.d/local` will work).

Once this is done you might want to check the mails that are sent, for the first few days/weeks/months. If you find you are sent messages you do not wish to receive, just add the regular expressions (see `regex(7)`) that correspond to these messages to the `/etc/ignore.d/local`. It's an ongoing tuning process; once the messages that are sent are always relevant you can consider the tuning finished. Note that if `logcheck` does not find anything relevant in your system it will not mail you even if it does run (so you might get a mail only once a week, if you are lucky).

4.12.2 Configuring where alerts are sent

Debian comes with a standard `syslog` configuration (in `/etc/syslog.conf`) that logs messages to the appropriate files depending on the system facility. You should be familiar with this; have a look at the `syslog.conf` file and the documentation if not. If you intend to maintain a secure system you should be aware of where log messages are sent so they do not go unnoticed.

For example, sending messages to the console also is an interesting setup useful for many production-level systems. But for many such systems it is also important to add a new machine that will serve as `loghost` (i.e. it receives logs from all other systems).

Root's mail should be considered also, many security controls (like `snort`) send alerts to root's mailbox. This mailbox usually points to the first user created in the system (check `/etc/aliases`). Take care to send root's mail to some place where it will be read (either locally or remotely).

There are other role accounts and aliases on your system. On a small system, it's probably simplest to make sure that all such aliases point to the root account, and that mail to root is forwarded to the system administrator's personal mailbox.

FIXME: it would be interesting to tell how a Debian system can send/receive SNMP traps related to security problems (jfs). Check: `snmptraplogd`, `snmp` and `snmpd`.

4.12.3 Using a loghost

A loghost is a host which collects `syslog` data remotely over the network. If one of your machines is cracked, the intruder is not able to cover his tracks, unless he hacks the loghost as well. So, the loghost should be especially secure. Making a machine a loghost is simple. Just start the `syslogd` with `syslogd -r` and a new loghost is born. In order to do this permanently in Debian, edit `/etc/init.d/sysklogd` and change the line

```
SYSLOGD= " "
```

to

```
SYSLOGD= "-r"
```

Next, configure the other machines to send data to the loghost. Add an entry like the following to `/etc/syslog.conf`:

```
facility.level @your_loghost
```

See the documentation for what to use in place of *facility* and *level* (they should not be entered verbatim like this). If you want to log everything remotely, just write:

```
*.* @your_loghost
```

into your `syslog.conf`. Logging remotely as well as locally is the best solution (the attacker might presume to have covered his tracks after deleting the local log files). See the `syslog(3)`, `syslogd(8)` and `syslog.conf(5)` manpages for additional information.

4.12.4 Log file permissions

It is not only important to decide how alerts are used, but also who has read/modify access to the log files (if not using a remote loghost). Security alerts which the attacker can change or disable are not worth much in the event of an intrusion. Also, you have to take into account that log files might reveal quite a lot of information about your system to an intruder if he has access to them.

Some log file permissions are not perfect after the installation (but of course this really depends on your local security policy). First `/var/log/lastlog` and `/var/log/faillog` do not need to be readable by normal users. In the `lastlog` file you can see who logged in recently, and in the `faillog` you see a summary of failed logins. The author recommends `chmod 660` for both. Take a brief look at your log files and decide very carefully which log files to make readable/writable for a user with a UID other than 0 and a group other than 'adm' or 'root'. You can easily check this in your system with:

```
# find /var/log -type f -exec ls -l {} \; | cut -c 17-35 | sort -u
(see to what users do files in /var/log belong)
# find /var/log -type f -exec ls -l {} \; | cut -c 26-34 | sort -u
(see to what groups do files in /var/log belong)
# find /var/log -perm +004
(files which are readable by any user)
# find /var/log \! -group root \! -group adm -exec ls -ld {} \;
(files which belong to groups not root or adm)
```

To customize how log files are created you will probably have to customize the program that generates them. If the log file gets rotated, however, you can customize the behavior of creation and rotation.

4.13 Using chroot

`chroot` is one of the most powerful possibilities to restrict a daemon or a user or another service. Just imagine a jail around your target, which the target cannot escape from (normally, but there are still a lot of conditions that allow one to escape out of such a jail). If you do not trust a user, you can create a change root environment for him. This can use quite a bit of disk space as you need to copy all needed executables, as well as libraries, into the jail. Even if the user does something malicious, the scope of the damage is limited to the jail.

A good example for this case is, if you do not authenticate against `/etc/passwd` but use LDAP or MySQL instead. So your `ftp-daemon` only needs a binary and perhaps a few libraries. A `chrooted` environment would be an excellent security improvement; if a new exploit is found for this `ftp-daemon`, then attackers can only exploit the UID of the `ftp-daemon-user` and nothing else.

Of course, many other daemons could benefit from this sort of arrangement as well.

However, be forewarned that a `chroot` jail can be broken if the user running in it is the super-user. So, you need to make the service run as a non-privileged user. By limiting its environment you are limiting the world readable/executable files the service can access, thus, you limit the possibilities of a privilege escalation by use of local system security vulnerabilities. Even in this situation you cannot be completely sure that there is no way for a clever attacker to somehow break out of the jail. Using only server programs which have a reputation for being secure is a good additional safety measure. Even minuscule holes like open file handles can be used by a skilled attacker for breaking into the system. After all, `chroot` was not designed as a security tool but as a testing tool.

As an additional note, the Debian default BIND (the Internet name service) is not shipped `chroot'ed` per default; in fact, no daemons come `chroot'ed`.

There is also some software (not currently in Debian but which might be packaged in the future) that can help set up `chroot` environments. See

4.14 Adding kernel patches

FIXME: More content

Debian GNU/Linux provides some of the patches for the Linux kernel that enhance its security. These include:

- Linux Intrusion Detection (in package `lids-2.2.19`)

- Linux Capabilities (in package `lcap`)
- Linux Trustees (in package `trustees`)
- NSA Enhanced Linux (in package `selinux` also available from the developer's website (<http://www.coker.com.au/selinux/>))
- `kernel-patch-2.2.18-openwall` (<http://packages.debian.org/kernel-patch-2.2.18-openwall>).
- `kernel-patch-2.2.19-harden`
- Linux capabilities (in package `lcap`)
- IPSEC kernel support (in package `kernel-patch-freeswan`)
- `kernel-patch-int`

However, some patches have not been provided in Debian yet. If you feel that some of these should be included please ask for it at the Work Needing and Prospective Packages (<http://wnpp.debian.org>). Some of these are:

- PaX patch (<http://pageexec.virtualave.net/>)
- HAP patch (<http://www.theaimsgroup.com/~hleinh/hap-linux/>)
- Stealth patch (<http://www.energymech.net/madcamel/fm/>)

4.15 Protecting against buffer overflows

Buffer overflow is the name of a common attack to software which makes use of insufficient boundary checking (a common programming error) in order to execute machine code through a program's inputs. These attacks, against server software which listens to connections remotely and against local software which grant higher privileges to users (`setuid` or `setgid`) can result in the compromise of any given system.

There are mainly four methods to protect against buffer overflows:

- patch the kernel to prevent stack execution.
- using a library, such as `libsafe`, to overwrite vulnerable functions and introduce proper checking (for information on how to install `libsafe` read this (<http://www.Linux-Sec.net/harden/libsafe.uhow2.txt>)).
- recompile code to introduce proper checks that prevent overflows, using, for example, `stackguard`.
- use tools to find and fix code that might introduce this vulnerability.

Debian GNU/Linux, as of the 3.0 release, only provides software to implement the first and last of these methods (kernel patches and tools to detect possible buffer overflows). The use of tools to detect buffer overflows requires, in any case, of programming experience in order to fix (and recompile) the code. Debian provides, for example: `bfbtester` (a buffer overflow tester that brute-forces binaries through command line and environment overflows) and `njamd`.

As for kernel patches (described in the section ‘Adding kernel patches’ on page 54), the Openwall patch provides protection against buffer overflows in 2.2 linux kernels. However, for 2.4 kernels, you need to use the Grsecurity patch (in the `kernel-patch-2.4-grsecurity` which includes the Openwall patch and many more features (<http://www.grsecurity.net/features.htm>) (including ACLs and network randomness to make it more difficult to remote OS fingerprinting), or the Linux Security Modules (in the `kernel-patch-2.4-lsm` and `kernel-patch-2.5-lsm` packages).

In any case, be aware, that even these workarounds might not prevent buffer overflows since there are ways to circumvent these, as described in phrack’s magazine issue 58 (<http://packetstorm.linuxsecurity.com/mag/phrack/phrack58.tar.gz>).

4.16 Secure file transfers

During normal system administration one usually needs to transfer files in and out from the installed system. Copying files in a secure manner from a host to another can be achieved by using the `sshd` server package. Another possibility is the use of `ftpd-ssl`, a ftp server which uses the *Secure Socket Layer* to encrypt the transmissions.

Any of these methods needs, of course, special clients. Debian provides clients, for example the `ssh` provides `scp`. It works like `rcp` but is encrypted completely, so the *bad guys* cannot even find out WHAT you copy. There is also a `ftpd-ssl` client package for the equivalent server. You can find clients for these software even for other operating systems (non-UNIX), `putty` and `winscp` provide secure copy implementations for any version of Microsoft’s operating system.

Note that using `scp` provides access to the users to all the file system unless `chroot`’ed as described in ‘Chrooting ssh’ on page 69. FTP access can be `chroot`’ed, probably easier depending on you chosen daemon, as described in ‘Securing FTP’ on page 71. If you are worried about users browsing your local files and want to have encrypted communication you can either use an ftp daemon with SSL support or combine cleartext ftp and a VPN setup (see ‘Virtual Private Networks’ on page 111).

4.17 File System limits and control

4.17.1 Using quotas

Having a good quota policy is important, as it keeps users from filling up the hard disk(s).

You can use two different quota systems: user quota and group quota. As you probably figured out, user quota limits the amount of space a user can take up, group quota does the equivalent for groups. Keep this in mind when you're working out quota sizes.

There are a few important points to think about in setting up a quota system:

- Keep the quotas small enough, so users do not eat up your disk space.
- Keep the quotas big enough, so users do not complain or their mail quota keeps them from accepting mail over a longer period.
- Use quotas on all user-writable areas, on `/home` as well as on `/tmp`.

Every partition or directory which users have full write access should be quota enabled. Calculate and assign a workable quota size for those partitions and directories which combines usability and security.

So, now you want to use quotas. First of all you need to check whether you enabled quota support in your kernel. If not, you will need to recompile it. After this, control whether the package `quota` is installed. If not you will need this one as well.

Enabling quota for the respective file systems is as easy as modifying the `defaults` setting to `defaults,usrquota` in your `/etc/fstab` file. If you need group quota, substitute `usrquota` to `grpquota`. You can also use them both. Then create empty `quota.user` and `quota.group` files in the roots of the file systems you want to use quotas on (e.g. `touch /home/quota.user /home/quota.group` for a `/home` file system).

Restart quota by doing `/etc/init.d/quota stop;/etc/init.d/quota start`. Now quota should be running, and quota sizes can be set.

Editing quotas for a specific user (say 'ref') can be done by `edquota -u ref`. Group quotas can be modified with `edquota -g <group>`. Then set the soft and hard quota and/or inode quotas as needed.

For more information about quotas, read the quota man page, and the quota mini-howto(`/usr/share/doc/HOWTO/en-html/mini/Quota.html`).

You might or might not like `lshell`, since it violates the FHS. Also take into account that `pam_limits` so might provide the same functionality and `lshell` is currently orphaned (<http://bugs.debian.org/93894>)

4.17.2 `chattr`/`lsattr`

These two commands are very useful, but they only work for the `ext2` file system. With '`lsattr`' you can list the attributes of a file and with '`chattr`' you can change them. Note that attributes are not the same thing as permissions. There are many attributes, but only the most important for increasing security are mentioned here. There are two flags which can only be set by the superuser.

First there is the 'a' flag. If set on a file, this file can only be opened for appending. This attribute is useful for some of the files in `/var/log/`, though you should consider they get moved sometimes due to the log rotation scripts.

The second flag is the 'i' flag, short for immutable. If set on a file, it can neither be modified nor deleted or renamed and no link be created to it. If you do not want users to look into your config files you could set this flag and remove readability. Furthermore it can give you a little bit more security against intruders, because the cracker might be confused by not being able to remove a file. Nevertheless, you should never assume that the cracker is blind. After all, he got into your system.

You can, also, remove the `chattr` and `lsattr` programs from the system so that an intruder with root access cannot change (or list) this attributes. Since they are part of the `e2fsprogs` and it's *Required* priority you cannot simply remove it. However, you can safely delete these two applications (and probably some others) from the file system. Copy them before to a removable media (floppy disk?) along with they `md5sums`.

An intruder in the system would have to download his own copies of the binaries in the system (probably even compile them in it) which might give you a littler more time to detect and recover from the compromise before the whole system is overrun.

FIXME: This is a bug that could be reported, are any of the binaries provided by the program useful in production systems? If not, and since the libraries are needed by many packages a new package `e2fsprogs-utils` could be included with less than *Required* priority.

Remember: `lsattr` and `chattr` are only available on ext2 file systems.

4.17.3 Checking file system integrity

Are you sure `/bin/login` on your hard drive is still the binary you installed there some months ago? What if it is a hacked version, which stores the entered password in a hidden file or mails it in cleartext version all over the internet?

The only method to have some kind of protection is to check your files every hour/day/month (I prefer daily) by comparing the actual and the old `md5sum` of this file. Two files cannot have the same `md5sum` (the MD5 digest is 128 bits, so the chance that two different files will have the same `md5sum` is roughly one in 3.4e3803), so you're on the safe site here, unless someone has also hacked the algorithm that creates `md5sums` on that machine. This is, well, extremely difficult and very unlikely. You really should consider this auditing of your binaries as very important, since it is an easy way to recognize changes at your binaries. Common tools used for this are `sXid`, `AIDE` (Advanced Intrusion Detection Environment), `TripWire` (non-free; the new version will be GPL), `integrit` and `samhain`.

Installing `debsums` will help to check the file system integrity, by comparing the `md5sums` of every file against the `md5sums` used in the Debian package archive. But beware, those files can easily be changed.

Furthermore you can replace `locate` with `slocate`. `slocate` is a security enhanced version of GNU `locate`. When using `slocate`, the user only sees the files he really has access to and you can exclude any files or directories on the system.

FIXME: put references to the snapshot taken after installation.

FIXME: Add a note regarding packages not providing debsums for all apps installed (not mandatory).

4.17.4 Setting up setuid check

Debian provides a cron job that runs daily in `/etc/cron.daily/standard`. This cron job will run the `/usr/sbin/checksecurity` script that will store information of this changes.

In order for this check to be made you must set `CHECKSECURITY_DISABLE="FALSE"` in `/etc/checksecurity.conf`. Note, this is the default, so unless you have changed something, this option will already be set to "FALSE".

The default behavior does not send this information to the superuser but, instead keeps daily copies of the changes in `/var/log/setuid.changes`. You should set the `CHECKSECURITY_EMAIL` (in `/etc/checksecurity.conf`) to 'root' to have this information mailed to him. . See `checksecurity(8)` for more configuration info.

4.18 Securing network access

FIXME. More (Debian-specific) content needed

4.18.1 Configuring kernel network features

FIXME: Content missing

Many features of the kernel can be modified while running by echoing something into the `/proc` file system or by using `sysctl`. By entering `/sbin/sysctl -A` you can see what you can configure and what the options are, and it can be modified running `/sbin/sysctl -w variable=value` (see `sysctl(8)`). Only in rare cases do you need to edit something here, but you can increase security that way as well. For example:

```
net/ipv4/icmp_echo_ignore_broadcasts = 1
```

This is a *Windows emulator* because it acts like Windows on broadcast ping if this option is set to 1. That is, ICMP_ECHO request sent to the broadcast address will be ignored. Otherwise, it does nothing.

If you want to block any ICMP echo requests on your system, enable this configuration option:

```
net/ipv4/icmp_echo_ignore_all = 0
```

To log packets with impossible addresses (due to wrong routes) on your network use:

```
/proc/sys/net/ipv4/conf/all/log_martians = 1
```

For more information on what things can be done with `/proc/sys/net/ipv4/*` read `/usr/src/linux/Documentation/filesystems/proc.txt`. All the options are describe thoroughly under `/usr/src/linux/Documentation/networking/ip-sysctl.txt` ⁵.

Configuring Syncookies

This option is a double-edged sword. On the one hand it protects your system against syn flooding; on the other hand it violates defined standards (RFCs).

```
net/ipv4/tcp_syncookies = 1
```

If you want to change this option you each time the kernel is working you need to change it in `/etc/network/options` by setting `syncookies=yes`. This will take effect whener `/etc/init.d/networking` is run (which is done at boot time) whileas this will only work with the current running kernel:

```
echo 1 > /proc/sys/net/ipv4/tcp_syncookies
```

This option will only be available if the kernel is compiled with the `CONFIG_SYNCOOKIES`. All Debian kernels are compiled with this option builtin but you can verify it running:

```
$ sysctl -A |grep syncookies
net/ipv4/tcp_syncookies = 1
```

For more information on TCP syncookies read <http://cr.yp.to/syncookies.html>.

4.18.2 Securing the network on boot-time

When setting configuration options for the kernel networking you need configure it so that it's loaded every time the system is restarted. The following example enables many of the previous options as well as other useful options.

Create the script in `/etc/network/interface-secure` (the name is given as an example) and call it from `/etc/network/interfaces` like this:

```
auto eth0
iface eth0 inet static
    address xxx.xxx.xxx.xxx
```

⁵In Debian the `kernel-image` package installs the sources under `/usr/src/kernel-souce-2.X.X`, just substitute `linux` to whatever kernel is installed

```
netmask 255.255.255.xxx
broadcast xxx.xxx.xxx.xxx
gateway xxx.xxx.xxx.xxx
pre-up /etc/network/interface-secure

# Script-name: /etc/network/interface-secure
# Modifies some default behaviour in order to secure against
# some TCP/IP spoofing & attacks
#
# Contributed by Dariusz Puchalak
#
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
# broadcast echo protection enable
echo 0 > /proc/sys/net/ipv4/ip_forward # ip forwarding disabled
echo 1 > /proc/sys/net/ipv4/tcp_syncookies # TCP syn cookie protection enable
echo 1 > /proc/sys/net/ipv4/conf/all/log_martians
# Log packets with impossible address
# but be careful with this on heavy loaded web servers
echo 1 > /proc/sys/net/ipv4/ip_always_defrag
# defragging protection always enabled
echo 1 > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses
# bad error message protection enabled

# now ip spoofing protection
for f in /proc/sys/net/ipv4/conf/*/rp_filter; do
    echo 1 > $f
done

# and finally some more things:
# Disable ICMP Redirect Acceptance
for f in /proc/sys/net/ipv4/conf/*/accept_redirects; do
    echo 0 > $f
done

for f in /proc/sys/net/ipv4/conf/*/send_redirects; do
    echo 0 > $f
done

# Disable Source Routed Packets
for f in /proc/sys/net/ipv4/conf/*/accept_source_route; do
    echo 0 > $f
done

# Log Spoofed Packets, Source Routed Packets, Redirect Packets
for f in /proc/sys/net/ipv4/conf/*/log_martians; do
    echo 1 > $f
```

done

You can also create a `init.d` script and have it run on bootup (using `update-rc.d` to create the appropriate `rc.d` links).

4.18.3 Configuring firewall features

In order to have firewall capabilities, either to protect the local system or others *behind* it, the kernel needs to be compiled with firewall capabilities. The standard Debian 2.2 kernel (also 2.2) provides the packet filter `ipchains` firewall, Debian 3.0 standard kernel (kernel 2.4) provides the *stateful* packet filter `iptables` (netfilter) firewall. Older Debian distributions would need the appropriate kernel patch (Debian 2.1 uses kernel 2.0.34).

In any case, it is pretty easy to use a kernel different from the one provided by Debian. You can find pre-compiled kernels as packages you can easily install in the Debian system. You can also download the kernel sources using the `kernel-source-x` and build custom kernel packages using `make-kpkg`.

Setting up firewalls in Debian is discussed more thoroughly in ‘Adding firewall capabilities’ on page 87.

4.18.4 Disabling weak-end hosts issues

Systems with more than one interface on different networks can have services configured so that they will bind only to a given IP address. This usually prevents services when requested through a given address. However, this does not mean (although it’s a common misconception even I had) that the service is bound to a given *hardware* address (interface card).⁶

This is not an ARP issue and it’s not an RFC violation (it’s called *weak end host* in RFC1122 (<ftp://ftp.isi.edu/in-notes/rfc1122.txt>), section 3.3.4.2). Remember, IP addresses have nothing to do with physical interfaces.

On 2.2 (and previous) kernels this can be fixed with:

```
# echo 1 > /proc/sys/net/ipv4/conf/all/hidden
# echo 1 > /proc/sys/net/ipv4/conf/eth0/hidden
# echo 1 > /proc/sys/net/ipv4/conf/eth1/hidden
.....
```

On later kernels this can be fixed either with:

⁶To reproduce this (example provided by Felix von Leitner on the bugtraq mailing list):

```
host a (eth0 connected to eth0 of host b): ifconfig eth0 10.0.0.1 ifconfig eth1
23.0.0.1 tcpserver -RHl localhost 23.0.0.1 8000 echo fnord host b: ifconfig eth0
10.0.0.2 route add 23.0.0.1 gw 10.0.0.1 telnet 23.0.0.1 8000
```

It seems, however, not to work with services bound to 127.0.0.1, you might need to write the tests using raw sockets.

- iptables rules.
- properly configured routing.⁷
- kernel patching.⁸

Along this text there will be many occasions in which it is shown how to configure some services (sshd server, apache, printer service...) in order to have them listening on any given address, the reader should take into account that, without the fixes given here, the fix would not prevent accesses from within the same (local) network.⁹

FIXME: comments on bugtraq indicate there is a Linux specific method to bind to a given interface.

FIXME: Submit a bug against netbase so that the routing fix is standard behaviour in Debian?

4.18.5 Protecting against ARP attacks

When you don't trust the other boxes on your LAN (which should always be the case, because it's the safest attitude) you should protect yourself from the various existing ARP attacks.

As you know the ARP protocol is used to link IP addresses to MAC addresses. (see RFC826 (<ftp://ftp.isi.edu/in-notes/rfc826.txt>) for all the details). Every time you send a packet to an IP address an arp resolution is done (first by looking into the local ARP cache then if the IP isn't present in the cache by broadcasting an arp query) to find the target's hardware address. All the ARP attacks aim to fool your box into thinking that box B's IP address is associated to the intruder's box's MAC address; Then every packet that you want to send to the IP associated to box B will be send to the intruder's box...

Those Attacks (Cache poisoning, ARP spoofing...) allow the attacker to sniff the traffic even on switched networks, to easily hijack connections, to disconnect any host from the network... Arp attack are powerful and simple to implement, several tools exists : arpspoof (present in package `dsniff`), arpmim (<http://www.team-teso.net/releases/arpmim-0.2.tar.gz>), arpoison (<http://web.syr.edu/~sabuer/arpoison/>)...

However, there is always a solution:

⁷The fact that this behaviour can be changed through routing was described by Matthew G. Marsh in the bugtraq thread:

```
eth0 = 1.1.1.1/24 eth1 = 2.2.2.2/24 ip rule add from 1.1.1.1/32 dev lo table 1 prio
15000 ip rule add from 2.2.2.2/32 dev lo table 2 prio 16000 ip route add default dev
eth0 table 1 ip route add default dev eth1 table 2
```

⁸There are some patches available for this behaviour as described in bugtraq's thread at <http://www.linuxvirtualserver.org/~julian/#hidden> and <http://www.fefe.de/linux-eth-forwarding.diff>.

⁹An attacker might have many problems pulling the access through after configuring the IP-address binding if he is not on the same broadcast domain (same network) as the attacked host. If the attack goes through a router it might be quite difficult for the answers to return somewhere.

- Use a static arp cache. You can set up “static” entries in your arp cache :

```
arp -s host_name hwaddr
```

By setting static entries for each important host in your network you ensure that nobody will create/modify a (fake) entry for these hosts (static entries don't expire and can't be modified) and spoofed arp replies will be ignored.

- Detect suspicious ARP traffic. You can use `arpwatch`, `karpiski` or more general IDS that can also detect suspicious arp traffic (`snort`, `prelude` (<http://www.mandrakelinux.com/prelude>)...).
- Implement IP traffic filtering validating the the MAC address.

4.19 Taking a snapshot of the system

Before putting the system into production system you could take a snapshot of the whole system. This snapshot could be used in the event of a compromise (see 'After the compromise' on page 127). You should remake this upgrade whenever the system is upgraded, specially if you upgrade to a new Debian release.

For this you can use a writable removable-media that can be set up read-only, this could be a floppy disk (read protected after use) or a CD on a CD-ROM unit (you could use a rewriteable CD-ROM so you could even keep backups of md5sums in different dates).

The following script creates such a snapshot:

```
#!/bin/bash
/bin/mount /dev/fd0 /mnt/floppy
/bin/cp /usr/bin/md5sum /mnt/floppy
echo "Calculating md5 database"
>/mnt/floppy/md5checksums.txt
for dir in /bin/ /sbin/ /usr/bin/ /usr/sbin/ /lib/ /usr/lib/
do
    find $dir -type f | xargs /usr/bin/md5sum >>/mnt/floppy/md5checksums-lib.t
done
/bin/umount /dev/fd0
echo "post installation md5 database calculated"
```

Note that the `md5sum` binary is placed on the floppy drive so it can be used later on to check the binaries of the system (just in case it gets trojaned).

The snapshot does not include the files under `/var/lib/dpkg/info` which includes the md5 hashes of installed packages (in files ended with `.md5sums`). You could copy this information along too, however you should notice:

- the md5sums provided by the Debian packages include all the files provided by them, which makes the database bigger (5 Mbs versus 600kbs in a Debian GNU/Linux system with graphical system and around 2.5 Gbs of software installed)
- not all Debian packages provide md5sums for the files installed since it is not (currently) mandated policy.

Once the snapshot is done you should make sure to set the medium read-only. You can then store it for backup or place it in the drive and use it to drive a cron check nightly comparing the original md5sums against those on the snapshot.

4.20 Other recommendations

4.20.1 Do not use software depending on svgalib

SVGAlib is very nice for console lovers like me, but in the past it has been proven several times that it is very insecure. Exploits against zgv were released, and it was simple to become root. Try to prevent using SVGAlib programs wherever possible.

Chapter 5

Securing services running on your system

Services can be secured in a running system in two ways:

- Making them only accessible in the access points (interfaces) they need to be in.
- Configuring them properly so that they can only be used by legitimate users in an authorised manner.

Restricting services so that they can only be accessed from a given place can be done by restricting access to them at the kernel (i.e. firewall) level, configure them to listen only on a given interface (some services might not provide this feature) or using some other methods, for example the linux vserver patch (for 2.4.16) can be used to force processes to use only one interface.

Regarding the services running from `inetd` (`telnet`, `ftp`, `finger`, `pop3`...) it is worth noting that `inetd` cannot be configured so that services only listen on a given interface. However, its substitute, the `xinetd` meta-daemon includes a `bind` just for this matter. See `xinetd.conf(5)`.

```
service nntp
{
    socket_type      = stream
    protocol        = tcp
    wait            = no
    user            = news
    group          = news
    server          = /usr/bin/env
    server_args     = POSTING_OK=1 PATH=/usr/sbin:/usr/bin:/sbin:/bin
+ /usr/sbin/snntpd logger -p news.info
    bind            = 127.0.0.1
}
```

The following sections detail how determined services can be configured properly depending on their intended use.

5.1 Securing ssh

If you are still running telnet instead of ssh, you should take a break from this manual and change this. Ssh should be used for all remote logins instead of telnet. In an age where it is easy to sniff internet traffic and get cleartext passwords, you should use only protocols which use cryptography. So, perform an `apt-get install ssh` on your system now.

Encourage all the users on your system to use ssh instead of telnet, or even better, uninstall telnet/telnetd. In addition you should avoid logging into the system using ssh as root and use alternative methods to become root instead, like `su` or `sudo`. Finally, the `sshd_config` file, in `/etc/ssh`, should be modified to increase security as well:

- `ListenAddress 192.168.0.1`

Have ssh listen only on a given interface, just in case you have more than one (and do not want ssh available on it) or in the future add a new network card (and don't want ssh connections from it).

- `PermitRootLogin No`

Try not to permit Root Login wherever possible. If anyone wants to become root via ssh, now two logins are needed and the root password cannot be brute forced via SSH.

- `Listen 666`

Change the listen port, so the intruder cannot be completely sure whether a `sshd` daemon runs (be forewarned, this is security by obscurity).

- `PermitEmptyPasswords no`

Empty passwords make a mockery of system security.

- `AllowUsers alex ref me@somewhere`

Allow only certain users to have access via ssh to this machine. `user@host` can also be used to restrict a given user from accessing only at a given host.

- `AllowGroups wheel admin`

Allow only certain group members to have access via ssh to this machine. `AllowGroups` and `AllowUsers` have equivalent directives for denying access to a machine. Not surprisingly they are called "DenyUsers" and "DenyGroups".

- `PasswordAuthentication yes`

It is completely your choice what you want to do. It is more secure only to allow access to machine from users with ssh-keys placed in the `~/.ssh/authorized_keys` file. If you want so, set this one to "no".

- Disable any forms of authentication you do not really need, if you do not use, for example `RhostsRSAAuthentication`, `HostbasedAuthentication`, `KerberosAuthentication` or `RhostsAuthentication` (for exaple) you should disable them, even if they are already by default (see the manpage `sshd_config(5)`).
- Protocol 2
Disable the protocol version 1, since it has some design flaws that make it easier to crack passwords. For more information read a paper regarding ssh protocol problems (<http://paris.cs.berkeley.edu/~dawnson/papers/ssh-timing.pdf>) or the Xforce advisory (<http://xforce.iss.net/static/6449.php>).
- Banner /etc/some_file
Add a banner (it will retrieved from the file) to users connecting to the ssh server, in some countries sending a warning before access to a given system warning about unauthorised access or user monitoring should be added to have legal protection.

You can also restrict access to the ssh server using `pam_listfile` or `pam_wheel` in the PAM control file for ssh to restrict ssh logins. For example, you could keep anyone not listed in `/etc/loginusers` by adding this line to `/etc/pam.d/ssh`:

```
auth required pam_listfile.so sense=allow onerr=fail item=user file
```

As a final note, be aware that these directives are from a OpenSSH configuration file. Right now, there are three commonly used SSH daemons, `ssh1`, `ssh2`, and OpenSSH by the OpenBSD people. `Ssh1` was the first ssh daemon available and it is still the most commonly used (there are rumors that there is even a Windows port). `Ssh2` has many advantages over `ssh1` except it is released under an closed-source license. OpenSSH is completely free ssh daemon, which supports both `ssh1` and `ssh2`. OpenSSH is the version installed on Debian when the package `ssh` is chosen.

You can read more information on how to set up SSH with PAM support in the security mailing list archives (<http://lists.debian.org/debian-security/2001/debian-security-200111/msg00395.html>).

5.1.1 Chrooting ssh

Currently OpenSSH does not provide a way to chroot automatically users upon connection (the commercial version does provide this functionality). However there is a project to provide this functionality for OpenSSH too, see <http://chrootssh.sourceforge.net>, it is not currently packaged for Debian, though. You could use, however, the `pam_chroot` module as described in 'Restricting users's access' on page 44.

In 'Chroot environment for SSH' on page 177 you can several options to make chroot environment for SSH.

5.1.2 Ssh clients

If you are using an SSH client against the SSH server you must make sure that it supports the same protocols that are enforced on the server. For example, if you use the `mindterm` package, it only supports protocol version 1. However, the `sshd` server is, by default, configured to only accept version 2 (for security reasons).

5.1.3 Disallowing file transfers

If you do *not* want users to transfer files to and from the ssh server you need to restrict access to the `sftp-server` *and* the `scp` access. You can restrict `sftp-server` by configuring the proper Subsystem in the `/etc/ssh/sshd_config`. However, to restrict `scp` access, however, you must either:

- disallow users from login to the ssh server (as described above either through the configuration file or PAM configuration).
- do not give valid shells to users which are not allowed secure transfers. The shells provided, however, should be programs that would make connecting to the ssh server was useful at all, such as menu programs (ala BBS). Otherwise the previous option is preferred.

5.2 Securing Squid

Squid is one of the most popular proxy/cache server, and there are some security issues that should be taken into account. Squid's default configuration file denies all users requests. However the Debian package allows access from 'localhost', you just need to configure your browser properly. You should configure Squid to allow access to trusted users, hosts or networks defining an Access Control List on `/etc/squid.conf`, see the Squid User's Guide (<http://squid-docs.sourceforge.net/latest/html/book1.htm>) for more information about defining ACLs rules.

Also, if not properly configured, someone may relay a mail message through Squid, since the HTTP and SMTP protocols are designed similarly. Squid's default configuration file denies access to port 25. If you wish to allow connections to port 25 just add it to `Safe_ports` lists. However, this is *NOT* recommended.

Setting and configuring the proxy/cache server properly is only part of keeping your site secure. Another necessary task is to analyse Squid's logs to assure that all things are working as they should be working. There are some packages in Debian GNU/Linux that can help an administrator to do this. The following packages are available in woody (Debian 3.0):

- `calamaris` - Log analyzer for Squid or Oops proxy log files.
- `modlogan` - A modular logfile analyzer.

- `squidtailed` - Squid log monitoring program.

When using Squid in Accelerator Mode it acts as a web server too. Turning on this option code complexity increases, making it less reliable. By default Squid is not configured to act as a web server, so you don't need to worry about this. Note that if you want to use this feature be sure that it is really necessary. To find more information about Accelerator Mode on Squid see the Squid User's Guide #Chapter9 (<http://squid-docs.sourceforge.net/latest/html/c2521.htm>).

5.3 Securing FTP

If you really have to use FTP (without wrapping it with `sslwrap` or inside a SSL or SSH tunnel), you should `chroot ftp` into the ftp users' home directory, so that the user is unable to see anything else than their own directory. Otherwise they could traverse your root file system just like if they had a shell in it. You can add the following line in your `proftpd.conf` in your global section to enable this `chroot` feature:

```
DefaultRoot ~
```

Restart `proftpd` by `/etc/init.d/proftpd restart` and check whether you can escape from your `homedir` now.

To prevent Proftpd DoS attacks using `../../../../`, add the following line in `/etc/proftpd.conf`:
`DenyFilter *.*/*`

Always remember that FTP sends login and authentication passwords in clear text (this is not an issue if you are providing an anonymous public service) and there are better alternatives in Debian for this. For example, `sftp` (provided by `ssh`). There are also free implementations of SSH for other operating systems: `putty` (<http://www.chiark.greenend.org.uk/~sgtatham/putty/>) and `cygwin` (<http://www.cygwin.com>) for example.

However, if you still maintain the FTP server while making users access through SSH you might encounter a typical problem. Users accessing Anonymous FTP servers inside SSH-secured systems might try to log in the *FTP server*. While the access will be refused, the password will nevertheless be sent through the net in clear form. To avoid that, ProFTPD developer TJ Saunders has created a patch that prevents users feeding the anonymous FTP server with valid SSH accounts. More information and patch available at: ProFTPD Patches (<http://www.castaglia.org/proftpd/#Patches>).

5.4 Securing access to the X Window System

Today, X terminals are used by more and more companies where one server is needed for a lot of workstations. This can be dangerous, because you need to allow the file server to connect

to the the clients (X server from the X point of view. X switches the definition of client and server). If you follow the (very bad) suggestion of many docs, you type `xhost +` on your machine. This allows any X client to connect to your system. For slightly better security, you can use the command `xhost +hostname` instead to only allow access from specific hosts.

A much more secure solution, though, is to use ssh to tunnel X and encrypt the whole session. This is done automatically when you ssh to another machine. This has to be enabled in `/etc/ssh/ssh_config` by setting `X11Forwarding` to `yes`. In times of SSH, you should drop the `xhost` based access control completely.

For best security, if you do not need X access from other machines, is to switch off the binding on tcp port 6000 simply by typing:

```
$ startx -- -nolisten tcp
```

This is the default behavior in Xfree 4.1.0 (the Xserver provided in Debian 3.0). If you are running Xfree 3.3.6 (i.e. you have Debian 2.2 installed) you can edit `/etc/X11/xinit/xserverrc` to have it something along the lines of:

```
#!/bin/sh
exec /usr/bin/X11/X -dpi 100 -nolisten tcp
```

If you are using XDM set `/etc/X11/xdm/Xservers` to: `:0 local /usr/bin/X11/X vt7 -dpi 100 -nolisten tcp`. If you are using Gdm make sure that the `-nolisten tcp` option is set in the `/etc/gdm/gdm.conf` (which is the default in Debian) such as this:

```
[server-Standard]
name=Standard Server
command=/usr/bin/X11/X -nolisten tcp
```

You can also set the default's system timeout for `xscreensaver` locks. Even if the user can override it, you should edit the `/etc/X11/app-defaults/XScreenSaver` configuration file and change the lock line:

```
*lock:                                False
```

(which is the default in Debian) to:

```
*lock:                                True
```

FIXME: add information on how to disable the screensavers which show the user desktop (which might have sensitive information).

Read more on X Window security in XWindow-User-HOWTO (<http://www.linuxdoc.org/HOWTO/XWindow-User-HOWTO.html>)(`/usr/share/doc/HOWTO/en-txt/XWindow-User-HOWTO`

FIXME: Add info on thread of debian-security on how to change config files of XFree 3.3.6 to do this.

5.4.1 Check your display manager

If you only want to have a display manager installed for local usage (having a nice graphical login, that is), make sure the XDMCP (X Display Manager Control Protocol) stuff is disabled. In XDM you can do this with this line in `/etc/X11/xdm/xdm-config`:

```
DisplayManager.requestPort: 0
```

Normally, all display managers are configured not to start XDMCP services per default in Debian.

5.5 Securing printing access (The lpd and lprng issue)

Imagine, you arrive at work, and the printer is spitting out endless amounts of paper because someone is DoSing your line printer daemon. Nasty, isn't it?

In any unix printing architecture, there has to be a way to get the client's data to the host's print server. In traditional `lpr` and `lp`, the client command copies or symlinks the data into the spool directory (which is why these programs is usually SUID or SGID).

In order to avoid any issues you should keep your printer servers specially secure. This means you need to configure your printer service so it will only allow connections from a set of trusted servers. In order to do this, add the servers you want to allow printing to your `/etc/hosts.lpd`.

However, even if you do this, the `lpr` daemon accepts incoming connections on port 515 of any interface. You should consider firewalling connections from networks/hosts which are not allowed printing (the `lpr` daemon cannot be limited to listen only on a given IP address).

`Lprng` should be preferred over `lpr` since it can be configured to do IP access control. And you can specify which interface to bind to (although somewhat weirdly).

If you are using a printer in your system, but only locally, you will not want to share this service over a network. You can consider using other printing systems, like the one provided by `cups` or PDQ (<http://pdq.sourceforge.net/>) which is based on user permissions of the `/dev/lp0` device.

In `cups`, the print data is transferred to the server via the `http` protocol. This means the client program doesn't need any special privileges, but does require that the server be listening on a port somewhere.

However, if you want to use `cups`, but only locally, you can configure it to bind to the loopback interface by changing `/etc/cups/cupsd.conf`:

```
Listen 127.0.0.1:631
```

There are many other security options like allowing or denying networks and hosts in this config file. However, if you do not need them you might be better off just limiting the listening port. CUPS also serves documentation through the HTTP port, if you do not want to disclose potential useful information to outside attackers (and the port is open) add also:

```
<Location />
  Order Deny,Allow
  Deny From All
  Allow From 127.0.0.1
</Location>
```

This configuration file can be modified to add some more features including SSL/TLS certificates and crypto. The manuals are available at <http://localhost:631/> or at cups.org.

FIXME: Add more content (the article on Amateur Fortress Building (<http://www.rootprompt.org>) provides some very interesting views).

FIXME: Check if PDG is available in Debian, and if so, suggest this as the preferred printing system.

FIXME: Check if Farmer/Wietse has a replacement for printer daemon and if it's available in Debian.

5.6 Securing the mail service

If your server is not a mailing system, you do not really need to have a mail daemon listening for incoming connections, but you might want local mail delivered in order, for example, to receive mail for the root user from any alert systems you have in place.

If you have `exim` you do not need the daemon to be working in order to do this since the standard cron job flushes the mail queue. See 'Disabling daemon services' on page 27 on how to do this.

5.6.1 Configuring a Nullmailer

However you might want to have a local mailer daemon so that it can relay the mails sent locally to a real system. This is common when you have to administer a number of systems and do not want to connect to each of them to read the local mail sent, it is usual to relay the local mail to a remote system that consolidates all of it (similarly to what should be done to logging for those systems).

Such a *relay-only* system should be configured properly for this. The daemon could, as well, be configured to only listen on the loopback address.

To do this in a Debian system, you will have to remove the `smtp` daemon from `inetd`:

```
$ update-inetd --disable smtp
```

and configure the mailer daemon to only listen on the loopback interface. In `exim` (the default MTA) you can do this by editing the file `/etc/exim.conf` and adding the following line:

```
local_interfaces = "127.0.0.1"
```

Restart both daemons (`inetd` and `exim`) and you will have `exim` listening on the `127.0.0.1:25` socket only. Be careful, and first disable `inetd`, otherwise, `exim` will not start since the `inetd` daemon is already handling incoming connections.

For postfix edit `/etc/postfix/main.conf`:

```
inet_interfaces = localhost
```

If you only want local mail, this approach is better than tcp-wrapping the mailer daemon or adding firewalling rules to limit anybody accessing it. However, if you do need it to listen on other interfaces, you might consider launching it from `inetd` and adding a tcp wrapper so incoming connections are checked against `/etc/hosts.allow` and `/etc/hosts.deny`. Also, you will be aware of when an unauthorized access is attempted against your mailer daemon, if you set up proper logging for any of the methods above.

In any case, to reject mail relay attempts at the SMTP level, you can change `/etc/exim/exim.conf` to include:

```
receiver_verify = true
```

Even if your mail server will not relay the message, this kind of configuration is needed for the relay tester at <http://www.abuse.net/relay.html> to determine that your server is *not* relay capable.

If you want a relay-only setup, however, you can consider changing the mailer daemon to programs that can *only* be configured to forward the mail to a remote mail server. Debian provides currently both `ssmtp` and `nullmailer` for this purpose. In any case, you can evaluate for yourself any of the mail transport agents ¹ provided by Debian and see which one suits best to the system's purposes.

¹To retrieve the list of mailer daemons available in Debian try:

```
$ apt-cache search mail-transport-agent
```

The list will not include `qmail`, which is distributed only as source code in the `qmail-src` package.

5.6.2 Providing secure access to mailboxes

If you want to give remote access to mailboxes there are a number of POP3 and IMAP daemons available². However, if you provide IMAP access note that it is a general file access protocol, it can become the equivalent of a shell access because users might be able to retrieve any file that they can through it.

Try, for example, to configure as your inbox path `{server.com}/etc/passwd` if it succeeds your IMAP daemon is not properly configured to prevent this kind of access.

Of the IMAP servers in Debian the `cyrus` server (in the `cyrus-imapd` package) gets around this by having all access be to a database in a restricted part of the file system. Also, `uw-imapd` (either installing the `uw-imapd` or, better if your IMAP clients support it, `uw-imapd-ssl`) can be configured to chroot the users mail directory but this is not enabled by default. The documentation provided gives more information on how to configure it.

Also, you might want to run an IMAP server that does not need valid users to be created on the local system (which would might grant shell access too), both `courier-imap` (for IMAP) and `courier-pop` `teapop` (for POP3) and `cyrus-imapd` (for both POP3 and IMAP) provide servers with authentication methods beside the local user accounts. `cyrus` can use any authentication method that can be configured through PAM whileas `teapop` might use databases (such as `postgresql` and `mysql`) for user authentication.

FIXME: Check: `uw-imapd` might be configured with user authentication through PAM too..

5.6.3 Receiving mail securely

Reading/receiving mail is the most common cleartext protocol. If you use either POP3 or IMAP to get your mail, you send your cleartext password across the net, so almost anyone can read your mail from now on. Instead, use SSL (Secure Sockets Layer) to receive your mail. The other alternative is `ssh`, if you have a shell account on the box which acts as your POP or IMAP server. Here is a basic `fetchmailrc` to demonstrate this:

```
poll my-imap-mailserver.org via "localhost"
  with proto IMAP port 1236
    user "ref" there with password "hackme" is alex here warnings 3600
  folders
    .Mail/debian
  preconnect 'ssh -f -P -C -L 1236:my-imap-mailserver.org:143 -l ref
    my-imap-mailserver.org sleep 15 </dev/null > /dev/null'
```

The `preconnect` is the important line. It fires up a `ssh` session and creates the necessary tunnel, which automatically forwards connections to `localhost` port 1236 to the IMAP mail server, but encrypted. Another possibility would be to use `fetchmail` with the `ssl` feature.

²A list of servers/daemons which support these protocols in Debian can be retrieved with:

```
$ apt-cache search pop3-server $ apt-cache search imap-server
```

If you want to provide encrypted mail services like POP and IMAP, `apt-get install stunnel` and start your daemons this way:

```
stunnel -p /etc/ssl/certs/stunnel.pem -d pop3s -l /usr/sbin/popd
```

This command wraps the provided daemon (-l) to the port (-d) and uses the specified ssl cert (-p).

5.7 Securing BIND

There are different issues that can be tackled in order to secure the Domain server daemon, which are similar to the ones considered when securing any given service:

- configure the daemon itself properly so it cannot be misused from the outside. This includes limiting possible queries from clients: zone transfers and recursive queries.
- limit the access of the daemon to the server itself so if it is used to break in, the damage to the system is limited. This includes running the daemon as a non-privileged user and chrooting it.

You should restrict some of the information that is served from the DNS server to outside clients so that it cannot be used to retrieve valuable information from your organization that you do not want to give away. This includes adding the following options: *allow-transfer*, *allow-query*, *allow-recursive* and *version*. You can either limit this on the global section (so it applies to all the zones served) or on a per-zone basis. This information is documented on the `bind-doc` package, read more on this on `/usr/share/doc/bind/html/index.html` once the package is installed.

Imagine that your server is connected to the Internet and to your internal (your internal IP is 192.168.1.2) network (a basic multi-homed server), you do not want to give any service to the Internet and you just want to enable DNS lookups from your internal hosts. You could restrict it by including in `/etc/bind/named.conf`:

```
options {
    allow-query { 192.168.1/24; } ;
    allow-transfer { none; } ;
    allow-recursive { 192.168.1/24; } ;
    listen-on { 192.168.1.2; } ;
    forward { only; } ;
    forwarders { A.B.C.D; } ;
};
```

The *listen-on* option makes the DNS bind to only the interface that has the internal address, but, even if this interface is the same as the interface that connects to the Internet (if you are using

NAT, for example), queries will only be accepted if coming from your internal hosts. If the system has multiple interfaces and the *listen-on* is not present, only internal users could query, but, since the port would be accessible to outside attackers, they could try to crash (or exploit buffer overflow attacks) on the DNS server. You could even make it listen only on 127.0.0.1 if you are not giving DNS service for any other systems than yourself.

The `version.bind` record in the `chaos` class contains the version of the of the currently running `bind` process. This information is often used by automated scanners and malicious individuals who wish to determine if one's `bind` is vulnerable to a specific attack. By providing false or no information in the `version.bind` record, one limits the probability that one's server will be attacked based on its publicized version. To provide your own version, use the *version* directive in the following manner:

```
options {
    ... various options here ...
    version "Not available.";
};
```

Changing the `version.bind` record does not provide actual protection against attacks, but it might be considered a useful safeguard.

A sample `named.conf` configuration file might be the following:

```
acl internal {
    127.0.0.1/32;           // localhost
    10.0.0.0/8;           // internal
    aa.bb.cc.dd;         // eth0 IP
};

acl friendly {
    ee.ff.gg.hh;         // slave DNS
    aa.bb.cc.dd;         // eth0 IP
    127.0.0.1/32;       // localhost
    10.0.0.0/8;         // internal
};

options {
    directory "/var/cache/bind";
    allow-query { internal; };
    allow-recursive { internal; };
    allow-transfer { none; };
};
// From here to the mysite.bogus zone
// is basically unmodified from the debian default
logging {
    category lame-servers { null; };
};
```

```
        category cname { null; };
};

zone "." {
    type hint;
    file "/etc/bind/db.root";
};

zone "localhost" {
    type master;
    file "/etc/bind/db.local";
};

zone "127.in-addr.arpa" {
    type master;
    file "/etc/bind/db.127";
};

zone "0.in-addr.arpa" {
    type master;
    file "/etc/bind/db.0";
};

zone "255.in-addr.arpa" {
    type master;
    file "/etc/bind/db.255";
};

// zones I added myself
zone "mysite.bogus" {
    type master;
    file "/etc/bind/named.mysite";
    allow-query { any; };
    allow-transfer { friendly; };
};
```

Please (again) check the Bug Tracking System regarding Bind, specifically Bug #94760 (regarding ACLs on zone transfers) (<http://bugs.debian.org/94760>). Feel free to contribute to the bug report if you think you can add useful information.

5.7.1 Changing BIND's user

Regarding limiting BIND's privileges you must be aware that if a non-root user runs BIND, then BIND cannot detect new interfaces automatically. For example, if you stick a PCMCIA card into your laptop. Check the README.Debian file in your named documentation (/usr

/share/doc/bind/README.Debian) directory for more information about this issue. There have been many recent security problems concerning BIND, so switching the user is useful when possible. We will detail here the steps needed in order to do this, however, if you want to do this in an automatic way you might try the script provided in 'Sample script to change the default Bind installation.' on page 169.

To run BIND under a different user, first create a separate user and group for it (it is *not* a good idea to use nobody or nogroup for every service not running as root). In this example, the user and group named will be used. You can do this by entering:

```
addgroup named
adduser --system --home /home/named --no-create-home --ingroup named \
    --disabled-password --disabled-login named
```

Notice that the user named will be quite restricted. If you want, for whatever reason, to have a less restrictive setup use:

```
adduser --system --ingroup named named
```

Now edit /etc/init.d/bind with your favorite editor and change the line beginning with

```
start-stop-daemon --start
```

to

```
start-stop-daemon --start --quiet --exec /usr/sbin/named -- -g named -u named
```

Also, in order to avoid running anything as root, change the reload line commenting out:

```
reload)
    /usr/sbin/ndc reload
```

And change it to:

```
reload)
    $0 stop
    sleep 1
    $0 start
```

Note: Depending on your Debian version you might have to change the restart line too. This was fixed in Debian's bind version 1:8.3.1-2.

All you need to do now is to restart bind via '/etc/init.d/bind restart', and then check your syslog for two entries like this:

```
Sep  4 15:11:08 nexus named[13439]: group = named
Sep  4 15:11:08 nexus named[13439]: user = named
```

Voilà! Your named now *does not* run as root. If you want to read more information on why BIND does not run as non-root user on Debian systems, please check the Bug Tracking System regarding Bind, specifically Bug #50013: bind should not run as root (<http://bugs.debian.org/50013>) and Bug #132582: Default install is potentially insecure (<http://bugs.debian.org/132582>), Bug #53550 (<http://bugs.debian.org/53550>), Bug #128120 (<http://bugs.debian.org/128120>), Bug #52745 (<http://bugs.debian.org/52745>), and Bug #128120 (<http://bugs.debian.org/128120>). Feel free to contribute to the bug reports if you think you can add useful information.

5.7.2 Chrooting the name server

To achieve maximum BIND security, now build a chroot jail (see ‘Using chroot’ on page 54) around your daemon. There is an easy way to do this: the `-t` option (see the `named(8)` manpage). This will make Bind chroot itself into the given directory without you needing to set up a chroot jail and worry about dynamic libraries. The only files there needs to be in these chroot jail:

```
dev/null
etc/bind/          - should hold named.conf and all the server zones
sbin/named-xfer    - if you do name transfers
var/run/named/     - should hold the pid and the name server cache (if
                    any) this directory needs to be writable by named
                    user
var/log/named      - if you set up logging to a file, needs to be writable
                    for the named user
dev/log            - syslogd should be listening here if named is configure to
                    log through it
```

In order for your Bind daemon to work properly it needs permission in the named files. This is an easy task since the configuration files are always at `/etc/named/`. Take in account that it only needs read-only access to the zone files, unless it is a secondary or cache name server. If this is your case you will have to give read-write permissions to the necessary zones (so that zone transfers from the primary server work).

Also, you can find more information regarding Bind chrooting in the Chroot-BIND-HOWTO (<http://www.linuxdoc.org/HOWTO/Chroot-BIND-HOWTO.html>) (regarding Bind 9) and Chroot-BIND8-HOWTO (<http://www.linuxdoc.org/HOWTO/Chroot-BIND8-HOWTO.html>) (regarding Bind 8). This same documents should be available through the installation of the `doc-linux-text` (text version) or `doc-linux-html` (html version). Another useful document is <http://www.psionic.com/papers/dns/dns-linux>.

If you are setting up a full chroot jail (i.e. not just `-t`) for Bind 8.2.3 in Debian (potato), make sure you have the following files in it:

```
dev/log - syslogd should be listening here
dev/null
etc/bind/named.conf
etc/localtime
etc/group - with only a single line: "named:x:GID:"
etc/ld.so.cache - generated with ldconfig
lib/ld-2.1.3.so
lib/libc-2.1.3.so
lib/ld-linux.so.2 - symlinked to ld-2.1.3.so
lib/libc.so.6 - symlinked to libc-2.1.3.so
sbin/ldconfig - may be deleted after setting up the chroot
sbin/named-xfer - if you do name transfers
var/run/
```

And modify also `syslogd` listen on `$CHROOT/dev/log` so the named server can write syslog entries into the local system log.

If you want to avoid problems with dynamic libraries, you can compile bind statically. You can use `apt-get` for this, with the `source` option. It can even download the packages you need to properly compile it. You would need to do something similar to:

```
$ apt-get --download-only source bind build-dep bind
$ cd bind-8.2.5-2
(edit the Makefile.in so CFLAGS includes the '-static' option
before the @CFLAGS@ definition substituted by autoconf)
$ dpkg-buildpackage -rfakeroot
$ cd ..
$ dpkg -i bind-8.2.5-2*deb
```

After installation, you will need to move around the files to the chroot jail³ you can keep the `init.d` scripts in `/etc/init.d` so that the system will automatically start the name server, but edit them to add `--chroot /location_of_chroot` in the calls to `start-stop-daemon` in those scripts.

For more information on how to set up chroots see 'General chroot and suid paranoia' on page 85.

FIXME, merge info from <http://people.debian.org/~pzn/howto/chroot-bind.sh.txt>, <http://people.pdxlinux.org/~karlheg/> (Bind9 on Debian), <http://www.cryptio.net/~ferlatte/config/> (Debian-specific), <http://www.psionic.com/papers/whitep01.html>, <http://csrc.nist.gov/fasp/FASPDocs/NISTSecuringDNS.htm> and <http://www.acmebw.com/papers/securing.pdf>.

³unless you use the `instdir` option when calling `dpkg` but then the chroot jail might be a little more complex

5.8 Securing Apache

FIXME: Add content: modules provided with the normal Apache installation (under `/usr/lib/apache/X.X/m` and modules that can be installed separately in `libapache-mod-XXX` packages.

You can limit access to the Apache server if you only want to use it internally (for testing purposes, to access the `doc-central` archive, etc..) and do not want outsiders to access it. To do this use the `Listen` or `BindAddress` directives in `/etc/apache/http.conf`.

Using `Listen`:

```
Listen 127.0.0.1:80
```

Using `BindAddress`:

```
BindAddress 127.0.0.1
```

Then restart apache with `/etc/init.d/apache restart` and you will see that it is only listening on the loopback interface.

In any case, if you are not using all the functionality provided by Apache, you might want to take a look at other web servers provided in Debian like `dhttpd`.

The Apache Documentation (http://httpd.apache.org/docs/misc/security_tips.html) provides information regarding security measures to be taken on Apache webserver (this same information is provided in Debian by the `apache-doc` package).

FIXME: Add pointer to Intersect's Alliance.

5.8.1 Disabling users from publishing web contents

The default Apache installation in Debian permits users to publish contents under the `$HOME/publish_html`. These contents can be retrieved remotely using an URL such as: `http://your_apache_server/publish_html`.

If you do not want to permit this you must change the `/etc/apache/http.conf` configuration file which includes:

```
<Directory /home/*/public_html>
    AllowOverride FileInfo AuthConfig Limit
    Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
    <Limit GET POST OPTIONS PROPFIND>
        Order allow,deny
        Allow from all
    </Limit>
    <Limit PUT DELETE PATCH PROPPATCH MKCOL COPY MOVE LOCK UNLOCK>
        Order deny,allow
```

```
        Deny from all
    </Limit>
</Directory>
```

Change it to:

```
<Directory /home/*/public_html>
    AllowOverride None
    Order deny,allow
    Deny from all
</Directory>
```

Note: An attacker might still do user enumeration, since the answer of the web server will be a *403 Permission Denied* and not a *404 Not available*.

5.8.2 Logfiles permissions

Apache logfiles, since 1.3.22-1, are owned by user 'root' and group 'adm' with permissions 640 this permissions are changed after rotation. An intruder that accessed the system through the web server would not be able (without privilege escalation) to remove old log file entries.

5.8.3 Published web files

Apache files are located under `/var/www`. Just after installation the default file provides some information on the system (mainly that it's a Debian system running Apache). The default webpages are owned by user root and group root by default, whileas the Apache process runs as user www-data and group www-data. This should make attackers that compromise the system through the web server harder to deface the site. You should, of course, substitute the default web pages (which might provide information you do not want to show to outsiders) with your own.

5.9 Securing finger

If you want to run the finger service first ask yourself if you need to do so. If you do, you will find out that Debian provides many finger daemons (output from `apt-cache search fingerd`):

- cfingerd - Configurable finger daemon
- efingerd - Another finger daemon for unix capable of fine-tuning your output.
- ffingerd - a secure finger daemon

- `fingerd` - Remote user information server.
- BSD-like finger daemon with `qmail` support.

`fingerd` is the recommended finger daemon if you are going to use it for a public service. In any case, you are encouraged to, when setting it up through `inetd`, `xinetd` or `tcpserver`: limit the number of processes that will be running at the same time, limit access to the finger daemon from a given number of hosts (using `tcp wrappers`) and having it only listening to the interface you need it to be in.

5.10 General chroot and suid paranoia

It is probably fair to say that the complexity of BIND is the reason why it has been exposed to a lot of attacks in recent years. (see ‘Securing BIND’ on page 77)

Other programs with complex features and a large installed user base include Sendmail and some ftp daemons (e.g. `WUftpd`). (Of course, a program with no features and no satisfied users can be just as insecure, besides being useless.)

Anyway, if you run any of these, consider similar arrangements for them — revoking root privileges, running in a chroot jail — or replacing them with a more secure equivalent.

5.10.1 Automaking chrooting programs

There are several programs to chroot automatically servers and services. Debian currently (accepted in may 2002) provides Wietse Venema’s `chrootuid` in the `chrootuid` package. This program can be used to set up a restricted environment for executing a command (including running it as a restricted user). However, you must set up the chroot environment yourself.

In the near future Debian will provide tools to set up the chroot environment easily. The `makejail` program for example, can create and update a chroot jail with short configuration files (it provides sample configuration files for `bind`, `apache`, `postgresql` and `mysql`). It attempts to guess and install into the jail all files required by the daemon using `strace`, `stat` and Debian’s package dependancies. More information at <http://www.floc.net/makejail/>. Jailer is a similar tool which can be retrieved from <http://www.balabit.hu/downloads/jailer/>.

FIXME: I have packages ready for `makejail` and `jailer`, update this when they get accepted.

Also useful to create chroots (or jails) is `deb.pl`, a script that analyses dependencies of a set of files.

5.11 General cleartext password paranoia

You should try to avoid any network service which sends and receives passwords in cleartext over a net like FTP/Telnet/NIS/RPC. The author recommends the use of `ssh` instead of `telnet`

and ftp to everybody.

Keep in mind that migrating from telnet to ssh, but using other cleartext protocols does not increase your security in ANY way! Best would be to remove ftp, telnet, pop, imap, http and to supersede them with their respective encrypted services. You should consider moving from these services to their SSL versions, ftp-ssl, telnet-ssl, pop-ssl, https ...

Most of these above listed hints apply to every Unix system (you will find them if reading any other hardening-related document related to Linux and other Unices).

5.12 Disabling NIS

You should not use NIS, the Network Information Service, if it is possible, because it allows password sharing. This can be highly insecure if your setup is broken.

If you need password sharing between machines, you might want to consider using other alternatives. For example, you can set a LDAP server and configure PAM on your system in order to contact the LDAP server for user authentication. You can find a detailed setup in the LDAP-HOWTO (<http://www.linuxdoc.org/HOWTO/LDAP-HOWTO.html>) (`/usr/share/doc/HOWTO/en-txt/LDAP-HOWTO.txt.gz`).

Read more on NIS security in NIS-HOWTO (<http://www.linuxdoc.org/HOWTO/NIS-HOWTO.html>) (`/usr/share/doc/HOWTO/en-txt/NIS-HOWTO.txt.gz`).

FIXME (jfs): Add info on how to set up this in Debian

5.13 Disabling RPC services

You should disable RPC wherever possible, that is, when you do not need it. ⁴ Many security holes for both the portmapper service and RPC-based services are known and could be easily exploited. On the other hand NFS services are quite important in some networks, so find a balance of security and usability in your network. Some of the DDoS (distributed denial of service) attacks use rpc exploits to get into the system and act as a so called agent/handler. Read more on NFS security in NFS-HOWTO (<http://www.linuxdoc.org/HOWTO/NFS-HOWTO.html>) (`/usr/share/doc/HOWTO/en-txt/NFS-HOWTO.txt.gz`).

Disabling portmap is quite simple. There are different methods. The simplest one in a Debian 3.0 system is to do `uninstall` the `portmap` package. If you are running another version you will have to disable the service as seen in 'Disabling daemon services' on page 27, this is due to the program being a part of the `net-base` package (which cannot be de-installed without breaking the system).

This in fact removes every symlink relating to portmap in `/etc/rc${runlevel}.d/`, which is something you could also do manually. Another possibility is to `chmod 644 /etc/init.d/portmap`,

⁴You only probably need it if using NFS (Network File System), NIS (Network Information System) or some other RPC-based service.

but that gives an error message when booting. You can also strip off the `start-stop-daemon` part in `/etc/init.d/portmap` shell script.

5.14 Adding firewall capabilities

The Debian GNU/Linux operating system has the built-in capabilities provided by the Linux kernel. This means that if you install a potato (Debian 2.2 release) system (default kernel is 2.2) you will have `ipchains` firewalling available in the kernel, you need to install the `ipchains` which will be surely (due to its priority) be already installed. If you install a woody (Debian 3.0 release) system (default kernel is 2.4) you will have `iptables` (neftfilter) firewalling available. The main different between `ipchains` and `iptables` is that the later is based on *stateful packet inspection* which provides for more secure (and easier to build) filtering configurations.

5.14.1 Firewalling the local system

You can use firewall rules as a way to secure the access to your local system and, even, to limit the outbound communications made by it. Firewall rules can be used also to protect processes that cannot be properly configured *not* to provide services to some networks, IP addresses, etc..

However, this step is presented last in this manual basically because it is *much* better to not depend solely on firewalling capabilities in order to protect a given system. Security in a system is made up of layers, firewalling should be the last to include, once all services have been hardened. You can easily imagine a setup in which the system is solely protected by a built-in firewall and an administrator blissfully removes the firewall rules for whatever reason (problems with the setup, annoyance, human error...), this system would be wide open to an attack if there were no other hardening in the system to protect from it.

On the other hand, having firewall rules on the local system also prevents some bad things from happening. Even if the services provided are configured securely, a firewall can protect from misconfigurations or from fresh installed services that have not yet been properly configured. Also, a tight configuration will prevent trojans *calling home* from working unless the firewalling code is removed. Note that an intruder does *not* need superuser access to install a trojan locally that could be remotely controlled (since binding on ports is allowed if they are not privileged ports and capabilities have not been removed).

Thus, a proper firewall setup would be one with a default deny policy, that is:

- incoming connections are allowed only to local services by allowed machines.
- outgoing connections are only allowed to services used by your system (DNS, web browsing, pop, email...)⁵
- the forward rule denies everything (unless you are protecting other systems, see below).

⁵Unlike personal firewalls in other operating systems, Debian GNU/Linux does not (yet) provide firewall generation interfaces that can make rules limiting them per process or user. However, the `iptables` code can be configured to do this (see the `owner` module in the `iptables(8)` manpage)

- all other incoming or outgoing connections are denied.

5.14.2 Using a firewall to protect other systems

A Debian firewall can also be installed in order to protect, with filtering rules, access to systems *behind* it, limiting their exposure to the Internet. The firewall can be configured to prevent systems outside the local network to access services (ports) that are not public. For example, on a mail server, only port 25 (where the mail service is being given) needs to be accessible from the outside. A firewall can be configured to, even if there are other services besides the public ones, throw away packets (this is known as *filtering*) directed towards them.

You can even set up a Debian GNU/Linux box as a bridge firewall, i.e. a filtering firewall completely transparent to the network that lacks an IP address and thus cannot be attacked directly. Depending on the kernel you have installed, you might need to install the bridge firewall patch and then go to *802.1d Ethernet Bridging* when configuring the kernel and a new option *netfilter (firewalling) suport*. See the 'Setting up a bridge firewall' on page 163 for more information on how to set this up in a Debian GNU/Linux system).

5.14.3 Configuring the firewall

Of course, the configuration of the firewall is always system and network dependant. An administrator must know beforehand what is the network layout and the systems he wants to protect, the services that need to be accessed, and wether or not other network considerations (like NAT or routing) need to be taken into account. Be careful when configuring your firewall, as Laurence J. Lane says in the `iptables` package:

The tools can easily be misused, causing enormous amounts of grief by completely cripple network access to a computer system. It is not terribly uncommon for a remote system administrator to accidentally lock himself out of a system hundreds or thousands of miles away. One can even manage to lock himself out of a computer who's keyboard is under his fingers. Please, use due caution.

Remember this: just installing the `iptables` (or the older `firewalling` code) does not give you any protection, just provides the software. In order to have a firewall you need to *configure* it!

If you do not know much about firewalling, read the `Firewalling-HOWTO` that can be found in `doc-linux-text` package (other document formats also available). See 'Be aware of general security problems' on page 19 for more (general) pointers.

Doing it the Debian way

If you are using Debian 3.0, you will notice that you have the `iptables` package installed. This is the support for the 2.4.4+ kernels `netfilter` implementation. Since just after installation the system cannot *know* any firewall rules (firewall rules are too system-specific) you have to enable `iptables`. However, the scripts have been configured so that the administrator can set up firewall rules and then have the `init` scripts *learn* them and use them always as the setup for the firewall.

In order to do so you must:

- Configure the package so that it starts with the system. On newer versions (since 1.2.6a-1) this is asked for when the package is installed. You can configure it afterwards with `dpkg-reconfigure -pnow iptables`. *Note*: on older versions this was done by editing `/etc/default/iptables` so that the variable `enable_iptables_initd` was set to `true`.
- create a firewall setup using iptables, you can use the command line (see `iptables(8)`) or some of the tools provided by the Debian firewall packages (see ‘Using Firewall packages’ on the current page). You need to create one set of firewall rules to be used when the firewall is in *active* state and another to be used when the firewall is in *inactive* state (these can be just empty rules).
- save the rules you created using `/etc/init.d/iptables save_active` and `/etc/init.d/iptables save_active` by running these scripts with the firewall rules you want enabled.

Once this is done your firewall setup is saved in the `/var/lib/iptables/` directory and will be executed when the system boots (or when running the `initd` script with `start` and `stop` arguments). Please notice that the default Debian setups starts the firewalling code in the multiuser runlevels (2 to 5) pretty soon (10). Also, it is stopped in singleuser runlevel (1), change this if it does not match your local policy.

If you do not have a clue on how to set up your firewall rules manually consult the *Packet Filtering HOWTO* and *NAT HOWTO* provided by `iptables` for offline reading at `/usr/share/doc/iptables/html/`. Also, the configuration file `/etc/default/iptables` provides some more information about the issues regarding this package.

Using Firewall packages

Setting up manually a firewall can be complicated for novice (and sometimes even expert) administrators. However, the free software community has created a number of tools that can be used to easily configure a local firewall. Be forewarned that some of these tools are oriented more towards local-only protection (also known as *personal firewall*) and some are more versatile and can be used to configure complex rules to protect whole networks.

Some software that can be used to set up firewall rules in a Debian system is:

- `firestarter`, oriented towards end-users including a wizard to quickly define the firewall rules.
- `knetfilter`
- `fwbuilder`, an object oriented GUI which includes policy compilers for various firewall platforms including iptables as well as router’s access-lists.
- `shorewall` which provides support for IPsec as well as limited support for traffic shaping as well as the definition of the firewall rules.

- `mason`, which can propose firewall rules based on the network traffic your system “sees”.
- `bastille` (among the hardening steps that can make new versions of `bastille` is the possibility of adding firewall rules to the system to be executed on startup)
- `ferm`
- `fwctl`
- `easyfw`
- `firewall-easy`
- `ipac-ng`
- `gfcc`
- `lokkit` or `gnome-lokkit`

The last packages: `gfcc`, `firestarter` and `knetfilter` are administration GUIs using either GNOME (first two) or KDE (last one) which are much more user-oriented (i.e. for home users) than the other packages in the list which might be more administrator-oriented.

Be forewarned that some of the packages outlined previousline will probably introduce firewalling scripts to be run when the system boots, this will undoubtedly conflict with the common setup (if configured) and you might undesireds effects. Usually, the firewalling script that runs last will be the one that configures the system (which might not be what you pretend). Consult the package documentation and use either one of these setups. Generally, other programs that help you set up the firewalling rules can tweak othe conrfiguration files.

FIXME: Add more info regarding this packages

FIXME: Check Information on Debian firewalling and what/how does it change from other distributions.

FIXME: Where should the custom firewalling code be enabled (common FAQ in `debian-firewall`?)

FIXME: Add information on Zorp (<http://www.balabit.hu/downloads/zorp/debian/>) in Debian (see Bug #88347 (<http://bugs.debian.org/88347>)). Debian packages are provided but they depend on `libglib1.3` which is not yet available in Debian distribution.

Chapter 6

Automatic hardening of Debian systems

After reading through all the information in the previous chapters you might be wondering “I have to do quite a lot of things in order to harden my system, couldn’t this things be automated?”. The question is yes, but be careful with automated tools. Some people believe, that a hardening tool does not eliminate the need for good administration. So do not be fooled to think that you can automate all the process and will fix all the related issues. Security is an ever-ongoing process in which the administrator must participate and cannot just stand away and let the tools do all the work since no single tool can cope: with all the possible security policy implementations, all the attacks and all the environments.

Since woody (Debian 3.0) there are two specific packages that are useful for security hardening. The `hardening` which takes an approach based on the package dependencies to quickly install valuable security packages and remove those with flaws, configuration of the packages must be done by the administrator. The `bastille` that implements a given security policy on the local system based on previous configuration by the administrator (the building of the configuration can be a guided process done with simple yes/no questions).

6.1 Harden

The `hardening` package tries to make it more easy to install and administer hosts that need good security . This package should be used by people that want some quick help to enhance the security of the system. To do this it conflicts with packages with known flaws, including (but not limited to): known security bugs (like buffer overflows), use of plaintext passwords, lack of access control, etc. It also automatically installs some tools that should enhance security in some way: intrusion detection tools, security analysis tools, etc. Harden installs the following *virtual* packages (i.e. no contents, just dependencies on others):

- `hardening-tools`: tools to enhance system security (integrity checkers, intrusion detection, kernel patches...)

- `hardened-doc`: provides this same manual and other security-related documentation packages.
- `hardened-environment`: helps configure a hardened environment (currently empty).
- `hardened-servers`: removes servers considered insecure for some reason.
- `hardened-clients`: removes clients considered insecure for some reason.
- `hardened-remoteflaws`: removes packages with known security holes that could be used by a remote attacker to compromise the system (uses versioned *Conflicts:*).
- `hardened-localflaws`: removes packages with known security holes that could be used by a local attacker to compromise the system (uses versioned *Conflicts:*).
- `hardened-remoteaudit`: tools to remotely audit a system.

Be careful because if you have software you need (and which you do not wish to uninstall for some reason) and it conflicts with some of the packages above you might not be able to fully use `hardened`. The `hardened` packages do not (directly) do a thing. They do have, however, intentional package conflicts with known non-secure packages. This way, the Debian packaging system will not approve the installation of these packages. For example, when you try to install a `telnetd` daemon with `hardened-servers`, `apt` will say:

```
# apt-get install telnetd
The following packages will be REMOVED:
  hardened-servers
The following NEW packages will be installed:
telnetd
Do you want to continue (Y/n)
```

This should set off some warnings in the administrator head, who should reconsider his actions.

6.2 Bastille Linux

Bastille Linux (<http://www.bastille-linux.org>) is an automatic hardening tool originally oriented towards the RedHat and Mandrake Linux distributions. However, the `bastille` package provided in Debian (since woody) is patched in order to provide the same functionality for the Debian GNU/Linux system.

Bastille can be used with different frontends (all are documented in their own manpage in the Debian package) which enables the administrator to:

- Answer questions step by step regarding the desired security of your system (using `InteractiveBastille(8)`)

- Use a default setting for security (amongst three: Lax, Moderate or Paranoia) in a given setup (server or workstation) and let Bastille decide which security policy to implement (using `BastilleChooser(8)`)
- Take a predefined configuration file (could be provided by Bastille or made by the administrator) and implement a given security policy (using `AutomatedBastille(8)`)

Chapter 7

Debian Security Infrastructure

7.1 The Debian Security Team

Debian has a Security Team, made up of five members and two secretaries who handle security in the *stable* distribution. Handling security means they keep track of vulnerabilities that arise in software (watching forums such as bugtraq, or vuln-dev) and determine if the *stable* distribution is affected by it.

Also, the Debian Security Team, is the contact point for problems that are coordinated by upstream developers or organisations such as CERT (<http://www.cert.org>) which might affect multiple vendors. That is, when problems are not Debian-specific. There are two contact points with the Security Team:

- team@security.debian.org (<mailto:team@security.debian.org>) which only the members of the security team read.
- security@debian.org (<mailto:security@debian.org>) which is read by all Debian developers (including the security team). Mails sent to this list are not published in the Internet (it's not a public mailing list).

Sensitive information should be sent to the first address and, in some cases, should be encrypted with the Debian Security Contact key (key ID 363CCD95).

Once a probable problem is received by the Security Team it will investigate if the *stable* distribution is affected, if it is, a fix is worked for the source code base. This fix will sometimes include backporting the patch made upstream (which usually is some versions ahead of the one distributed by Debian). After testing of the fix is done, new packages are prepared and published in the security.debian.org site so they can be retrieved through apt (see 'Execute a security update' on page 38). At the same time a *Debian Security Advisory* (DSA) is published on the web site and sent to public mailing lists including [debian-security-announce](http://lists.debian.org/debian-security-announce) (lists.debian.org/debian-security-announce) and bugtraq.

Some other frequently asked questions on the Debian Security Team can be found at 'Questions regarding the Debian security team' on page 147.

7.2 Debian Security Advisories

Debian Security Advisories are made whenever a security vulnerability is discovered that affects a Debian package. These advisories, signed by one of the Security Team members, include information of the versions affected as well as the location of the updates and their MD5sums. This information is:

- version number for the fix.
- problem type.
- whether it is remote or locally exploitable.
- short description of the package.
- description of the problem.
- description of the exploit.
- description of the fix.

DSAs are published both in Debian's mainserver frontpage (<http://www.debian.org/>) and in the Debian security pages (<http://www.debian.org/security/>). Usually this does not happen until the website is rebuilt (once daily) so they might not be present immediately, the preferred channel is the `debian-security-announce` mailing list.

Interested users can, however (and this is done in some Debian-related portals) use the RDF channel to download automatically the DSAs to their desktop. Some applications, such as `Evolution` (an email client and personal information assistant) and `Multiticker` (a GNOME applet), can be used to retrieve the advisories automatically. The RDF channel is available at <http://www.debian.org/security/dsa.rdf>.

DSAs published on the website might be updated after being sent to the public-mailing lists. A common update is adding cross references to security vulnerability databases such as CVE (<http://cve.mitre.org>), CERT/CC vulnerability notes (<http://www.cert.org>) or Bugtraq (<http://www.securityfocus.com/bid/>). This feature was added to the website on June 2002.

One of the advantages of adding cross references to these vulnerability databases is that:

- it makes Debian users easier to see and track which general (published) advisories have already been covered by Debian.
- system administrators can learn more of the vulnerability and its impact by following the cross references.
- this information can be used to cross-check output from vulnerability scanners that include references to CVE to remove false positives (see 'Vulnerability assessment scanner X says my Debian system is vulnerable!' on page 143).

7.3 Debian Security Build Infrastructure

Since Debian is currently supported in a large number of architectures, administrators sometimes wonder if a given architecture might take more time to receive security updates than another. As a matter of fact, except for rare circumstances, updates are available to all architectures at the same time.

While previously the task to build security updates was done by hand, it is currently not (as Anthony Thowns describes in a mail sent to the `debian-devel-announce` mailing list dated 8th june 2002. FIXME: add pointer).

Packages uploaded by the security team (to [security.debian.org:/org/security.debian.org/queue/unchecked](https://security.debian.org/org/security.debian.org/queue/unchecked) or <ftp://security.debian.org/pub/SecurityUploadQueue> with an appropriate patch are checked for signatures within fifteen minutes of being uploaded, once this is done they get added to the list of the autobuilders (which no longer do a daily archive run). Thus, packages can get automatically built for *all* architectures thirty minutes or an hour or so after they're uploaded. However, security updates are a little more different than normal uploads sent by package maintainers since, in some cases, before being published they need to wait until they can be tested further, an advisory written, or need to wait for a week or more to avoid publicising the flaw until all vendors have had a reasonable chance to fix it.

Thus, the security upload archive works with the following procedure (called “*Accepted-Autobuilding*”):

- Someone finds a security problem.
- Someone fixes the problem, and makes an upload to `security.debian.org`'s incoming (this *someone* is usually a Security Team member but can be also a package maintainer with an appropriate fix that has contacted the Security Team previously). The Changelog includes a *testing-security* or *stable-security* as target distribution.
- The upload gets checked and processed by a Debian system and moved into `queue/accepted`, and the buildbots are notified. Files in here can be accessed by the security team and (somewhat indirectly) by the buildbots.
- Security-enabled buildbots pick up the source package (prioritized over normal builds), build it, and send the logs to the security team.
- The security team reply to the logs, and the newly built packages are uploaded to `queue/unchecked`, where they're processed by a Debian system, and moved into `queue/accepted`.
- When the security team find the source package acceptable (i.e., that it's been correctly built for all applicable architectures and that it fixes the security hole and doesn't introduce new problems of its own) they run a script which:
 - installs the package into the security archive.
 - updates the Packages, Sources and Release files of `security.debian.org` in the usual way (`dpkg-scanpackages`, `dpkg-scansources...`)

- sets up a template advisory that the security team can finish off.
- (optionally) forwards the packages to the appropriate proposed-updates so that it can be included in the real archive as soon as possible.

This procedure, previously done by hand, was tested and put through during the freezing stage of Debian 3.0 woody (july 2002). Thanks to this infrastructure the Security Team was able to have updated packages ready for the apache and OpenSSH issues for all the supported (almost twenty) architectures in less than a day.

7.3.1 Developer's guide to security updates

This mail was sent by Wichert Akkerman to the Debian-devel-announce mailing list ([+http://lists.debian.org/debian-devel-announce/2002/debian-devel-announce-200206/msg00004.html](http://lists.debian.org/debian-devel-announce/2002/debian-devel-announce-200206/msg00004.html)) in order to describe Debian developer's behaviour for handling security problems in their packages. It is published here both for the benefit of developers as well as for users to understand better how security is handled in Debian.

Coordinating with the security team

If a developer learns of a security problem, either in his package or someone else's he should always contact the security team (at team@security.debian.org). They keep track of outstanding security problems, can help maintainers with security problems or fix them themselves, are responsible for sending security advisories and maintaining security.debian.org.

Please note that security advisories are only done for release distributions, not for testing, unstable (see 'How is security handled for testing and unstable?' on page 148) or older distributions (see 'I use an older version of Debian, is it supported by the Debian Security Team?' on page 148).

Learning of security problems

There are a few ways a developer can learn of a security problem:

- he notices it on a public forum (mailing list, website, etc.):
- someone files a bugreport (the *Security* tag should be used, or added by the developer)
- someone informs him via private email.

In the first two cases the information is public and it is important to have a fix as soon as possible. In the last case however it might not be public information. In that case there are a few possible options for dealing with the problem:

- if it is a trivial problem (like insecure temporary files) there is no need to keep the problem a secret and a fix should be made and released.
- if the problem is severe (remote exploitable, possibility to gain root privileges) it is preferable to share the information with other vendors and coordinate a release. The security team keeps contacts with the various organizations and individuals and can take care of that.

In all cases if the person who reports the problem asks to not disclose the information that should be respected, with the obvious exception of informing the security team (the developer should make sure he tells the security team that the information can not be disclosed).

Please note that if secrecy is needed the developer can also not upload a fix to unstable (or anywhere else), since the changelog information for unstable is public information.

There are two reasons for releasing information even though secrecy is requested/required: the problem has been known for too long, or the information becomes public.

Building a package

The most important guideline when making a new package that fixes a security problem is to make as few changes as possible. People are relying on the exact behaviour of a release once it is made, so any change made to it can possibly break someone's system. This is especially true of libraries: the developer must make sure he never changes the API or ABI, no matter how small the change.

This means that moving to a new upstream version is not a good solution, instead the relevant changes should be backported. Generally upstream maintainers are willing to help if needed, if not the Debian security team might be able to help.

In some cases it is not possible to backport a security fix, for example when large amounts of sourcecode need to be modified or rewritten. If that happens it might be necessary to move to a new upstream version, but it should always be coordinated with the security team beforehand.

Related to this is another important aspect: developers must always test your change. If their is an exploit the developer should try if it indeed succeeds on the unpatched package and fails on the fixed package. The developer should try normal usage as well, sometimes a security fix can break normal use subtly.

Finally a few technical things for developers to keep in mind:

- Make sure you target the right distribution in your debian/changelog. For stable this is stable-security and for testing this is testing-security. Do not target <codename>-proposed-updates.
- Make sure the version number is proper. It has to be higher than the current package, but lower than package versions in later distributions. For testing this means there has to be a higher version in unstable. If there is none yet (testing and unstable have the same version for example) upload a new version to unstable first.

- Do not make source-only uploads if your package has any binary-all packages. The build infrastructure will not build those.
- Make sure when compiling a package you compile on a clean system which only has package installed from the distribution you are building for. If you do not have such a system yourself you can try a [debian.org](http://db.debian.org/machines.cgi) machine (see <http://db.debian.org/machines.cgi> or set up a chroot (the `pbuilder` and `debootstrap` packages can be helpful in that case).)

Uploading security fixes

After the developer has created and tested the new package it needs to be uploaded so it can be installed in the archives. For security uploads the place to upload to is `ftp://security.debian.org/pub/Security`.

Once an upload to the security queue has been accepted the package will automatically be rebuilt for all architectures and stored for verification by the security team.

Uploads waiting for acceptance or verification are only accessible by the security team. This is necessary since there might be fixes for security problems that can not be disclosed yet.

If a member of the security team accepts a package it will be installed on `security.debian.org` as well as the proper `<codename>-proposed-updates` in `ftp-master` or non-US archive.

The security advisory

Security advisories are written and posted by the security team. However they certainly do not mind if a maintainer can supply (part of) the text for them. Information that should be in an advisory is described in 'Debian Security Advisories' on page 96.

7.4 Package signing in Debian

This section could also be titled "how to upgrade/update safely your Debian GNU/Linux system" and it deserves its own section basically because it is an important part of the Security Infrastructure. Package signing is an important issue since it avoids tampering of packages distributed in mirrors and of downloads with man-in-the-middle attacks. Automatic software update is an important feature but it's also important to remove security threats that could help the distribution of trojans and the compromise of systems during updates ¹

As of today (december 2001) Debian does not provide signed packages for the distribution and the *woody* release (3.0) does not integrate that feature. There is a solution for signed packages which will be, hopefully, provided in the next release (*sarge*).

¹Some operating systems have already been plagued with automatic-updates problems such as the Mac OS X Software Update vulnerability (<http://www.cunap.com/~hardingr/projects/osx/exploit.html>). FIXME: probably the Internet Explorer vulnerability handling certificate chains has an impact on security updates on Microsoft Windows.

This issue is better described in the Strong Distribution HOWTO (http://www.cryptnet.net/fdp/crypto/strong_distro.html) by V. Alex Brennen.

7.4.1 The proposed scheme for package signature checks

The current (unimplemented) scheme for package signature checking using apt is:

- the Release file includes the md5sum of Packages.gz (which contains the md5sums of packages) and will be signed. The signature is one of a trusted source.
- This signed Release file is downloaded by 'apt-get update' and stored in the HD along with Packages.gz.
- When a package is going to be installed, it is first downloaded, then the md5sum is generated.
- The signed Release file is checked (signature ok) and it extracts from it the md5sum for the Packages.gz file, the Packages.gz checksum is generated and (if ok) the md5sum of the downloaded package is extracted from it.
- If the md5sum from the downloaded package is the same as the one in the Packages.gz file the package will be installed otherwise the administrator will be alerted and the package will be left in cache (so the administrator can decide whether to install it or not). If the package is not in the Packages.gz and the administrator has configured the system to only install checked packages it will not be installed either.

By following the chain of MD5 sums apt is capable of verifying that a package originates from a specific release. This is less flexible than signing each package one by one, but can be combined with that scheme too (see below).

Package signing has been discussed in Debian for quite some time, for more information you can read: <http://www.debian.org/News/weekly/2001/8/> and <http://www.debian.org/News/weekly/2000/11/>.

7.4.2 Alternative per-package signing scheme

The additional scheme of signing each and every packages allows packages to be checked when they are no longer referenced by an existing Packages file, and also third-party packages where no Packages ever existed for them can be also used in Debian but will not be default scheme.

This package signing scheme can be implemented using `debsig-verify` and `debsigs`. These two packages can sign and verify embedded signatures in the .deb itself. Debian already has the capability to do this now, but implementing the policy and tools won't be started until after woody releases.

Latest dpkg versions (since 1.9.21) incorporate a patch (<http://lists.debian.org/debian-dpkg/2001/debian-dpkg-200103/msg00024.html>) that provides this functionality as soon as `debsig-verify` is installed.

NOTE: Currently `/etc/dpkg/dpkg.cfg` ships with “no-debsig” as per default.

NOTE2: Signatures from developers are currently stripped when they enter off the package archive since the currently preferred method is release checks as described previously.

7.4.3 Checking distribution releases

In case you want to add now the additional security checks you can use the script below, provided by Anthony Thown. This script can automatically do some new security checks to allow the user to be sure that the software s/he’s downloading matches the software Debian’s distributing. This stops Debian developers from hacking into someone’s system without the accountability provided by uploading to the main archive, or mirrors mirroring something almost, but not quite like Debian, or mirrors providing out of date copies of unstable with known security problems.

This sample code, renamed as `apt-check-sigs`, should be used in the following way:

```
# apt-get update
# apt-check-sigs
(...results...)
# apt-get dist-upgrade
```

First you need to:

- get the keys the archive software uses to sign Release files, http://ftp-master.debian.org/ziyi_key_2002.asc and add them to `~/ .gnupg/trustedkeys.gpg` (which is what `gpgv` uses by default).

```
gpg --no-default-keyring --keyring trustedkeys.gpg --import ziyi_key_
```

- remove any `/etc/apt/sources.list` lines that don’t use the normal “dists” structure, or change the script so that it works with them.
- be prepared to ignore the fact that Debian security updates don’t have signed Release files, and that Sources files don’t have appropriate checksums in the Release file (yet).
- be prepared to check that the appropriate sources are signed by the appropriate keys.

This is the example code for `apt-check-sigs`, the latest version can be retrieved from <http://people.debian.org/~ajt/apt-check-sigs>. This code is currently in beta, for more information read <http://lists.debian.org/debian-devel/2002/debian-devel-200207/msg00421.html>.

```
#!/bin/bash
# This script is copyright (c) 2001, Anthony Towns
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.

rm -rf /tmp/apt-release-check
mkdir /tmp/apt-release-check || exit 1
cd /tmp/apt-release-check

>OK
>MISSING
>NOCHECK
>BAD

arch=`dpkg --print-installation-architecture`

am_root () {
    [ `id -u` -eq 0 ]
}

get_md5sumsize () {
    cat "$1" | awk '/^MD5Sum:\/,\/^SHA1:\/' |
        MYARG="$2" perl -ne '@f = split /\s+\/; if ($f[3] eq $ENV{"MYARG"})'
}

checkit () {
    local FILE="$1"
    local LOOKUP="$2"

    Y=`get_md5sumsize Release "$LOOKUP"``
    Y=`echo "$Y" | sed 's/^ *\/;\/s/ *\/ /g'``

    if [ ! -e "/var/lib/apt/lists/$FILE" ]; then
        if [ "$Y" = "" ]; then
            # No file, but not needed anyway
            echo "OK"
            return
        fi
        echo "$FILE" >>MISSING
    fi
}
```

```

        echo "MISSING $Y"
        return
    fi
    if [ "$Y" = "" ]; then
        echo "$FILE" >>NOCHECK
        echo "NOCHECK"
        return
    fi
    X=`md5sum < /var/lib/apt/lists/$FILE` `wc -c < /var/lib/apt/lists/$FILE`
    X=`echo "$X" | sed 's/^ *//;s/  */ /g'`
    if [ "$X" != "$Y" ]; then
        echo "$FILE" >>BAD
        echo "BAD"
        return
    fi
    echo "$FILE" >>OK
    echo "OK"
}

echo
echo "Checking sources in /etc/apt/sources.list:"
echo "~~~~~"
echo
(echo "You should take care to ensure that the distributions you're downloading
echo "are the ones you think you are downloading, and that they are as up to
echo "date as you would expect (testing and unstable should be no more than
echo "two or three days out of date, stable-updates no more than a few weeks
echo "or a month).") | fmt
echo

cat /etc/apt/sources.list |
    sed 's/^ *//' | grep '^#[^#]' |
    while read ty url dist comps; do
        if [ "${url%:*}" = "http" -o "${url%:*}" = "ftp" ]; then
            baseurl="${url#*://}"
        else
            continue
        fi
        echo "Source: ${ty} ${url} ${dist} ${comps}"

        rm -f Release Release.gpg
        wget -q -O Release "${url}/dists/${dist}/Release"

        if ! grep -q '^' Release; then
            echo " * NO TOP-LEVEL Release FILE"
        fi
    done

```

```

else
    origline=`sed -n 's/^Origin: */p' Release | head -1`
    lablline=`sed -n 's/^Label: */p' Release | head -1`
    suitline=`sed -n 's/^Suite: */p' Release | head -1`
    codeline=`sed -n 's/^Codename: */p' Release | head -1`
    dateline=`grep "^Date:" Release | head -1`
    dscline=`grep "^Description:" Release | head -1`
    echo "  o Origin: $origline/$lablline"
    echo "  o Suite: $suitline/$codeline"
    echo "  o $dateline"
    echo "  o $dscline"

    if [ "${dist%%/*}" != "$suitline" -a "${dist%%/*}" != "$codeline" ]
    then
        echo " * WARNING: asked for $dist, got $suitline/$codeline"
    fi

    wget -q -O Release.gpg "${url}/dists/${dist}/Release.gpg"
    sigline=`gpgv --status-fd 3 Release.gpg Release 3>&1 >/dev/null`
    if [ "$sigline" ]; then
        echo "  o Signed by: $sigline"
    else
        echo " * NO VALID SIGNATURE"
        >Release
    fi
fi

okaycomps=""
for comp in $comps; do
    if [ "$ty" = "deb" ]; then
        X=$(checkit "`echo "${baseurl}/dists/${dist}/${comp}/`
        Y=$(checkit "`echo "${baseurl}/dists/${dist}/${comp}/`
        if [ "$X $Y" = "OK OK" ]; then
            okaycomps="$okaycomps $comp"
        else
            echo " * PROBLEMS WITH $comp ($X, $Y)"
        fi
    elif [ "$ty" = "deb-src" ]; then
        X=$(checkit "`echo "${baseurl}/dists/${dist}/${comp}/`
        Y=$(checkit "`echo "${baseurl}/dists/${dist}/${comp}/`
        if [ "$X $Y" = "OK OK" ]; then
            okaycomps="$okaycomps $comp"
        else
            echo " * PROBLEMS WITH component $comp ($X, $Y)"
        fi
    fi
done
[ "$okaycomps" = "" ] || echo "  o Okay:$okaycomps"

```

```
        echo
    done

    echo "Results"
    echo "~~~~~"
    echo

    allokay=true

    cd /tmp/apt-release-check
    diff <(cat BAD MISSING NOCHECK OK | sort) <(cd /var/lib/apt/lists && find . -

    cd /tmp/apt-release-check
    if grep -q ^ UNVALIDATED; then
        allokay=false
        (echo "The following files in /var/lib/apt/lists have not been validated.
        echo "This could turn out to be a harmless indication that this script"
        echo "is buggy or out of date, or it could let trojaned packages get onto
        echo "your system."
        ) | fmt
        echo
        sed 's/^/    //' < UNVALIDATED
        echo
    fi

    if grep -q ^ BAD; then
        allokay=false
        (echo "The contents of the following files in /var/lib/apt/lists does not
        echo "match what was expected. This may mean these sources are out of dat
        echo "that the archive is having problems, or that someone is actively"
        echo "using your mirror to distribute trojans."
        if am_root; then
            echo "The files have been renamed to have the extension .FAILED and"
            echo "will be ignored by apt."
            cat BAD | while read a; do
                mv /var/lib/apt/lists/$a /var/lib/apt/lists/${a}.FAILED
            done
        fi) | fmt
        echo
        sed 's/^/    //' < BAD
        echo
    fi

    if grep -q ^ MISSING; then
        allokay=false
        (echo "The following files from /var/lib/apt/lists were missing. This"
```

```

        echo "may cause you to miss out on updates to some vulnerable packages."
    ) | fmt
    echo
    sed 's/^/    /' < MISSING
    echo
fi

if grep -q ^ NOCHECK; then
    allokay=false
    (echo "The contents of the following files in /var/lib/apt/lists could not
    echo "be validated due to the lack of a signed Release file, or the lack"
    echo "of an appropriate entry in a signed Release file. This probably"
    echo "means that the maintainers of these sources are slack, but may mean"
    echo "these sources are being actively used to distribute trojans."
    if am_root; then
        echo "The files have been renamed to have the extension .FAILED and"
        echo "will be ignored by apt."
        cat NOCHECK | while read a; do
            mv /var/lib/apt/lists/$a /var/lib/apt/lists/${a}.FAILED
        done
    fi) | fmt
    echo
    sed 's/^/    /' < NOCHECK
    echo
fi

if $allokay; then
    echo 'Everything seems okay!'
    echo
fi

rm -rf /tmp/apt-release-check

```

You might need to apply the following patch for *sid* since `md5sum` adds an `'-'` after the sum when the input is stdin:

```

@@ -37,7 +37,7 @@
     local LOOKUP="$2"

     Y="\`get_md5sumsize Release "$LOOKUP"``"
-    Y="\`echo "$Y" | sed 's/^ *//;s/ */ /g'``"
+    Y="\`echo "$Y" | sed 's/-//;s/^ *//;s/ */ /g'``"

     if [ ! -e "/var/lib/apt/lists/$FILE" ]; then
         if [ "$Y" = "" ]; then

```

```
@@ -55,7 +55,7 @@
        return
    fi
    X=`md5sum < /var/lib/apt/lists/$FILE` `wc -c < /var/lib/apt/lists/$F
-   X=`echo "$X" | sed 's/^ *//;s/ */ /g'`
+   X=`echo "$X" | sed 's/-//;s/^ *//;s/ */ /g'`
    if [ "$X" != "$Y" ]; then
        echo "$FILE" >>BAD
        echo "BAD"
```

Chapter 8

Security tools in Debian

FIXME: More content needed.

Debian provides also a number of security tools that can make a Debian box suited for security purposes. This purposes include protection of information systems through firewalls (either packet or application-level), intrusion detection (both network and host based), vulnerability assesment, antivirus, private networks, etc.

Since Debian 3.0 (*woody*), the distribution features cryptographic software integrated into the main distribution. OpenSSH and GNU Privacy Guard are included in the default install, and strong encryption is now present in web browsers and web servers, databases, and so forth. Further integration of cryptography is planned for future releases. This software, due to export restrictions in the US, was not distributed along with the main distribution but included only in non-US sites.

8.1 Remote vulnerability assesment tools

The tools provided by Debian to perform remote vulnerability assesment are: ¹

- nessus
- raccess
- whisker
- nikto (whisker's replacement)
- bass (non-free)
- satan (non-free)

¹Some of them are provided when installing the `hardened-remoteaudit` package.

By far, the most complete and up-to-date tools is `nessus` which is composed of a client (`nessus`) used as a GUI and a server (`nessusd`) which launches the programmed attacks. Nessus includes remote vulnerabilities for quite a number of systems including network appliances, ftp servers, www servers, etc. The latest releases are able even to parse a web site and try to discover which interactive pages are available which could be attacked. There are also Java and Win32 clients (not included in Debian) which can be used to contact the management server.

`whisker` is a web-only vulnerability assessment scanner including anti-IDS tactics (most of which are not *anti-IDS* anymore). It is one of the best cgi-scanners available, being able to detect WWW servers and launch only a given set of attacks against it. The database used for scanning can be easily modified to provide for new information.

`Bass` (Bulk Auditing Security Scanner) and `Satan` (Security Auditing Tool for Analysing Networks) must be thought of more like “proof of concept” programs than as tools to be used while performing audits. Both are quite ancient and are not kept up-to-date. However, `SATAN` was the first tool to provide vulnerability assesment in a simple (GUI) way and `Bass` is still a very high-performance assesment tool.

8.2 Network scanner tools

Debian does provide some tools used for remote scanning of hosts (but not vulnerability assessment). These tools are, in some cases, used by vulnerability assesment scanners as the first type of “attack” run against remote hosts in an attempt to determine remote services available. Currently Debian provides:

- `nmap`
- `xprobe`
- `queso`
- `knocker`
- `hping2`
- `isic`
- `icmpush`
- `nbtscan`
- `fragrouter`

Whileas `queso` and `xprobe` provide only remote operating system detection (using TCP/IP fingerprinting), `nmap` and `knocker` do both operating system detection and port scanning of the remote hosts. On the other hand, `hping2` and `icmpush` can be used for remote ICMP attack techniques.

Designed specifically for Netbios networks, `nbtscan` can be used to scan IP networks and retrieve name information from SMB-enabled servers, including: usernames, network names, MAC addresses...

On the other hand, `fragrouter` can be used to test network intrusion detection systems and see if the NIDS can be eluded by fragmentation attacks.

FIXME: Check Bug #153117 (<http://bugs.debian.org/153117>) (ITP `fragrouter`) to see if it's included.

FIXME add information based on Debian Linux Laptop for Road Warriors (http://rr.sans.org/linux/debian_laptop.php) which describes how to use Debian and a laptop to scan for wireless (803.1) networks.

8.3 Internal audits

Currently, only the `tiger` tool used in Debian can be used to perform internal (also called white box) audit of hosts in order to determine if the file system is properly set up, which processes are listening on the host, etc.

8.4 Auditing source code

Debian provides two packages that can be used to audit C/C++ source code programs and find programming errors that might lead to potential security flaws:

- `flawfinder`
- `rats`

8.5 Virtual Private Networks

A virtual private network (VPN) is a group of two or more computer systems, typically connected to a private network with limited public network access, that communicate securely over a public network. VPNs may connect a single computer to a private network (client-server), or a remote LAN to a private network (server-server). VPNs often include the use of encryption, strong authentication of remote users or hosts, and methods for hiding the private network's topology.

Debian provides quite a few packages to set up encrypted virtual private networks:

- `vtun`
- `tunnelv`

- cipe
- vpnd
- tinc
- secvpn
- pptpd
- freeswan

The FreeSWAN package is probably the best choice overall, since it promises to interoperate with almost anything that uses the IP security protocol, IPsec (RFC 2411). However, the other packages listed above can also help you get a secure tunnel up in a hurry. The point to point tunneling protocol (PPTP) is a proprietary Microsoft protocol for VPN. It is supported under Linux, but is known to have serious security issues.

For more information see the VPN-Masquerade HOWTO (<http://www.linuxdoc.org/HOWTO/VPN-Masquerade-HOWTO.html>) (covers IPsec and PPTP), VPN HOWTO (<http://www.linuxdoc.org/HOWTO/VPN-HOWTO.html>) (covers PPP over SSH), and Cipe mini-HOWTO (<http://www.linuxdoc.org/HOWTO/mini/Cipe+Masq.html>), and PPP and SSH mini-HOWTO (<http://www.linuxdoc.org/HOWTO/mini/ppp-ssh/index.html>).

8.5.1 Point to Point tunneling

If you want to provide a tunneling server for a mixed environment (both Microsoft operating systems and Linux clients) and IPsec is not an option (since it's only provided for Windows 2000 and Windows XP), you can use *PoPToP* (Point to Point Tunneling Server), provided in the pptpd package.

If you want to use Microsoft's authentication and encryption with the server provided in the ppp package, note the following from the FAQ:

```
It is only necessary to use PPP 2.3.8 if you want Microsoft compatible
MSCHAPv2/MPPE authentication and encryption. The reason for this is that
the MSCHAPv2/MPPE patch currently supplied (19990813) is against PPP
2.3.8. If you don't need Microsoft compatible authentication/encryption
any 2.3.x PPP source will be fine.
```

However, you also have to apply the kernel patch provided by the kernel-patch-mppe package, which provides the pp_mppe module for pppd.

Take into account that the encryption in pptpd forces you to store user passwords in clear text, and that the MS-CHAPv2 protocol contains known security holes (http://mopo.informatik.uni-freiburg.de/pptp_mschapv2/).

8.6 Public Key Infrastructure (PKI)

Public Key Infrastructure (PKI) is a security architecture introduced to provide an increased level of confidence for exchanging information over insecure networks. It makes use of the concept of public and private cryptographic keys to verify the identity of the sender (signing) and to ensure privacy (encryption).

When considering a PKI, you are confronted with a wide variety of issues:

- a Certificate Authority (CA) that can issue and verify certificates, and that can work under a given hierarchy
- a Directory to hold user's public certificates
- a Database (?) to maintain Certificate Revocation Lists (CRL)
- devices that interoperate with the CA in order to print out smart cards/USB tokens/whatever to securely store certificates
- certificate-aware applications that can use certificates issued by a CA to enroll in encrypted communication and check given certificates against CRL (for authentication and full Single Sign On solutions)
- a Time stamping authority to digitally sign documents
- a management console from which all of this can be properly used (certificate generation, revocation list control, etc...)

Debian GNU/Linux has software packages to help you with some of these PKI issues. They include `OpenSSL` (for certificate generation), `OpenLDAP` (as a directory to hold the certificates), `gnupg` and `freeswan` (with X.509 standard support). However, as of the Woody release (Debian 3.0), Debian does not have any of the freely available Certificate Authorities such as `pyCA`, `OpenCA` (<http://www.openca.org>) or the CA samples from `OpenSSL`. For more information read the Open PKI book (<http://ospkibook.sourceforge.net/>).

8.7 SSL Infrastructure

Debian does provide some SSL certificates with the distribution so that they can be installed locally. They are found in the `ca-certificates` package. This package provides a central repository of certificates that have been submitted to Debian and approved (that is, verified) by the package maintainer, useful for any `OpenSSL` applications which verify SSL connections.

FIXME: read `debian-devel` to see if there was something added to this.

8.8 Anti-virus tools

There are not many anti-virus tools included with Debian GNU/Linux, probably because GNU/Linux users are not plagued by viruses. The UN*X security model makes a distinction between privileged (root) processes and user-owned processes, therefore a “hostile” executable that a non-root user receives or creates and then executes cannot “infect” or otherwise manipulate the whole system. However, GNU/Linux worms and viruses do exist, although there has not (yet, hopefully) been any that has spread in the wild over any Debian distribution. In any case, administrators might want to build up anti-virus gateways that protect against viruses arising on other, more vulnerable systems in their network.

Debian GNU/Linux currently provides the following tools for building anti-virus environments:

- sanitizer (<http://packages.debian.org/sanitizer>), a tool that uses the `procmail` package, which can scan email attachments for viruses, block attachments based on their filenames, and more.
- amavis-postfix (<http://packages.debian.org/amavis-postfix>), a script that provides an interface from a mail transport agent to one or more commercial virus scanners (this package is built with support for the `postfix` MTA only).
- scannerdaemon, a daemon written in Java that accepts incoming requests to scan files for viruses.

As you can see, Debian does not currently provide any anti-virus software in the main distribution. There are, however, free software anti-virus projects which might be included in future Debian GNU/Linux releases:

- Open Antivirus (<http://sourceforge.net/projects/openantivirus/>) (see Bug #150698 (ITP `oav-scannerdaemon` (<http://bugs.debian.org/150698>) and Bug #150695 (ITP `oav-update` (<http://bugs.debian.org/150695>)).
- Clam Antivirus (<http://clamav.elektrapro.com/>)
- jvirus (<http://sourceforge.net/projects/jvirus/>).
- Amavis Next Generation (<http://www.sourceforge.net/projects/amavis>), a mail virus scanner which integrates with your MTA and supports multiple virus scanning engines (see Bug #154294 (<http://bugs.debian.org/154294>)).

There is also a `virussignatures` package, which provides signatures for all packages, this package provides a script to download the latest virus signatures from <http://www.openantivirus.org/latest.php>.

FIXME: Check to determine which packages are available for antivirus. Is `clamav` available? (there seem to be Debian packages for it).

FIXME: check if scannerdaemon is the same as the open antivirus scanner daemon (read ITPs).

However, Debian will *never* provide commercial anti-virus software such as: Panda Anti-virus (<http://www.pandasoftware.com/com/linux/linux.asp>), NAI Netshield (uvs-can) (<http://www.nai.com/naicommon/buy-try/try/products-evals.asp>), Sophos Sweep (<http://www.sophos.com/>), TrendMicro Interscan (<http://www.antivirus.com/products/>), or RAV (<http://www.ravantivirus.com>). For more pointers see the Linux anti-virus software mini-FAQ (http://www.computer-networking.de/~link/security/av-linux_e.txt). This does not mean that this software can be installed properly in a Debian system.

For more information on how to set up an a virus detection system read Dave Jones' article Building an E-mail Virus Detection System for Your Network (<http://www.linuxjournal.com/article.php?sid=4882>).

8.9 GPG agent

It is very common nowadays to digitally sign (and sometimes encrypt) e-mail. You might, for example, find that many people participating on mailing lists sign their list e-mail. Public key signatures are currently the only means to verify that an e-mail was sent by the sender and not by some other person.

Debian GNU/Linux provides a number of e-mail clients with built-in e-mail signing capabilities that interoperate either with `gnupg` or `pgp`:

- Evolution.
- mutt.
- kmail.
- sylpheed. Depending on how the stable version of this package evolves, you may need to use the *bleeding edge version*, `sylpheed-claws`.
- gnus, which when installed with the `mailcrypt` package, is an emacs interface to `gnupg`.
- kuvert, which provides this functionality independently of your chosen mail user agent (MUA) by interacting with the mail transport agent (MTA).

Key servers allow you to download published public keys so that you may verify signatures. One such key server is <http://wwwkeys.pgp.net>. `gnupg` can automatically fetch public keys that are not already in your public keyring. For example, to configure `gnupg` to use the above key server, edit the file `~/.gnupg/options` and add the following line: ²

```
keyserver wwwkeys.pgp.net
```

²For more examples of how to configure `gnupg` check `/usr/share/doc/mutt/examples/gpg.rc`.

Most key servers are linked, so that when your public key is added to one server, the addition is propagated to all the other public key servers. There is also a Debian GNU/Linux package `debian-keyring`, that provides all the public keys of the Debian developers. The `gnupg` keyrings are installed in `/usr/share/keyrings/`.

For more information:

- GnuPG FAQ (<http://www.gnupg.org/faq.html>).
- GnuPG Handbook (<http://www.gnupg.org/gph/en/manual.html>).
- GnuPG Mini Howto (English) (http://www.dewinter.com/gnupg_howto/english/GPGMiniHowto.html).
- comp.security.pgp FAQ (<http://www.uk.pgp.net/pgpnet/pgp-faq/>).
- Keysigning Party HOWTO (<http://www.cryptnet.net/fdp/crypto/gpg-party.html>).

Chapter 9

Before the compromise

9.1 Continuously update the system

You should conduct security updates frequently. The vast majority of exploits result from known vulnerabilities that have not been patched in time, as this <http://www.cs.umd.edu/~waa/vulnerability.html> name="paper by Bill Arbaugh"> (presented at the 2001 IEEE Symposium on Security and Privacy) explains. Updates are described under 'Execute a security update' on page 38.

9.1.1 Using Tiger to check for security updates

If you're looking for a tool to quickly check and report system security vulnerabilities, try the `tiger` package. This package is a set of Bourne shell scripts, C programs and data files used to perform security audits. The Debian GNU/Linux package has additional enhancements oriented toward the Debian distribution, providing more functionality than the Tiger scripts provided by TAMU (or even TARA, a tiger version distributed by ARSC). See the README.Debian file and the man page `tiger(8)` for more information.

One of these enhancements is the `deb_checkadvisories` script. This script takes a list of DSA's and checks against the installed package base, reporting back any packages that are vulnerable according to the Debian Security Team. This is a slightly different, more general approach than is implemented by the Tiger `check_signatures` script, which checks MD5sums of known vulnerable programs.

Since Debian currently does not ship a list of MD5sums of known vulnerable programs (utilized by some other operating systems like Sun Solaris), the *check-against-DSA* approach is used. The DSA approach and the MD5sums approach both suffer from the problem that signatures have to be updated regularly.

This is currently solved by making new versions of the Tiger package, but the package maintainer might not make a new version every time a DSA is announced. A nice addition, which is not yet implemented, might be to do this proactively. That is, download the DSAs from the

web, make the list and then run the check. The DSAs are currently updated from the maintainer's local CVS update of the WML sources used to build <http://security.debian.org> (the web server, that is).

A program to parse published DSAs, either received through e-mail or available in security.debian.org, and then generate the file used by 'deb_checkadvisories' to confirm vulnerabilities would be appreciated. Send it as a bug report for `tiger`.

The mentioned check is run through the standard program configuration once installed (see `/etc/tiger/cronrc`):

```
# Check for Debian security measures every day at 1 AM
#
1 * *    deb_checkmd5sums deb_nopackfiles deb_checkadvisories
#
```

There is an additional check that you might want to add, which is not yet part of the standard cron scripts. That check is the script `check_patches`, which works in the following way:

- run `# apt-get update`
- checks if there are new packages available

If you are running a *stable* system and add the security.debian.org apt source line to your `/etc/apt/sources.list` (as described in 'Execute a security update' on page 38), this script will be able to tell you if there are new packages that you need to install. Since the only packages changing in this setup are security updates, then you have just what you wanted.

Of course, this will not work if you are running *testing* or *sid/unstable*, since currently, the new packages are probably much more than security updates.

You can add this script to the checks done by the cron job (in the above configuration file) and `tigercron` would mail (to whomever `Tiger-Mail_RCPT` was set to in `/etc/tiger/tigerrc`) the new packages:

```
# Check for Debian security measures every day at 1 am
#
1 * *    deb_checkmd5sums deb_nopackfiles check_patches
#
```

9.1.2 Avoid using the unstable branch

Unless you want to dedicate time to patch packages yourself when a vulnerability arises, you should *not* use Debian's unstable branch for production-level systems. The main reason for this is that there are no security updates for *unstable* (see 'How is security handled for *testing* and *unstable*?' on page 148).

The fact is that some security issues might appear in *unstable* and *not* in the *stable* distribution. This is due to new functionality constantly being added to the applications provided there, as well as new applications being included which might not yet have been thoroughly tested.

In order to do security upgrades in the *unstable* branch, you might have to do full upgrades to new versions (which might update much more than just the affected package). Although there have been some exceptions, security patches are usually only back ported into the *stable* branch. The main idea being that between updates, *no new code* should be added, just fixes for important issues.

9.1.3 Avoid using the testing branch

If you are using the *testing* branch, there are some issues that you must take into account regarding the availability of security updates:

- When a security fix is prepared, packages are prepared for *unstable* and the patch is back ported to *stable* (since *stable* is usually some minor or major versions behind). Packages for the *stable* distribution are more thoroughly tested than *unstable*, since the latter might just provide the latest upstream release.
- Security updates are available immediately for both branches (but not yet for the *testing* branch).
- If no (new) bugs are detected in the *unstable* version of the package, it moves to *testing* after several days (usually over a week). However, this does depend on the release state of the distribution.

9.1.4 Automatic updates in a Debian GNU/Linux system

First of all, automatic updates are not fully recommended, since administrators should review the DSAs and understand the impact of any given security update.

If you want to update your system automatically you should:

- Configure `apt` so that those packages that you do not want to update stay at their current version, either with `apt`'s *pinning* feature or marking them as *hold* with `dpkg` or `dselect`.

To pin the packages under a given release, you must edit `/etc/apt/preferences` (see `apt_preferences(5)`) and add:

```
Package: *
Pin: release a=stable
Pin-Priority: 100
```

FIXME: verify if this configuration is OK.

- Add a `cron` entry so that the update is run daily, for example:

```
apt-get update && apt-get -y upgrade
```

The `-y` option will have `apt` assume 'yes' for all the prompts that might arise during the update. In some cases, you might want to use the `--trivial-only` option instead of the `--assume-yes` (equivalent to `-y`).¹

- Configure `cron` so that `debconf` will not ask for any input during upgrades, that way they are done non-interactively.²
- Check the results of the `cron` execution, which will be mailed to the superuser (unless changed with `MAILTO` environment variable in the script).

A safer alternative might be to use the `-d` (or `--download-only`) option, which will download but not install the necessary packages. Then if the `cron` execution shows that the system needs to be updated, it can be done manually.

In order to accomplish any of these tasks, the system must be properly configured to download security updates as discussed in 'Execute a security update' on page 38.

However, this is not recommended for *unstable* without careful analysis, since you might bring your system into an unusable state if some serious bug creeps into an important package and gets installed in your system. *Testing* is slightly more *secure* with regard to this issue, since serious bugs have a better chance of being detected before the package is moved into the testing branch (although, you may have *no* security updates available whatsoever).

If you have a mixed distribution, that is, a *stable* installation with some packages updated to *testing* or *unstable*, you can fiddle with the pinning preferences as well as the `--target-release` option in `apt-get` to update *only* those packages that you have updated.³

9.2 Set up Intrusion Detection

Debian GNU/Linux includes tools for intrusion detection, which is the practice of detecting inappropriate or malicious activity on your local system, or other systems in your private network. This kind of defense is important if the system is very critical or you are truly paranoid. The most common approaches to intrusion detection are statistical anomaly detection and pattern-matching detection.

¹You may also want to use the `--quiet` (`-q`) option to reduce the output of `apt-get`, which will stop the generation of any output if no packages are installed.

²Note that some packages might *not* use `debconf` and updates will stall due to packages asking for user input during configuration.

³This is a common issue since many users want to maintain a stable system while updating some packages to *unstable* to gain the latest functionality. This need arises due to some projects evolving faster than the time between Debian's *stable* releases.

Always be aware that in order to really improve the system's security with the introduction of any of these tools, you need to have an alert+response mechanism in place. Intrusion detection is a waste of time if you are not going to alert anyone.

When a particular attack has been detected, most intrusion detection tools will either log the event with `syslogd` or send e-mail to the root user (the mail recipient is usually configurable). An administrator has to properly configure the tools so that false positives do not trigger alerts. Alerts may also indicate an ongoing attack and might not be useful, say, one day later, since the attack might have already succeeded. So be sure that there is a proper policy on handling alerts and that the technical mechanisms to implement this policy are in place.

An interesting source of information is CERT's Intrusion Detection Checklist (http://www.cert.org/tech_tips/intruder_detection_checklist.html)

9.2.1 Network based intrusion detection

Network based intrusion detection tools monitor the traffic on a network segment and use this information as a data source. Specifically, the packets on the network are examined, and they are checked to see if they match a certain signature.

`Snort` is a flexible packet sniffer or logger that detects attacks using an attack signature dictionary. It detects a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, and much more. `Snort` also has real-time alerting capability. You can use `snort` for a range of hosts on your network as well as for your own host. This is a tool which should be installed on every router to keep an eye on your network. Just install it with `apt-get install snort`, follow the questions, and watch it log.

Debian's `snort` package has many security checks enabled by default. However, you should customize the setup to take into account the particular services you run on your system. You may also want to seek additional checks specific to these services.

There are other, simpler tools that can be used to detect network attacks. `portsentry` is an interesting package that can tip you off to port scans against your hosts. Other tools like `ipp1` or `iplogger` will also detect some IP (TCP and ICMP) attacks, even if they do not provide the kind of advanced techniques `snort` does.

You can test any of these tools with the Debian package `idswakeup`, a shell script which generates false alarms, and includes many common attack signatures.

9.2.2 Host based intrusion detection

Host based intrusion detection involves loading software on the system to be monitored which uses log files and/or the systems auditing programs as a data source. It looks for suspicious processes, monitors host access, and may even monitor changes to critical system files.

`Tiger` is an older intrusion detection tool which has been ported to Debian since the Woody branch. `Tiger` provides checks of common issues related to security break-ins, like password

strength, file system problems, communicating processes, and other ways root might be compromised. This package includes new Debian-specific security checks including: MD5sums checks of installed files, locations of files not belonging to packages, and analysis of local listening processes. The default installation sets up `tiger` to run each day, generating a report that is sent to the superuser about possible compromises of the system.

Log analysis tools, such as `logcheck` can also be used to detect intrusion attempts. See ‘Using and customising `logcheck`’ on page 51.

In addition, packages which monitor file system integrity (see ‘Checking file system integrity’ on page 58) can be quite useful in detecting anomalies in a secured environment. It is most likely that an effective intrusion will modify some files in the local file system in order to circumvent local security policy, install Trojans, or create users. Such events can be detected with file system integrity checkers.

9.3 Useful kernel patches

FIXME: This section needs to cover how these specific patches can be installed in Debian using the `kernel-2.x.x-patch-XXX` packages.

The following are some kernel patches which can significantly enhance system security:

- *OpenWall patch, by Solar Designer.* This is a useful set of kernel restrictions, like restricted links, FIFOs in `/tmp`, a restricted `/proc` file system, special file descriptor handling, non-executable user stack area and other. Homepage: <http://www.openwall.com/linux/>
- *LIDS — Linux intrusion detection system by Huagang Xie & Philippe Biondi.* This kernel patch makes the process of hardening your Linux system easier by allowing you to restrict, hide and protect processes, even from root. It also allows you to protect or hide certain files so that even root cannot modify them. Furthermore, you can also set capabilities for certain processes. A must for the paranoid system administrator. Homepage: <http://www.lids.org>
- *POSIX Access Control Lists (ACLs) for Linux* This kernel patch adds access control lists, an advanced method for restricting access to files. Homepage: <http://acl.bestbits.at/>
- *Linux trustees.* This patch adds a decent advanced permissions management system to your Linux kernel. Special objects (called trustees) are bound to every file or directory, and are stored in kernel memory, which allows fast lookup of all permissions. Homepage: <http://trustees.sourceforge.net/>
- *International kernel patch.* This is a crypt-oriented kernel patch, therefore you have to pay attention to your local laws regarding the use of cryptography. It basically adds the possibility of using encrypted file systems. Homepage: <http://www.kerneli.org>

- *SubDomain*. A kernel extension designed to provide least privilege confinement to possibly insecure programs. SubDomain complements and extends native access controls. While similar to a chroot environment, it claims to be easier to construct and more flexible than a chroot environment. Homepage: <http://www.immunix.org/subdomain.html>
- *UserIPAcct*. This is not really a security related patch, but it does allow you to create per user quotas for the traffic on your system. You can also fetch statistics about the user traffic. Homepage: <http://ramses.smeyers.be/useripacct>.
- *FreeSWAN*. If you want to use the IPsec protocol with Linux, you need this patch. You can create VPNs with this quite easily, even to Windows machines, as IPsec is a common standard. Homepage: <http://www.freeswan.org>

9.4 Avoiding root-kits

9.4.1 Loadable Kernel Modules (LKM)

Loadable kernel modules are files containing dynamically loadable kernel components used to expand the functionality of the kernel. The main benefit of using modules is the ability to add additional devices, like an Ethernet or sound card, without patching the kernel source and recompiling the entire kernel. However, crackers are now using LKMs for root-kits (knark and adore), opening up back doors in GNU/Linux systems.

LKM back doors are more sophisticated and less detectable than traditional root-kits. They can hide processes, files, directories and even connections without modifying the source code of binaries. For example, a malicious LKM can force the kernel into hiding specific processes from `ps`, so that even a known good copy of the binary `ps` would not list accurate information about the current processes on the system.

9.4.2 Detecting root-kits

There are two approaches to defending your system against LKM root-kits, a proactive defense and a reactive defense. The detection work can be simple and painless, or difficult and tiring, depending on the approach taken.

Proactive defense

The advantage of this kind of defense is that it prevents damage to the system in the first place. One such strategy is *getting there first*, that is, loading a LKM designed to protect the system from other malicious LKMs. A second strategy is to remove capabilities from the kernel itself. For example, you can remove the capability of loadable kernel modules entirely.

Debian GNU/Linux has a few packages that can be used to mount a proactive defense:

- `kernel-patch-2.4-lsm` - LSM is the Linux Security Modules framework.
- `lcap` - A user friendly interface to remove *capabilities* (kernel-based access control) in the kernel, making the system more secure. For example, executing `lcap CAP_SYS_MODULE`⁴ will remove module loading capabilities (even for the root user).⁵

If you don't really need many kernel features on your GNU/Linux system, you may want to disable loadable modules support during kernel configuration. To disable loadable module support, just set `CONFIG_MODULES=n` during the configuration stage of building your kernel, or in the `.config` file. This will prevent LKM root-kits, but you lose this powerful feature of the Linux kernel. Also, disabling loadable modules can sometimes overload the kernel, making loadable support necessary.

Reactive defense

The advantage of a reactive defense is that it does not overload system resources. It works by comparing the system call table with a known clean copy in a disk file, `System.map`. Of course, a reactive defense will only notify the system administrator after the system has already been compromised.

Detection of some root-kits in Debian can be accomplished with the `chkrootkit` package. The `Chkrootkit` (<http://www.chkrootkit.org>) program checks for signs of several known root-kits on the target system, but is not a definitive test.

Another helpful tool is `KSTAT` (<http://www.s0ftpj.org/en/site.html>) (Kernel Security Therapy Anti Trolls) by the `S0ftpj` group. `KSTAT` checks the kernel memory area `/dev/kmem` for information about the target host to assist the system administrator in finding and removing malicious LKMs.

9.5 Genius/Paranoia Ideas — what you could do

This is probably the most unstable and funny section, since I hope that some of the “duh, that sounds crazy” ideas might be realized. The following are just some ideas for increasing security — maybe genius, paranoid, crazy or inspired depending on your point of view.

- Playing around with Pluggable Authentication Modules (PAM). As quoted in the Phrack 56 PAM article, the nice thing about PAM is that “You are limited only by what you can

⁴There are over 28 capabilities including: `CAP_BSET`, `CAP_CHOWN`, `CAP_FOWNER`, `CAP_FSETID`, `CAP_FS_MASK`, `CAP_FULL_SET`, `CAP_INIT_EFF_SET`, `CAP_INIT_INH_SET`, `CAP_IPC_LOCK`, `CAP_IPC_OWNER`, `CAP_KILL`, `CAP_LEASE`, `CAP_LINUX_IMMUTABLE`, `CAP_MKNOD`, `CAP_NET_ADMIN`, `CAP_NET_BIND_SERVICE`, `CAP_NET_RAW`, `CAP_SETGID`, `CAP_SETPCAP`, `CAP_SETUID`, `CAP_SYS_ADMIN`, `CAP_SYS_BOOT`, `CAP_SYS_CHROOT`, `CAP_SYS_MODULE`, `CAP_SYS_NICE`, `CAP_SYS_PACCT`, `CAP_SYS_PTRACE`, `CAP_SYS_RAWIO`, `CAP_SYS_RESOURCE`, `CAP_SYS_TIME`, and `CAP_SYS_TTY_CONFIG`. All of them can be activated or de-activated to harden your kernel.

⁵You don't need to install `lcap` to do this, but it's easier than setting `/proc/sys/kernel/cap-bound` by hand.

think of.” It is true. Imagine root login only being possible with fingerprint or eye scan or cryptocard (why did I use an OR conjunction instead of AND?).

- Fascist Logging. I would refer to all the previous logging discussion above as “soft logging”. If you want to perform real logging, get a printer with fanfold paper, and send all logs to it. Sounds funny, but it’s reliable and it cannot be tampered with or removed.
- CD distribution. This idea is very easy to realize and offers pretty good security. Create a hardened Debian distribution, with proper firewall rules. Turn it into a boot-able ISO image, and burn it on a CDROM. Now you have a read-only distribution, with about 600 MB space for services. Just make sure all data that should get written is done over the network. It is impossible for intruders to get read/write access on this system, and any changes an intruder does make can be disabled with a reboot of the system.
- Switch module capability off. As discussed earlier, when you disable the usage of kernel modules at kernel compile time, many kernel based back doors are impossible to implement because most are based on installing modified kernel modules.
- Logging through serial cable. (contributed by Gaby Schilders) As long as servers still have serial ports, imagine having one dedicated logging system for a number of servers. The logging system is disconnected from the network, and connected to the servers via a serial-port multiplexer (Cyclades or the like). Now have all your servers log to their serial ports, write only. The log-machine only accepts plain text as input on its serial ports and only writes to a log file. Connect a CD/DVD-writer, and transfer the log file to it when the log file reaches the capacity of the media. Now if only they would make CD writers with auto-changers... Not as hard copy as direct logging to a printer, but this method can handle larger volumes and CDROMs use less storage space.
- Change file attributes using `chattr`. (taken from the Tips-HOWTO, written by Jim Dennis). After a clean install and initial configuration, use the `chattr` program with the `+i` attribute to make files unmodifiable (the file cannot be deleted, renamed, linked or written to). Consider setting this attribute on all the files in `/bin`, `/sbin`, `/usr/bin`, `/usr/sbin`, `/usr/lib` and the kernel files in root. You can also make a copy of all files in `/etc`, using `tar` or the like, and mark the archive as immutable.

This strategy will help limit the damage that you can do when logged in as root. You won’t overwrite files with a stray redirection operator, and you won’t make the system unusable with a stray space in a `rm -fr` command (you might still do plenty of damage to your data — but your libraries and binaries will be safer.)

This strategy also makes a variety of security and denial of service (DoS) exploits either impossible or more difficult (since many of them rely on overwriting a file through the actions of some SETUID program that *isn’t providing an arbitrary shell command*).

One inconvenience of this strategy arises during building and installing various system binaries. On the other hand, it prevents the `make install` from over-writing the files. When you forget to read the Makefile and `chattr -i` the files that are to be overwritten, (and the directories to which you want to add files) - the make command fails, and you just use the `chattr` command and rerun it. You can also take that opportunity to move your old bin’s and libs out of the way, into a `.old/` directory or tar archive for example.

Note that this strategy also prevents you from upgrading your system's packages, since the files updated packages provide cannot be overwritten. You might want to have a script or other mechanism to disable the immutable flag on all binaries right before doing an `apt-get update`.

9.5.1 Building a honeypot

FIXME: More Content specific to Debian needed.

A honeypot is a system designed to teach system administrators how crackers probe for and exploit a system. It is a system setup with the expectation and goal that the system will be probed, attacked and potentially exploited. By learning the tools and methods employed by the cracker, a system administrator can learn to better protect their own systems and network.

A Debian GNU/Linux system can easily be setup as a honeypot, if you dedicate the time to implement and monitor it. Simply setup the fake server with a firewall and some sort of network intrusion detector, put it on the Internet, and wait. Do take care that if the system is exploited, you are alerted in time (see 'The importance of logs and alerts' on page 50) so that you can take appropriate measures and terminate the compromise when you've seen enough. Here are some of the packages and issues to consider when setting up your honeypot:

- The firewall technology you will use (provided by the Linux kernel).
- `syslog-ng`, useful for sending logs from the honeypot to a remote syslog server.
- `snort`, to set up capture of all the incoming network traffic to the honeypot and detect the attacks.
- `osh`, a SETUID root, security enhanced, restricted shell with logging (see Lance Spitzner's article below).
- Of course, all the daemons you will be using for your fake server honeypot (but do *not* harden the honeypot).
- The Deception Toolkit, which uses deception to counter attacks. Homepage: Deception Toolkit (<http://www.all.net/dtk/>)
- Integrity checkers (see 'Checking file system integrity' on page 58) and The Coroner's Toolkit (`tct`) to do post-attack audits.

You can read more about building honeypots in Lanze Spitzner's excellent article To Build a Honeypot (<http://www.net-security.org/text/articles/spitzner/honeypot.shtml>) (from the Know your Enemy series), or David Raikow's Building your own honeypot (<http://www.zdnetindia.com/techzone/resources/security/stories/7601.htm>). Also, the Honeynet Project (<http://project.honeynet.org/>) provides valuable information about building honeypots and auditing the attacks made on them.

Chapter 10

After the compromise

10.1 General behavior

If you are physically present when an attack is happening, your first response should be to remove the machine from the network by unplugging the network card (if this will not adversely affect any business transactions). Disabling the network at layer 1 is the only true way to keep the attacker out of the compromised box (Phillip Hofmeister's wise advice).

However, some rootkits or back doors are able to detect this event and react to it. Seeing a `rm -rf /` executed when you unplug the network from the system is not really much fun. If you are unwilling to take the risk, and you are sure that the system is compromised, you should *unplug the power cable* (all of them if more than one) and cross your fingers. This may be extreme but, in fact, will avoid any logic-bomb that the intruder might have programmed. In this case, the compromised system *should not be re-booted*. Either the hard disks should be moved to another system for analysis, or you should use other media (a CD-ROM) to boot the system and analyze it. You should *not* use Debian's rescue disks to boot the system, but you *can* use the shell provided by the installation disks (remember, Alt+F2 will take you to it) to analyze the system. ¹

The most recommended method for recovering a compromised system is to use a live-filesystem on CDROM with all the tools (and kernel modules) you might need to access the compromised system. You can use the `mkinitrd-cd` package to build such a CDROM². You might find the Biatchux (<http://biatchux.dmzs.com/>) CDROM useful here too, since it's also a live CDROM with forensic tools useful in these situations. There is not (yet) a Debian-based tool such as this, nor an easy way to build the CDROM using your own selection of Debian packages and `mkinitrd-cd` (so you'll have to read the documentation provided with it to make your own CDROMs).

¹If you are adventurous, you can login to the system and save information on all running processes (you'll get a lot from `/proc/nnn/`). It is possible to get the whole executable code from memory, even if the attacker has deleted the executable files from disk. Then pull the power cord.

²In fact, this is the tool used to build the CDROMs for the Gibraltar (<http://www.gibraltar.at/>) project (a firewall on a live CDROM based on the Debian distribution).

If you really want to fix the compromise quickly, you should remove the compromised host from your network and re-install the operating system from scratch. Of course, this may not be effective because you will not learn how the intruder got root in the first place. For that case, you must check everything: firewall, file integrity, log host, log files and so on. For more information on what to do following a break-in, see Sans' Incident Handling Guide (<http://www.sans.org/y2k/DDoS.htm>) or CERT's Steps for Recovering from a UNIX or NT System Compromise (http://www.cert.org/tech_tips/root_compromise.html).

Some common questions on how to handle a compromised Debian GNU/Linux system are also available in 'My system is vulnerable! (Are you sure?)' on page 143.

10.2 Backing up the system

Remember that if you are sure the system has been compromised you cannot trust the installed software or any information that it gives back to you. Applications might have been Trojanized, kernel modules might be installed, etc.

The best thing to do is a complete file system backup copy (using `dd`) after booting from a safe medium. Debian GNU/Linux CDROMs can be handy for this since they provide a shell in console 2 when the installation is started (jump to it using `Alt+2` and pressing `Enter`). From this shell, backup the information to another host if possible (maybe a network file server through NFS/FTP). Then any analysis of the compromise or re-installation can be performed while the affected system is offline.

If you are sure that the only compromise is a Trojan kernel module, you can try to run the kernel image from the Debian CDROM in *rescue* mode. Make sure to startup in *single user* mode, so no other Trojan processes run after the kernel.

10.3 Forensic analysis

If you wish to gather more information, the `tct` (The Coroner's Toolkit from Dan Farmer and Wietse Venema) package contains utilities which perform a 'post mortem' of a system. `tct` allows the user to collect information about deleted files, running processes and more. See the included documentation for more information.

Some other tools that can be used for forensic analysis provided in the Debian distribution are:

- `Fenris`.
- `Strace`.
- `Ltrace`.

Any of these packages can be used to analyze rogue binaries (such as back doors), in order to determine how they work and what they do to the system. Some other common tools include `ldd` (in `libc6`), `strings` and `objdump` (both in `binutils`).

If you try to do forensic analysis with back doors or suspected binaries retrieved from compromised systems, you should do so in a secure environment (for example in a bochs or flex86 image or a chroot'ed environment using a user with low privileges). Otherwise your own system can be back doored/r00ted too!

Also, remember that forensics analysis should be done always on the backup copy of the data, *never* on the data itself, in case the data is altered during analysis and the evidence is lost.

FIXME: This paragraph will hopefully provide more information about forensics in a Debian system in the coming future.

FIXME: talk on how to do a debsums on a stable system with the MD5sums on CD and with the recovered file system restored on a separate partition.

FIXME add pointers to forensic analysis papers (like the HoneyNet's reverse challenge or David Dittirch's papers (<http://staff.washington.edu/dittrich/>)).

Chapter 11

Frequently asked Questions (FAQ)

This chapter introduces some of the most common questions from the Debian security mailing list. You should read them before posting there or else people might tell you to RTFM.

11.1 Security in the Debian operating system

11.1.1 Is Debian more secure than X?

A system is only as secure as its administrator is capable of making it. Debian's default installation of services aims to be *secure*, but may not be as paranoid as some other operating systems which install all services *disabled by default*. In any case, the system administrator needs to adapt the security of the system to his local security policy.

For a collection of data regarding security vulnerabilities for many operating systems, see <http://securityfocus.com/vulns/stats.shtml>. Is this data useful? The site lists several factors to consider when interpreting the data, and warns that the data cannot be used to compare the vulnerabilities of one operating system versus another.¹ Also, keep in mind that some Bugtraq vulnerabilities regarding Debian apply only to the *unstable* branch.

Is Debian more secure than other Linux distributions (such as RedHat, SuSE...)?

There are not really many differences between Linux distributions, with exceptions to the base installation and package management system. Most distributions share many of the same applications, with differences mainly in the versions of these applications that are shipped with the distribution's stable release. For example, the kernel, Bind, Apache, OpenSSH, XFree, gcc, zlib, etc. are all common across Linux distributions.

¹For example, based on the Securityfocus data, it might seem that Windows NT is more secure than Linux, which is a questionable assertion. After all, Linux distributions usually provide many more applications compared to Microsoft's Windows NT.

For example, RedHat was unlucky and shipped when foo 1.2.3 was current, which was then later found to have a security hole. Debian, on the other hand, was lucky enough to ship foo 1.2.4, which incorporated the bug fix. That was the case in the big `rpc.statd` (<http://www.cert.org/advisories/CA-2000-17.html>) problem from a couple years ago.

There is a lot of collaboration between the respective security teams for the major Linux distributions. Known security updates are rarely, if ever, left unfixed by a distribution vendor. Knowledge of a security vulnerability is never kept from another distribution vendor, as fixes are usually coordinated upstream, or by CERT (<http://cert.org>). As a result, necessary security updates are usually released at the same time, and the relative security of the different distributions is very similar.

One of Debian's main advantages with regards to security is the ease of system updates through the use of `apt`. Here are some other aspects of security in Debian to consider:

- Debian provides more security tools than other distributions, see 'Security tools in Debian' on page 109.
- Debian's standard installation is smaller (less functionality), and thus more secure. Other distributions, in the name of usability, tend to install many services by default, and sometimes they are not properly configured (remember the Ramen or Lion worms (<http://www.sans.org/y2k/lion.htm>)). Debian's installation is not as limited as OpenBSD (no daemons are active per default), but it's a good compromise.²
- Debian documents best security practices in documents like this one.

11.1.2 There are many Debian bugs in Bugtraq. Does this mean that it is very vulnerable?

The Debian distribution boasts a large and growing number of software packages, probably more than provided by many proprietary operating systems. The more packages installed, the greater the potential for security issues in any given system.

More and more people are examining source code for flaws. There are many advisories related to source code audits of the major software components included in Debian. Whenever such source code audits turn up security flaws, they are fixed and an advisory is sent to lists such as Bugtraq.

Bugs that are present in the Debian distribution usually affect other vendors and distributions as well. Check the "Debian specific: yes/no" section at the top of each advisory (DSA).

11.1.3 Does Debian have any certification related to security?

Short answer: no.

²Without diminishing the fact that some distributions, such as RedHat or Mandrake, are also taking into account security in their standard installations by having the user select *security profiles*, or using wizards to help with configuration of *personal firewalls*.

Long answer: certification costs money and nobody has dedicated the resources in order to certify Debian GNU/Linux to any level of, for example, the Common Criteria. If you are interested in having a certified GNU/Linux distribution, try to provide the resources needed to make it possible.

11.1.4 Are there any hardening programs for Debian?

Yes. Bastille Linux (<http://www.bastille-linux.org>), originally oriented toward other Linux distributions (RedHat and Mandrake), currently works for Debian. Steps are being taken to integrate the changes made to the upstream version into the Debian package, named `bastille`.

Some people believe, however, that a hardening tool does not eliminate the need for good administration.

11.1.5 I want to run XYZ service, which one should I choose?

One of Debian's great strengths is the wide variety of choice available between packages that provide the same functionality (DNS servers, mail servers, ftp servers, web servers, etc.). This can be confusing to the novice administrator when trying to determine which package is right for you. The best match for a given situation depends on a balance between your feature and security needs. Here are some questions to ask yourself when deciding between similar packages:

- Is the software maintained upstream? When was the last release?
- Is the package mature? The version number really does *not* tell you about its maturity. Try to trace the software's history.
- Is the software bug-ridden? Have there been security advisories related to it?
- Does the software provide all the functionality you need? Does it provide more than you really need?

11.1.6 How can I make service XYZ more secure in Debian?

You will find information in this document to make some services (FTP, Bind) more secure in Debian GNU/Linux. For services not covered here, check the program's documentation, or general Linux information. Most of the security guidelines for Unix systems also apply to Debian. In most cases, securing service X in Debian is like securing that service in any other Linux distribution (or Un*x, for that matter).

11.1.7 How can I remove all the banners for services?

If you do not like users connecting to your POP3 daemon, for example, and retrieving information about your system, you might want to remove (or change) the banner the service shows to users.³ Doing so depends on the software you are running for a given service. For example, in `postfix`, you can set your SMTP banner in `/etc/postfix/main.cf`:

```
smtpd_banner = $myhostname ESMTP $mail_name (Debian/GNU)
```

Other software is not as easy to change. OpenSSH will need to be recompiled in order to change the version that it prints. Take care not to remove the first part (`SSH-2.0`) of the banner, which clients use to identify which protocol(s) is supported by your package.

11.1.8 Are all Debian packages safe?

The Debian security team cannot possibly analyze all the packages included in Debian for potential security vulnerabilities, since there are just not enough resources to source code audit the whole project. However, Debian does benefit from the source code audits made by upstream developers or by other projects like the Linux Kernel Security Audit Project (<http://kernel-audit.sourceforge.net/>), or the Linux Security-Audit Project (<http://www.lsap.org/>).

As a matter of fact, a Debian developer could distribute a Trojan in a package, and there is no possible way to check it out. Even if introduced into a Debian branch, it would be impossible to cover all the possible situations in which the Trojan would execute. This is why Debian has a “*no guarantees*” license clause.

However, Debian users can take confidence in the fact that the stable code has a wide audience and most problems would be uncovered through use. Installing untested software is not recommended in a critical system (if you cannot provide the necessary code audit). In any case, if there were a security vulnerability introduced into the distribution, the process used to include packages (using digital signatures) ensures that the problem can be ultimately traced back to the developer. The Debian project has not taken this issue lightly.

11.1.9 Why are some log files/configuration files world-readable, isn't this insecure?

Of course, you can change the default Debian permissions on your system. The current policy regarding log files and configuration files is that they are world readable *unless* they provide sensitive information.

Be careful if you do make changes since:

³Note that this is ‘security by obscurity’, and will probably not be worth the effort in the long term.

- Processes might not be able to write to log files if you restrict their permissions.
- Some applications may not work if the configuration file they depend on cannot be read. For example, if you remove the world-readable permission from `/etc/samba/smb.conf`, the `smbclient` program will not work when run by a normal user.

FIXME: Check if this is written in the Policy. Some packages (i.e. ftp daemons) seem to enforce different permissions.

11.1.10 Why does /root/ (or UserX) have 755 permissions?

As a matter of fact, the same questions stand for any other user. Since Debian's installation does not place *any* file under that directory, there's no sensitive information to protect there. If you feel these permissions are too broad for your system, consider tightening them to 750. For users, read 'Limiting access to other user's information' on page 47.

This Debian security mailing list thread (<http://lists.debian.org/debian-devel/2000/debian-devel-200011/msg00783.html>) has more on this issue.

11.1.11 After installing a grsec/firewall, I started receiving many console messages! How do I remove them?

If you are receiving console messages, and have configured `/etc/syslog.conf` to redirect them to either files or a special TTY, you might be seeing messages sent directly to the console.

The default console log level for any given kernel is 7, which means that any message with lower priority will appear in the console. Usually, firewalls (the LOG rule) and some other security tools log lower than this priority, and thus, are sent directly to the console.

To reduce messages sent to the console, you can use `dmesg (-n option, see dmesg(8))`, which examines and *controls* the kernel ring buffer. To fix this after the next reboot, change `/etc/init.d/klogd` from:

```
KLOGD= " "
```

to:

```
KLOGD=" -c 4 "
```

Use a lower number for `-c` if you are still seeing them. A description of the different log levels can be found in `/usr/include/sys/syslog.h`:

```
#define LOG_EMERG      0      /* system is unusable */
#define LOG_ALERT      1      /* action must be taken immediately */
```

```
#define LOG_CRIT      2      /* critical conditions */
#define LOG_ERR       3      /* error conditions */
#define LOG_WARNING   4      /* warning conditions */
#define LOG_NOTICE    5      /* normal but significant condition */
#define LOG_INFO      6      /* informational */
#define LOG_DEBUG     7      /* debug-level messages */
```

11.1.12 Operating system users and groups

Are all system users necessary?

Yes and no. Debian comes with some predefined users (user id (UID) < 99 as described in Debian Policy (<http://www.debian.org/doc/debian-policy/>) or `/usr/share/doc/base-passwd/README`) to ease the installation of some services that require that they run under an appropriate user/UID. If you do not intend to install new services, you can safely remove those users who do not own any files in your system and do not run any services. In any case, the default behavior is that UID's from 0 to 99 are reserved in Debian, and UID's from 100 to 999 are created by packages on install (and deleted when the package is purged).

To easily find users who don't own any files, execute the following command (run it as root, since a common user might not have enough permissions to go through some sensitive directories):

```
cut -f 1 -d : /etc/passwd | \
while read i; do find / -user "$i" | grep -q . && echo "$i"; done
```

These users are provided by `base-passwd`. Look in its documentation for more information on how these users are handled in Debian. The list of default users (with a corresponding group) follows:

- `root`: Root is (typically) the superuser.
- `daemon`: Some unprivileged daemons that need to write to files on disk run as `daemon.daemon` (e.g., `portmap`, `atd`, probably others). Daemons that don't need to own any files can run as `nobody.nogroup` instead, and more complex or security conscious daemons run as dedicated users. The `daemon` user is also handy for locally installed daemons.
- `bin`: maintained for historic reasons.
- `sys`: same as with `bin`. However, `/dev/vcs*` and `/var/spool/cups` are owned by group `sys`.
- `sync`: The shell of user `sync` is `/bin/sync`. Thus, if its password is set to something easy to guess (such as `""`), anyone can sync the system at the console even if they have don't have an account.

- games: Many games are SETGID to games so they can write their high score files. This is explained in policy.
- man: The man program (sometimes) runs as user man, so it can write cat pages to `/var/cache/man`
- lp: Used by printer daemons.
- mail: Mailboxes in `/var/mail` are owned by group mail, as explained in policy. The user and group are used for other purposes by various MTA's as well.
- news: Various news servers and other associated programs (such as suck) use user and group news in various ways. Files in the news spool are often owned by user and group news. Programs such as inews that can be used to post news are typically SETGID news.
- uucp: The uucp user and group is used by the UUCP subsystem. It owns spool and configuration files. Users in the uucp group may run uucico.
- proxy: Like daemon, this user and group is used by some daemons (specifically, proxy daemons) that don't have dedicated user id's and that need to own files. For example, group proxy is used by pdnsd, and squid runs as user proxy.
- majordom: Majordomo has a statically allocated UID on Debian systems for historical reasons. It is not installed on new systems.
- postgres: Postgresql databases are owned by this user and group. All files in `/var/lib/postgresql` are owned by this user to enforce proper security.
- www-data: Some web browsers run as www-data. Web content should *not* be owned by this user, or a compromised web server would be able to rewrite a web site. Data written out by web servers, including log files, will be owned by www-data.
- backup: So backup/restore responsibilities can be locally delegated to someone without full root permissions.
- operator: Operator is historically (and practically) the only 'user' account that can login remotely, and doesn't depend on NIS/NFS.
- list: Mailing list archives and data are owned by this user and group. Some mailing list programs may run as this user as well.
- irc: Used by irc daemons. A statically allocated user is needed only because of a bug in ircd, which SETUID(s) itself to a given UID on startup.
- gnats.
- nobody, nogroup: Daemons that need not own any files run as user nobody and group nogroup. Thus, no files on a system should be owned by this user or group.

Other groups which have no associated user:

- **adm**: Group `adm` is used for system monitoring tasks. Members of this group can read many log files in `/var/log`, and can use `xconsole`. Historically, `/var/log` was `/usr/adm` (and later `/var/adm`), thus the name of the group.
- **tty**: TTY devices are owned by this group. This is used by `write` and `wall` to enable them to write to other people's TTYs.
- **disk**: Raw access to disks. Mostly equivalent to root access.
- **kmem**: `/dev/kmem` and similar files are readable by this group. This is mostly a BSD relic, but any programs that need direct read access to the system's memory can thus be made SETGID `kmem`.
- **dialout**: Full and direct access to serial ports. Members of this group can reconfigure the modem, dial anywhere, etc.
- **dip**: The group's name stands for "Dial-up IP", and membership in `dip` allows you to use tools like `ppp`, `dip`, `wvdial`, etc. to dial up a connection. The users in this group cannot configure the modem, but may run the programs that make use of it.
- **fax**: Allows members to use fax software to send / receive faxes.
- **voice**: Voicemail, useful for systems that use modems as answering machines.
- **cdrom**: This group can be used locally to give a set of users access to a CDROM drive.
- **floppy**: This group can be used locally to give a set of users access to a floppy drive.
- **tape**: This group can be used locally to give a set of users access to a tape drive.
- **sudo**: Members of this group don't need to type their password when using `sudo`. See `/usr/share/doc/sudo/OPTIONS`.
- **audio**: This group can be used locally to give a set of users access to an audio device.
- **src**: This group owns source code, including files in `/usr/src`. It can be used locally to give a user the ability to manage system source code.
- **shadow**: `/etc/shadow` is readable by this group. Some programs that need to be able to access the file are SETGID `shadow`.
- **utmp**: This group can write to `/var/run/utmp` and similar files. Programs that need to be able to write to it are SETGID `utmp`.
- **video**: This group can be used locally to give a set of users access to an video device.
- **staff**: Allows users to add local modifications to the system (`/usr/local`, `/home`) without needing root privileges. Compare with group "adm", which is more related to monitoring/security.
- **users**: While Debian systems use the private user group system by default (each user has their own group), some prefer to use a more traditional group system, in which each user is a member of this group.

What is the difference between the adm and the staff group?

The 'adm' group are usually administrators, and this group permission allows them to read log files without having to su. The 'staff' group are usually help-desk/junior sysadmins, allowing them to work in `/usr/local` and create directories in `/home`.

11.1.13 Why is there a new group when I add a new user? (or Why does Debian give each user one group?)

The default behavior in Debian is that each user has its own, private group. The traditional UN*X scheme assigned all users to the *users* group. Additional groups were created and used to restrict access to shared files associated with different project directories. Managing files became difficult when a single user worked on multiple projects because when someone created a file, it was associated with the primary group to which they belong (e.g. 'users').

Debian's scheme solves this problem by assigning each user to their own group; so that with a proper umask (0002) and the SETGID bit set on a given project directory, the correct group is automatically assigned to files created in that directory. This makes it easier for people who work on multiple projects, because they will not have to change groups or umasks when working on shared files.

You can, however, change this behavior by modifying `/etc/adduser.conf`. Change the *USERGROUPS* variable to 'no', so that a new group is not created when a new user is created. Also, set *USERS_GID* to the GID of the users group which all users will belong to.

11.1.14 Question regarding services and open ports

Why are all services activated upon installation?

That's just an approach to the problem of being, on one side, security conscious and on the other side user friendly. Unlike OpenBSD, which disables all services unless activated by the administrator, Debian GNU/Linux will activate all installed services unless deactivated (see 'Disabling daemon services' on page 27 for more information). After all you installed the service, didn't you?

There has been much discussion on Debian mailing lists (both at `debian-devel` and at `debian-security`) regarding which is the better approach for a standard installation. However, as of this writing (March 2002), there still isn't a consensus.

Can I remove `inetd`?

Inetd is not easy to remove since `netbase` depends on the package that provides it (`netkit-inetd`). If you want to remove it, you can either disable it (see 'Disabling daemon services' on page 27) or remove the package by using the `equivs` package.

Why do I have port 111 open?

Port 111 is sunrpc's portmapper, and it is installed by default as part of Debian's base installation since there is no need to know when a user's program might need RPC to work correctly. In any case, it is used mostly for NFS. If you do not need it, remove it as explained in 'Disabling RPC services' on page 86.

What use is `identd` (port 113) for?

Identd service is an authentication service that identifies the owner of a specific TCP/IP connection to the remote server accepting the connection. Typically, when a user connects to a remote host, `inetd` on the remote host sends back a query to port 113 to find the owner information. It is often used by mail, FTP and IRC servers, and can also be used to track down which user in your local system is attacking a remote system.

There has been extensive discussion on the security of `identd` (See mailing list archives (<http://lists.debian.org/debian-security/2001/debian-security-200108/msg00297.html>)). In general, `identd` is more helpful on a multi-user system than on a single user workstation. If you don't have a use for it, disable it, so that you are not leaving a service open to the outside world. If you decide to firewall the `identd` port, *please* use a reject policy and not a deny policy, otherwise a connection to a server utilizing `identd` will hang until a timeout expires (see reject or deny issues (http://logi.cc/linux/reject_or_deny.php3)).

I have services using port 1 and 6, what are they and how can I remove them?

If you have run the command `netstat -an` and receive:

```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
raw      0      0 0.0.0.0:1               0.0.0.0:*               7
-
raw      0      0 0.0.0.0:6               0.0.0.0:*               7
-
```

You are *not* seeing processes listening on TCP/UDP port 1 and 6. In fact, you are seeing a process listening on a *raw* socket for protocols 1 (ICMP) and 6 (TCP). Such behavior is common to both Trojans and some intrusion detection systems such as `iip1`, `iplogger` and `portsentry`. If you have these packages simply remove them. If you do not, try `netstat's -p` (process) option to see which process is running these listeners.

I found the port XYZ open, can I close it?

Yes, of course. The ports you are leaving open should adhere to your individual site's policy regarding public services available to other networks. Check if they are being opened by `inetd`

(see 'Disabling `inetd` services' on page 28), or by other installed packages and take the appropriate measures (i.e, configure `inetd`, remove the package, avoid it running on boot-up).

Will removing services from `/etc/services` help secure my box?

No, `/etc/services` only provides a mapping between a virtual name and a given port number. Removing names from this file will not (usually) prevent services from being started. Some daemons may not run if `/etc/services` is modified, but that's not the norm. To properly disable the service, see 'Disabling daemon services' on page 27.

11.1.15 Common security issues

I have lost my password and cannot access the system!

The steps you need to take in order to recover from this depend on whether or not you have applied the suggested procedure for limiting access to `lilo` and your system's BIOS.

If you have limited both, you need to disable the BIOS setting that only allows booting from the hard disk before proceeding. If you have also forgotten your BIOS password, you will have to reset your BIOS by opening the system and manually removing the BIOS battery.

Once you have enabled booting from a CD-ROM or diskette enable, try the following:

- Boot-up from a rescue disk and start the kernel
- Go to the virtual console (Alt+F2)
- Mount the hard disk where your `/root` is
- Edit (Debian 2.2 rescue disk comes with the editor `ae`, and Debian 3.0 comes with `nano-tiny` which is similar to `vi`) `/etc/shadow` and change the line:

```
root:asdfj1290341274075:XXXX:X:XXXX:X::: (X=any number)
```

to:

```
root::XXXX:X:XXXX:X:::
```

This will remove the forgotten root password, contained in the first colon separated field after the user name. Save the file, reboot the system and login with root using an empty password. Remember to reset the password. This will work unless you have configured the system more tightly, i.e. if you have not allowed users to have null passwords or not allowed root to login from the console.

If you have introduced these features, you will need to enter into single user mode. If LILLO has been restricted, you will need to rerun `lilo` just after the root reset above. This is quite tricky

since your `/etc/lilo.conf` will need to be tweaked due to the root (`/`) file system being a ramdisk and not the real hard disk.

Once LILO is unrestricted, try the following:

- Press the Alt, shift or Control key just before the system BIOS finishes, and you should get the LILO prompt.
- Type `linux single`, `linux init=/bin/sh` or `linux 1` at the prompt.
- This will give you a shell prompt in single-user mode (it will ask for a password, but you already know it)
- Re-mount read/write the root (`/`) partition, using the mount command.

```
# mount -o remount,rw /
```

- Change the superuser password with `passwd` (since you are superuser it will not ask for the previous password).

11.1.16 How do I accomplish setting up a service for my users without giving out shell accounts?

For example, if you want to set up a POP service, you don't need to set up a user account for each user accessing it. It's best to set up directory-based authentication through an external service (like Radius, LDAP or an SQL database). Just install the appropriate PAM library (`libpam-radius-auth`, `libpam-ldap`, `libpam-pgsql` or `libpam-mysql`), read the documentation (for starters, see 'User authentication: PAM' on page 39) and configure the PAM-enabled service to use the back end you have chosen. This is done by editing the files under `/etc/pam.d/` for your service and modifying the

```
auth required pam_unix_auth.so shadow nullok use_first_pass
```

to, for example, `ldap`:

```
auth required pam_ldap.so
```

In the case of LDAP directories, some services provide LDAP schemas to be included in your directory that are required in order to use LDAP authentication. If you are using a relational database, a useful trick is to use the *where* clause when configuring the PAM modules. For example, if you have a database with the following table attributes:

```
(user_id, user_name, realname, shell, password, UID, GID, homedir, sys, pop
```

By making the services attributes boolean fields, you can use them to enable or disable access to the different services just by inserting the appropriate lines in the following files:

- `/etc/pam.d/imap:where=imap=1.`
- `/etc/pam.d/qpopper:where=pop=1.`
- `/etc/nss-mysql*.conf:users.where_clause = user.sys = 1;.`
- `/etc/proftpd.conf:SQLWhereClause "ftp=1".`

11.2 My system is vulnerable! (Are you sure?)

11.2.1 Vulnerability assessment scanner X says my Debian system is vulnerable!

Many vulnerability assessment scanners give false positives when used on Debian systems, since they only use version checks to determine if a given software package is vulnerable, but do not really test the security vulnerability itself. Since Debian does not change software versions when fixing a package (many times the fix made for newer releases is back ported), some tools tend to think that an updated Debian system is vulnerable when it is not.

If you think your system is up to date with security patches, you might want to use the cross references to security vulnerability databases published with the DSAs (see 'Debian Security Advisories' on page 96) to weed out false positives, if the tool you are using includes CVE references.

11.2.2 I've seen an attack in my system's logs. Is my system compromised?

A trace of an attack does not always mean that your system has been compromised, and you should take the usual steps to determine if the system is indeed compromised (see 'After the compromise' on page 127). Also, notice that the fact that you see the attacks in the log might mean your system is already vulnerable to it (a determined attacker might have used some other vulnerability besides the ones you have seen, however).

11.2.3 I have found strange 'MARK' lines in my logs: Am I compromised?

You might find the following lines in your system logs:

```
Dec 30 07:33:36 debian -- MARK --
Dec 30 07:53:36 debian -- MARK --
Dec 30 08:13:36 debian -- MARK --
```

This does not indicate any kind of compromise, and users changing between Debian releases might find it strange. If your system does not have high loads (or many active services), these lines might appear throughout your logs. This is an indication that your `syslogd` daemon is running properly. From `syslogd(8)`:

```
-m interval
    The syslogd logs a mark timestamp regularly. The
    default interval between two -- MARK -- lines is 20
    minutes. This can be changed with this option.
    Setting the interval to zero turns it off entirely.
```

11.2.4 I found users using 'su' in my logs: Am I compromised?

You might find lines in your logs like:

```
Apr  1 09:25:01 server su[30315]: + ??? root-nobody
Apr  1 09:25:01 server PAM_unix[30315]: (su) session opened for user nobody
```

Don't worry too much. Check to see if these entries are due to cron jobs (usually `/etc/cron.daily/find` or `logrotate`):

```
$ grep 25 /etc/crontab
25 6 * * * root test -e /usr/sbin/anacron || run-parts --report
/etc/cron.daily
$ grep nobody /etc/cron.daily/*
find:cd / && updatedb --localuser=nobody 2>/dev/null
```

11.2.5 I have found possible 'SYN flooding' in my logs: Am I under attack?

If you see entries like these in your logs:

```
May 1 12:35:25 linux kernel: possible SYN flooding on port X. Sending cooki
May 1 12:36:25 linux kernel: possible SYN flooding on port X. Sending cooki
May 1 12:37:25 linux kernel: possible SYN flooding on port X. Sending cooki
May 1 13:43:11 linux kernel: possible SYN flooding on port X. Sending cooki
```

Check if there is a high number of connections to the server using `netstat`, for example:

```
linux:~# netstat -ant | grep SYN_RECV | wc -l
9000
```

This is an indication of a denial of service (DoS) attack against your system's X port (most likely against a public service such as a web server or mail server). You should activate TCP syncookies in your kernel, see 'Configuring Syncookies' on page 60. Note, however, that a DoS attack might flood your network even if you can stop it from crashing your systems (due to file descriptors being depleted, the system might become unresponsive until the TCP connections timeout). The only effective way to stop this attack is to contact your network provider.

11.2.6 I have found strange root sessions in my logs: Am I compromised?

You might see these kind of entries in your `/var/log/auth.log` file:

```
May 2 11:55:02 linux PAM_unix[1477]: (cron) session closed for user root
May 2 11:55:02 linux PAM_unix[1476]: (cron) session closed for user root
May 2 12:00:01 linux PAM_unix[1536]: (cron) session opened for user root by
(UID=0)
May 2 12:00:02 linux PAM_unix[1536]: (cron) session closed for user root
```

These are due to a cron job being executed (in this example, every five minutes). To determine which program is responsible for these jobs, check entries under: `/etc/crontab`, `/etc/cron.d`, `/etc/crond.daily` and root's crontab under `/var/spool/cron/crontabs`.

11.2.7 I have suffered a break-in, what do I do?

There are several steps you might want to take in case of a break-in:

- Check if your system is up to date with security patches for published vulnerabilities. If your system is vulnerable, the chances that the system is in fact compromised are increased. The chances increase further if the vulnerability has been known for a while, since there is usually more activity related to older vulnerabilities. Here is a link to SANS Top 20 Security Vulnerabilities (<http://www.sans.org/top20.htm>).
- Read this document, especially the 'After the compromise' on page 127 section.
- Ask for assistance. You might use the `debian-security` mailing list and ask for advice on how to recover/patch your system.
- Notify your local CERT (<http://www.cert.org>) (if it exists, otherwise you may want to consider contacting CERT directly). This might or might not help you, but, at the very least, it will inform CERT of ongoing attacks. This information is very valuable in determining which tools and attacks are being used by the *blackhat* community.

11.2.8 How can I trace an attack?

By watching the logs (if they have not been tampered with), using intrusion detection systems (see ‘Set up Intrusion Detection’ on page 120), `traceroute`, `whois` and similar tools (including forensic analysis), you may be able to trace an attack to the source. The way you should react to this information depends solely on your security policy, and what *you* consider is an attack. Is a remote scan an attack? Is a vulnerability probe an attack?

11.2.9 Program X in Debian is vulnerable, what do I do?

First, take a moment to see if the vulnerability has been announced in public security mailing lists (like Bugtraq) or other forums. The Debian Security Team keeps up to date with these lists, so they are may also be aware of the problem. Do not take any further actions if you see an announcement at <http://security.debian.org>.

If no information seems to be published, please send e-mail about the affected package(s), as well as a detailed description of the vulnerability (proof of concept code is also OK), to security@debian.org. This will get you in touch with Debian’s security team.

11.2.10 The version number for a package indicates that I am still running a vulnerable version!

Instead of upgrading to a new release, Debian back ports security fixes to the version that was shipped in the stable release. The reason for this is to make sure that the stable release changes as little as possible, so that things will not change or break unexpectedly as a result of a security fix. You can check if you are running a secure version of a package by looking at the package changelog, or comparing its exact (upstream version -slash- debian release) version number with the version indicated in the Debian Security Advisory.

11.2.11 Specific software

proftpd is vulnerable to a Denial of Service attack.

Add `DenyFilter *.*/*` to your configuration file, and for more information see <http://www.proftpd.org/critbugs.html>.

After installing portsentry, there are a lot of ports open.

That’s just the way `portsentry` works. It opens about twenty unused ports to try to detect port scans.

11.3 Questions regarding the Debian security team

11.3.1 What is a Debian Security Advisory (DSA)?

It is information sent by the Debian Security Team (see below) regarding the discovery and fix for a security related vulnerability in a package available in Debian GNU/Linux. Signed DSAs are sent to public mailing lists (debian-security-announce) and posted on Debian's web site (both in the front page and in the security area (<http://www.debian.org/security/>)).

DSAs include information on the affected package(s), the security flaw that was discovered and where to retrieve the updated packages (and their MD5 sums).

11.3.2 The signature on Debian advisories does not verify correctly!

This is most likely a problem on your end. The debian-security-announce list has a filter that only allows messages with a correct signature from one of the security team members to be posted.

Most likely some piece of mail software on your end slightly changes the message, thus breaking the signature. Make sure your software does not do any MIME encoding or decoding, or tab/space conversions. Known culprits are evolution, fetchmail (with the mimedecode option enabled) and formail (from procmail 3.14 only).

11.3.3 How are security incidents handled in Debian?

Once the Security Team receives a notification of an incident, one or more members review it and consider Debian/stable vulnerable or not. If our system is vulnerable, it is worked on a fix for the problem. The package maintainer is contacted as well, if he didn't contact the Security Team already. Finally, the fix is tested and new packages are prepared, which then are compiled on all stable architectures and uploaded afterward. After all this tasks are done a DSA is sent to the public mailing lists.

11.3.4 How much time will it take Debian to fix vulnerability XXXX?

The Debian security team works quickly to send advisories and produce fixed packages for the stable branch once a vulnerability is discovered. A report published in the debian-security mailing list (<http://lists.debian.org/debian-security/2001/debian-security-200112/msg00257.html>) showed that in the year 2001, it took the Debian Security Team an average of 35 days to fix security-related vulnerabilities. However, over 50% of the vulnerabilities were fixed in a 10-day time frame, and over 15% of them were fixed the *same day* the advisory was released.

However, when asking this question people tend to forget that:

- DSAs are not sent until:

- packages are available for *all* architectures supported by Debian (which takes some time for packages that are part of the system core, especially considering the number of architectures supported in the stable release).
 - new packages are thoroughly tested in order to ensure that no new bugs are introduced
- Packages might be available before the DSA is sent (in the incoming queue or on the mirrors).
 - Debian is a volunteer-based project.
 - Debian is licensed with a “no guarantees” clause.

If you want more in-depth analysis on the time it takes for the Security Team to work on vulnerabilities, you should consider that new DSAs (see ‘Debian Security Advisories’ on page 96) published on the security website (<http://security.debian.org>), and the metadata used to generate them, include links to vulnerability databases. You could download the sources from the web server (from the CVS (<http://cvs.debian.org>)) or use the HTML pages to determine the time that it takes for Debian to fix vulnerabilities and correlate this data with public databases.

11.3.5 How is security handled for testing and unstable?

The short answer is: it’s not. Testing and unstable are rapidly moving targets and the security team does not have the resources needed to properly support those branches. If you want to have a secure (and stable) server you are strongly encouraged to stay with stable.

However, the unstable branch usually gets security fixes quite quickly, because those fixes are usually available upstream faster (other versions, like those in the stable branch, usually need to be back ported).

11.3.6 I use an older version of Debian, is it supported by the Debian Security Team?

No. Unfortunately, the Debian Security Team cannot handle both the stable release (unofficially, also the unstable) and other older releases. However, you can expect security updates for a limited period of time (usually several months) immediately following the release of a new Debian distribution.

11.3.7 Why are there no official mirrors for security.debian.org?

The purpose of security.debian.org is to make security updates available as quickly and easily as possible. Mirrors would add extra complexity that is not needed and can cause frustration if they are not up to date.

11.3.8 I've seen DSA 100 and DSA 102, what happened to DSA 101?

Several vendors (mostly of GNU/Linux, but also of BSD derivatives) coordinate security advisories for some incidents and agree to a particular timeline so that all vendors are able to release an advisory at the same time. This was decided in order to not discriminate against some vendors that need more time to prepare patches.

In some cases, the Debian Security Team prepares advisories in advance, and holds the advisory number until the advisory can be released. Hence, the gaps in DSA numbers.

11.3.9 How can I reach the security team?

Security information can be sent to security@debian.org, which is read by all Debian developers. If you have sensitive information please use team@security.debian.org which only the members of the security team read. If desired email can be encrypted with the Debian Security Contact key (key ID 0x363CCD95).

11.3.10 What difference is there between security@debian.org and debian-security@lists.debian.org?

When you send messages to security@debian.org, they are sent to the developers mailing list (`debian-private`). All Debian developers are subscribed to this list and posts are kept private (i.e. are not archived at the public website). The public mailing list, debian-security@lists.debian.org, is open to anyone that wants to subscribe (<http://www.debian.org/MailingLists/>), and there are searchable archives available `id="http://lists.debian.org/search.html" name="here">`.

11.3.11 How can I contribute to the Debian security team?

- By contributing to this document, fixing FIXMEs or providing new content. Documentation is important and reduces the overhead of answering common issues. Translation of this documentation into other languages is also of great help.
- By packaging applications that are useful for checking or enhancing security in a Debian GNU/Linux system. If you are not a developer, file a WNPP bug (<http://www.debian.org/devel/wnpp/>) and ask for software you think would be useful, but is not currently provided.
- Audit applications in Debian or help solve security bugs and report issues to security@debian.org. Other projects' work like the Linux Kernel Security Audit Project (<http://kernel-audit.sourceforge.net/>) or the Linux Security-Audit Project (<http://www.lsap.org/>) increase the security of Debian GNU/Linux, since contributions will eventually help here, too.

In all cases, please review each problem before reporting it to security@debian.org. If you are able to provide patches, that would speed up the process. Do not simply forward Bugtraq mails, since they are already received. Providing additional information, however, is always a good idea.

11.3.12 Who is the Security Team composed of?

The Debian Security Team currently consists of five members and two secretaries. The Security Team itself appoints people to join the team.

11.3.13 Does the Debian Security team check every new package in Debian?

No, the Debian security team does not check every new package and there is no automatic (lintian) check to detect malicious new packages, since those checks are rather impossible to detect automatically. Maintainers, however, are fully responsible for the packages they introduce into Debian, and all packages are first signed by an authorized developer(s). The developer is in charge of analyzing the security of all packages that they maintain.

Appendix A

The hardening process step by step

Below is a post-installation, step-by-step procedure for hardening a Debian 2.2 GNU/Linux system. This is one possible approach to such a procedure and is oriented toward the hardening of network services. It is included to show the entire process you might use during configuration. Also, see 'Configuration checklist' on page [155](#).

- Install the system, taking into account the information regarding partitioning included earlier in this document. After base installation, go into custom install. Do not select task packages. Select shadow passwords.
- Using `dselect`, remove all unneeded but selected packages before doing `[I]ninstall`. Keep the bare minimum of packages for the system.
- Update all software from the latest packages available at security.debian.org as explained previously in 'Execute a security update' on page [38](#).
- Implement the suggestions presented in this manual regarding user quotas, login definitions and `lilo`
- Make a list of services currently running on your system. Try:

```
$ ps -aux
$ netstat -pn -l -A inet
# /usr/sbin/lsof -i | grep LISTEN
```

You will need to install `lsof-2.2` for the third command to work (run it as root). You should be aware that `lsof` can translate the word `LISTEN` to your locale settings.

- In order to remove unnecessary services, first determine what package provides the service and how it is started. This can be accomplished by checking the program that listens in the socket. The following shell script, which uses the programs `lsof` and `dpkg`, does just that:

```
#!/bin/sh
# FIXME: this is quick and dirty; replace with a more robust script
for i in `sudo lsof -i | grep LISTEN | cut -d " " -f 1 | sort -u` ; do
  pack=`dpkg -S $i |grep bin |cut -f 1 -d : | uniq`
  echo "Service $i is installed by $pack";
  init=`dpkg -L $pack |grep init.d/ `
  if [ ! -z "$init" ]; then
    echo "and is run by $init"
  fi
done
```

- Once you find any unwanted services, remove the associated package (with `dpkg -purge`), or disable the service from starting automatically at boot time using `update-rc.d` (see ‘Disabling daemon services’ on page 27).
- For `inetd` services (launched by the superdaemon), check which services are enabled in `/etc/inetd.conf` using:

```
$ grep -v "^#" /etc/inetd.conf | sort -u
```

Then disable those services that are not needed by commenting out the line that includes them in `/etc/inetd.conf`, removing the package, or using `update-inetd`.

- If you have wrapped services (those using `/usr/sbin/tcpd`), check that the files `/etc/hosts.allow` and `/etc/hosts.deny` are configured according to your service policy.
- If the server uses more than one external interface, depending on the service, you may want to limit the service to listen on a specific interface. For example, if you want internal FTP access only, make the FTP daemon listen only on your management interface, not on all interfaces (i.e, 0.0.0.0:21).
- Re-boot the machine, or switch from single user `user` and then back to multiuser using the commands:

```
$ init 1
(... )
$ init 2
```

- Check the services now available, and, if necessary, repeat the steps above.
- Now install the needed services, if you have not done so already, and configure them properly.
- Use the following shell command to determine what user each available service is running as:

```
$ for i in `ls /usr/sbin/ | grep LISTEN | cut -d " " -f 1 | sort -u`
> do user=`ps -ef | grep $i | grep -v grep | cut -f 1 -d " "` ; \
> echo "Service $i is running as user $user"; done
```

Consider changing these services to a specific user/group and maybe chroot'ing them for increased security. You can do this by changing the `/etc/init.d` scripts which start the service. Most services in Debian use `start-stop-daemon`, which has `$$$options` (`--change-uid` and `--chroot`) for accomplishing this. A word of warning regarding the chroot'ing of services: you may need to put all the files installed by the package (use `dpkg -L`) providing the service, as well as any packages it depends on, in the chroot'ed environment. Information about setting up a chroot environment for the `ssh` program can be found in 'Chroot environment for SSH' on page 177.

- Repeat the steps above in order to check that only desired services are running and that they are running as the desired user/group combination.
- Test the installed services in order to see if they work as expected.
- Check the system using a vulnerability assessment scanner (like `nessus`), in order to determine vulnerabilities in the system (i.e., misconfigurations, old services or unneeded services).
- Install network and host intrusion measures like `snort` and `logentry`.
- Repeat the network scanner step and verify that the intrusion detection systems are working correctly.

For the truly paranoid, also consider the following:

- Add firewalling capabilities to the system, accepting incoming connections only to offered services and limiting outgoing connections only to those that are authorized.
- Re-check the installation with a new vulnerability assessment using a network scanner.
- Using a network scanner, check outbound connections from the system to an outside host and verify that unwanted connections do not find their way out.

FIXME: this procedure considers service hardening but not system hardening at the user level, include information regarding checking user permissions, SETUID files and freezing changes in the system using the ext2 file system.

Appendix B

Configuration checklist

This appendix briefly reiterates points from other sections in this manual in a condensed checklist format. This is intended as a quick summary for someone who has already read the manual. There are other good checklists available, including Kurt Seifried's Securing Linux Step by Step (<http://seifried.org/security/os/linux/20020324-securing-linux-step-by-step.html>) and CERT's Unix Security Checklist (http://www.cert.org/tech_tips/usc20_full.html).

FIXME: This is based on v1.4 of the manual and might need to be updated.

- Limit physical access and booting capabilities
 - Enable BIOS password
 - Disable floppy/cdrom/... booting
 - Set a LILO or GRUB password (`/etc/lilo.conf` or `/boot/grub/menu.lst`, respectively); check that the LILO or GRUB configuration file is read-protected.
 - Disallow MBR floppy booting back door by overwriting the MBR (maybe not?)
- Partitioning
 - Separate user-writable data, non-system data, and rapidly changing run-time data to their own partitions
 - Set `nosuid`, `noexec`, `nodev` mount options in `/etc/fstab` on ext2 partitions such as `/tmp`.
- Password hygiene and login security
 - Set a good root password
 - Enable password shadowing and MD5
 - Install and use PAM

- * Add MD5 support to PAM and make sure that (generally speaking) entries in `/etc/pam.d/` files which grant access to the machine have the second field in the pam.d file set to `requisite` or `required`.
 - * Tweak `/etc/pam.d/login` so as to only permit local root logins.
 - * Also mark authorized `tty:s` in `/etc/security/access.conf` and generally set up this file to limit root logins as much as possible.
 - * Add `pam_limits.so` if you want to set per-user limits
 - * Tweak `/etc/pam.d/passwd`: set minimum length of passwords higher (6 characters maybe) and enable MD5
 - * Add group `wheel` to `/etc/group` if desired; add `pam_wheel.so group=wheel` entry to `/etc/pam.d/su`
 - * For custom per-user controls, use `pam_listfile.so` entries where appropriate
 - * Have an `/etc/pam.d/other` file and set it up with tight security
 - Set up limits in `/etc/security/limits.conf` (note that `/etc/limits` is not used if you are using PAM)
 - Tighten up `/etc/login.defs`; also, if you enabled MD5 and/or PAM, make sure you make the corresponding changes here, too
 - Disable root ftp access in `/etc/ftputers`
 - Disable network root login; use `su(1)` or `sudo(1)`. (consider installing `sudo`)
 - Use PAM to enforce additional constraints on logins?
- Other local security issues
 - Kernel tweaks (see ‘Configuring kernel network features’ on page 59)
 - Kernel patches (see ‘Useful kernel patches’ on page 122)
 - Tighten up log file permissions (`/var/log/{last, fail}log`, Apache logs)
 - Verify that SETUID checking is enabled in `/etc/checksecurity.conf`
 - Consider making some log files append-only and configuration files immutable using `chattr` (ext2 file systems only)
 - Set up file integrity (see ‘Checking file system integrity’ on page 58). Install `debsums`
 - Consider replacing `locate` with `slocate`
 - Log everything to a local printer?
 - Burn your configuration on a boot-able CD and boot off that?
 - Disable kernel modules?
 - Limit network access
 - Install and configure `ssh` (suggest `PermitRootLogin No` in `/etc/ssh/sshd_config`, `PermitEmptyPasswords No`; note other suggestions in text also)
 - Consider disabling or removing `in.telnetd`

- Generally, disable gratuitous services in `/etc/inetd.conf` using `update-inetd -disable` (or disable `inetd` altogether, or use a replacement such as `xinetd` or `rlinead`)
 - Disable other gratuitous network services; mail, ftp, DNS, WWW etc should not be running if you do not need them and monitor them regularly.
 - For those services which you do need, do not just use the most common programs, look for more secure versions shipped with Debian (or from other sources). Whatever you end up running, make sure you understand the risks.
 - Set up `chroot` jails for outside users and daemons.
 - Configure firewall and `tcpwrappers` (i.e. `hosts_access(5)`); note trick for `/etc/hosts.deny` in text.
 - If you run ftp, set up your `ftpd` server to always run `chroot`'ed to the user's home directory
 - If you run X, disable `xhost` authentication and go with `ssh` instead; better yet, disable remote X if you can (add `-nolisten tcp` to the X command line and turn off XDMCP in `/etc/X11/xdm/xdm-config` by setting the `requestPort` to 0)
 - Disable outside access to printers
 - Tunnel any IMAP or POP sessions through SSL or `ssh`; install `stunnel` if you want to provide this service to remote mail users
 - Set up a log host and configure other machines to send logs to this host (`/etc/syslog.conf`)
 - Secure BIND, Sendmail, and other complex daemons (run in a `chroot` jail; run as a non-root pseudo-user)
 - Install `snort` or a similar logging tool.
 - Do without NIS and RPC if you can (disable `portmap`).
- Policy issues
 - Educate users about the whys and hows of your policies. When you have prohibited something which is regularly available on other systems, provide documentation which explains how to accomplish similar results using other, more secure means.
 - Prohibit use of protocols which use clear-text passwords (`telnet`, `rsh` and `friends`; `ftp`, `imap`, `http`, ...).
 - Prohibit programs which use `SVGAlib`.
 - Use disk quotas.
 - Keep informed about security issues
 - Subscribe to security mailing lists
 - Configure `apt` for security updates – add to `/etc/apt/sources.list` an entry (or entries) for `http://security.debian.org/debian-security`
 - Also remember to periodically run `apt-get update ; apt-get upgrade` (perhaps install as a `cron` job?) as explained in 'Execute a security update' on page 38.

Appendix C

Setting up a stand-alone IDS

You can easily set up a dedicated Debian system as a stand-alone Intrusion Detection System using `snort`.

Some guidelines:

- Install a base Debian system and select no additional packages.
- Download and manually (with `dpkg`) install necessary packages (see installed packages list below).
- Download and install ACID (Analysis Console for Intrusion Databases).

ACID is currently packaged for Debian as `acidlab`. It provides a graphical WWW interface to `snort`'s output. It can also be downloaded from <http://www.cert.org/kb/acid/>, <http://acidlab.sourceforge.net> or <http://www.andrew.cmu.edu/~rdanyliw/snort/>. You might also want to read the Snort Statistics HOWTO (<http://www.linuxdoc.org/HOWTO/Snort-Statistics-HOWTO/index.html>).

This system should be set up with at least two interfaces: one interface connected to a management LAN (for accessing the results and maintaining the system), and one interface with no IP address attached to the network segment being analyzed.

The standard Debian `/etc/network/interfaces` file normally used to configure network cards cannot be used, since the `ifup` and `ifdown` programs expect an IP address. Instead, simply use `ifconfig eth0 up`.

Besides the base installation, `acidlab` also depends on the packages `php4` and `apache` among others. Download the following packages (Note: the versions might vary depending on which Debian distribution you are using, this list is from Debian *woody* September 2001):

```
ACID-0.9.5b9.tar.gz
adduser_3.39_all.deb
apache-common_1.3.20-1_i386.deb
```

```
apache_1.3.20-1_i386.deb
debconf_0.9.77_all.deb
dialog_0.9a-20010527-1_i386.deb
fileutils_4.1-2_i386.deb
klogd_1.4.1-2_i386.deb
libbz2-1.0_1.0.1-10_i386.deb
libc6_2.2.3-6_i386.deb
libdb2_2.7.7-8_i386.deb
libdbd-mysql-perl_1.2216-2_i386.deb
libdbi-perl_1.18-1_i386.deb
libexpat1_1.95.1-5_i386.deb
libgdbmg1_1.7.3-27_i386.deb
libmm11_1.1.3-4_i386.deb
libmysqlclient10_3.23.39-3_i386.deb
libncurses5_5.2.20010318-2_i386.deb
libpcap0_0.6.2-1_i386.deb
libpcre3_3.4-1_i386.deb
libreadline4_4.2-3_i386.deb
libstdc++2.10-glibc2.2_2.95.4-0.010703_i386.deb
logrotate_3.5.4-2_i386.deb
mime-support_3.11-1_all.deb
mysql-client_3.23.39-3_i386.deb
mysql-common_3.23.39-3.1_all.deb
mysql-server_3.23.39-3_i386.deb
perl-base_5.6.1-5_i386.deb
perl-modules_5.6.1-5_all.deb
perl_5.6.1-5_i386.deb
php4-mysql_4.0.6-4_i386.deb
php4_4.0.6-1_i386.deb
php4_4.0.6-4_i386.deb
snort_1.7-9_i386.deb
sysklogd_1.4.1-2_i386.deb
zlib1g_1.1.3-15_i386.deb
```

Installed packages (dpkg -l):

```
ii  adduser          3.39
ii  ae               962-26
ii  apache           1.3.20-1
ii  apache-common   1.3.20-1
ii  apt              0.3.19
ii  base-config     0.33.2
ii  base-files      2.2.0
ii  base-passwd     3.1.10
ii  bash            2.03-6
```

ii	bsdutils	2.10f-5.1
ii	console-data	1999.08.29-11.
ii	console-tools	0.2.3-10.3
ii	console-tools-	0.2.3-10.3
ii	cron	3.0p11-57.2
ii	debconf	0.9.77
ii	debianutils	1.13.3
ii	dialog	0.9a-20010527-
ii	diff	2.7-21
ii	dpkg	1.6.15
ii	e2fsprogs	1.18-3.0
ii	elvis-tiny	1.4-11
ii	fbset	2.1-6
ii	fdflush	1.0.1-5
ii	fdutils	5.3-3
ii	fileutils	4.1-2
ii	findutils	4.1-40
ii	ftp	0.10-3.1
ii	gettext-base	0.10.35-13
ii	grep	2.4.2-1
ii	gzip	1.2.4-33
ii	hostname	2.07
ii	isapnptools	1.21-2
ii	joe	2.8-15.2
ii	klogd	1.4.1-2
ii	ldso	1.9.11-9
ii	libbz2-1.0	1.0.1-10
ii	libc6	2.2.3-6
ii	libdb2	2.7.7-8
ii	libdbd-mysql-p	1.2216-2
ii	libdbi-perl	1.18-1
ii	libexpat1	1.95.1-5
ii	libgdbmg1	1.7.3-27
ii	libmm11	1.1.3-4
ii	libmysqlclient	3.23.39-3
ii	libncurses5	5.2.20010318-2
ii	libnewt0	0.50-7
ii	libpam-modules	0.72-9
ii	libpam-runtime	0.72-9
ii	libpam0g	0.72-9
ii	libpcap0	0.6.2-1
ii	libpcre3	3.4-1
ii	libpopt0	1.4-1.1
ii	libreadline4	4.2-3
ii	libssl09	0.9.4-5
ii	libstdc++2.10	2.95.2-13

ii	libstdc++2.10-	2.95.4-0.01070
ii	libwrap0	7.6-4
ii	lilo	21.4.3-2
ii	locales	2.1.3-18
ii	login	19990827-20
ii	makedev	2.3.1-46.2
ii	mawk	1.3.3-5
ii	mbr	1.1.2-1
ii	mime-support	3.11-1
ii	modutils	2.3.11-13.1
ii	mount	2.10f-5.1
ii	mysql-client	3.23.39-3
ii	mysql-common	3.23.39-3.1
ii	mysql-server	3.23.39-3
ii	ncurses-base	5.0-6.0potato1
ii	ncurses-bin	5.0-6.0potato1
ii	netbase	3.18-4
ii	passwd	19990827-20
ii	pciutils	2.1.2-2
ii	perl	5.6.1-5
ii	perl-base	5.6.1-5
ii	perl-modules	5.6.1-5
ii	php4	4.0.6-4
ii	php4-mysql	4.0.6-4
ii	ppp	2.3.11-1.4
ii	pppconfig	2.0.5
ii	procps	2.0.6-5
ii	psmisc	19-2
ii	pump	0.7.3-2
ii	sed	3.02-5
ii	setserial	2.17-16
ii	shellutils	2.0-7
ii	slang1	1.3.9-1
ii	snort	1.7-9
ii	ssh	1.2.3-9.3
ii	sysklogd	1.4.1-2
ii	syslinux	1.48-2
ii	sysvinit	2.78-4
ii	tar	1.13.17-2
ii	tasksel	1.0-10
ii	tcpd	7.6-4
ii	telnet	0.16-4potato.1
ii	textutils	2.0-2
ii	update	2.11-1
ii	util-linux	2.10f-5.1
ii	zlib1g	1.1.3-15

Appendix D

Setting up a bridge firewall

This information was contributed by Francois Bayart in order to help users set up a Linux bridge/firewall with the 2.4.x kernel and iptables. The only features needed are the bridge firewall patch, available at sourceforge download page (<http://bridge.sourceforge.net/download.html>).

For example, if you are using a 2.4.18 kernel, first install the kernel source package, then download the patch (<http://bridge.sourceforge.net/devel/bridge-nf/bridge-nf-0.0.6-against-2.4.18.diff>) and apply it, as shown in this example:

```
Zipowz:/usr/src# apt-get install kernel-source-2.4.18
Zipowz:/usr/src# cd kernel-source-2.4.18
Zipowz:/usr/src/kernel-source-2.4.18# patch -p1 < ../bridge-nf-0.0.6-against-
patching file include/linux/netfilter.h
patching file include/linux/netfilter_ipv4.h
patching file include/linux/skbuff.h
patching file net/bridge/br.c
patching file net/bridge/br_forward.c
patching file net/bridge/br_input.c
patching file net/bridge/br_netfilter.c
patching file net/bridge/br_private.h
patching file net/bridge/Makefile
patching file net/Config.in
patching file net/core/netfilter.c
patching file net/core/skbuff.c
patching file net/ipv4/ip_output.c
patching file net/ipv4/netfilter/ip_tables.c
patching file net/ipv4/netfilter/ipt_LOG.c
```

Now, configure the kernel (i.e., `make menuconfig`, `make xconfig`). In the section *Networking options*, enable the following options:

```

[*] Network packet filtering (replaces ipchains)
[ ] Network packet filtering debugging (NEW)
<*> 802.1d Ethernet Bridging
[*] netfilter (firewalling) support (NEW)

```

Caution: you must disable this if you want to apply some firewalling rules or else iptables will not work.

```
[ ] Network packet filtering debugging (NEW)
```

Next, add the correct options in the section *IP: Netfilter Configuration*. Then, compile and install the kernel. If you want to do it the *Debian way*, install `kernel-package` and run `make-kpkg` to create a custom Debian kernel package you can install on your server using `dpkg`. Once the new kernel is compiled and installed, install the `bridge-utils` package.

Once these steps are complete, you can complete the configuration of your bridge. The next section presents two different possible configurations for the bridge, each with a hypothetical network map and the necessary commands.

D.1 A bridge providing NAT and firewall capabilities

The first configuration uses the bridge as a firewall with network address translation (NAT) that protects a server and internal LAN clients. A diagram of the network configuration is shown below:

```

Internet ---- router ( 62.3.3.25 ) ---- bridge (62.3.3.26 gw 62.3.3.25 / 192.
|
|----- WWW Server (62.3.3.27 gw 62.3
|
LAN --- Zipowz (192.168.0.2 gw 192.1

```

The following commands show how this bridge can be configured.

```

# Create the interface br0
/usr/sbin/brctl addbr br0

# Add the Ethernet interface to use with the bridge
/usr/sbin/brctl addif br0 eth0
/usr/sbin/brctl addif br0 eth1

# Start up the Ethernet interface

```

```

/sbin/ifconfig eth0 0.0.0.0
/sbin/ifconfig eth1 0.0.0.0

# Configure the bridge ethernet
# The bridge will be correct and invisible ( transparent firewall ).
# It's hidden in a traceroute and you keep your real gateway on the
# other computers. Now if you want you can config a gateway on your
# bridge and choose it as your new gateway for the other computers.

/sbin/ifconfig br0 62.3.3.26 netmask 255.255.255.248 broadcast 62.3.3.32

# I have added this internal IP to create my NAT
ip addr add 192.168.0.1/24 dev br0
/sbin/route add default gw 62.3.3.25

```

D.2 A bridge providing firewall capabilities

A second possible configuration is a system that is set up as a transparent firewall for a LAN with a public IP address space.

```

Internet ---- router (62.3.3.25) ---- bridge (62.3.3.26)
                                     |
                                     |---- WWW Server (62.3.3.28 gw 62.3.3.26)
                                     |
                                     |---- Mail Server (62.3.3.27 gw 62.3.3.26)

```

The following commands show how this bridge can be configured.

```

# Create the interface br0
/usr/sbin/brctl addbr br0

# Add the Ethernet interface to use with the bridge
/usr/sbin/brctl addif br0 eth0
/usr/sbin/brctl addif br0 eth1

# Start up the Ethernet interface
/sbin/ifconfig eth0 0.0.0.0
/sbin/ifconfig eth1 0.0.0.0

# Configure the bridge Ethernet
# The bridge will be correct and invisible ( transparent firewall ).

```

```
# It's hidden in a traceroute and you keep your real gateway on the
# other computers. Now if you want you can config a gateway on your
# bridge and choose it as your new gateway for the other computers.

/sbin/ifconfig br0 62.3.3.26 netmask 255.255.255.248 broadcast 62.3.3.32
```

If you traceroute the Linux Mail Server, you won't see the bridge. If you want access to the bridge with ssh, you must have a gateway or you must first connect to another server, such as the "Mail Server", and then connect to the bridge through the internal network card.

D.3 Basic IPtables rules

This is an example of the basic rules that could be used for either of these setups.

```
iptables -F FORWARD
iptables -P FORWARD DROP
iptables -A FORWARD -s 0.0.0.0/0.0.0.0 -d 0.0.0.0/0.0.0.0 -m state --state
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

# Some funny rules but not in a classic IPtables sorry ...
# Limit ICMP
# iptables -A FORWARD -p icmp -m limit --limit 4/s -j ACCEPT
# Match string, a good simple method to block some VIRUS very quickly
# iptables -I FORWARD -j DROP -p tcp -s 0.0.0.0/0 -m string --string "cmd.e

# Block all MySQL connection just to be sure
iptables -A FORWARD -p tcp -s 0/0 -d 62.3.3.0/24 --dport 3306 -j DROP

# Linux Mail Server Rules

# Allow FTP-DATA ( 20 ) , FTP ( 21 ) , SSH ( 22 )
iptables -A FORWARD -p tcp -s 0.0.0.0/0 -d 62.3.3.27/32 --dport 20:22 -j ACCEP

# Allow the Mail Server to connect to the outside
# Note: This is *not* needed for the previous connections
# (remember: stateful filtering) and could be removed.
iptables -A FORWARD -p tcp -s 62.3.3.27/32 -d 0/0 -j ACCEPT

# WWW Server Rules

# Allow HTTP ( 80 ) connections with the WWW server
iptables -A FORWARD -p tcp -s 0.0.0.0/0 -d 62.3.3.28/32 --dport 80 -j ACCEP

# Allow HTTPS ( 443 ) connections with the WWW server
```

```
iptables -A FORWARD -p tcp -s 0.0.0.0/0 -d 62.3.3.28/32 --dport 443 -j ACCE  
  
# Allow the WWW server to go out  
# Note: This is *not* needed for the previous connections  
# (remember: stateful filtering) and could be removed.  
iptables -A FORWARD -p tcp -s 62.3.3.28/32 -d 0/0 -j ACCEPT
```


Appendix E

Sample script to change the default Bind installation.

This script automates the procedure for changing the bind name server's default installation so that it does *not* run as the superuser. It will create the user and groups to be used for the name server. Use with extreme care since it has not been tested thoroughly.

```
#!/bin/sh
# Change the default Debian bind configuration to have it run
# with a non-root user and group.
#
# WARN: This script has not been tested thoroughly, please
# verify the changes made to the INITD script

# (c) 2002 Javier Fernandez-Sanguino Peña
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 1, or (at your option)
# any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# Please see the file 'COPYING' for the complete copyright notice.
#

restore() {
# Just in case, restore the system if the changes fail
echo "WARN: Restoring to the previous setup since I'm unable to properly c
```

```
    echo "WARN: Please check the $INITDERR script."
    mv $INITD $INITDERR
    cp $INITDBAK $INITD
}

USER=named
GROUP=named
INITD=/etc/init.d/bind
INITDBAK=$INITD.preuserchange
INITDERR=$INITD.changeerror
START="start-stop-daemon --start --quiet --exec /usr/sbin/named -- -g $GROU
AWKS="awk ' /start-stop-daemon --start/ { print \"\$START\"; noprint = 1; };"

[ `id -u` -ne 0 ] && {
    echo "This program must be run by the root user"
    exit 1
}

RUNUSER=`ps -eo user,fname |grep named |cut -f 1 -d " "`

if [ "$RUNUSER" = "$USER" ]
then
    echo "WARN: The name server running daemon is already running as $USER"
    echo "ERR:  This script will not many any changes to your setup."
    exit 1
fi
if [ ! -f $INITD ]
then
    echo "ERR:  This system does not have $INITD (which this script tries to c
RUNNING=`ps -eo fname |grep named`
    [ -z "$RUNNING" ] && \
        echo "ERR:  In fact the name server daemon is not even running (is it
    echo "ERR:  No changes will be made to your system"
    exit 1
fi

# Check if named group exists
if [ -z "`grep $GROUP /etc/group`" ]
then
    echo "Creating group $GROUP:"
    addgroup $GROUP
else
    echo "WARN: Group $GROUP already exists. Will not create it"
fi
# Same for the user
```

```
if [ -z "`grep $USER /etc/passwd`" ]
then
  echo "Creating user $USER:"
  adduser --system --home /home/$USER \
  --no-create-home --ingroup $GROUP \
  --disabled-password --disabled-login $USER
else
  echo "WARN: The user $USER already exists. Will not create it"
fi

# Change the init.d script

# First make a backup (check that there is not already
# one there first)
if [ ! -f $INITDBAK ]
then
  cp $INITD $INITDBAK
fi

# Then use it to change it
cat $INITDBAK |
eval $AWKS > $INITD

echo "WARN: The script $INITD has been changed, trying to test the changes."
echo "Restarting the named daemon (check for errors here)."
$INITD restart
if [ $? -ne 0 ]
then
  echo "ERR: Failed to restart the daemon."
  restore
  exit 1
fi

RUNNING=`ps -eo fname |grep named`
if [ -z "$RUNNING" ]
then
  echo "ERR: Named is not running, probably due to a problem with the chang
  restore
  exit 1
fi

# Check if it's running as expected
RUNUSER=`ps -eo user,fname |grep named |cut -f 1 -d " "`

if [ "$RUNUSER" = "$USER" ]
```

```
then
  echo "All has gone well, named seems to be running now as $USER."
else
  echo "ERR: The script failed to automatically change the system."
  echo "ERR: Named is currently running as $RUNUSER."
  restore
  exit 1
fi

exit 0
```

The previous script, run on Woody's (Debian 3.0) custom bind, will produce the following initd file after creating the 'named' user and group:

```
#!/bin/sh

PATH=/sbin:/bin:/usr/sbin:/usr/bin

test -x /usr/sbin/named || exit 0

start () {
  echo -n "Starting domain name service: named"
  start-stop-daemon --start --quiet \
    --pidfile /var/run/named.pid --exec /usr/sbin/named
  echo "."
}

stop () {
  echo -n "Stopping domain name service: named"
  # --exec doesn't catch daemons running deleted instances of named,
  # as in an upgrade. Fortunately, --pidfile is only going to hit
  # things from the pidfile.
  start-stop-daemon --stop --quiet \
    --pidfile /var/run/named.pid --name named
  echo "."
}

case "$1" in
  start)
    ;;

  stop)
    ;;

  *)
    ;;
esac
```

```
        restart|force-reload)
stop
sleep 2
start
    ;;

        reload)
/usr/sbin/ndc reload
    ;;

*)
echo "Usage: /etc/init.d/bind {start|stop|reload|restart|force-reload}" >&
exit 1
    ;;
esac

exit 0
```


Appendix F

Security update protected by a firewall

After a standard installation, a system may still have some security vulnerabilities. Unless you can download updates for the vulnerable packages on another system (or you have mirrored security.debian.org for local use), the system will have to be connected to the Internet for the downloads.

However, as soon as you connect to the Internet you are exposing this system. If one of your local services is vulnerable, you might be compromised even before the update is finished! This may seem paranoid but, in fact, analysis from the HoneyNet Project (<http://www.honeynet.org>) has shown that systems can be compromised in less than three days, even if the system is not publicly known (i.e., not published in DNS records).

When doing an update on a system not protected by an external system like a firewall, it is possible to properly configure your local firewall to restrict connections involving only the security update itself. The example below shows how to set up such local firewall capabilities, which allow connections from security.debian.org only, logging all others.

FIXME: add IP address for security.debian.org (since otherwise you need DNS up to work) on /etc/hosts.

FIXME: test this setup to see if it works properly

FIXME: this will only work with HTTP URLs since ftp might need the ip_conntrack_ftp module, or use passive mode.

```
# iptables -F
# iptables -L
Chain INPUT (policy ACCEPT)
target      prot opt source                destination

Chain FORWARD (policy ACCEPT)
target      prot opt source                destination

Chain OUTPUT (policy ACCEPT)
```

```
target      prot opt source                destination
# iptables -P INPUT DROP
# iptables -P FORWARD DROP
# iptables -P OUTPUT DROP
# iptables -A OUTPUT -d security.debian.org -p 80 -j ACCEPT
# iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A INPUT -p icmp -j ACCEPT
# iptables -A INPUT -j LOG
# iptables -A OUTPUT -j LOG
# iptables -L
Chain INPUT (policy DROP)
target      prot opt source                destination
ACCEPT     all  --  0.0.0.0/0             0.0.0.0/0          state RELATED,E
ACCEPT     icmp --  0.0.0.0/0             0.0.0.0/0
LOG        all  --  anywhere              anywhere            LOG level warni

Chain FORWARD (policy DROP)
target      prot opt source                destination

Chain OUTPUT (policy DROP)
target      prot opt source                destination
ACCEPT     80  --  anywhere              security.debian.org
LOG        all  --  anywhere              anywhere            LOG level warni
```

Appendix G

Chroot environment for SSH

Creating a restricted environment for SSH is a tough job due to its dependencies and the fact that, unlike other servers, SSH provides a remote shell to users. Thus, you will also have to consider the applications users will be allowed to use in the environment. If you create this file structure in, for example `/var/chroot/ssh`, you could start the `ssh` server chroot'ed with this command:

```
# chroot /var/chroot/ssh /sbin/sshd -f /etc/sshd_config
```

G.1 Automatically making the environment (the easy way)

You can easily create a restricted environment with the `makejail` package, since it automatically takes care of tracing the server daemon (with `strace`), and makes it run under the restricted environment.

The advantage of programs that automatically generate chroot environments is that they are capable of copying any package to the chroot environment (even following the package's dependencies and making sure it's complete). Thus, providing user applications is easier.

To set up the environment using `makejail`'s provided examples, just use the command:

```
# makejail /usr/share/doc/examples/bind.py
```

Read the sample file to see what other changes need to be made to the environment. Some of these changes, such as copying user's home directories, cannot be done automatically. Also, limit the exposure of sensitive information by only copying the data from a given number of users from the files `/etc/shadow` or `/etc/group`.

The following sample environment has been (slightly) tested and is built with the configuration file provided in the package and includes the `fileutils` package:

```
.
|-- bin
|   |-- ash
|   |-- bash
|   |-- chgrp
|   |-- chmod
|   |-- chown
|   |-- cp
|   |-- csh -> /etc/alternatives/csh
|   |-- dd
|   |-- df
|   |-- dir
|   |-- fdflush
|   |-- ksh
|   |-- ln
|   |-- ls
|   |-- mkdir
|   |-- mknod
|   |-- mv
|   |-- rbash -> bash
|   |-- rm
|   |-- rmdir
|   |-- sh -> bash
|   |-- sync
|   |-- tcsh
|   |-- touch
|   |-- vdir
|   |-- zsh -> /etc/alternatives/zsh
|   `-- zsh4
|-- dev
|   |-- null
|   |-- ptmx
|   |-- pts
|   |-- ptya0
(... )
|   |-- tty
|   |-- tty0
(... )
|   `-- urandom
|-- etc
|   |-- alternatives
|   |   |-- csh -> /bin/tcsh
|   |   `-- zsh -> /bin/zsh4
|   |-- environment
|   |-- hosts
|   |-- hosts.allow
```

```
|
|  |-- hosts.deny
|  |-- ld.so.conf
|  |-- localtime -> /usr/share/zoneinfo/Europe/Madrid
|  |-- motd
|  |-- nsswitch.conf
|  |-- pam.conf
|  |-- pam.d
|  |   |-- other
|  |   `-- ssh
|  |-- passwd
|  |-- resolv.conf
|  |-- security
|  |   |-- access.conf
|  |   |-- chroot.conf
|  |   |-- group.conf
|  |   |-- limits.conf
|  |   |-- pam_env.conf
|  |   `-- time.conf
|  |-- shadow
|  |-- shells
|  `-- ssh
|     |-- moduli
|     |-- ssh_host_dsa_key
|     |-- ssh_host_dsa_key.pub
|     |-- ssh_host_rsa_key
|     |-- ssh_host_rsa_key.pub
|     `-- sshd_config
-- home
  `-- userX
-- lib
  |-- ld-2.2.5.so
  |-- ld-linux.so.2 -> ld-2.2.5.so
  |-- libc-2.2.5.so
  |-- libc.so.6 -> libc-2.2.5.so
  |-- libcap.so.1 -> libcap.so.1.10
  |-- libcap.so.1.10
  |-- libcrypt-2.2.5.so
  |-- libcrypt.so.1 -> libcrypt-2.2.5.so
  |-- libdl-2.2.5.so
  |-- libdl.so.2 -> libdl-2.2.5.so
  |-- libm-2.2.5.so
  |-- libm.so.6 -> libm-2.2.5.so
  |-- libncurses.so.5 -> libncurses.so.5.2
  |-- libncurses.so.5.2
  |-- libnsl-2.2.5.so
  |-- libnsl.so.1 -> libnsl-2.2.5.so
```

```
|-- libnss_compat-2.2.5.so
|-- libnss_compat.so.2 -> libnss_compat-2.2.5.so
|-- libnss_db-2.2.so
|-- libnss_db.so.2 -> libnss_db-2.2.so
|-- libnss_dns-2.2.5.so
|-- libnss_dns.so.2 -> libnss_dns-2.2.5.so
|-- libnss_files-2.2.5.so
|-- libnss_files.so.2 -> libnss_files-2.2.5.so
|-- libnss_hesiod-2.2.5.so
|-- libnss_hesiod.so.2 -> libnss_hesiod-2.2.5.so
|-- libnss_nis-2.2.5.so
|-- libnss_nis.so.2 -> libnss_nis-2.2.5.so
|-- libnss_nisplus-2.2.5.so
|-- libnss_nisplus.so.2 -> libnss_nisplus-2.2.5.so
|-- libpam.so.0 -> libpam.so.0.72
|-- libpam.so.0.72
|-- libpthread-0.9.so
|-- libpthread.so.0 -> libpthread-0.9.so
|-- libresolv-2.2.5.so
|-- libresolv.so.2 -> libresolv-2.2.5.so
|-- librt-2.2.5.so
|-- librt.so.1 -> librt-2.2.5.so
|-- libutil-2.2.5.so
|-- libutil.so.1 -> libutil-2.2.5.so
|-- libwrap.so.0 -> libwrap.so.0.7.6
|-- libwrap.so.0.7.6
`-- security
    |-- pam_access.so
    |-- pam_chroot.so
    |-- pam_deny.so
    |-- pam_env.so
    |-- pam_filter.so
    |-- pam_ftp.so
    |-- pam_group.so
    |-- pam_issue.so
    |-- pam_lastlog.so
    |-- pam_limits.so
    |-- pam_listfile.so
    |-- pam_mail.so
    |-- pam_mkhomedir.so
    |-- pam_motd.so
    |-- pam_nologin.so
    |-- pam_permit.so
    |-- pam_rhosts_auth.so
    |-- pam_rootok.so
    |-- pam_securetty.so
```

```
|      |-- pam_shells.so
|      |-- pam_stress.so
|      |-- pam_tally.so
|      |-- pam_time.so
|      |-- pam_unix.so
|      |-- pam_unix_acct.so -> pam_unix.so
|      |-- pam_unix_auth.so -> pam_unix.so
|      |-- pam_unix_passwd.so -> pam_unix.so
|      |-- pam_unix_session.so -> pam_unix.so
|      |-- pam_userdb.so
|      |-- pam_warn.so
|      `-- pam_wheel.so
-- sbin
  `-- start-stop-daemon
-- usr
  |-- bin
  |   |-- dircolors
  |   |-- du
  |   |-- install
  |   |-- link
  |   |-- mkfifo
  |   |-- shred
  |   |-- touch -> /bin/touch
  |   `-- unlink
  |-- lib
  |   |-- libcrypto.so.0.9.6
  |   |-- libdb3.so.3 -> libdb3.so.3.0.2
  |   |-- libdb3.so.3.0.2
  |   |-- libz.so.1 -> libz.so.1.1.4
  |   `-- libz.so.1.1.4
  |-- sbin
  |   `-- sshd
  `-- share
      |-- locale
      |   `-- es
      |       |-- LC_MESSAGES
      |       |   |-- fileutils.mo
      |       |   |-- libc.mo
      |       |   `-- sh-utils.mo
      |       `-- LC_TIME -> LC_MESSAGES
      `-- zoneinfo
          `-- Europe
              `-- Madrid
-- var
  `-- run
      |-- sshd
```

```
`-- sshd.pid
```

```
27 directories, 733 files
```

G.2 Patching SSH to enable chroot functionality

Debian's `sshd` does not allow restriction of a user's movement through the server, since it lacks the `chroot` function that the commercial program `sshd2` includes (using 'ChrootGroups' or 'ChrootUsers', see `sshd2_config(5)`). However, there is a patch available to add this functionality available from Bug report 139047 (<http://bugs.debian.org/139047>) or <http://www.cag.lcs.mit.edu/~raoul/>. The patch may be included in future releases of the OpenSSH package. Emanuel Lacour has `ssh` deb packages with this feature at <http://debian.home-dn.net/woody/ssh/>. Completing the compilation step is recommended, though.

A description of all the necessary steps can be found at <http://mail.incredimail.com/howto/openssh/> (though it is aimed at RedHat 7.2 users, almost all of them are applicable to Debian). After applying the patch, modify `/etc/passwd` by changing the home path of the users (with the special `./` token):

```
joeuser:x:1099:1099:Joe Random User:/home/joe/./:/bin/bash
```

This will restrict *both* remote shell access, as well as remote copy through the `ssh` channel.

Make sure to have all the needed binaries and libraries in the `chroot`'ed path for users. These files should be owned by `root` to avoid tampering by the user (so as to exit the `chroot`'ed jailed). A sample might include:

```
./bin:
total 660
drwxr-xr-x    2 root    root          4096 Mar 18 13:36 .
drwxr-xr-x    8 guest   guest          4096 Mar 15 16:53 ..
-r-xr-xr-x    1 root    root        531160 Feb  6 22:36 bash
-r-xr-xr-x    1 root    root         43916 Nov 29 13:19 ls
-r-xr-xr-x    1 root    root        16684 Nov 29 13:19 mkdir
-rwxr-xr-x    1 root    root        23960 Mar 18 13:36 more
-r-xr-xr-x    1 root    root         9916 Jul 26  2001 pwd
-r-xr-xr-x    1 root    root        24780 Nov 29 13:19 rm
lrwxrwxrwx    1 root    root           4 Mar 30 16:29 sh -> bash

./etc:
total 24
drwxr-xr-x    2 root    root          4096 Mar 15 16:13 .
drwxr-xr-x    8 guest   guest          4096 Mar 15 16:53 ..
```

```

-rw-r--r--    1 root    root           54 Mar 15 13:23 group
-rw-r--r--    1 root    root          428 Mar 15 15:56 hosts
-rw-r--r--    1 root    root           44 Mar 15 15:53 passwd
-rw-r--r--    1 root    root           52 Mar 15 13:23 shells

./lib:
total 1848
drwxr-xr-x    2 root    root          4096 Mar 18 13:37 .
drwxr-xr-x    8 guest   guest          4096 Mar 15 16:53 ..
-rwxr-xr-x    1 root    root         92511 Mar 15 12:49 ld-linux.so.2
-rwxr-xr-x    1 root    root        1170812 Mar 15 12:49 libc.so.6
-rw-r--r--    1 root    root         20900 Mar 15 13:01 libcrypt.so.1
-rw-r--r--    1 root    root          9436 Mar 15 12:49 libdl.so.2
-rw-r--r--    1 root    root        248132 Mar 15 12:48 libncurses.so.5
-rw-r--r--    1 root    root         71332 Mar 15 13:00 libnsl.so.1
-rw-r--r--    1 root    root         34144 Mar 15 16:10
libnss_files.so.2
-rw-r--r--    1 root    root         29420 Mar 15 12:57 libpam.so.0
-rw-r--r--    1 root    root        105498 Mar 15 12:51 libpthread.so.0
-rw-r--r--    1 root    root         25596 Mar 15 12:51 librt.so.1
-rw-r--r--    1 root    root          7760 Mar 15 12:59 libutil.so.1
-rw-r--r--    1 root    root         24328 Mar 15 12:57 libwrap.so.0

./usr:
total 16
drwxr-xr-x    4 root    root          4096 Mar 15 13:00 .
drwxr-xr-x    8 guest   guest          4096 Mar 15 16:53 ..
drwxr-xr-x    2 root    root          4096 Mar 15 15:55 bin
drwxr-xr-x    2 root    root          4096 Mar 15 15:37 lib

./usr/bin:
total 340
drwxr-xr-x    2 root    root          4096 Mar 15 15:55 .
drwxr-xr-x    4 root    root          4096 Mar 15 13:00 ..
-rwxr-xr-x    1 root    root        10332 Mar 15 15:55 env
-rwxr-xr-x    1 root    root        13052 Mar 15 13:13 id
-r-xr-xr-x    1 root    root        25432 Mar 15 12:40 scp
-rwxr-xr-x    1 root    root        43768 Mar 15 15:15 sftp
-r-sr-xr-x    1 root    root       218456 Mar 15 12:40 ssh
-rwxr-xr-x    1 root    root         9692 Mar 15 13:17 tty

./usr/lib:
total 852
drwxr-xr-x    2 root    root          4096 Mar 15 15:37 .
drwxr-xr-x    4 root    root          4096 Mar 15 13:00 ..
-rw-r--r--    1 root    root       771088 Mar 15 13:01

```

```

libcrypto.so.0.9.6
-rw-r--r--    1 root    root    54548 Mar 15 13:00 libz.so.1
-rwxr-xr-x    1 root    root    23096 Mar 15 15:37 sftp-server

```

G.3 Handmade environment (the hard way)

It is possible to create an environment, using a trial-and-error method, by monitoring the `sshd` server traces and log files in order to determine the necessary files. The following environment, contributed by José Luis Ledesma, is a sample listing of files in a `chroot` environment for `ssh`:

1

```

.:
total 36
drwxr-xr-x  9 root root 4096 Jun 5 10:05 ./
drwxr-xr-x 11 root root 4096 Jun 3 13:43 ../
drwxr-xr-x  2 root root 4096 Jun 4 12:13 bin/
drwxr-xr-x  2 root root 4096 Jun 4 12:16 dev/
drwxr-xr-x  4 root root 4096 Jun 4 12:35 etc/
drwxr-xr-x  3 root root 4096 Jun 4 12:13 lib/
drwxr-xr-x  2 root root 4096 Jun 4 12:35 sbin/
drwxr-xr-x  2 root root 4096 Jun 4 12:32 tmp/
drwxr-xr-x  2 root root 4096 Jun 4 12:16 usr/
./bin:
total 8368
drwxr-xr-x  2 root root 4096 Jun 4 12:13 ./
drwxr-xr-x  9 root root 4096 Jun 5 10:05 ../
-rwxr-xr-x  1 root root 109855 Jun 3 13:45 a2p*
-rwxr-xr-x  1 root root 387764 Jun 3 13:45 bash*
-rwxr-xr-x  1 root root 36365 Jun 3 13:45 c2ph*
-rwxr-xr-x  1 root root 20629 Jun 3 13:45 dprofpp*
-rwxr-xr-x  1 root root 6956 Jun 3 13:46 env*
-rwxr-xr-x  1 root root 158116 Jun 3 13:45 fax2ps*
-rwxr-xr-x  1 root root 104008 Jun 3 13:45 faxalter*
-rwxr-xr-x  1 root root 89340 Jun 3 13:45 faxcover*
-rwxr-xr-x  1 root root 441584 Jun 3 13:45 faxmail*
-rwxr-xr-x  1 root root 96036 Jun 3 13:45 faxrm*
-rwxr-xr-x  1 root root 107000 Jun 3 13:45 faxstat*
-rwxr-xr-x  1 root root 77832 Jun 4 11:46 grep*
-rwxr-xr-x  1 root root 19597 Jun 3 13:45 h2ph*
-rwxr-xr-x  1 root root 46979 Jun 3 13:45 h2xs*
-rwxr-xr-x  1 root root 10420 Jun 3 13:46 id*

```

¹Notice that there are no SETUID files. This makes it more difficult for remote users to escape the `chroot` environment. However, it also prevents users from changing their passwords, since the `passwd` program cannot modify the files `/etc/passwd` or `/etc/shadow`.

```
-rwxr-xr-x 1 root root 4528 Jun 3 13:46 ldd*
-rwxr-xr-x 1 root root 111386 Jun 4 11:46 less*
-r-xr-xr-x 1 root root 26168 Jun 3 13:45 login*
-rwxr-xr-x 1 root root 49164 Jun 3 13:45 ls*
-rwxr-xr-x 1 root root 11600 Jun 3 13:45 mkdir*
-rwxr-xr-x 1 root root 24780 Jun 3 13:45 more*
-rwxr-xr-x 1 root root 154980 Jun 3 13:45 pal2rgb*
-rwxr-xr-x 1 root root 27920 Jun 3 13:46 passwd*
-rwxr-xr-x 1 root root 4241 Jun 3 13:45 pl2pm*
-rwxr-xr-x 1 root root 2350 Jun 3 13:45 pod2html*
-rwxr-xr-x 1 root root 7875 Jun 3 13:45 pod2latex*
-rwxr-xr-x 1 root root 17587 Jun 3 13:45 pod2man*
-rwxr-xr-x 1 root root 6877 Jun 3 13:45 pod2text*
-rwxr-xr-x 1 root root 3300 Jun 3 13:45 pod2usage*
-rwxr-xr-x 1 root root 3341 Jun 3 13:45 podchecker*
-rwxr-xr-x 1 root root 2483 Jun 3 13:45 podselect*
-r-xr-xr-x 1 root root 82412 Jun 4 11:46 ps*
-rwxr-xr-x 1 root root 36365 Jun 3 13:45 pstruct*
-rwxr-xr-x 1 root root 7120 Jun 3 13:45 pwd*
-rwxr-xr-x 1 root root 179884 Jun 3 13:45 rgb2ycbcr*
-rwxr-xr-x 1 root root 20532 Jun 3 13:45 rm*
-rwxr-xr-x 1 root root 6720 Jun 4 10:15 rmdir*
-rwxr-xr-x 1 root root 14705 Jun 3 13:45 s2p*
-rwxr-xr-x 1 root root 28764 Jun 3 13:46 scp*
-rwxr-xr-x 1 root root 385000 Jun 3 13:45 sendfax*
-rwxr-xr-x 1 root root 67548 Jun 3 13:45 sendpage*
-rwxr-xr-x 1 root root 88632 Jun 3 13:46 sftp*
-rwxr-xr-x 1 root root 387764 Jun 3 13:45 sh*
-rws--x--x 1 root root 744500 Jun 3 13:46 slogin*
-rwxr-xr-x 1 root root 14523 Jun 3 13:46 splain*
-rws--x--x 1 root root 744500 Jun 3 13:46 ssh*
-rwxr-xr-x 1 root root 570960 Jun 3 13:46 ssh-add*
-rwxr-xr-x 1 root root 502952 Jun 3 13:46 ssh-agent*
-rwxr-xr-x 1 root root 575740 Jun 3 13:46 ssh-keygen*
-rwxr-xr-x 1 root root 383480 Jun 3 13:46 ssh-keyscan*
-rwxr-xr-x 1 root root 39 Jun 3 13:46 ssh_europa*
-rwxr-xr-x 1 root root 107252 Jun 4 10:14 strace*
-rwxr-xr-x 1 root root 8323 Jun 4 10:14 strace-graph*
-rwxr-xr-x 1 root root 158088 Jun 3 13:46 thumbnail*
-rwxr-xr-x 1 root root 6312 Jun 3 13:46 tty*
-rwxr-xr-x 1 root root 55904 Jun 4 11:46 useradd*
-rwxr-xr-x 1 root root 585656 Jun 4 11:47 vi*
-rwxr-xr-x 1 root root 6444 Jun 4 11:45 whoami*
./dev:
total 8
drwxr-xr-x 2 root root 4096 Jun 4 12:16 ./
```

```
drwxr-xr-x 9 root root 4096 Jun 5 10:05 ../
crw-r--r-- 1 root root 1, 9 Jun 3 13:43 urandom
./etc:
total 208
drwxr-xr-x 4 root root 4096 Jun 4 12:35 ./
drwxr-xr-x 9 root root 4096 Jun 5 10:05 ../
-rw----- 1 root root 0 Jun 4 11:46 .pwd.lock
-rw-r--r-- 1 root root 653 Jun 3 13:46 group
-rw-r--r-- 1 root root 242 Jun 4 11:33 host.conf
-rw-r--r-- 1 root root 857 Jun 4 12:04 hosts
-rw-r--r-- 1 root root 1050 Jun 4 11:29 ld.so.cache
-rw-r--r-- 1 root root 304 Jun 4 11:28 ld.so.conf
-rw-r--r-- 1 root root 235 Jun 4 11:27 ld.so.conf~
-rw-r--r-- 1 root root 88039 Jun 3 13:46 moduli
-rw-r--r-- 1 root root 1342 Jun 4 11:34 nsswitch.conf
drwxr-xr-x 2 root root 4096 Jun 4 12:02 pam.d/
-rw-r--r-- 1 root root 28 Jun 4 12:00 pam_smb.conf
-rw-r--r-- 1 root root 2520 Jun 4 11:57 passwd
-rw-r--r-- 1 root root 7228 Jun 3 13:48 profile
-rw-r--r-- 1 root root 1339 Jun 4 11:33 protocols
-rw-r--r-- 1 root root 274 Jun 4 11:44 resolv.conf
drwxr-xr-x 2 root root 4096 Jun 3 13:43 security/
-rw-r----- 1 root root 1178 Jun 4 11:51 shadow
-rw----- 1 root root 80 Jun 4 11:45 shadow-
-rw-r----- 1 root root 1178 Jun 4 11:48 shadow.old
-rw-r--r-- 1 root root 161 Jun 3 13:46 shells
-rw-r--r-- 1 root root 1144 Jun 3 13:46 ssh_config
-rw----- 1 root root 668 Jun 3 13:46 ssh_host_dsa_key
-rw-r--r-- 1 root root 602 Jun 3 13:46 ssh_host_dsa_key.pub
-rw----- 1 root root 527 Jun 3 13:46 ssh_host_key
-rw-r--r-- 1 root root 331 Jun 3 13:46 ssh_host_key.pub
-rw----- 1 root root 883 Jun 3 13:46 ssh_host_rsa_key
-rw-r--r-- 1 root root 222 Jun 3 13:46 ssh_host_rsa_key.pub
-rw-r--r-- 1 root root 2471 Jun 4 12:15 sshd_config
./etc/pam.d:
total 24
drwxr-xr-x 2 root root 4096 Jun 4 12:02 ./
drwxr-xr-x 4 root root 4096 Jun 4 12:35 ../
lrwxrwxrwx 1 root root 4 Jun 4 12:02 other -> sshd
-rw-r--r-- 1 root root 318 Jun 3 13:46 passwd
-rw-r--r-- 1 root root 546 Jun 4 11:36 ssh
-rw-r--r-- 1 root root 479 Jun 4 12:02 sshd
-rw-r--r-- 1 root root 370 Jun 3 13:46 su
./etc/security:
total 32
drwxr-xr-x 2 root root 4096 Jun 3 13:43 ./
```

```

drwxr-xr-x 4 root root 4096 Jun 4 12:35 ../
-rw-r--r-- 1 root root 1971 Jun 3 13:46 access.conf
-rw-r--r-- 1 root root 184 Jun 3 13:46 chroot.conf
-rw-r--r-- 1 root root 2145 Jun 3 13:46 group.conf
-rw-r--r-- 1 root root 1356 Jun 3 13:46 limits.conf
-rw-r--r-- 1 root root 2858 Jun 3 13:46 pam_env.conf
-rw-r--r-- 1 root root 2154 Jun 3 13:46 time.conf
./lib:
total 8316
drwxr-xr-x 3 root root 4096 Jun 4 12:13 ./
drwxr-xr-x 9 root root 4096 Jun 5 10:05 ../
-rw-r--r-- 1 root root 1024 Jun 4 11:51 cracklib_dict.hwm
-rw-r--r-- 1 root root 214324 Jun 4 11:51 cracklib_dict.pwd
-rw-r--r-- 1 root root 11360 Jun 4 11:51 cracklib_dict.pwi
-rwxr-xr-x 1 root root 342427 Jun 3 13:46 ld-linux.so.2*
-rwxr-xr-x 1 root root 4061504 Jun 3 13:46 libc.so.6*
lrwxrwxrwx 1 root root 15 Jun 4 12:11 libcrack.so -> libcrack.so.2.7*
lrwxrwxrwx 1 root root 15 Jun 4 12:11 libcrack.so.2 -> libcrack.so.2.7*
-rwxr-xr-x 1 root root 33291 Jun 4 11:39 libcrack.so.2.7*
-rwxr-xr-x 1 root root 60988 Jun 3 13:46 libcrypt.so.1*
-rwxr-xr-x 1 root root 71846 Jun 3 13:46 libdl.so.2*
-rwxr-xr-x 1 root root 27762 Jun 3 13:46 libhistory.so.4.0*
lrwxrwxrwx 1 root root 17 Jun 4 12:12 libncurses.so.4 -> libncurses.so.4.2*
-rwxr-xr-x 1 root root 503903 Jun 3 13:46 libncurses.so.4.2*
lrwxrwxrwx 1 root root 17 Jun 4 12:12 libncurses.so.5 -> libncurses.so.5.0*
-rwxr-xr-x 1 root root 549429 Jun 3 13:46 libncurses.so.5.0*
-rwxr-xr-x 1 root root 369801 Jun 3 13:46 libnsl.so.1*
-rwxr-xr-x 1 root root 142563 Jun 4 11:49 libnss_compat.so.1*
-rwxr-xr-x 1 root root 215569 Jun 4 11:49 libnss_compat.so.2*
-rwxr-xr-x 1 root root 61648 Jun 4 11:34 libnss_dns.so.1*
-rwxr-xr-x 1 root root 63453 Jun 4 11:34 libnss_dns.so.2*
-rwxr-xr-x 1 root root 63782 Jun 4 11:34 libnss_dns6.so.2*
-rwxr-xr-x 1 root root 205715 Jun 3 13:46 libnss_files.so.1*
-rwxr-xr-x 1 root root 235932 Jun 3 13:49 libnss_files.so.2*
-rwxr-xr-x 1 root root 204383 Jun 4 11:33 libnss_nis.so.1*
-rwxr-xr-x 1 root root 254023 Jun 4 11:33 libnss_nis.so.2*
-rwxr-xr-x 1 root root 256465 Jun 4 11:33 libnss_nisplus.so.2*
lrwxrwxrwx 1 root root 14 Jun 4 12:12 libpam.so.0 -> libpam.so.0.72*
-rwxr-xr-x 1 root root 31449 Jun 3 13:46 libpam.so.0.72*
lrwxrwxrwx 1 root root 19 Jun 4 12:12 libpam_misc.so.0 ->
libpam_misc.so.0.72*
-rwxr-xr-x 1 root root 8125 Jun 3 13:46 libpam_misc.so.0.72*
lrwxrwxrwx 1 root root 15 Jun 4 12:12 libpamc.so.0 -> libpamc.so.0.72*
-rwxr-xr-x 1 root root 10499 Jun 3 13:46 libpamc.so.0.72*
-rwxr-xr-x 1 root root 176427 Jun 3 13:46 libreadline.so.4.0*
-rwxr-xr-x 1 root root 44729 Jun 3 13:46 libutil.so.1*

```

```
-rwxr-xr-x 1 root root 70254 Jun 3 13:46 libz.a*
lrwxrwxrwx 1 root root 13 Jun 4 12:13 libz.so -> libz.so.1.1.3*
lrwxrwxrwx 1 root root 13 Jun 4 12:13 libz.so.1 -> libz.so.1.1.3*
-rwxr-xr-x 1 root root 63312 Jun 3 13:46 libz.so.1.1.3*
drwxr-xr-x 2 root root 4096 Jun 4 12:00 security/
./lib/security:
total 668
drwxr-xr-x 2 root root 4096 Jun 4 12:00 ./
drwxr-xr-x 3 root root 4096 Jun 4 12:13 ../
-rwxr-xr-x 1 root root 10067 Jun 3 13:46 pam_access.so*
-rwxr-xr-x 1 root root 8300 Jun 3 13:46 pam_chroot.so*
-rwxr-xr-x 1 root root 14397 Jun 3 13:46 pam_cracklib.so*
-rwxr-xr-x 1 root root 5082 Jun 3 13:46 pam_deny.so*
-rwxr-xr-x 1 root root 13153 Jun 3 13:46 pam_env.so*
-rwxr-xr-x 1 root root 13371 Jun 3 13:46 pam_filter.so*
-rwxr-xr-x 1 root root 7957 Jun 3 13:46 pam_ftp.so*
-rwxr-xr-x 1 root root 12771 Jun 3 13:46 pam_group.so*
-rwxr-xr-x 1 root root 10174 Jun 3 13:46 pam_issue.so*
-rwxr-xr-x 1 root root 9774 Jun 3 13:46 pam_lastlog.so*
-rwxr-xr-x 1 root root 13591 Jun 3 13:46 pam_limits.so*
-rwxr-xr-x 1 root root 11268 Jun 3 13:46 pam_listfile.so*
-rwxr-xr-x 1 root root 11182 Jun 3 13:46 pam_mail.so*
-rwxr-xr-x 1 root root 5923 Jun 3 13:46 pam_nologin.so*
-rwxr-xr-x 1 root root 5460 Jun 3 13:46 pam_permit.so*
-rwxr-xr-x 1 root root 18226 Jun 3 13:46 pam_pwcheck.so*
-rwxr-xr-x 1 root root 12590 Jun 3 13:46 pam_rhosts_auth.so*
-rwxr-xr-x 1 root root 5551 Jun 3 13:46 pam_rootok.so*
-rwxr-xr-x 1 root root 7239 Jun 3 13:46 pam_securetty.so*
-rwxr-xr-x 1 root root 6551 Jun 3 13:46 pam_shells.so*
-rwxr-xr-x 1 root root 55925 Jun 4 12:00 pam_smb_auth.so*
-rwxr-xr-x 1 root root 12678 Jun 3 13:46 pam_stress.so*
-rwxr-xr-x 1 root root 11170 Jun 3 13:46 pam_tally.so*
-rwxr-xr-x 1 root root 11124 Jun 3 13:46 pam_time.so*
-rwxr-xr-x 1 root root 45703 Jun 3 13:46 pam_unix.so*
-rwxr-xr-x 1 root root 45703 Jun 3 13:46 pam_unix2.so*
-rwxr-xr-x 1 root root 45386 Jun 3 13:46 pam_unix_acct.so*
-rwxr-xr-x 1 root root 45386 Jun 3 13:46 pam_unix_auth.so*
-rwxr-xr-x 1 root root 45386 Jun 3 13:46 pam_unix_passwd.so*
-rwxr-xr-x 1 root root 45386 Jun 3 13:46 pam_unix_session.so*
-rwxr-xr-x 1 root root 9726 Jun 3 13:46 pam_userdb.so*
-rwxr-xr-x 1 root root 6424 Jun 3 13:46 pam_warn.so*
-rwxr-xr-x 1 root root 7460 Jun 3 13:46 pam_wheel.so*
./sbin:
total 3132
drwxr-xr-x 2 root root 4096 Jun 4 12:35 ./
drwxr-xr-x 9 root root 4096 Jun 5 10:05 ../
```

```
-rwxr-xr-x 1 root root 178256 Jun 3 13:46 choptest*
-rwxr-xr-x 1 root root 184032 Jun 3 13:46 cqttest*
-rwxr-xr-x 1 root root 81096 Jun 3 13:46 dialtest*
-rwxr-xr-x 1 root root 1142128 Jun 4 11:28 ldconfig*
-rwxr-xr-x 1 root root 2868 Jun 3 13:46 lockname*
-rwxr-xr-x 1 root root 3340 Jun 3 13:46 ondelay*
-rwxr-xr-x 1 root root 376796 Jun 3 13:46 pagesend*
-rwxr-xr-x 1 root root 13950 Jun 3 13:46 probemodem*
-rwxr-xr-x 1 root root 9234 Jun 3 13:46 recvstats*
-rwxr-xr-x 1 root root 64480 Jun 3 13:46 sftp-server*
-rwxr-xr-x 1 root root 744412 Jun 3 13:46 sshd*
-rwxr-xr-x 1 root root 30750 Jun 4 11:46 su*
-rwxr-xr-x 1 root root 194632 Jun 3 13:46 tagtest*
-rwxr-xr-x 1 root root 69892 Jun 3 13:46 tsitest*
-rwxr-xr-x 1 root root 43792 Jun 3 13:46 typetest*
./tmp:
total 8
drwxr-xr-x 2 root root 4096 Jun 4 12:32 ./
drwxr-xr-x 9 root root 4096 Jun 5 10:05 ../
./usr:
total 8
drwxr-xr-x 2 root root 4096 Jun 4 12:16 ./
drwxr-xr-x 9 root root 4096 Jun 5 10:05 ../
lrwxrwxrwx 1 root root 7 Jun 4 12:14 bin -> ../bin//
lrwxrwxrwx 1 root root 7 Jun 4 11:33 lib -> ../lib//
lrwxrwxrwx 1 root root 8 Jun 4 12:13 sbin -> ../sbin//
```