

Securing a Unix Server

MSc Thesis in Information Security
Royal Holloway, University of London

Kapetanakis Ioannis
bilias@edu.physics.uoc.gr

August 27, 2003

*To my grandfather,
Ματθαίο Χρηστάκη*

Abstract

The aim of this project is concentrated on producing a helpful guide on how to protect and secure a networked Unix server. It can be used as a security manual by System Administrators. It covers general and more specific aspects of Unix and network security. Recommendations are also given on protecting services like DNS, Mail and Web. Particular examples of the specific commands and configuration options are presented and analyzed after their integrity has been tested and verified. Great consideration was given so that they can be clearly followed. The whole implementation was based on Linux and open source software.

Acknowledgments

I would like to thank a number of people who helped me complete this thesis. Firstly my supervisor, Jason Crampton, for his guidance and advice throughout this period of study.

A great thanks to Ifiyenia, Mixalis and Olga (in a random order) for their patience, help and also their support during this summer. Without them the result would have been much different.

Thanks to Dimitris and Sotiris for providing me with ideas and technical recommendations not only now, but over the last seven years.

Finally a big thanks to my family for their love and support.

Conventions Used

The following conventions are used in this document:

Constant Width

Used for Unix command, user, group and program names.
It is also used for entries in configuration files.

Constant Width Italic

Used for URL and IP addresses as well as for computer and network names.

Sans-Serif

Used for Unix file, directory and filesystem names.
It is also used for system output such as logged data.

Italic

Used to emphasize new terms and concepts when they are introduced.

Command 0.1 An example of Commands

```
server# make bzImage
server$ lynx http://www.kernel.org
client$ ssh server -l admin
```

`server#`

Command is executed by the `root` user on the *server*.

`server$`

Command is executed by a non-privileged user on the *server*.

`client$`

Command is executed by a non-privileged user on the *client*.

```
root:x:0:0:root:/home/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
wwwadm:x:49:49:./usr/local/www:/bin/false
```

File 0.1: Example entries in a configuration File

```
Aug 11 05:52:38 client kernel: LIDS: sshd (dev 22:6 inode 50428)
pid 692 ppid 645 uid/gid (0/0) on (null tty) : access hidden file
/etc/shadow - logging disabled for (60)s
Aug 11 05:52:38 client sshd(pam_unix)[692]: check pass; user unknown
Aug 11 05:52:40 client sshd[692]: Failed password for user from 10.0.0.1
port 32777 ssh2
```

Output 0.1: An example Output from the system's log files

server

The hostname of the computer which is used as a server.

client

The hostname of the computer which is used as a client.

example.com

A domain name used only for the purposes of this document.

10.0.0.1

The IP address of *server*.

10.0.0.10

The IP address of *client*.

10.0.0.0/255.255.255.0

A class C network address that defines the network.

It includes the *10.0.0.0 - 10.0.0.255* IP address range.

Contents

1	Introduction	1
2	Linux Security	3
2.1	Choosing a Linux distribution	3
2.2	RedHat Installation	4
2.2.1	Several filesystems	4
2.2.2	Enabling security features	4
2.2.3	Do not install unnecessary software	6
2.3	Main Security	6
2.3.1	User accounts	6
2.3.2	Strong user passwords	7
2.3.3	File access permissions	9
2.3.4	The root user	10
2.3.5	Dangerous SUID and SGID files	11
2.3.6	Disabling unnecessary services	13
2.3.7	Securing X Windows	14
2.3.8	Limiting system's resources	14
2.3.9	SSH installation	15
2.3.10	System logging	18
2.3.11	TCP Wrappers	20
2.4	Advanced Security Features	21
2.4.1	Process accounting	21
2.4.2	File attributes	22
2.4.3	Setting boot options	23
2.4.4	Kernel options	25
2.4.5	Linux Intrusion Detection System (LIDS)	26
2.5	Conclusion	27
3	Certification Authority for Internal Use	31
3.1	Public Key Cryptography	31
3.2	Digital Signatures	31
3.3	Digital Certificates	33
3.4	Certification Authorities	33
3.5	Root Certification Authority Creation	34
3.6	Non-root CA Creation	35
3.7	Creating and Signing Certificates	36
3.8	Conclusion	38

4	DNS Server Security	39
4.1	BIND Installation	40
4.2	BIND Configuration	40
4.2.1	rndc.conf	40
4.2.2	Zone files	41
4.2.3	named.conf	42
4.2.4	Security logging	46
4.2.5	Permissions	46
4.3	Server Control	47
4.4	Testing Access Control	48
4.5	TSIG	49
4.6	DNSSEC	50
4.7	Conclusion	51
5	Mail Server Security	53
5.1	Sendmail Installation	53
5.2	Sendmail Configuration	55
5.2.1	sendmail.mc	55
5.2.2	Accept emails for specific hosts only	60
5.2.3	Restricting relay access	60
5.3	Access Control	61
5.4	Resource Limits	61
5.5	Fighting Spam Mail and Viruses	63
5.5.1	Internal mechanisms	63
5.5.2	External mechanisms for spam mail	64
5.5.3	External mechanisms for viruses	65
5.6	E-mail Gateway	67
5.7	SMTP Authentication	68
5.8	STARTTLS	69
5.9	Sendmail Alternatives	69
6	Web Server Security	71
6.1	Apache Installation	71
6.2	Secure Web Site Preparation	72
6.3	Apache Configuration	73
6.4	User Authentication	76
6.4.1	Basic authentication	76
6.4.2	Digest authentication	77
6.5	SSL/TLS Secure Web	79
6.5.1	SSL client authentication	81
6.6	Conclusion	82

7	Firewalls	85
7.1	Different Types of Firewalls	85
7.2	Firewall Policy Configuration	86
7.2.1	Setting the default policy rules	86
7.2.2	Define traffic policy	87
7.3	Conclusion	92
8	Intrusion Detection	95
8.1	Intrusion Detection Techniques	95
8.2	Ad Hoc Intrusion Detection	96
8.2.1	System log files	96
8.2.2	Detection of invalid login records	96
8.2.3	Detection of invalid processes	97
8.2.4	Detection of invalid network services	98
8.2.5	Detection of invalid SUID files	99
8.2.6	Integrity verification	100
8.2.7	<i>Spying</i> on your users	100
8.2.8	Additional traces	101
8.3	Automated Host-based Intrusion Detection	102
8.3.1	Integrity verification	102
8.4	Network-based Intrusion Detection	105
8.5	Conclusion	106
9	Final Considerations	109
9.1	Further improvements	109
	Appendices	113
A	DNS	113
A.1	Zones files	113
A.2	DNS startup script	115
B	Configuration Files	117
B.1	Virus definitions in Sendmail	117
B.2	Certificate extensions in Openssl	118
C	Scripts	119
C.1	Enabling NAT	119
C.2	ICMP filtering	119
	Glossary	121

References	125
------------	-----

List of Figures

1	Signing and verifying a signature of a message	32
2	A hierarchical CA scheme	38
3	DNS hierarchy	39
4	E-mail gateway	67

List of Files

0.1	Example entries in a configuration File	ix
2.1	<code>/etc/pam.d/system-auth</code>	8
2.2	Password aging in <code>/etc/login.defs</code>	9
2.3	Placing resource limits in <code>/etc/security/limits.conf</code>	15
2.4	SSH configuration in <code>/etc/ssh/sshd_config</code>	17
2.5	System logging configuration in <code>/etc/syslog.conf</code>	18
2.6	A tool for <code>syslogd</code> manipulation	19
2.7	Allow access to specific hosts only	21
2.8	<code>/etc/inittab</code>	24
2.9	Protecting the boot-loader	25
2.10	Kernel hardening	29
4.1	<code>rndc.conf</code>	41
4.2	<code>named.conf</code> part 1	42
4.3	<code>named.conf</code> part 2	43
4.4	DNS security logging	46
4.5	TSIG for secure zone transfers, <code>named.conf</code>	51
5.1	<code>site.config.m4</code>	53
5.2	<code>sendmail.mc</code> part 1	56
5.3	<code>sendmail.mc</code> part 2	58
5.4	Access control for connections to mail server	62
5.5	<code>MailScanner.conf</code>	65
5.6	Mail routing with <code>mailertable</code>	68
6.1	Apache configuration file <code>httpd.conf</code> part 1	73
6.2	Apache configuration file <code>httpd.conf</code> part 2	75
6.3	Web client basic authentication, <code>httpd.conf</code> part 3	78
6.4	Web client digest authentication, <code>httpd.conf</code> part 4	78
6.5	Secure web site configuration, <code>ssl.conf</code> part 1	80
6.6	Web SSL client authentication, <code>ssl.conf</code> part 2	82

6.7	Deny normal access to a web page	82
A.1	0.0.10.in-addr.arpa zone file	113
A.2	example.com zone file	114
A.3	DNS startup script part 1	115
A.4	DNS startup script part 2	116
B.1	sendmail.mc part 3	117
B.2	openssl.cnf	118

List of Commands

0.1	An example of Commands	ix
2.1	Find world-writable files	10
2.2	Applying the sticky bit to world-writable directories	10
2.3	Detection of public SUID and SGID binaries	12
2.4	Locating and disabling services that automatically start	13
2.5	Restricting access to <code>crond</code> and <code>atd</code>	14
2.6	SSH installation	16
2.7	Enabling automatic startup of SSH server	16
2.8	Firewall entry for a log-server	20
2.9	Enabling process accounting	22
2.10	Hack the system from the console	24
3.1	Creation of CA storage space	34
3.2	Creation of self signed certificate for the root CA	34
3.3	Creation of a non-root CA	36
3.4	Signing the certificate request of the non-root CA	36
3.5	Reading a certificate or a private key	36
3.6	Creation of a certification request	37
3.7	Reading and signing a server certification request	37
3.8	Reading and signing a user certification request	37
3.9	Finding the purpose of a certificate	37
3.10	Revoking a certificate and creating a revocation list	38
4.1	BIND installation	40
4.2	Key generation for server control	41
4.3	Version discovery	44
4.4	File permissions	47
4.5	Server startup	47
4.6	Zone transfer attempt	48
4.7	Attempt for recursive queries	49
4.8	Nodes in <code>blackhole</code> ACL never get response back	50
4.9	Key generation for secure host authentication	50

5.1	Sendmail Installation	54
5.2	sendmail.cf creation	60
5.3	Database files creation	61
5.4	Installing MailScanner	64
6.1	Apache installation	71
6.2	Enabling support for specific modules only	72
6.3	Secure web site preparation	72
6.4	Creating new users for basic web authentication	77
6.5	Creating new users for digest web authentication	78
6.6	Put certificates in place	79
6.7	Starting an SSL Apache server	80
6.8	Creating a PKCS#12 certificate	81
7.1	Setting the default rules	87
7.2	Control the loopback interface	88
7.3	SYN-flood protection	88
7.4	Drop XMAS and NULL TCP port scans	88
7.5	Extensive protection against SYN-flood and furtive port scanners	89
7.6	Spoofing protection	89
7.7	Block unwanted broadcast	89
7.8	Block identd requests	90
7.9	Protect X-resources	90
7.10	Allow DNS	90
7.11	Allow Web for everyone	90
7.12	Allow Mail for everyone	90
7.13	Allow SSH from admin's host only	91
7.14	Allow internal network to access the portmapper	91
7.15	Allow valid traffic	91
7.16	Ping of Death protection	91
7.17	Block rest of the traffic	91
7.18	Creating and using the policy file	92
8.1	Creation of login records	97
8.2	Listing of current processes	97
8.3	Listing of open sockets and ports	98
8.4	Reporting RPC and mount information	98
8.5	Detection of SUID and SGID files on the system	99
8.6	Verifying the integrity of RPM software packages	100
8.7	Detecting device files and unowned files	101
8.8	Tripwire installation	103
8.9	Tripwire configuration	103
8.10	Tripwire initialization	104

8.11	Integrity verification with Tripwire	104
8.12	Running automated integrity check	104
8.13	Updating Tripwire's database	104
8.14	Updating Tripwire's policy file	105
8.15	Hardening file permissions for Tripwire	105
A.1	Retrieving the latest list of root DNS servers	113
A.2	Enabling a startup script	117
C.1	Enabling NAT on firewall gateway	119
C.2	Allow some incoming ICMP traffic	119
C.3	Allow some outgoing ICMP traffic	119
C.4	Block unwanted ICMP traffic	119

List of Outputs

0.1	An example Output from the system's log files	x
2.1	TCPD refusing or allowing a connection	20
3.1	Calculating the hash value of a message	31
3.2	Creation of root certificate	35
4.1	Attacker trying to control the server	48
4.2	Denying access to zone transfers	49
4.3	Denying access to recursive queries	49
5.1	File permissions for mail	55
8.1	Listing logged users	96
8.2	TCP SYN port scan on the server	99

List of Tables

1	Recommended filesystems	5
2	Dangerous SUID/SGID files	12
3	List of minimum enabled services	13
4	Disable network connections in X server and XFS	14
5	Security attributes	23

1 Introduction

Networks are being used to a great extent by a large number of people. Banks, governmental authorities, enterprises, universities and home users rely on networks such as the Internet to transfer sensitive information, confidential data or exchange personal details for E-commerce services. Security should be a main concern, as it is a vital factor for any kind of transaction.

Threats are generally frequent in this area due to the rapid technology changes. Computers can be accessed by anyone while the number of hackers increases constantly. Sophisticated tools which provide automated features and do not require any special abilities, can be easily obtained and are in fact being used by attackers. Detailed information on how vulnerabilities of a system can be detected and exploited is actually available.

Common targets include the server itself as well as major services like the Web, the Mail and the DNS. The platform that has been certified as an industry standard for such services is the UNIX Operating System (OS) which is literally the Internet's backbone.

The examples concerning the security of an OS like Unix that are going to be presented in this thesis, have been implemented on a free version of Unix which is called Linux. Linux was created in 1991 by Linus Torvalds, a computer science student. It consists of the Linux's kernel (<http://www.kernel.org>) and other freely distributed tools, mainly available by the GNU project (<http://www.gnu.org>). It is a fast, stable multiuser OS which is suitable for high-availability servers or even home workstations. Furthermore, over the last few years it is under a constant development and use by the entire computer industry.

A spherical approach of security is required in order for Unix to be adequately protected. The primary aim is the security of the operating system and the network, followed by commonly used servers like the Web, the Mail and the DNS. For that reason both prevention and intrusion detection mechanisms have been implemented in great detail.

Regarding the manner these security measures are going to be presented, we plan to use clear examples of configuration files and commands, displayed in distinct tables. The main focus of this thesis falls into the specific procedures that should be followed by a system administrator, in order to protect his server effectively against possible attacks. We have concentrated our efforts on presenting the appropriate methods-actions that should take place in a productive server which is administered by advanced users.

2 Linux Security

Linux is configured by default to run in multiple environments. It can be used by servers that need 24-7 availability or home users that need a large number of user friendly applications. Creating an OS that meets such different requirements is not only hard to achieve, but it generates a lot of security considerations as well.

The default installation includes software that is not necessary, depending on the purpose the system is used for. Home users have different requirements for security than productive servers which are always connected to an untrusted network such as the Internet. Furthermore, the main servers included in the default installation (DNS, Mail and Web server) are configured for running on different environments, thus supporting multiple features, instead of being optimized for security.

Our main objective in the following sections is to present a more secure solution for a server using the Linux OS. Note that most of these features can be used in other Unix systems as well. The installation procedure, the main security requirements and also the advanced security features will all be analyzed in the following pages.

2.1 Choosing a Linux distribution

Linux is supported by many different distributions. All of them use the same kernel and the same GNU utilities. The main difference between the available distributions is the support for user utilities, graphical interfaces, automated configuration utilities and other goodies. The most important part in choosing a distribution should be the customer support options available by each company, their response to bugs and vulnerabilities and the amount of fine documentation offered with the operating system.

However, whatever distribution you chose to install is insignificant, if the system is going to be optimized for security. Only the basic operating system should be installed, which is more or less the same on every distribution. Furthermore, most of the software provided, is created by the *open source* community, which follows an autonomous process for updating and fixing their software.

RedHat v8, which is commonly used by the Linux community, was used as a platform for implementing the security measures that are presented. Despite its “age” it has not yet lost its ability for upgrading. Either way a lot of the included software was removed. For what it was left, updated software versions were installed including the most recent version of the stable kernels (kernel-2.4.21). A second, perhaps even better, choice in a real production

environment would be the Slackware Linux. Its quality is that it sustains its minimalistic character thus making it easier to secure it.

Patches for defective or vulnerable software should always be installed by the administrator. RedHat's updates are located in <http://www.redhat.com/errata>. Subscribing to the security-announce mailing list is also recommended as the security reports and patches are announced there. The same actions also apply to all the types of the Linux OS.

2.2 RedHat Installation

The configuration of the system starts at the point when you decide to install it. Various options can be defined at the installation part of an operating system. Some of those have to do with the security of the system.

A possible configuration is presented in the following Sections. Security has been the first criterion for any decision making. Choosing the type of the system is the first task the installation procedure requires. The system should be configured as a **Custom** one and not as a **Server** or a **Workstation** one. Both **Server** and **Workstation** systems include predefined configuration options which are far from the level of security we are seeking to reach. **Upgrading** an existing system is also not recommended as it might affect the stability of the new system.

2.2.1 Several filesystems

A server must have several disk partitions in order for files to be distinctly separated. In Unix, partitions are called filesystems. The most important of those is the **root** filesystem (/) which is the first filesystem *mounted* when the system boots. This should hold only what is necessary for the system to start. The kernel*, the system's configuration files as well as the most important binaries and libraries, are stored in the **root** filesystem. The rest of the filesystems are mounted under the **root** filesystem. A recommended setup is presented in Table 1.

2.2.2 Enabling security features

Further security features can be configured in the installation procedure. This part can be skipped as it is possible for an administrator to enable

*The kernel is also frequently stored in a different filesystem in Linux which is called `/boot`.

Filesystems	Size MB	Filesystem's Purpose
/	$\simeq 160$	The core of Unix OS
swap	≥ 512	Memory paging space
/boot	$\simeq 16$	Stores the kernel
/tmp	$\simeq 128$	Stores temporary data
/usr	$> 3,000$	Installation programs
/var	≥ 256	Log files, audit trails
/usr/src/misc	$\simeq 1,000$	Compile new programs
/usr/local	$\simeq 2,000$	Install new programs
/usr/local/www	\sim	Storing web site

Table 1: Recommended filesystems

or modify them after the end of the installation. Nevertheless, if they are required it is preferred to be configured at that moment.

A boot-loader is required that will be used to boot the system. There is a choice between Grub and Lilo. Lilo is considered to be less secure than Grub, as multiple vulnerabilities have been discovered in the past for it. Both of them can protect the boot process with a password. This way if an attacker has access to the system's console, he will not be able to manipulate the system's startup options [Section 2.4.3]. Using a strong password is highly recommended [Section 2.3.2].

The system asks if a firewall is needed to protect it against network attacks and remote intruders. The administrator should enable the firewall. However, the firewall's rules should be revised after the system is configured, as new services will have been added at that time. New options must be added in order to have a more complete protection which satisfies the server's needs. Firewall configurations are presented in Section 7.

The system's `root` password is inserted while the system is being installed. Bear in mind that this password is important for the security of the system. A temporary or a permanent password can be chosen. Nevertheless, it must be a strong one in order for the `root` account and the whole system to be protected [Section 2.3].

The next two options will definitely improve the security of the user accounts. These are the Shadow Password Suite and the MD5 algorithm. Shadowed passwords are stored in a separate location than other user information which is readable by anyone on the system. This makes it harder for an intruder to gain access to the encrypted* passwords. Even if the attacker

*Unix passwords are often called encrypted although they are not. A hash function

manages to obtain the file with the passwords, these will be hashed with the MD5 algorithm which is a strong one; at least more secure than the alternative of the `crypt(3)` function. MD5 produces a 2^{128} bit value, while `crypt` offers only 2^{56} bits.

2.2.3 Do not install unnecessary software

Custom installation offers the administrator the ability to choose which software will be installed in the system. Unnecessary software should never be installed in systems where security is the primary goal. The more software a server has, the easier it is for an attacker to gain access to the system.

The administrator should either install only what is necessary for its server to operate, or he can choose to remove installed software that is not needed after the installation has been completed [3]. Bear in mind that custom installation of the DNS, the Mail and the Web server are extensively covered in the next Sections.

2.3 Main Security

2.3.1 User accounts

Unix/Linux is a multiuser operating system meaning that a large number of users* can simultaneously use the system's resources. In order for the system to differentiate users and their privileges, user accounts must be created.

Each user is assigned a personal account which is uniquely identified by a username and a user-id (UID). The username has no meaning to the operating system as the OS separates users by the UID number, which must be different for each user. Each account also consists of a password, a group-id (GID), a home directory and a shell. The GID is a number identifying the primary group each user belongs to. The home directory is the disk space, where the user can store his data. The shell is a command language interpreter used to execute commands.

Computer system's security is primarily based on account protection. Each user must be assigned with a different username, a different user-id and a different home location. Unix stores this information in the file `/etc/passwd` which must be readable by everyone logged in the system.

transforms the password to a *digest* or the *hash value* of the password.

*Individuals using a computer are called users.

2.3.2 Strong user passwords

The front line of account protection is the password, which must be known only to the owner of each account. Each user is requested to insert his username and the corresponding password whenever authentication in the system is required. The password must remain secret, otherwise attackers *will* use it to compromise the security of the system or violate user's privacy.

Furthermore, the password must be strong [2], meaning that guessing it is highly difficult. Passwords that are difficult to guess share the following characteristics*:

- Have both uppercase and lowercase letters
- Have digits and/or punctuation characters as well as letters
- May include some control characters and/or spaces
- Can be typed quickly
- Are at least eight characters long
- Are easy to remember, so they do not have to be written down
- DO NOT include your name, your username or parts of them
- DO NOT include your boy/girlfriend's name
- DO NOT include place names or your favourite band's name
- DO NOT include any name
- DO NOT include phone numbers or license plate numbers
- DO NOT include letters or numbers only
- DO NOT include anybody's birth date or any other date in any format
- DO NOT include information easily obtained about you
- DO NOT include words from an English dictionary or a foreign one
- DO NOT include simple patterns on the keyboard like 123qwe or 1q0o2w9i
- DO NOT include common words like **sex**, **police**, **money** or **television**

*Some of the characteristics were taken from [1].

- DO NOT include any of the above spelled backwards
- DO NOT include any of the above followed or prepended by a single digit

The system can be configured to require strong user passwords when these are initiated. This is defined in file `/etc/pam.d/system-auth` [File 2.1]. Using these entries, when a user tries to change his password to a weaker one, his action will be rejected. Furthermore if an account does not have a password, the user will not be able to login.

```
# User changes will be destroyed next time authconfig is run.
auth      required    /lib/security/pam_env.so
auth      sufficient  /lib/security/pam_unix.so likeauth
auth      required    /lib/security/pam_deny.so
account   required    /lib/security/pam_unix.so
password  required    /lib/security/pam_cracklib.so retry=3
           difok=3 type=Unix minlen=12

password  sufficient  /lib/security/pam_unix.so use_authtok
nulloke  md5 shadow remember=10

password  required    /lib/security/pam_deny.so
session   required    /lib/security/pam_limits.so
session   required    /lib/security/pam_unix.so
```

File 2.1: `/etc/pam.d/system-auth`

Additionally, an old password must not be reused, as there is always the possibility that it has been successfully guessed or eavesdropped. We configured `/etc/pam.d/system-auth` to keep track of the last 10 passwords each user had. These are stored in file `/etc/security/opasswd`, which must already exist, otherwise the user will be able to use old passwords.

Passwords should also change frequently. Users must be enforced to do so by the system. The password aging configuration is in file `/etc/login.defs`. Minimum password length is also defined in this configuration file [File 2.2].

In order for passwords to be better protected, the system does not keep cleartext versions of them. On the contrary, encrypted versions are stored which are being used when identity verification and authentication is required. Older versions of Unix OS used to store the encrypted passwords in `/etc/passwd`. However, this is not secure as anyone who has access to the

```
# Password aging controls:
#
#PASS_MAX_DAYS Maximum number of days a password may be used.
#PASS_MIN_DAYS Minimum number of days between password changes.
#PASS_MIN_LEN Minimum acceptable password length.
#PASS_WARN_AGE Number of days warning given before a password
#expires.
PASS_MAX_DAYS 100
PASS_MIN_DAYS 1
PASS_MIN_LEN 8
PASS_WARN_AGE 7
```

File 2.2: Password aging in `/etc/login.defs`

system can read that file and decrypt the passwords. Linux, among others, supports *shadow* passwords. These passwords are stored in a different file, `/etc/shadow`, which is not readable by non-privileged users. This is the reason we enabled shadow password during the installation of the OS.

Apart from the enforcement of strong password by automated procedures, the administrator should frequently run password cracking programs to detect weak passwords. Such programs are Crack and John the Ripper.

2.3.3 File access permissions

Unix treats everything in the system as a file. User's data, programs, devices like the hard disk or the CD-Rom, even the physical memory are represented by a file. According to the type of access that should be allowed on a file, only the appropriate file permissions should be granted. The more restrictive those permissions are, the better it is for the security of the system.

The basic file and directory permissions are `read (-r--)`, `write (--w-)` and `execute (---x)`. These can be set using `chmod(1)*` command. These access permissions are explicitly set for the owner and the group the file belongs to as well as anyone in the system that is not the owner or a member of the group (owner, group, other). By using combination of those permissions, access can be restricted to a limited set of allowed actions [4]. A sensitive file can be made non-readable at all or readable by a specific group of people as long as this is required.

*When a command appears with a number inside parentheses, it means that there exists a manual page for this specific command. The reader can execute `man 1 chmod` or `man chmod` for more details about the `chmod` command.

Furthermore, modification of configuration files must be restricted only to the administrator of the system. Access to commands and programs, that should not be executed by unauthorized users, can be denied to anyone who is not a member of the authorized group of people. File permissions and file ownership will be extensively used in the following sections where critical services are being secured.

When files and directories are created default permissions are applied. These are controlled by the `umask` value. Common values are `077` which grants permissions only to the owner and `022` which grants all permissions to the owner as well as a read-only access to the group and others. The default `umask` for all users can be set in the startup files of the login shells: `/etc/bashrc` for `bash` and `/etc/csh.cshrc` for `csh` and `tcsh`.

The system should also be searched for dangerous files like world-writable files. These files can be modified or deleted by anyone. Command 2.1 must be executed in order for these files to be listed. These type of write access must be removed unless the file does not contain any sensitive data at all.

Command 2.1 Find world-writable files

```
server# find /. -perm -2 ! -type l -print
```

Nevertheless, there are two directories in the system that should be writable by anyone. These are the `/tmp/` and `/var/tmp/`. Most of the programs store temporary data there and everyone must be able to create files inside these directories. However, a user must not be allowed to delete files owned by other users. A special permission, which is called the “sticky bit”, must be set in these directories*. If this is set [Command 2.2] users will not be able to delete files they do not own or files for which they do not have write permission granted.

Command 2.2 Applying the sticky bit to world-writable directories

```
server# chmod 1777 /tmp/
server# chmod 1777 /var/tmp/
```

2.3.4 The root user

Unix/Linux is configured with a special account that uses the `root` username. The UID of this account is always set to `0`, granting administrator’s privileges on the system. The `root` or the super-user can delete any file in the

*Linux has the sticky bit enabled in `/tmp/`. Other operating systems like HP-UX, are not protected by default and the administrator should execute Command 2.2.

system, enable or disable any services, create or delete other user accounts and generally speaking, any action can be performed by this powerful user. What `root` user cannot do is unmount filesystems which are being used or make a program listen for connections on a port being used by another. Nevertheless, he is able to stop (`kill(1)`) any process that occupies a filesystem or has been binded to a specific port.

This account must be protected by a very strong password which will be frequently changed. This password must only be known to a very limited group of people, the administrators of the system. Their role is the installation, configuration and maintenance of the system. None of the administrators should be able to directly login as `root` in the system, at least from a remote location. This helps in auditing each administrator's actions. Furthermore, it is more difficult for an attacker to bypass two user accounts than one.

When a certain action that requires `root` access must be performed, the administrator should use the `su(1)` utility to enhance his privileges. As soon as `root` privileges are no more required, the administrator should logout from that account.

2.3.5 Dangerous SUID and SGID files

Apart from the basic file permissions, Unix has implemented the SUID and SGID permission bits. When a user executes a program that has the SUID bit enabled, it is as if it has been executed by the owner of that program. If the SGID bit is set then the privileges of the group that the program belongs to are granted to the user.

SUID and SGID files owned by user `root` can be very dangerous*. Historically, most of vulnerabilities in the Unix OS, have been exploited because of such files. Nevertheless, there are tasks that require `root` privileges, which must be performed by normal users. Such an example is the execution of the `passwd` command, which allows users to change their passwords. These passwords are stored in `/etc/shadow` which is a file that cannot be read or modified by anyone except `root`. That is why `passwd` is made SUID and owned by `root`.

However, Linux comes with a large number of `root` SUID files. Most of them are used for administrative purposes and by default they can be executed by anyone in the system. For security reasons the administrator should detect all the SUID and SGID files installed [Command 2.3].

Most of these files should either be removed or modified to be available for

*These SUID files grant the privileges of `root` user. SGID files grant privileges of the `root` group.

Command 2.3 Detection of public SUID and SGID binaries

```
server# foreach i in ( /bin /sbin /usr/bin /usr/sbin \
/usr/X11R6/bin /usr/local/bin /usr/local/sbin )
foreach? find $i/. -type f -user root -perm -4001 -ls
foreach? find $i/. -type f -group root -perm -2001 -ls
foreach? end
```

execution only by a specific group of users. A listing of such files is presented in Table 2. Bear in mind that if custom installation of the operating system

/bin/ping	/usr/bin/rcp
/bin/mount	/usr/bin/rlogin
/bin/umount	/usr/bin/rsh
/sbin/netreport	/usr/bin/remsh
/usr/bin/suidperl	/usr/bin/sperl5.6.1
/usr/bin/chage	/usr/sbin/ping6
/usr/bin/gpasswd	/usr/sbin/traceroute6
/usr/bin/at	/usr/sbin/usernetctl
/usr/bin/chfn	/usr/sbin/userhelper
/usr/bin/chsh	/usr/sbin/traceroute
/usr/bin/newgrp	/usr/sbin/rscsi
/usr/bin/crontab	/usr/X11R6/bin/Xwrapper
/usr/bin/kdesud	/usr/local/bin/xlock
/usr/bin/rdist	/usr/bin/rexec

Table 2: Dangerous SUID/SGID files

is performed, other SUID/SGID files may be installed as well. Either the unnecessary programs or the SUID bit should be removed.

Finally, the system can be configured to allow SUID bits to take effect only in specific filesystems like / (root filesystem), /usr and /usr/local. The rest of the filesystems* can be mounted with the nosuid[†] option enabled in /etc/fstab configuration file.

*This applies especially to the /home filesystem.

[†]A more restrictive setup would be to set the **nodev** and **noexec** options as well. These would deny interpretation of device files and execution of any binaries in those filesystems.

2.3.6 Disabling unnecessary services

Servers are being exploited from remote attackers due to vulnerable or bad configured services. The more services a host is running the more possible it is that a vulnerability will be found and will be used to penetrate the system. Moreover, insecure servers like `telnetd` and `ftpd*` are enabled by default, thus making the system's penetration easier and more possible.

The administrator should disable all unnecessary services which are enabled by startup scripts located in `/etc/rc.d/`. Only needed services should be configured to automatically startup when the system boots. Such services are listed in Table 3. Using Command 2.4, the administrator can query the

keytable
syslog
network
random
rawdevices
apmd
iptables
crond

Table 3: List of minimum enabled services

system for all the services which are automatically started when it boots. All, except those listed in Table 3, should be disabled.

Command 2.4 Locating and disabling services that automatically start

```
server# chkconfig --list | grep ":on"
server# chkconfig --del sendmail
server# chkconfig --del xinetd
```

The `crond(8)` daemon is started by default. This is used to execute scheduled commands at a later time. Only `root` should be able to invoke the `crond` server. The same applies for the `atd(8)` as well. Using Command 2.5, the necessary configuration files will be created that will only allow `root` user to use these services.

*Other insecure services that should not be enabled are `fingerd`, `rexecd`, `rlogind` and `rshd`.

Command 2.5 Restricting access to `crond` and `atd`

```
server# echo root >> /etc/cron.allow
server# echo root >> /etc/at.allow
server# chmod 600 /etc/cron.allow /etc/at.allow
server# chown root:root /etc/cron.allow /etc/at.allow
```

2.3.7 Securing X Windows

X Windows is a graphical interface allowing a user to interact with the system. The user can be remotely located or logged in the system. The support for remote connections in the X protocol introduces vulnerabilities to the system.

Both the X server and the X font server (`xfst`) are configured to use TCP ports and listen for local and remote connections. This default behaviour of listening for remote connections can be disabled by making the necessary changes listed in Table 4. If it is required for remote users to connect to

Securing the X server	
<u>Files</u>	<u>Entries</u>
~/ <code>.xserverrc</code>	<code>exec X -nolisten tcp</code>
<code>/etc/X11/xdm/Xservers</code>	<code>:0 local /usr/X11R6/bin/X -nolisten tcp</code>
<code>/etc/X11/fs/config</code>	<code>no-listen = tcp</code>
<code>/etc/X11/xdm/Xaccess</code>	<code>no entries</code>
Securing the X font server	

Table 4: Disable network connections in X server and XFS

the X server, this should be performed using authentication mechanisms like Kerberos, DES private keys or Cookies* [5]. Even if TCP support is disabled, the user can tunnel his graphical applications through the SSH server [Section 2.3.9]. Using this feature of SSH is highly recommended, as it provides with strong authentication and encryption for the communication channel. Finally a firewall can also be used to protect the X resources from a remote attacker [Command 7.9 on page 90].

2.3.8 Limiting system's resources

Users must not have unlimited access to resources like the CPU and the memory. If they are not restricted they are able to run malicious programs

*`Xsecurity(7)` manual page lists all the available authentication mechanisms provided by the X server.

that will cause Denial of Service (DoS) to the system. Such programs do not require advanced programming skills that is why they have been used in the past to crash a system or delay the server's response in critical applications. Resources for users or groups are listed in file `/etc/security/limits.conf` [File

```
# /etc/security/limits.conf
# Custom entries by Kapetanakis Giannis
# Use with caution !

@users hard core      0      # coredumpsize
@users hard cpu       2      # cputime
@users hard nofile    32     # descriptors
@users hard rss       16384   # memoryuse
@users hard as        65536   # virtual memory
@users hard stack     8192   # stacksize
@users hard nproc     16     # max processes
@users hard locks     32     # max file locks
*      - maxlogins    2      # max logins for each user
```

File 2.3: Placing resource limits in `/etc/security/limits.conf`

2.3] which is controlled by a PAM module named `pam_limits` [6]. Custom entries have been added that restrict the available CPU time, the memory use, the maximum number of open files, the maximum number of processes, the default core size and the maximum login sessions allowed for each user.

When a program fails abnormally, all data stored in the program's memory is dumped to a `core` file. Such files might hold sensitive information like user passwords. This function was disabled by setting the default size of `core` files equal to zero.

2.3.9 SSH installation

In Section 2.3.6 it was mentioned that insecure servers like `telnetd`, `ftpd` and `rlogind` should be disabled. These services are used to remotely connect to a Unix system. However, they send passwords in cleartext which makes it easy for an eavesdropper to capture them.

An alternative for those services is the Secure Shell (SSH) server. It supports remote logging, remote file transfers and tunneling of various protocols like the X protocol or even NFS [7]. Furthermore, it provides a secure, encrypted communication channel which can be used in insecure networks like

the Internet. Finally strong, mutual authentication mechanisms can be used to authenticate both client and server in a secure manner.

A free version of the SSH software can be downloaded from <http://www.openssh.org>. An administrator can install it using Command 2.6. Compatibility with MD5 passwords, PAM and TCP Wrappers [Section 2.3.11] was added. Support for Smart-cards and Kerberos can also be included if it is required.

Command 2.6 SSH installation

```
server# gtar -zxvf openssh-3.6.1p2.tar.gz
server# cd openssh-3.6.1p2/
server# ./configure --prefix=/usr --sysconfdir=/etc/ssh \
--with-tcp-wrappers --with-pam --with-md5-passwords
server# make
server# mkdir /var/empty
server# chown root:sys /var/empty
server# chmod 755 /var/empty
server# groupadd -g 72 sshd
server# useradd -u 72 -g 72 -d /var/empty -s /bin/false -M sshd
server# make install
server# cp contrib/redhat/sshd.pam /etc/pam.d/
```

The SSH server should be configured to cover the specific needs of each host it runs. The configuration file is located in `/etc/ssh/sshd_config` and a sample of this is presented in File 2.4. Only the administrators are allowed to login into the server. Normal users should not use the server for login. Other hosts with SSH installed can be configured to accept users. Furthermore root login is disabled for security reasons. The administrators should use `su` or `sudo` to login to the root account. The server can be enabled by using Command 2.7.

Command 2.7 Enabling automatic startup of SSH server

```
server# cp contrib/redhat/sshd.init /etc/init.d/sshd
server# chkconfig --level 345 sshd on
server# /usr/sbin/sshd
```

Finally it should be added that SSH should be installed in both servers and clients. The same commands and configuration options can be used to install SSH in a Unix host that is used as a client. The only difference between a client and a server is the access control. Only authorized administrators, connecting from authorized hosts, should be allowed to connect to the server. On the contrary, in clients all the users are usually allowed to connect.

```
# /etc/ssh/sshd_config for OpenSSH
# Created by Kapetanakis Giannis

Port 22
Protocol 2
ListenAddress 10.0.0.1
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key

SyslogFacility AUTH
LogLevel INFO

RhostsAuthentication no
PermitEmptyPasswords no

ChallengeResponseAuthentication yes
X11Forwarding yes
UsePrivilegeSeparation yes

PermitRootLogin no
AllowUsers admin1 admin2

Subsystem      sftp      /usr/libexec/sftp-server
```

File 2.4: SSH configuration in `/etc/ssh/sshd_config`

Entity authentication is controlled by the configuration file of SSH and the PAM library which provides with a large number of authentication modules for Linux. Users can either be authenticated using a password, a public key* or by using Kerberos. Host based access control can either be performed by public keys, TCP wrappers or a firewall [Command 7.13 on page 91]. A combination of those mechanisms should be used to provide an adequate level of security.

*The server has a public key stored, for which the client knows the corresponding private key. Digital signatures are then produced for entity authentication. For more information about digital signatures and public key cryptography the reader can reader to Section 3.

2.3.10 System logging

Unix/Linux is provided with a server which is responsible for logging and documenting various events. This server is called `syslogd(8)`. Kernel messages, system logs and messages from other servers like the Mail server or the DNS server are forwarded to the `syslogd` server.

System logs are important for a complete system security. Security events like valid or invalid authentication attempts, firewall logs, network attacks and configuration errors are all recorded. The administrator can detect an attack by examining the log files [Section 8] and improve the system's security by correcting the errors which are reported.

The main configuration file of `syslogd` is located in `/etc/syslog.conf`. The server can be instructed to keep different types of logs in different files or even different hosts. All the log files are kept in `/var/log/`.

```
# /etc/syslog.conf by Kapetanakis Giannis
# Log kernel messages
kern.*                /var/log/kernel

# Save boot messages
local7.*              /var/log/boot

# Everybody gets emergency messages
*.emerg               *

# Log warnings and errors
*.warn;*.err         /var/log/syslog

# Log all logins
auth.*;authpriv.*    /var/log/loginlog
user.*;daemon.none   /var/log/loginlog

# Log everything
*.*                  /var/log/messages
*.*                  /dev/tty9
*.*                  @loghost.example.com
```

File 2.5: System logging configuration in `/etc/syslog.conf`

Log files should not be readable by normal users, in case user logins are permitted. The file permissions should be modified to allow read access only

to the group of the administrators. Finally, only `root` should be able to delete or modify those files.

Bear in mind that the `syslogd` server is not much of a secure one. Anyone on the system can forward fake messages by using simple programs like the one presented in File 2.6. Such messages can be fake authentication or error

```
#include<stdio.h>
#include<syslog.h>

/* Compile: gcc -o syslog-write syslog-write.c
 * Execute: ./syslog-write [program] "message"
 */

main(int argc, char *argv[])
{
    openlog(argv[1], LOG_PID, LOG_AUTHPRIV);
    syslog(LOG_INFO, argv[2]);
    closelog();
}
```

File 2.6: A tool for `syslogd` manipulation

messages. By flooding the log files with these, an attack might not be noticed as it will be hidden among other fake messages. Furthermore, an attacker is able to forward messages to the log server even if he is located in a remote host. Such techniques are used by attackers when their aim is to disable the logging facility. By filling the log files with excessive data and using all the available space of the `/var/` filesystem, where the files are being kept, the logging is stopped as no more data can be saved.

The `syslogd` server is by default configured to listen for remote connections in many Unix distributions. Such behavior can be disabled in Linux by removing the `-r` flag inside the file `/etc/sysconfig/syslog`. This should be enabled only in log-servers dedicated to log messages from other hosts. Furthermore a firewall running on that log-server should be configured to accept connections only from a limited number of hosts [Command 2.8].

In case an attacker has successfully gained `root` access to a server, the logs cannot be trusted. He has the ability to delete them or manipulate them by will. That is why logs should be rotated often and stored in a secure place. Such places can be a printer, a CD-Rom, a log-server or a backup device. Furthermore, alternatives can be used which are more secure than the old `syslogd` daemon. Such alternatives are the `Msys-`

Command 2.8 Firewall entry for a log-server

```
server# iptables -A INPUT -s ! 10.0.0.0/8 -i eth0 -p udp \
--dport 514 -j LOG --log-prefix "Log attempt:"
server# iptables -A INPUT -s ! 10.0.0.0/8 -i eth0 -p udp \
--dport 514 -j DROP
```

log (<http://sourceforge.net/projects/msyslog>) and the SDSC Syslog (<http://sourceforge.net/projects/sdscsyslog>).

2.3.11 TCP Wrappers

At the time when firewalls were not included as a standard in every Unix and Linux distribution, Wietse Venema produced TCP Wrappers (TCPD), an access control facility for network services. TCPD is a simple and flexible wrapper for services like `telnet`, `ftp`, `portmap`, `ssh` and generally every network service that is linked to it.

Whenever a request for a supported service arrives, TCPD allows or denies the connection according to its host access tables. All the requests are logged through the `syslog` facility [Output 2.1].

```
Aug 20 20:10:01 server sshd[5052]: connection from 192.168.2.10
Aug 20 20:10:01 server sshd[5058]: WARNING: Denied connection
from 192.168.2.10 by tcp wrappers
Aug 30 20:11:57 server sshd[5052]: connection from 10.0.0.2
Aug 30 20:12:01 server PAM_pwdb[5086]: (sshd2) session
opened for user admin
```

Output 2.1: TCPD refusing or allowing a connection

Its access control decisions are based on IP entries that are listed in the two configuration files; `/etc/hosts.deny` and `/etc/hosts.allow`. A flexible and secure setup is to deny access to everyone in the first file and explicitly allow access to specific services from specific hosts only, in `/etc/hosts.allow`. For that, a single entry is required only in `/etc/hosts.deny`:

```
ALL: ALL
```

meaning that the default policy for ALL services is to deny access to ALL hosts, unless access is permitted by entries in `/etc/hosts.allow` [File 2.7].

TCP Wrappers, although it is a great security software, is not a complete solution to network-based access control. It must be combined with a firewall for a better result. However, there is always the possibility that an attacker

```
# /etc/hosts.allow access control entries
# /etc/hosts.deny must have only:
#     ALL: ALL@ALL
#
ALL: 127.0.0.1
sshd: 10.0.0.1, admin1@10.0.0.2, admin2@10.0.0.2

sendmail: 127.0.0.1, 10.0.0.0/255.255.255.0
portmap: 10.0.0.0/255.255.255.0
```

File 2.7: Allow access to specific hosts only

bypasses the firewall and the TCP Wrappers as they are both based on IP addresses which can be spoofed easily. The IPsec [8] protocol provides strong authentication of the network traffic and its mechanisms can be used for network-based access control. Only authenticated hosts can be allowed access to services without the risk provided by the IPv4 protocol.

2.4 Advanced Security Features

2.4.1 Process accounting

Linux includes support for process accounting. This facility can be enabled in the kernel, by setting the `CONFIG_BSD_PROCESS_ACCT=y` option. Whenever a process exits, information about it will be written to a file. Information recorded includes the user who executed the process, the command executed, memory use, CPU time and the controlling terminal. This facility is called accounting and is part of the auditing mechanisms that Linux supports.

In order for the accounting to be enabled Commands 2.9 must be executed. All the information is stored in `/var/account/pacct` and it can be viewed with the `lastcomm(1)` and the `sa(8)` programs. `Acctail` (<http://www.vanheusden.com/acctail/>) can also be used if real-time monitoring is required. Other accounting information which is important from a security point of view, is the login record of each user, which is examined in Section 8.2.2 on page 96.

Bear in mind that if an attacker manages to gain `root` access on the server he is able to delete or modify the accounting record in order to hide his evidence. Simple scripts can be created by the administrator that will store these records and the log files into a secure location. This documented information can be a starting point to a further investigation, in case a successful

Command 2.9 Enabling process accounting

```

server# wget ftp://ftp.gnu.org/pub/gnu/acct/acct-6.3.2.tar.gz
server# gtar -zxvf acct-6.3.2.tar.gz
server# cd acct-6.3.2
server# ./configure
server# make
server# make install
server# mkdir -m 700 /var/account
server# touch /var/account/pacct
server# touch /var/account/savacct
server# touch /var/account/usracct
server# chown -R root:root /var/account
server# ln -s /var/account/pacct /var/log/pacct
server# ln -s /usr/local/sbin/accton /sbin/accton
server# /usr/sbin/accton /var/account/pacct

```

intrusion is detected [Section 8].

However, the records are not as complete as they should. Commands executed are recorded but without any information about the arguments appended to each command. Files which have been successfully (or not) accessed are not documented. Finally if more than one administrators are using the `root` account, their actions cannot be separated as they are logged in using the same username*.

Auditing of security relevant events should be recorded with an auditing mechanism that is compatible with the *controlled access protection* defined in C2 class security of the Orange Book's evaluation criteria [9]. A C2 auditing mechanism is not available for Linux although its necessity has been suggested by the NSA.

2.4.2 File attributes

File attributes is an access control mechanism that can be used to make file permissions more restrictive than they are. Table 5 lists some of the available attributes that can provide security features. These can be applied using `chattr(1)` command. The attributes that are set on a file can be listed with `lsattr(1)`. Attributes like 'a' and 'i' are highly recommended for log files and files with sensitive data.

*Other Unix systems, like AIX can log more information, like accesses files and the initial account name prior the user changes to `root` with `su`.

Attributes	Result if applied on a file
A	The last access time (atime) is not modified.
a	Can only be opened in append mode for writing. Useful for log files.
d	Is not included for backup with dump(8) .
i	Cannot be modified, deleted or renamed. No links can be created for this file. Set only by root user. Useful for files with sensitive data that do not change often.
s	If deleted, its blocks are zeroed.
u	If deleted, its contents are saved. Allows undeletion.

Table 5: Security attributes

2.4.3 Setting boot options

Unix supports different run-levels in which different operations are being performed. The run-level is controlled by the first process started when the system boots, the **init(8)**. This process always gets one (1) as a process id (PID) and is responsible for starting all the other subsystems, services and processes.

RedHat Linux is using the SYS(V) startup procedure. The **init** process is reading its configuration file which is located in **/etc/inittab**, like in most of the Unix operating systems. For each runlevel defined (0-6) there, there is a directory **/etc/rc.d/rc[0-6].d/** which includes links for scripts that should be executed when the system enters each runlevel. These scripts either start or stop the appropriate subsystems and services.

Furthermore, the file **/etc/inittab** defines the console terminals that should be available, the default runlevel and other options like the trusted path (Ctrl+Alt+Del). The administrator should set the default runlevel to a none graphical environment as it is less secure than the default console. Trapping of Ctrl+Alt+Del should also be disabled as it allows anyone logged on the system console to reboot the machine [File 2.8].

Runlevel 3 is the default runlevel as it provides with a multi-user environment. Runlevel 1 is used for system maintenance. None of the services is started, including the network. User **root** is only allowed to login and a password should restrict that.

Other Unix distributions, like AIX or Solaris, prompt the administrator for the root password when the system goes to the *single user mode*. RedHat Linux does not. It automatically starts a **root** shell which is not protected with a password. Furthermore, while the boot-loader is running, the user

```
# Default runlevel. The runlevels used by RHS are:
# 0 - halt (Do NOT set initdefault to this)
# 1 - Single user mode
# 2 - Multiuser, without NFS (The same as 3,
#     if you do not have networking)
# 3 - Full multiuser mode
# 4 - unused
# 5 - X11
# 6 - reboot (Do NOT set initdefault to this)
d:3:initdefault:

# Do NOT trap CTRL-ALT-DELETE
# ca::ctrlaltdel:/sbin/shutdown -t3 -r now

# Run xdm in runlevel 5
x:5:respawn:/etc/X11/prefdm -nodaemon
```

File 2.8: /etc/inittab

can insert options like those listed in Command 2.10 and instruct the system to boot into single user mode.

Command 2.10 Hack the system from the console

```
LILO: linux init 1
LILO: linux root=/bin/bash
```

If the boot loader is protected with a password, then no options can be passed without entering this password. To protect the boot-loader with a password, the entries listed in File 2.9 should be inserted in the boot-loader's configuration file.

The LILO boot-loader stores the password in cleartext. GRUB stores an MD5 hashed value of the password, that is why it is more recommended. In both boot-loaders the configuration files must not be readable by anyone except `root`.

Physical security is one of the most important parts of computer security [10]. Physical access to the system makes it easier for an attacker to gain `root` privileges and very difficult for an administrator to be protected against such an attack. Although this document does not refer to physical security at all, some minor information is listed on how to protect the Linux system running on a x86 platform.

```
# For GRUB, first run grub-md5-crypt to produce
# the password. Then insert the following entries
# in /boot/grub/grub.conf
password --md5 $1$WokL.0$j0XhPNj9JgljCjakDZ5D1.

# For LILO, insert the following entries
# in /etc/lilo.conf
image=/boot/vmlinuz-2.4.21
    label=linux
    read-only
    root=/dev/hda2
    restricted
    password=test_pass

# Then run lilo -v -v -v -C /etc/lilo.conf
```

File 2.9: Protecting the boot-loader

The PC BIOS must always be protected with a password in order to be more difficult for someone with access to the console to change its options. The system must only boot from the hard disk where the OS is installed. Booting from other devices like the CD-Rom, the floppy, from the network or other hard disks should be disabled. The box must be secured with a lock in order to prevent attackers from resetting the BIOS or installing other devices. The reset and the power-off button should be removed or other with a key should be installed.

More advanced features include room cameras, resistant doors, alarms, light detectors, movement detectors, temperature detectors, fire detectors and even guards which are heavily equipped.

Do not forget that physical phenomena like flood, fire, earthquakes and lightning strikes are threats to your computer security as they can easily destroy it. Theft, sabotage, terrorism or vandalism are also possible threats, depending on the organisation. Also it should be mentioned, that apart from the computers, other assets should be included in your list; backups, manuals, printouts, software, hardware personnel records and audit records are assets with a value and should be protected [1].

2.4.4 Kernel options

Administrators should always install a custom kernel for their system. Only the minimum options should be defined in the kernel in order to create a

small, flexible and stable kernel. Furthermore, the kernel can be optimized for security while the system runs. Various features can be enabled or disabled in order to improve the security of the system and make it more difficult for an attacker to compromise the system [11]. Examples of such features are presented in File 2.10.

2.4.5 Linux Intrusion Detection System (LIDS)

The main vulnerability of Unix and Linux has always been the presence of the `root` user. If someone gains unauthorized `root` access to the system all the possible privileges are granted. He is able to modify or delete files and audit record, start or stop services, delete or create accounts, change the file permissions and ownership, disable the firewall and even change the kernel.

Traditionally, there was no protection against such attacks. However, kernel patches have been created aiming at the protection of the system; mechanisms are included that can prevent the `root` user from tampering with important parts of the system.

LIDS is such a detection and prevention system that resides within the Linux kernel [12]. A system-wide *reference* model and a *mandatory** access control policy [9] is implemented which defines who is allowed to have access to particular parts of the system. The enhanced mechanisms provide protection for important files, directories and processes, preventing unauthorized users (including `root`) to access them. Additionally, access to raw input/output operations, network operations and privileged system calls can be restricted to a limited group of programs [13]. In case a policy violation is detected, it is reported to the administrator.

Access Control Lists (ACLs) which are stored within the objects (files, directories), define the allowed operations that can be performed. For example the password file `/etc/shadow` can be made writable only by the `/bin/login` program in order to authenticate users. If the file is protected with LIDS, then even `root` user will not be able to read or modify the file. The append-only operation can be attached to system log files and audit records that will prevent accidental or intentional deletion.

Furthermore, *Capabilities* conforming to POSIX.1e [14] are attached to the subjects (programs), specifying the subject's access rights on particular objects. The limitation is that only programs are treated as subjects. Capabilities cannot be associated with users in order to restrict their rights.

*Most operating systems, including Linux support only the concept of ownership when making access control decisions. Such policy is called *discretionary* access control policy [51].

However, an administrator can overcome this by setting appropriate file permissions to programs. A limited group of people may then be allowed to invoke particular programs, if access to protected files (objects) is required. Nevertheless, if capabilities were associated with users as well as with programs, access control would be more powerful and more flexible.

LIDS is primarily used for preventing and detecting unauthorized actions, according to the specified policy. Furthermore by using LIDS, the kernel is enhanced with a port scan detector which can alert the administrator to the warning signs of a possible intruder [15]. The alerts are logged and can be forwarded over the network using a mailer program included in the kernel.

Additional kernel patches have been created for Linux that can improve the security of the system. Such a project is the Security-Enhanced Linux (SELinux) which is maintained by the NSA [16]. It supports Type Enforcement, Role Based Access Control (RBAC) and optionally Multilevel Security. LIDS and SELinux have been merged into newer, beta versions of the Linux kernel like v2.5 and v2.6, under the Linux Security Module (LSM) framework [17, 18]. The LSM is a joint development effort which combines several security projects.

2.5 Conclusion

Unfortunately vulnerabilities are always found. As long as security is regarded a serious matter, then patches for incorrect software should always be installed as soon as they are available by the vendor. This procedure requires a constant maintenance by the system administrator. However, software that is not installed is not a threat to a system nor does it need to be updated. That is why only the required software should be in place. Furthermore, protection of a minimal system is easier than protection of a complex one. Services that are not needed should be removed or disabled as they can be potential access doors to your system.

The rest of the system being used should enable all the available security features. Account and password security is the first line of a system's defence and should never be underestimated. Unauthorized access to the system makes it easier for an attacker to bypass your security mechanisms and gain `root` access.

Despite the need for ACLs and Capabilities, only a minimum number of operating systems actually make use of these powerful access control mechanisms. Most of the Unix OS currently support only simple file permissions.

Although security threats which are enabled by the presence of the `root` account can be eliminated, this kind of approach is not yet fully adopted by Linux. Anomalies and contradictions have been detected throughout the

installation and use of both the LIDS kernel patch and a part of the LSM framework. Access was allowed to objects that were protected while unprotected objects were not accessible. Further consequences were instabilities to the system. Nevertheless, if future versions are error-free, their attractive mechanisms will contribute greatly to the overall computer security.

```
# Enable kernel's spoof protection.
echo 1 > /proc/sys/net/ipv4/conf/all/rp_filter
# or
for interface in /proc/sys/net/ipv4/conf/*/rp_filter; do
    echo "1" > ${interface}
done

# Log spoofed packets, source routed packets, redirect packets.
echo 1 > /proc/sys/net/ipv4/conf/all/log_martians

# Enable TCP SYN Cookie Protection.
# Protects against a SYN-Flood DoS.
# kernel: CONFIG_SYN_COOKIES=y
echo 1 > /proc/sys/net/ipv4/tcp_syncookies

# Make sure that IP forwarding is turned off.
# We only want this for a router
echo 0 > /proc/sys/net/ipv4/ip_forward

# Don't accept source routed packets.
echo 0 > /proc/sys/net/ipv4/conf/all/accept_source_route

# Disable ICMP redirect acceptance. ICMP redirects can be
# used to alter your routing tables, possibly to a bad end.
echo 0 > /proc/sys/net/ipv4/conf/all/accept_redirects

# Enable the filtering options -> Firewall
# kernel: CONFIG_NETFILTER=y
# kernel: CONFIG_NETFILTER_DEBUG=y
# kernel: CONFIG_FILTER=y
# kernel: CONFIG_IP_NF_IPTABLES=y

# Enable always defragging protection on the firewall
echo 1 > /proc/sys/net/ipv4/ip_always_defrag

# Enable broadcast echo Protection
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

# Enable bad error message protection.
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_bogus_error_responses

# Disable response to ping.
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_all
```

File 2.10: Kernel hardening

3 Certification Authority for Internal Use

This section is concentrated on the procedures that can be followed in order for a Certification Authority to be created. The digital certificates issued by this authority will be used in the next sections to provide an adequate level of security. Firstly, let us introduce some of the concepts needed for a better understanding.

3.1 Public Key Cryptography

Public key cryptography is based on asymmetric algorithms [19] where each entity possesses a public key and a corresponding private key. The public key is used for data encryption and digital signature verification, while the private key is used for decryption and production of digital signatures. The private key should be kept secret and available only to the owner of that key. On the other hand the public key can be made available to anyone.

3.2 Digital Signatures

A digital signature for a message is a cryptographic value that validates the message and verifies its origin [19]. It depends on the message and a secret parameter which is the private key of the sender. For example, in order to produce a digital signature, we first use a Hash function [20] on the message. This one way function takes input of arbitrary length and changes it to a fixed length. Furthermore, minor changes to the message will produce a totally different hash value [Output 3.1].

Message	MD5 Hash
I owe you \$100	0244cf8c60618ace0198af172f265623
I owe you \$1,000,000	4cdfd9a4f3d32240f239c73c50401831

Output 3.1: Calculating the hash value of a message

The next step is the encryption of the hash value with the private key of the sender [Figure 1]. This way we can settle disputes of what message (if any) was sent, and verify its integrity. The verification process compares the hash value of the message to the hash value generated from the decryption of the signature to test if they are identical. This can be done by anyone who knows the public key of the sender and therefore can decrypt the digital signature.

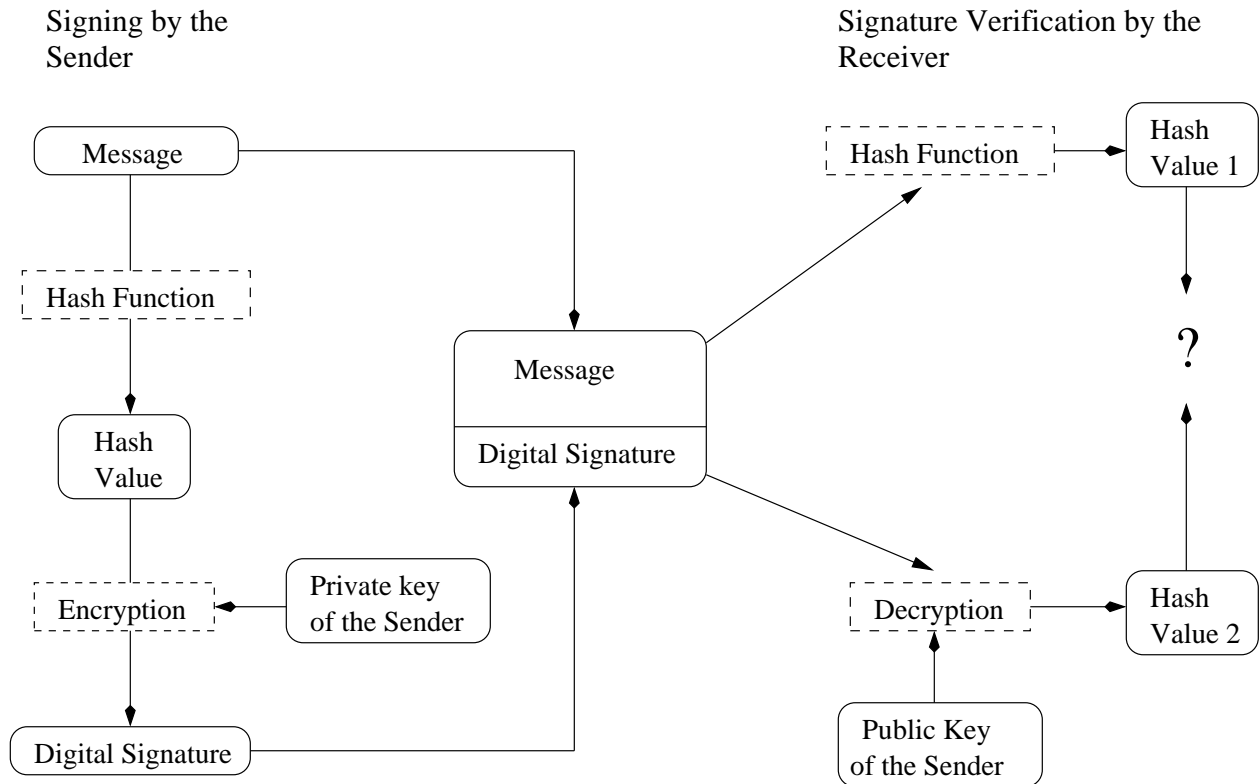


Figure 1: Signing and verifying a signature of a message

Furthermore, digital signatures can be used in challenge-response protocols where mutual authentication is required. Both entities are authenticated to each other by proving knowledge of the private keys. Each entity (user/host) produces a digital signature based on a *challenge* sent by the other. The challenge is usually a *nonce* which is a random number used only once. The signature attached to a new nonce is sent back in order for the other entity to verify the signature and produce a new one using his own private key. This way both entities authenticate each other.

The main difference between a hand-written signature and a digital signature is that the last one is not static; meaning that, if we sign two different documents, the signatures will be different. Therefore, a digital signature cannot be easily forged. Nevertheless, if an attacker has access to the private key he is able to forge signatures. In order to protect a system from this kind of attack, private keys should remain secret. Encrypting private keys and storing them in devices from which they cannot be easily extracted is a suggested solution.

The security of the system also relies on the authenticity of the public

key. If the authenticity of the public key cannot be guaranteed, the digital signature should not be trusted. An attack would be to impose the use of a different public key, for which the attacker knows the corresponding private key, on the receiver. Certificates are used to ensure the authenticity of an entity's public key.

3.3 Digital Certificates

A typical certificate like the X.509 v3 certificate [21] includes the public key, an expiration date, the name of the owner and the issuer of the certificate. It also includes a signature by the issuer, cryptographic algorithms used for the signature and the public key, and permitted use of the public key. By using the certificate we can verify the authenticity of an entity's public key. In order for this to happen, we must have an authentic copy of the issuer's public key which we will use to verify the signature which is attached to the certificate.

If a certificate has more than one public keys and signatures, it is called a certificate chain [22]. It includes the public key of the issuer and a signature for that key produced by the prime issuer. The procedure which is followed now, includes the verification of both signatures and public keys.

Certificates are frequently used in entity and data origin authentication. An entity can be a person, a device or even a server. The SSL/TLS protocol is highly dependent on digital certificates.

3.4 Certification Authorities

The main role of a Certification Authority (CA) is to provide the certificates that bind an entity's identity to the value of its public key [19]. In order for these certificates to be verified, the CA's public key must be known and must be trusted as an authentic one. If there is a need for two CAs to certify each other, then either a third one is required (root CA) or cross certification between the two CAs should take place.

The certification procedure is of great importance in Public Key Infrastructures (PKI). The two key issues involved in the certification procedure are *who* generates the keys and *how* the identity of an entity is verified* during the registration. Key distribution, key storage and certification revocation are also parts of the system that require great concern, as the security of the system relies on these as well [23].

*The system can be exploited if there is not a secure link between the CA and the Registration Authority (RA).

In order for PKI solutions to be successfully deployed, basic requirements must be satisfied. Defining business' requirements and planning of the PKI structure are the first considerations someone should have [24]. The cost*, the risks and the benefits [25] of becoming a CA should also be taken into account. Nevertheless, the technical part of CA can be easily implemented as shown in the following section.

3.5 Root Certification Authority Creation

We can easily create a Certification Authority based on the X.509 PKI [21], by using the `openssl v0.9.7b` software. Its primary use is to implement the SSL/TLS protocol. Furthermore, it can be used to create and run a medium size certification authority like the one a University, or an Internet Service Provider (ISP) needs. This setup is not suggested for e-commerce sites as it lacks the automation that is needed in such sites.

First we should create the directory where our CA will be stored. Then we must create the root certificate which will be self signed because we are creating a root CA [Command 3.1].

Command 3.1 Creation of CA storage space

```
server# mkdir /usr/local/CA
server# chmod 700 /usr/local/CA
server# cd /usr/local/CA
server# mkdir certs crl newcerts private
server# touch index.txt
server# echo 01 > serial
```

With the Command 3.2 we make a new certificate request, which will be self signed, and a new RSA private key (`private/rootkey.pem`) with size equal to 4096 bits. We plan to use the same key for 10 years that is why we have put such a large key size. The minimum size that should be used is 2048 bits. The certificate request will be hashed with the SHA-1 algorithm as it is considered to be more secure than MD5, which is the default option in `openssl`. The root certificate is stored in `rootcert.pem`.

Command 3.2 Creation of self signed certificate for the root CA

```
server# openssl req -new -x509 -newkey rsa:4096 -sha1 -keyout \
private/rootkey.pem -out rootcert.pem -days 3650
```

*Time, money and effort.

The private key must be encrypted using triple DES (168 bit symmetric encryption). This is why a password is requested. The administrator should choose a strong, lengthy password* to encrypt the private key because the security of the system relies on the secrecy of this key. Even if the key falls into the hands of an attacker it should be difficult for him to decrypt it and use it. Then, to complete the creation of the certificate, Command 3.2 requests from the administrator to specify the name and the location of the site [Output 3.2].

```

Generating a 4096 bit RSA private key
.....
writing new private key to 'private/cakey.pem'
Enter PEM pass phrase: *****
Verifying - Enter PEM pass phrase: *****
Country Name (2 letter code): GR
State or Province Name (full name): Crete
Locality Name (eg, city): Heraklion
Organization Name (eg, company): Example Organization
Organizational Unit Name (eg, section): ROOT CA
Common Name (eg, YOUR name): Example ROOT CA
Email Address: admin@example.com

```

Output 3.2: Creation of root certificate

3.6 Non-root CA Creation

We can also create a second non-root CA [Command 3.3] that will issue all the certificates for users and SSL servers[†]. This way the root CA can be off-line in order to be more secure. The second CA can be on-line and its certificate will be signed by the root CA. In addition to this, if the private key (of the non-root CA) is compromised we can issue a new certificate for the non-root CA, without redistributing the certificate of the root CA.

This new certificate request is not self signed (`-x509` argument is missing), the key size is only 2048 bits and it is valid for 5 years instead of 10. These options can be changed to fit your specific needs. Bear in mind that the

*If someone has access to the private key and knows the password, he is able to issue any certificate he wants. It will be considered as valid, from anyone who trusts the root certificate.

[†]A third scheme can be adopted as well, where two non-root CAs are created; one for servers and the other for users only [Figure 2].

Command 3.3 Creation of a non-root CA

```
server# openssl req -new -newkey rsa:2048 -sha1 -keyout \
private/cakey.pem -out ca.pem -days 1825
```

private key of the second CA should remain secret and protected as well. Again, note that a strong password must be chosen to encrypt it.

We should then sign* this request with the private key of the root CA [Command 3.4]. In order to read the certificate or the private key which we have just produced, Command 3.5 must be executed.

Command 3.4 Signing the certificate request of the non-root CA

```
server# openssl ca -policy policy_anything -extensions \
ca_cert -keyfile private/rootkey.pem -cert rootcert.pem -out \
cacert.pem -days 1825 -md sha1 -infiles ca.pem
```

Command 3.5 Reading a certificate or a private key

```
server# openssl x509 -in cacert.pem -text
server# openssl rsa -text -in private/cakey.pem
```

3.7 Creating and Signing Certificates

Our certification authority is now ready to issue production certificates. A certificate may have multiple purposes. To start with, it can be used for authentication of a Web server in order to initiate an SSL protected session. Moreover, it can be presented by a user to authenticate himself on a server. Finally, it can be used to sign the public key of another certification authority[†]. Both certification authorities will now be trusted[‡]. Nevertheless we do not want to allow a certificate to be used for a purpose it was not issued for; that is why limits should be imported.

Consequently we create three new extensions, aiming to restrict the purpose of each certificate. One for CAs, one for servers and the other for users. These extensions are included in the configuration file of `openssl` which resides in `/usr/share/ssl/openssl.cnf` [File B.2 on page 118].

*In order to sign the request we use the `ca_cert` extension in `openssl`. The reader is referred to Section 3.7 for more information about extensions.

[†]We did this on our second CA where we signed its public key with the private key of the root CA.

[‡]By “trusted”, we mean that the certificates are valid and that the user depends on the administrator to present him with the true certificate of the root CA.

Then we create two certificates. One will be used by our web server and the other for user authentication [Command 3.6]. Using the following commands an administrator can create more certificates to use with other services like DNS [Section 4.6] and SMTP [Section 5.8]. The administrator

Command 3.6 Creation of a certification request

```
server# openssl req -new -nodes -newkey rsa:1024 -sha1 \  
-keyout wwwkey.pem -out www.pem  
server# openssl req -new -nodes -newkey rsa:1024 -sha1 \  
-keyout userkey.pem -out user.pem
```

of the CA should first read the certificate requests and then sign them by using the private key belonging to the non-root CA. These procedures are listed in Commands 3.7 and 3.8. After signing the certificates it should be

Command 3.7 Reading and signing a server certification request

```
server# openssl req -in www.pem -text  
server# openssl ca -policy policy_anything -extensions \  
srv_cert -out wwwcert.pem -days 365 -md sha1 -infile www.pem
```

Command 3.8 Reading and signing a user certification request

```
server# openssl req -in user.pem -text  
server# openssl ca -policy policy_anything -extensions \  
usr_cert -out usercert.pem -days 365 -md sha1 -infile user.pem
```

verified [Command 3.9] that they only include the purpose they were issued for. These certificates will expire after one year and will not be accepted as valid, but new ones can always be issued.

In case a key (private key) is compromised or a certificate is expired we can revoke the certificate. This is done by creating a list with all the non-valid certificates [Command 3.10]. This is called revocation list and it should be placed in a public web server in order to be accessible by anyone. All the certificates we have created include a URL with the path to our revocation list [File B.2].

Command 3.9 Finding the purpose of a certificate

```
server# openssl x509 -in wwwcert.pem -text -purpose -noout  
server# openssl x509 -in usercert.pem -text -purpose -noout
```

Command 3.10 Revoking a certificate and creating a revocation list

```

server# openssl ca -revoke newcerts/01.pem -crl_reason \
keyCompromise -crl_compromise 20040508191200
server# openssl ca -gencrl -cert rootcert.pem -keyfile \
private/rootkey.pem -out root.crl
server# openssl ca -gencrl -out ca.crl

```

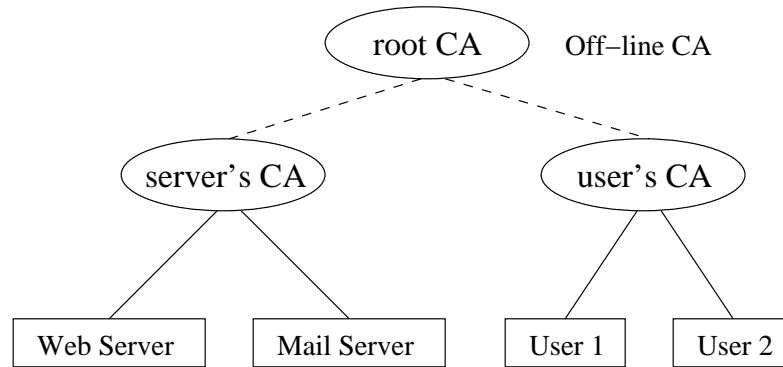


Figure 2: A hierarchical CA scheme

3.8 Conclusion

The certificates that have been issued will be used in Sections 4.6, 5.8 and 6.5. Apart from these applications a Certification Authority can be used in all the schemes that require certificates to operate. Instead of receiving certificates from a vendor, an internal CA can be created which will be maintained by the system administrator. Additionally, if there is a need for the root certificate to be trusted by individuals outside the organisation, this can be signed by the public key of a globally trusted CA. Only the root certificate needs to be signed in such occasions.

It should be mentioned that only the technical part of becoming a CA was introduced here. Other procedures, like the registration and the identity verification of users, as well as the necessary policy, were not covered. The aim of this project was different. However, the importance of these procedures is not to be overlooked.

4 DNS Server Security

The Internet Domain Name System (DNS), is a service that provides a hierarchical distributed database [26]. This database mainly consists of IP addresses of nodes, which are part of the Internet, and their corresponding hostnames. It maps each IP address to a unique hostname (PTR records) and a hostname to one or more IP addresses (A records). It also includes listings of *authoritative** name servers for a domain (NS records), aliases for hostnames (CNAME records), Mail servers which are responsible for a domain (MX records) as well as other types of records [27].

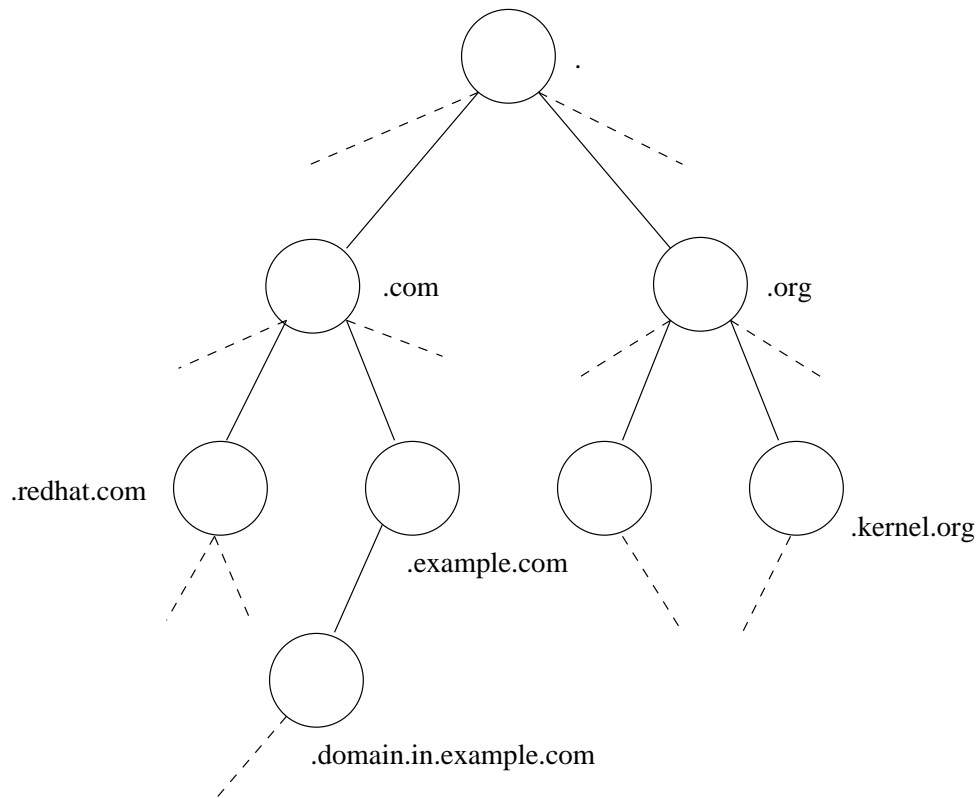


Figure 3: DNS hierarchy

A DNS Server, is a vital part of a network, not only for the network to operate, but for security as well. It must be appropriately secured, because if an attacker controls the DNS server, he can manipulate the zone files and

*A name server is called authoritative when it is responsible for a domain. The primary name server of this domain will include a record in its zone files, specifying all the valid authoritative servers for each of its sub-domains.

insert bogus entries in order to gain access (easily) to the whole network. Subsequently he can steal your outgoing mail or your credit card number by intercepting your “secure” web transactions. This can be done even if they are encrypted with SSL. Availability of the network is also dependent on the DNS server because, in case of DNS failure, all the network services fail as well.

We will install and configure a DNS server in our Linux or Unix server. For this purpose, we will use BIND 9.2.2 software. BIND is the most common choice for a DNS server; version 9.2.2 is the latest version and is highly recommended for security and functionality. Bear in mind that older versions of BIND software are vulnerable to specific security attacks which can be exploited by an attacker. Those security flaws are fixed in this version. We should also mention that Version 9 of BIND supports DNS Security [Section 4.6] and IPv6.

4.1 BIND Installation

The source tree for BIND can be found at <http://www.isc.org>. We download, extract and compile it by using the following commands:

Command 4.1 BIND installation

```
server$ tar -zxvf bind-9.2.2.tar.gz
server$ cd bind-9.2.2
server$ ./configure --prefix=/usr/local --with-openssl \
--with-randomdev=/dev/random
server$ make
server$ su -
server# cd /usr/src/misc/bind-9.2.2
server# make install
```

Help for this task can be provided by the README file, which is included in the tar ball, and a manual for BIND administrators [28], which is available in the `bind-9.2.2/doc` directory. We have added `openssl`, in order to include support for cryptography, which will be used in TSIG [Section 4.5] and DNSSEC [Section 4.6].

4.2 BIND Configuration

4.2.1 rndc.conf

The daemon (`named`) is installed in `/usr/local/sbin/`, and expects to find its configuration file in `/usr/local/etc/named.conf`. We will first create a 256

bit key, which will be used for controlling the operations of our server, with the `rndc` command. The key is a base-64 encoded string using HMAC-MD5 algorithm on our entropy spooler `/dev/random`. Its size ranges from 1 to 512 bits. The default size is 128 bits, but we will double it in order to achieve better security. Every command to the DNS server must be signed with this key in order to be authenticated, that is why we have to protect it from unauthorized viewers. The key is stored in `rndc.conf` and in `named.conf`.

Command 4.2 Key generation for server control

```
server# rndc-confgen -b 256 -r /dev/random > /etc/rndc.conf
server# ln -s /etc/rndc.conf /usr/local/etc/rndc.conf
```

```
# rndc.conf by Kapetanakis Giannis
key "rndc-key" {
algorithm hmac-md5;
secret "LPkj3Mw7rr/PXaRREziteEOYIQC760qZmQaislPyaMgc=";
};

options {
default-key "rndc-key";
default-server 127.0.0.1;
default-port 953;
};
```

File 4.1: `rndc.conf`

The `secret` value is our key. The other options are used by the `rndc` control program, defining that the default server binds to address `127.0.0.1`, on port 953 and that the `"rndc-key"` key must be used to sign its requests. Different keys should be used in order to control different servers.

4.2.2 Zone files

Our next job is to make the zone files. These files keep all the information about our domains. We serve two domains, `example.com` and the reverse mapping for `10.0.0.0/24` domain. The last one is called `0.0.10.in-addr.arpa`. The scope of this document does not cover the rules for creation of zone files, but example files are presented [Appendix A.1]. We should mention that keeping HINFO entries [27] with any kind of valid information is not advisable. Many administrators use this field for storing the operating system version or other data in order to keep track of every node

in the network. Such information might be useful to an attacker. You do not want to give the attacker more information than he should have. Keeping this field in comment might be considered an alternative solution.

4.2.3 named.conf

The `named.conf` file, holds all the configuration options for the DNS server. All the information about the domains that our server is authoritative for, is kept there. The security options are also stored in this file. We will put the file in `/etc/` and we will create a link from `/usr/local/etc/named.conf` to `/etc/named.conf`. This is because it is preferred to have all the configuration files in the root (`/`) filesystem. The listing of `named.conf` is presented in File 4.2 and 4.3.

```
# named.conf part_1 by Kapetanakis Giannis
acl internal { 10.0.0.0/24; localhost; };
acl spoof-nets { 0.0.0.0/8; 1.0.0.0/8; 2.0.0.0/8;
                192.168.0.0/16; 172.16.0.0/12; };
options {
    directory "/etc/namedb";
    pid-file "/var/log/named/named.pid";
    random-device "/dev/random";
    version "DNS server";
    dump-file "/var/log/named/named_dumb.db";
    statistics-file "/var/log/named/named.stats";
    listen-on port 53 { 127.0.0.1; 10.0.0.1; };

    allow-transfer { localhost; 10.0.0.2; };
    allow-query { internal; };
    allow-recursion { internal; };
    blackhole { spoof-nets; };
};
key "rndc-key" {
    algorithm hmac-md5;
    secret "LPkj3Mw7rr/PXaRREziteEOYIQC760qZmQaislPyaMgc=";
};
```

File 4.2: `named.conf` part 1

We will explain each step that is relevant to security. First we create an Access Control List (ACL) which is named `internal` (`acl internal`). This

```
# named.conf part_2 by Kapetanakis Giannis
controls {
    inet 127.0.0.1 port 953
    allow { localhost; } keys { "rndc-key"; };
};

zone "." in {
    type hint;
    file "zone/named.ca";
};

zone "0.0.127.in-addr.arpa" in {
    type master;
    file "zone/named.rev.loop";
    notify no;
    allow-query { localhost; };
};

zone "0.0.10.in-addr.arpa" in {
    type master;
    notify yes;
    file "zone/10.0.0";
    allow-query { any; };
    allow-update { none; };
};

zone "example.com" in {
    type master;
    notify yes;
    file "zone/example.com";
    allow-query { any; };
    allow-update { none; };
};
```

File 4.3: named.conf part 2

will be used to control access to DNS services. It is actually an alias for the addresses which are included in the statement. The second ACL is about address ranges defined by RFC 1918 [35] as private and non-routable. We will block those networks as they are used frequently for spoofing attacks.

In the options statement, we specify that our random number generator is the device file `/dev/random`. We also hide the version of the server, although this may raise conflicting opinions. Some administrators believe that, if you

hide the daemon's version, an attacker will not be able to recognize if your server is vulnerable to specific attacks and in that case they will launch the attack anyway. Others believe that, the less information you give to outsiders, the better it is for you. Bear in mind that hiding the version of your server will not help you if it is vulnerable and not appropriately patched. The version of a DNS server can be extracted with the following command:

Command 4.3 Version discovery

```
server$ dig @server example.com txt chaos version.bind
```

Next is our default access control list for all the zones. We only allow zone file transfers to be made by the server itself and a secondary (backup) DNS server with IP *10.0.0.2* (`allow-transfer`). If this is not restricted then anyone would be able to query the server and discover our network layout and every machine within the network. Normal queries can also be blocked. This is usually done in case someone wants to maintain a private domain, whose nodes must not be resolved outside of the organisation. This can be done with the `allow-query` option. In order to give answers about our two domains, *0.0.10.in-addr.arpa* and *example.com* to everyone, we explicitly allow queries inside the zone entry. This is done again with the `allow-query any`.

We will not allow queries to be made about domains or hosts, for which our server is not authoritative. This will apply to everyone except our internal network (`allow-recursion`). In this way, we defend ourselves against cache-poisoning attacks [29]. This is an attack where bogus data is inserted in the cache of the DNS server. This makes it easier to launch other attacks that use spoofing. Denying recursive queries also reduces the usage of our server. Note that even if the queries are blocked, the server will give an answer if it has already been cached. Otherwise, it will not search for it, unless it is a query coming from a node belonging to the internal network. Also keep in mind that an attacker might spoof his address to one of our local IP addresses. This will give him access to recursive queries, as the access control is only based on his IP address.

A more efficient way to defend the server against such attacks would be a *Split-Split DNS* [30]. This setup consists of using two DNS servers: one, outside our network, will answer queries about our domains and will have recursion disabled; and one inside, protected by a firewall, will do the recursive queries for our clients only. The router must also block, in both cases, every packet coming to its external interface that claims to have a source address which is part of our internal network. This is definitely a spoofed packet. We will discuss this further in our firewall configuration

[Section 7]. For a more extensive reading on DNS attacks and vulnerabilities you can look at [31, 32]. Finally if it is too expensive to maintain two name servers, it is possible to implement a similar setup using the `view` statement [36]. This feature was introduced into version 9 of BIND and it is used for Split DNS setups using one server only.

We are also blocking the spoof-networks with the powerful statement `blackhole`. The server will not accept queries from these addresses. Nor will it resolve queries for these addresses. The difference between this and `allow-query` is that now the server will not give any response back. If an administrator wants to deny access from another network (an attacker's network) this is a method for implementing access control.

Finally inside each zone entry, we disable updates from our secondary server. We do not want our *slave* server to be able to update our zone records. Only the primary *master* should be able to do that. More information about options that can be used inside or outside a zone entry, can be found in Chapter 6 of [28].

The key entry (`key "rndc-key"`) is the same key as the previous in `rndc.conf`. The server can be controlled only if the controller signs its request with this secret key. Next, in the `controls` entry, we define that our server should bind to local port 953 of `loopback` interface (`127.0.0.1`) and listen for commands. Only *localhost** is allowed to insert commands, and then only if it has knowledge of the key. The *localhost* will know the key because it can find it in `rndc.conf`.

If we want to control the DNS server remotely from another machine, then the IP address of this machine must be inserted in the `controls` statement. The remote controller must also have knowledge of the secret key. Our server should also bind to `10.0.0.1:953`, otherwise it cannot be contacted. These actions are not recommended. If there is such a need, then the remote machine must be trusted and action must be taken to preserve the secrecy of our key. It is also possible to create a second key for this remote host.

Resource limits can also be applied to the BIND daemon. These limits include how many simultaneous clients are allowed, the maximum number of open files, maximum cache size and the maximum amount of memory used. We use the default options, as different organisations require different tuning. The reader can find more about the resource limits and tuning in Chapter 6 of [28].

**localhost* is the hostname used for the `127.0.0.1` IP address. This address is used by the `loopback` interface, a virtual network adapter which is used for local connections only.

4.2.4 Security logging

The default logging of the BIND server is sufficient for our requirements. It uses the `syslog(2)` system call to log various events in `/var/adm/messages`. This logging can be extended to provide a great deal of detail. We will include an entry in our `named.conf` file that will enable the server to store the security events in a separate file. File 4.4 must be appended to our current `named.conf` file [28]. The logs will be kept in `/etc/namedb/security.log` file as well as in the `syslog` daemon files. Bear in mind that some security events, will not be logged, as they belong to different categories. For instance when an attacker tries to control our server without a proper authentication, he will not be logged, as this event belongs to the `general` category. Nevertheless this will be logged in `/var/log/messages` by `syslog`.

```
logging {
    channel "my_security_channel" {
        file "/var/log/named/security.log";
        severity info;
    };
    category "security" {
        "my_security_channel";
        "default_syslog";
        "default_debug";
    };
};
```

File 4.4: DNS security logging

4.2.5 Permissions

The BIND DNS server has the ability to run as a user other than `root`. However `root` user must start the daemon in order to bind to port 53, because only `root` processes can bind to ports lower than 1024. After the startup, our server can switch to another user. This improves our system security because even if our server is compromised, the attacker (most of the time) would only be able to gain normal user privileges and not root access. We will create a user called “`named`” for that purpose and we will tighten the file permissions [Command 4.4].

First a group called `named` is created and then a user account called `named`. Group-id and user-id 53 must not already exist in the system. The `named` account will be locked and no one will be able to login in this account. Even

Command 4.4 File permissions

```
server# groupadd -g 53 named
server# useradd -u 53 -g 53 -d /etc/namedb -s /bin/false \
-M -c "DNS admin" named
server# chown root:root /etc/rndc.conf
server# chmod 600 /etc/rndc.conf
server# chown root:named /etc/named.conf
server# chmod 640 /etc/named.conf
server# chown root:named /etc/namedb
server# chmod 750 /etc/namedb
server# cd /etc/namedb
server# chown root:named zone
server# chmod 750 zone ; cd zone
server# chmod 640 example.com 10.0.0 named.ca named.rev.loop
server# mkdir -m 700 /var/log/named
server# chown named:named /var/log/named
server# chmod 700 /var/log/named
```

root user cannot `su(1)` there. The `rndc.conf` is only readable by `root`. The `named.conf` must be readable by user `named` without write permissions. The `named` user should also not be able to write or delete the zone files. That is why we keep them in a separate directory (`/etc/namedb/zone/`) where he has only read access. The logs are kept in `/var/log/named/` and can be manipulated by that user.

This way our system will be protected, as the attacker will be able to gain only `named` access (sophisticated exploits would be required to give `root` access). Even if he gained that access (as `named`), he would not be able to harm the system or the DNS server's zone files.

4.3 Server Control

Our server should now be ready for startup. It can be started either by command line or with a shell script (Appendix A.2 contains a suitable startup script). The command line for starting the server is shown in Command 4.5; it can only be started by the `root` user. The control of our server is performed

Command 4.5 Server startup

```
server# /usr/local/sbin/named -u named
```

by the `rndc(8)` command. In order to be able to control the server, there

must be a match between the keys in `/etc/rndc.conf` and `/etc/named.conf`, otherwise the server will ignore the command and log the event. This might be an attacker trying to control your server remotely without proper authentication [Output 4.1].

```
Feb 4 15:56:09 server named[process-id]: invalid command from
attacker.at.ip.address#port: bad auth
```

Output 4.1: Attacker trying to control the server

If the server only listens for commands on the `loopback` interface, as proposed, such a threat is eliminated. Bear in mind that, although a key is needed to control the server, the `root` user is able to control it without the key by sending signals to its process (`kill -1 pid`, `kill -9 pid` etc). These enable him to stop or reload the server.

4.4 Testing Access Control

Various access control lists have been applied in our configuration, which exist in `named.conf`. These ACLs will now be tested in order to verify that nobody without proper access is able to query our server. This task also verifies that our DNS is working properly. Common tools for such testing are `nslookup(8)`, `dig(1)` and `host(1)`. The command lines and the logs on the server are presented below.

Only our secondary DNS server should be able to transfer whole zones (AXFR requests). We will check if everyone else is denied access. This can be performed on our client using Command 4.6. The server must log the event in `syslogd` [as shown in Output 4.2].

Command 4.6 Zone transfer attempt

```
client$ host -l example.com server
Using domain server:
Name: server
Address: 10.0.0.1#53
Host example.com. not found: 5(REFUSED)
; Transfer failed.
```

Zone transfer of domain `0.0.10.in-addr.arpa` must be denied and this should be verified as well. Recursive queries are denied to anyone except our local domain (`10.0.0.0/24`). We will use a node outside this range (for

```
Feb 26 17:10:57 server named[972]: client 10.0.0.10#1031: zone transfer
'example.com/IN' denied
```

Output 4.2: Denying access to zone transfers

example 10.0.2.20) in order to verify that this is the case [Command 4.7, Output 4.3].

Command 4.7 Attempt for recursive queries

```
attacker$ host www.isg.rhul.ac.uk server.example.com
Using domain server:
Name:   server.example.com
Address: 10.0.0.1#53
Host www.isg.rhul.ac.uk. not found: 5(REFUSED)
```

```
Feb 26 17:25:06 server named[972]: attacker 10.0.2.20#1056:
query (cache) denied
```

Output 4.3: Denying access to recursive queries

In order to verify the `blackhole` ACL, we temporarily add the previous node (10.0.2.20) there, reload the configuration file and request from the server normal and recursive queries [Command 4.8]. These requests get a timeout as the server does not send back any response. The server does not also log the event. It should be clear by now, how powerful the `blackhole` statement is.

Finally, we should mention again that these access control mechanisms are based on IP addresses only. If an attacker can spoof his address to a host on the local net, he is able to bypass the access control lists as well. More advanced techniques and anti-spoofing mechanisms on the router and the firewall must be applied in order to prevent such attacks.

4.5 TSIG

In order for BIND to achieve better access control between server-to-server communications, TSIG (Transaction SIGNature) can be adopted [33]. Shared secret keys and one-way hashing is used to secure the zone transfers and the recursive queries. It provides data origin authentication and data

Command 4.8 Nodes in blackhole ACL never get response back

```
attacker$ host www.example.com server.example.com
;; connection timed out; no servers could be reached

attacker$ host www.isg.rhul.ac.uk server.example.com
;; connection timed out; no servers could be reached
```

integrity. When dynamic updates are used between two servers, TSIG should be used to accept updates from valid servers only. This way IP spoofing will not be a threat as the access control will be key-based and not IP-based. It would be a nice feature if BIND supported both IP-based and key-based access control combined together.

TSIG can also be used in zone transfers between primary and secondary DNS servers. The setup is easy and the overhead is minimal. We will use it in our domain between *server* and *server2*. First a key is created by the server [Command 4.9] which is then transferred to *server2* through a secure communication channel*.

Command 4.9 Key generation for secure host authentication

```
server# dnssec-keygen -a HMAC-MD5 -b 256 -n HOST serv-serv2
```

The key is included in the file `Kserver-server2.+157+02615.private` and it is extracted from there. It should be placed in `named.conf` as shown in File 4.5. The `named.conf` file must also specify the keys to be used between the two servers. The `allow-transfer` statement is changed as well. Bear in mind that the clocks of the servers must be synchronized otherwise the transfer will not be allowed. This is to prevent replaying of previous requests.

4.6 DNSSEC

DNS Security can be used in order to cryptographically sign the zones of a server. These digital signatures are part of the secure zone. As defined in RFC 2535 [34], Security Extensions of the DNS provide an optional authentication layer to be used with DNS protocol transactions and requests. DNSSEC can also be used as an infrastructure for storing and distributing public keys. Support for DNSSEC exists in the BIND server.

A server can digitally sign its zone files. Public key algorithms (DSA[†])

*SSH and SFTP are recommended for transferring keys between two servers. For automating this process, the TKEY mechanism can be used.

[†]DSA and RSA are designated as “mandatory” by the IETF in an implementation of DNSSEC. RSA is not yet supported by BIND.

```
# TSIG in named.conf by Kapetanakis Giannis
key "serv-serv2" {
    algorithm hmac-md5;
    secret "7nhvP4YcXw9bkMW2ainvv45UqG0tYVAgN3gjQCVQkdo=";
};
server 10.0.0.10 {
    keys { serv-serv2. ; };
};
options { ...
    allow-transfer { localhost; key serv-serv2. ; };
    ...
};
```

File 4.5: TSIG for secure zone transfers. `named.conf`

are used for this purpose. The private key is used for the signatures and the public key is included in the zone file in order for others to verify the digital signature. After that the public key of the child server can be signed with its parent's private key in order to verify its authenticity. This can be extended up to the Root Name Servers*. In this way, every DNS zone can be authenticated and its integrity can be verified. The reader can refer to Chapter 4.7 of *BIND 9 Administrator Reference Manual* [28] and Chapter 11 of *DNS and BIND* [36] for examples of DNSSEC setup.

4.7 Conclusion

It is true that several bugs have been discovered for older versions of BIND. This may raise doubts as to whether BIND is really secure and if it should be used in sites where security is a concern. However, the ISC (Internet Software Consortium) has always made information, concerning security vulnerabilities in their software, publicly available immediately after a hole has been found. Vulnerable versions are usually fixed quite quickly. A page is held in their site [37] where all vulnerabilities of all BIND versions are presented. BIND 9.2.2 is the latest version so far and it is not vulnerable to any known attack.

Unfortunately, in major projects like BIND, vulnerabilities will always be discovered. Nevertheless, if you patch your server to the latest version

*Root Name Servers are placed on top of the DNS hierarchy [Figure 3] and they are authoritative for the `.` domain. Their main purpose is to advertise the Name Servers of top-level domains, like `.org` and `.com`.

and configure it with security in mind, then you are as safe as you can be. Finally it should be mentioned that at least nine of the thirteen Root Name Servers are running BIND DNS. This is significant because these nodes are considered to be the most vital part of the Internet.

An alternative solution to BIND is djbdns. It was built with security and efficiency in mind. It is very reliable and it serves millions of domain names every day. Its author D. J. Bernstein offers \$500 to the first person to publicly report a verifiable security hole in the latest version of djbdns. Unfortunately, DNSSEC and TSIG are not supported. Encrypted zone transfers can be done with `rsync` and `ssh`. More information about djbdns can be found in [38, 39].

The purpose of this section was to provide the reader with extensive information on how a DNS server can be secured. All these recommendations have been tested and verified as correct not only during the completion of this project, but in a real environment as well. The focus was on preventing the attackers from gaining `root` access to the system through a configuration error in the DNS server. Furthermore, the operation of DNS was appropriately secured to prevent a malicious exploitation of the DNS protocol. The result of this misuse would be defective network communication as well as a further exploitation of other protocols.

5 Mail Server Security

Internet Email system is the most commonly used service after the World Wide Web (WWW). For that reason it is very attractive to attackers. Viruses are usually spread with e-mails. E-mail forging, spamming and eavesdropping of confidential data sent by e-mail are also common type of attacks to the Mail system. The Mail server itself is a common target. Exploitation of one of its vulnerabilities enables fundamental threats like information leakage, denial of service, integrity violation and illegitimate use.

The scope of this chapter is to provide defences against these threats, by appropriately securing the mail server. We will install Sendmail v8.12.9 which is the most commonly used mail server on the Internet. It uses the Simple Mail Transfer Protocol (SMTP).

5.1 Sendmail Installation

The source file for Sendmail can be found at <http://www.sendmail.org>. The latest version is recommended because most of the time it fixes all the vulnerabilities of the previous versions. The source code is downloaded and extracted in a temporary directory. If someone wants to change the compilation options*, he should edit the file `devtools/Site/site.config.m4` [File 5.1]. We did not put support for TCP Wrappers in our example as we plan to

```
dn1 local compilation options:
define('confMAPDEF', '-DNIS')
dn1 define('confENVDEF', '-DTCPWRAPPERS')
dn1 define('confLIBS', '-lwrap')
dn1 add support for SMTP AUTH (sasl v2.1.14)
APPENDDEF('conf_sendmail_LIBS', '-lsasl2')
APPENDDEF('confLIBDIRS', '-Lusr/lib/sasl2/lib')
APPENDDEF('confINCDIRS', '-I/usr/include/sasl')
APPENDDEF('confENVDEF', '-DSASL=20114')
```

File 5.1: site.config.m4

enforce access control using internal mechanisms. The file listing includes comments (lines beginning with `dn1`) with the necessary commands in case

*Options passed to the compiler, the default location of the libraries or compatibility with other protocols like Network Information Service (NIS), Lightweight Directory Access Protocol (LDAP) or TCP Wrappers [Section 2.3.11].

someone wants to add support for TCPD as well. We also included support for SMTP Authentication (SMTP AUTH) [Section 5.7].

Command 5.1 Sendmail Installation

```
server$ cd sendmail-8.12.9
server$ ./Build
server# groupadd -g 12 mail
server# groupadd -g 25 smmsp
server# groupadd -g 26 sendmail
server# useradd -u 8 -g 12 -d /etc/mail -s /bin/false \
-M mail
server# useradd -u 25 -g 25 -d /var/spool/clientmqueue \
-s /bin/false -M smmsp
server# useradd -u 26 -g 26 -d /var/spool/mail -s /bin/false \
-M sendmail
server# ./Build install
server# mkdir /var/spool/mail
server# mkdir /var/spool/mqueue
server# mkdir /usr/adm/sm.bin
server# chown sendmail:sendmail /var/spool/mqueue
server# chown root:root /var/spool/mail
server# chmod 700 /var/spool/mqueue
server# chmod 1555 /var/spool/mail
```

The compilation and installation of Sendmail can be done with the commands listed in Command 5.1. We plan to run the mail server as a different user than `root`. By doing so, the security level is extended and in case of an exploitation of a vulnerability it will probably give normal user access instead of administrator privileges. For this reason the system accounts `mail`, `smmsp` and `sendmail` are created. User `sendmail` will only handle the incoming connections. User `smmsp` will write to the mail queue and user `mail` will run the local mailer*. The privileges at these accounts are restricted to these purposes only. All the mails received will be written in the directory `/var/spool/mail/` and only `root` will have access there (each user will only have access to their mailbox which resides there as well). Then we create all the directories and files needed by sendmail, with minimum file permissions [Output 5.1]. User `root` should also be added in group `sendmail`.

*The local mailer gathers all emails for the local users from the mail queue and puts them in their mailbox.

<i>Permissions</i>	<i>Owner</i>	<i>Group</i>	<i>File/Directory</i>
drwx-----	sendmail	sendmail	/var/spool/mqueue/
drwxrwx---	smmsp	smmsp	/var/spool/clientmqueue/
dr-xr-xr-t	root	root	/var/spool/mail/
-r--r--r--	root	root	/etc/mail/sendmail.cf
-r--r--r--	root	root	/etc/mail/submit.cf
-rw-----	sendmail	root	/etc/mail/relay-domains
-rw-----	sendmail	root	/etc/mail/local-host-names
-rw-----	root	root	/etc/mail/access
-rw-rw----	sendmail	root	/etc/mail/access.db
-rw-----	sendmail	root	/etc/mail/aliases
-rw-----	sendmail	root	/etc/mail/aliases.db
-rw-----	sendmail	root	/etc/mail/statistics
-rw-----	root	root	/etc/mail/virtusertable
-rw-----	sendmail	root	/etc/mail/virtusertable.db
-rw-----	root	root	/etc/mail/mailertable
-rw-----	sendmail	root	/etc/mail/mailertable.db
-rw-----	root	root	/etc/mail/genericstable
-rw-----	sendmail	root	/etc/mail/genericstable.db
-r-xr-sr-x	root	smmsp	/usr/sbin/sendmail
-r-sr-x---	root	sendmail	/usr/libexec/mailer/procmail

Output 5.1: File permissions for mail

5.2 Sendmail Configuration

5.2.1 sendmail.mc

Sendmail reads its runtime options from `/etc/mail/sendmail.cf` configuration file [40]. This is created from a source file named `sendmail.mc` which is compiled to give `sendmail.cf`. In order for this file to be customized with respect to security, we will create our own version which we will edit. Each directive in `sendmail.mc` [File 5.2] that affects security is explained:

```
define('confDEF_USER_ID', '8:12')
```

Sets the user-id (uid) and group-id (gid) for the mailer. The default option of uid:gid is 1:1 (user = `bin`, group = `bin`) but is recommended to be changed to 8:12 (user = `mail`, group = `mail`) for more restrictive permissions. The reason for this action is that the system account `bin` is more privileged than the `mail` account which has just been created.

```

divert(-1)
# This is a generic configuration file for Linux.
# created by Kapetanakis Giannis

divert(0)dnl
include('/usr/src/sendmail-8.12.9/cf/m4/cf.m4')
VERSIONID('$Id: linux-biliias.mc,v 8.12.9 2003/06/25')
OSTYPE(linux)
DOMAIN(generic)

dnl Security Options
dnl -----
define('confDEF_USER_ID', '8:12')
define('confrUN_AS_USER', 'sendmail')
define('confPRIVACY_FLAGS', 'goaway,noetrn,authwarnings')
define('confSMTP_LOGIN_MSG', 'Mail Server $j $b')
define('confLOG_LEVEL', '14')
FEATURE('access_db', 'hash -T<TMPF> -o /etc/mail/access')
FEATURE('relay_hosts_only')
FEATURE('blacklist_recipients')
FEATURE('delay_checks')

FEATURE('dnsbl', 'blackholes.easynet.nl', "550 5.7.1 ACCESS
DENIED to <"$&f"> thru "$&{client_name}" by easynet.nl
(http://blackholes.easynet.nl/errors.html)", '')

```

File 5.2: sendmail.mc part 1

```
define('confrUN_AS_USER', 'sendmail')
```

When the daemon accepts a new connection, it will change to user `sendmail`. This way all incoming connections are handled by a non privileged user (default is `root` user). In order for this to work, the local mailer must not have the `S` flag defined*. This configuration is very useful on firewalls or servers where users do not have an account.

```
define('confPRIVACY_FLAGS', 'goaway,noetrn,authwarnings')
```

Privacy options are set with this directive. We included `goaway` option which disables all SMTP status queries (`noexpn`, `novrfy`, `noverb`)

*The local mailer flags are defined with the directive of `FEATURE('local_procmail')` on page 59.

and `authwarnings` which adds logging about spoofing attempts. An attacker cannot verify if a user exists (`novrfy`) and cannot expand an address (`noexpn`). ETRN [41] should also be disabled, as it is stated in RFC 2476 [42]. This is done with the `noetrn` command.

Using these options we give the minimum information to an attacker about our site. It is generally true that, the less information you have, the harder it is to break or abuse a system. More details about privacy flags can be found in chapter 5.6 of Sendmail Installation and Operation Guide [40].

```
define('confSMTP_LOGIN_MSG', 'Mail Server $j $b')
```

We also hide the version of our server, by using “Mail Server” as the message which appears when a client connects. `$j` prints our server name (*server.example.com*) and `$b` the time.

```
define('confLOG_LEVEL', '14')
```

The default log level of sendmail is set to 9. We change it to 14 in order for more security events like refused connections, bad file permissions, authentication information and all SMTP connections [40] to be logged. Logs of all incoming and outgoing SMTP commands can be logged with level 15 but be aware that this will produce a large amount of information. Our log files are kept in `/var/adm/messages`.

```
FEATURE('access_db', 'hash -T<TMPF> -o /etc/mail/access')
```

This option turns on the access control database, feature of sendmail. It gives the ability to allow or refuse emails from specific domains or users (spammers, attackers). The format of this file is explained later in Section 5.3.

```
FEATURE('relay_hosts_only')
```

We change the default behaviour of *relay* access control in `/etc/mail/relay-domains` [Section 5.2.3] in order to include host names instead of domain names.

```
FEATURE('blacklist_recipients')
```

Deny specific local users or local hosts from receiving emails.

```
FEATURE('dnsbl', 'blackholes.easynet.nl', ...)
```

Turns on rejection of hosts found in a DNS based rejection list. Such lists are provided in real-time by various sites* and consist of hosts and e-mail accounts used by spammers and flooders [43, 44].

*Some of the real-time blackhole lists are *list.dsbl.org*, *relays.ordb.org*, *dnsbl.njabl.org*, *proxies.blackholes.easynet.nl* and *blackholes.easynet.nl*.

Another useful ability of sendmail is to hide the internal addresses of local hosts when users send emails from these hosts. We masquerade those addresses in order to be shown as *From:user@example.com* instead of addresses like *From:user@host.example.com*. This is done with the entries in the second part of `sendmail.mc` [File 5.3] which are analyzed below.

```

dnl Addressing Configuration
dnl -----
FEATURE('masquerade_envelope')
MASQUERADE_AS('example.com')
EXPOSED_USER('root')
FEATURE('mailertable', 'hash -o /etc/mail/mailertable.db')
FEATURE('genericstable', 'hash -o /etc/mail/genericstable.db')
FEATURE('virtusertable', 'hash -o /etc/mail/virtusertable.db')

dnl Local Configuration
dnl -----
undefine('UUCP_RELAY')
undefine('BITNET_RELAY')
FEATURE('nouucp', 'reject')
FEATURE('smrsh', '/usr/sbin/smrsh')
define('PROCMAIL_MAILER_PATH', /usr/libexec/mailler/procmail)
FEATURE('local_procmail', '', '', 'Pfhn9')
dnl define('LOCAL_MAILER_FLAGS', 'Pfhn9')
MAILER(local)
MAILER(smtp)

```

File 5.3: sendmail.mc part 2

`FEATURE('masquerade_envelope')`

If masquerading is enabled then this directive will cause envelope addresses to be masqueraded, in addition to header addresses.

`MASQUERADE_AS('example.com')`

All emails are masqueraded to be send from *example.com*.

`EXPOSED_USER('root')`

Mails coming from user `root` will not be masqueraded. This is useful because if you receive automated emails, from a large number of hosts, you will be able to separate them more easily*.

*The `crond(8)` daemon produces such automated mails.

```
FEATURE('mailertable', 'hash -o /etc/mail/mailertable.db')
```

This table is used to override the default mail routing for particular domains. It is also used on backup mail servers in case the primary server fails to respond. Each site should have at least one backup server (placed in a different network from the primary if possible) in case the primary server is a victim of a denial of service attack, or is temporarily offline for maintenance reasons.

In order to setup our backup server (*server2.example.com*) to receive all emails for our domain (*example.com*) while our primary server (*server.example.com*, *10.0.0.1*) is down, we should put the following entry in *server2*'s */etc/mail/mailertable*:

```
.example.com smtp:[10.0.0.1]
```

Additionally, the DNS server of *example.com* must include an MX record for the backup mail server [File A.2].

```
FEATURE('virtusertable', 'hash -o /etc/mail/virtusertable.db')
```

This is an alias table, allowing multiple virtual domains to be maintained by a single mail server.

```
FEATURE('smrsh', '/usr/sbin/smrsh')
```

The Sendmail Restricted Shell (*smrsh*) is used instead of the default system shell (*/bin/sh*) for mailing to programs. This way you can control what can be run via email. In the past, attackers used commands inserted in emails. When a user opened the email for reading, the command was executed, causing harm to the system, or revealing information to the attacker. The administrator can control what can be run, by placing symbolic links* in the directory */usr/adm/sm.bin/*, to the binaries allowed to be executed (*procmail*, *vacation*).

```
FEATURE('local_procmail', '', '', 'Pfhn9')
```

We are using *procmail* as the local mailer. A copy of *procmail* is copied in */usr/libexec/mailler/* and although it is turned to *suid*, it can only be executed by accounts belonging to the *sendmail* group [Output 5.1]. Bear in mind that the *S* is not included in the mailer's flags. Both of these actions are needed in order to enable the *confRUN_AS_USER* option (page 56) and avoid running the mail server as *root*.

Having finished the customization of *sendmail.mc*, we can now create the *sendmail.cf* configuration file. The command line is listed in Command 5.2.

*Symbolic links can be set with the command:

```
ln -s /usr/bin/procmail /usr/adm/sm.bin/procmail
```

More information about configuration options and directives that can be used

Command 5.2 `sendmail.cf` creation

```
server# m4 /usr/src/misc/sendmail-8.12.9/cf/cf/sendmail.mc > \  
/etc/mail/sendmail.cf
```

in `sendmail.mc` and `sendmail.cf`, can be found in `cf/README` inside the source tree of Sendmail.

5.2.2 Accept emails for specific hosts only

The mail server can accept emails for more than one host or domain. Those hosts for which our server is responsible, should be listed in the file `/etc/mail/local-host-names`, otherwise emails will be discarded. An entry for our domain must exist as well (*example.com*).

5.2.3 Restricting relay access

The mail server is used to send and receive messages. It is normally configured to accept emails from any user or host, except those explicitly rejected as spammers. The server should also be configured to forward emails sent by local users and hosts, to remote sites. Bear in mind that you should control who has access to use your server in order to relay his emails, otherwise this service will be used by attackers in order to hide their real address and to spam and flood others with unwanted mails. Open relay servers were often used in the past to forward viruses and worms.

Sendmail can control relay access with the `/etc/mail/relay-domains` file. All the domains that should have access to connect to our mail server in order to send email, and only those, should be listed there. If your site handles email for virtual domains, those should be listed here as well. Examples are presented below:

example.com

dialup.domain.com

another.domain.com

If the option `FEATURE('relay_hosts_only')` is defined in `sendmail.mc` [Section 5.2.1], then it is necessary for individual host names to be included as well. This file (`/etc/mail/relay-domains`) can also be used to give access to everyone to send email to particular domains or hosts, but this is not considered to be a secure solution, as anyone can take advantage of this option and flood the remote site. More options about relay access control can be found in the next section.

5.3 Access Control

Sendmail has internal mechanisms for controlling who can send emails to our server and to our users. We have already enabled this mechanism with the `FEATURE('access_db')` in the customization of `sendmail.mc`. The configuration file exists in `/etc/mail/access` [File 5.4].

Using this mechanism we can control who has access to our mail server. We can deny access to certain users, hosts or even whole domains. This is particularly helpful in case of spammers. This access control mechanism can also be used to allow relaying from certain hosts or disable specific local users and hosts from receiving email.

Command 5.3 Database files creation

```
server# makemap hash /etc/mail/access < /etc/mail/access
```

An example of this file is listed in File 5.4. Explanation for each entry is also listed in a comment above each statement. The actual file that Sendmail reads from is `/etc/mail/access.db`. In order to create this file from `/etc/mail/access` we use Command 5.3. The same command can be used for the creation of `genericstable.db`, `mailertable.db` and `virtusertable.db` [45].

5.4 Resource Limits

Sendmail has options in order to defend against spam and denial of service attacks by applying limits to your resources (hard disk, CPU). These must be adjusted to match your specific requirements.

```
define('confCONNECTION_RATE_THROTTLE',3)
```

This directive limits the number of new connections per second per daemon. Further connections will be delayed. This is very helpful against DoS but it can cause denial of service attack to legitimate users, if your site deals with large volumes of traffic.

```
define('confMAX_DAEMON_CHILDREN',12)
```

This limits the number of processes (children per daemon) sendmail will permit. New connections will be rejected.

```
define('confMAX_MESSAGE_SIZE',2097152)
```

Maximum message size is limited to 2MB.

```
# reject mails from this user
spammer@domain.com      REJECT

# reject mails from all users in this domain
spamdomain.com         REJECT

# accept mails from this user even if his domain
# is denied with the previous entry
friend@spamdomain.com  OK

# allow relaying from and to any host in the sendmail.org
# if you also use FEATURE('relay_hosts_only') in sendmail.mc
# then each host should be explicitly specified
sendmail.org           RELAY
host.sub.sendmail.org  RELAY
192.168                RELAY
IPv6:1:2:3:4:5:6:7     RELAY

# reject mails from this user in every domain
spammer@  ERROR:550 Spam not accepted
From: spammer@domain  REJECT

# allow mails
To:example.com         RELAY
Spam:spam.domain      FRIEND

# if FEATURE('blacklist_recipients') is used we can prevent
# certain local users and hosts from receiving mails
badlocaluser@         ERROR:550 Mailbox disabled for this user
client3.example.com   ERROR:550 That host does not accept mail
user@client4.example.com ERROR:550 Mailbox disabled

# avoid checking your local network against blacklists
# you can also add domains that you are sure they are no
# spammers but they were added in a blacklist by mistake
Connect:10.0.0        OK
Connect:127.0.0.1     OK
Connect:ok_domain.com OK
```

File 5.4: Access control for connections to mail server

```
define('confMAX_HEADERS_LENGTH',16384)
    Limits the header size of incoming emails.

define('confMAX_RCPTS_PER_MESSAGE',25)
    Limits the number of recipients per message.

define('confQUEUE_LA',8)
    CPU load average at which queue-only function is permitted.

define('confREFUSE_LA',12)
    CPU load average at which incoming SMTP connections are refused.

define('confMIN_FREE_BLOCKS',100)
    Minimum number of free blocks on queue filesystem to accept email.
```

These directives must be inserted, and tuned to your requirements, in `sendmail.mc` configuration file [Section 5.2.1]. More information about resource limits can be found in `cf/README` and [46].

5.5 Fighting Spam Mail and Viruses

5.5.1 Internal mechanisms

Spam mail or junk email is received by everyone in exceedingly large volumes. It is very annoying and is a problem that the user cannot do much about it. Filters can be added but long after the email is received and identified as an unwanted spam mail. On the other hand we can apply various mechanisms on the mail server, to deal with and partially solve this problem.

In the previous section we already specified how to reject mails coming from spammers. Domains and hosts that are used to send spam mail, should be placed in that access control list. Restricting relay access [Section 5.2.3] is also a defence mechanism but, in order for this to work efficiently, it should be applied to all the mail servers globally. This way we can be sure that only local users can send emails. If one of them starts flooding others with unwanted email, we can easily identify him and deny him access to the mail service.

As it was mentioned in the customization of `sendmail.cf` [Section 5.2.1], the `FEATURE('dnsbl')` directive is used for real-time rejection of hosts and e-mail accounts that belong to spammers. This is a very interesting feature of Sendmail, because it stops a great amount of junk mail reaching your mailbox. The only problem with this is that there is a possibility of normal email being rejected as well, if a domain is inserted by mistake in a blackhole

list. If that is the case, you can add that domain in `/etc/mail/access` and allow users from that domain to send mails to your users.

In addition to spam rejection we should block emails containing viruses. Sendmail is powerful enough to block viruses and worms, provided that you “teach” it how to do it. This can be accomplished by rule-sets inserted in `sendmail.mc` [Appendix B.1 on page 117]. These rule-sets must contain a characteristic (signature) of the virus you wish to block. Depending on the malicious code you want to detect, these rules can become extremely complicated. What’s more, this is only useful in case of an emergency; sendmail is a mail server, not an anti-virus software that will detect every possible virus that exists or will appear in the future. Anti-virus software scanning every email sent to or from your users, is considered to be a better solution [Section 5.5.3].

5.5.2 External mechanisms for spam mail

Additional software can be used to reject spam mail [46]. We will use a combination of an email parser (MailScanner) and a spam detector (SpamAssassin). We download these tools from <http://www.mailscanner.info> and <http://spamassassin.org> and compile them using the commands listed in Command 5.4. The MailScanner installation was transformed in order to

Command 5.4 Installing MailScanner

```
server# cd /opt
server# gtar -zxvf MailScanner-4.21-9.tar.gz
server# ln -s MailScanner-4.21-9 MailScanner
server# chown -R sendmail:sendmail MailScanner-4.21-9
server# chmod 700 MailScanner-4.21-9
server# mkdir /var/spool/mqueue.in
server# chown sendmail:sendmail /var/spool/mqueue.in
server# chmod 700 /var/spool/mqueue.in
server# mkdir -p /var/spool/MailScanner/incoming
server# mkdir /var/spool/MailScanner/quarantine
server# mkdir /var/spool/MailScanner/spamassassin
server# chown -R sendmail:sendmail /var/spool/MailScanner
server# chmod -R 700 /var/spool/MailScanner
```

run the program as a different user than `root` for security reasons. The file permissions and file ownerships are also more strict than the default. The installation of SpamAssassin is not covered here as we followed the default

recommendation found in the `INSTALL` file which is located inside the source tree.

The configuration file for MailScanner is `MailScanner.conf` and can be found in `/opt/MailScanner/etc/`. The default entries must be changed in order to support spam detection and virus detection [Section 5.5.3]. We also instructed the program to run using the non-privileged account `sendmail` instead of `root`. The entries that must be changed in order for these to work are listed in File 5.5.

```
Run As User = sendmail
Run As Group = sendmail

Incoming Queue Dir = /var/spool/mqueue.in
Outgoing Queue Dir = /var/spool/mqueue
Incoming Work Dir = /var/spool/MailScanner/incoming
Quarantine Dir = /var/spool/MailScanner/quarantine

MTA = sendmail
Sendmail = /usr/sbin/sendmail

Virus Scanning = yes
Virus Scanners = f-prot clamav panda
Spam Checks = yes
Spam List = ORDB-RBL
Spam Domain List = Easynet-DNSBL Easynet-Proxies
Use SpamAssassin = yes
```

File 5.5: MailScanner.conf

5.5.3 External mechanisms for viruses

A computer virus was defined by Fred Cohen as *a program that can “infect” other programs by modifying them to include a possibly evolved copy of itself* [47, 48]. Nowadays, computer viruses, have evolved to be the most significant security problem for the home user. The presence of anti-virus software is absolutely necessary for a workstation that is connected to the Internet. Both Windows and Unix systems are vulnerable to viruses, Trojan horses, logical time-bombs and worms [49, 50].

Most PCs are getting infected by malicious code which is attached to emails. Users, although they need protection from viruses, do not have the

security expertise to deploy an anti-virus solution or even update it as often as it is needed*. In addition to anti-virus software in each workstation, we should add it to our mail server to scan all incoming and outgoing emails for viruses.

There is a vast number of anti-virus products available for use. Some of them are free of charge and other free for non-commercial use. However most of them are not free and need to be purchased. Some of those are:

F-Prot Antivirus for Linux Workstations, <http://www.f-prot.com>

Free for personal and no commercial use. Suitable for home users. This edition was used in this project.

F-Prot Antivirus for Linux Mail Servers, <http://www.f-prot.com>

Needs to be purchased. Both of these are considered to be very efficient anti-virus products.

Clam AntiVirus, <http://clamav.elektropro.com>

Freeware anti-virus. This was tested in this project in combination with F-Prot and the result was more than satisfactory. This is also free for other Unix systems like Solaris, FreeBSD, OpenBSD, NetBSD, AIX and Mac OS X.

AntiVir MailGate for Linux, <http://antivir.de>

AntiVir MailGate for Linux is also free of charge for private (individual, non-commercial) use.

Panda Antivirus for Linux, <http://www.pandasoftware.com>

Panda is freeware for Linux but has no technical support. There is not support for automatic signature updating for the freeware edition.

InoculateIT, <http://www3.ca.com>

It is not clear if InoculateIT is free of charge or not. Nevertheless, it can be downloaded[†] without any restrictions. It is apparently the older version of eTrust Antivirus, which is not free. Binaries for other Unix platforms can be as well.

All of these tools can be used with MailScanner for virus detection. You can also combine two or more together [File 5.5]. Automatic updating of signature files (new virus definitions) should be

*Dieter Gollmann defines this as the fundamental dilemma of computer security: *Security-unaware users have specific security requirements but usually no security expertise.* [51].

[†]<ftp://ftp.ca.com/pub/getbbs/linux.eng/inocstar.LINUX.Z>

scheduled at least once a day. An alternative to MailScanner is AMaViS (<http://www.amavis.org>) and AMaViS “Next Generation” (<http://www.sourceforge.net/projects/amavis>).

5.6 E-mail Gateway

E-mail servers are often placed outside the internal network because they are common targets for attack. A good approach for this is to have multiple mail servers; one outside that receives incoming emails and forwards outgoing to remote servers*, and one (or more) inside the internal network, that receives emails from the gateway, and tunnels all the emails from users to the gateway [Figure 4].

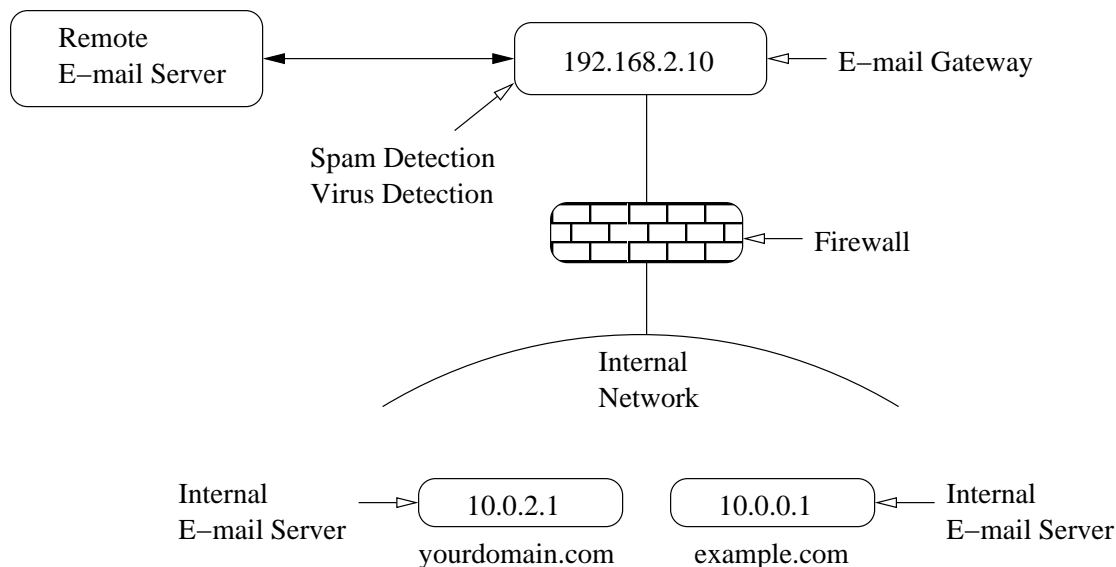


Figure 4: E-mail gateway

By applying this technique we avoid possible threats to the rest of our network in case an attacker gains access to the e-mail gateway. Extending our configuration to support this is no more complex than the previous configuration. Suppose we want to handle emails for two domains:

- *example.com*. The DNS server of *example.com* must have an MX record in its zone file that the primary mail server for this domain is the e-mail gateway with IP-address *192.168.2.10*. The e-mail gateway must forward all emails with address *user@example.com* to the internal

*This server is called e-mail gateway and is usually placed on a DMZ.

mail server with IP-address *10.0.0.1*. This is done with an entry in `/etc/mail/mailertable` [File 5.6].

- *yourdomain.com*. The DNS server of *yourdomain.com* must have an MX record in its zone file that the primary mail server for this domain is the e-mail gateway with IP-address *192.168.2.10*. The e-mail gateway must forward all emails with address *user@yourdomain.com* to the internal mail server (*10.0.2.1*). This is done with an entry in `/etc/mail/mailertable` [File 5.6].

<code>.example.com</code>	<code>smtp:[10.0.0.1]</code>
<code>example.com</code>	<code>smtp:[10.0.0.1]</code>
<code>.yourdomain.com</code>	<code>smtp:[10.0.2.1]</code>
<code>yourdomain.com</code>	<code>smtp:[10.0.2.1]</code>

File 5.6: Mail routing with mailertable

The e-mail gateway must also include the names of the two domains in `/etc/mail/relay-domains` and must not include them in `/etc/mail/local-host-names`. All mails send by our users should be forwarded to the appropriate internal mail server, then to the e-mail gateway and finally to the remote mail server. The opposite direction will be followed when a user receives an email.

5.7 SMTP Authentication

The procedure of receiving and sending emails is not particularly secure. Usually users are not authenticated to the mail server. Thus anyone who can relay mails through your mail server, (any user in the local network) can masquerade as any other. The mail transfer protocol [52] is quite old and security was not the first priority back in 1982. In the more recent RFC 2554 [53], extensions were added in the SMTP to provide authentication for the users.

This extension is a profile of the Simple Authentication and Security Layer (SASL) [54]. In order for SASL to be supported, sendmail must be linked with the Cyrus SASL library (<http://asg.web.cmu.edu/sasl>) when it is compiled [File 5.1]. The authentication methods that are supported include Kerberos 5 (GSSAPI), Kerberos 4, MD5 hashed passwords and plain unencrypted passwords. Furthermore, the authentication can be extended to cover not only communications between a server and a client, but between

servers as well. Documents exist [55, 56, 57] that provide help on the setup of SMTP Authentication.

5.8 STARTTLS

Besides authentication, confidentiality is a need for email services. If your email traffic is not protected, then an attacker can view private or sensitive data like passwords transmitted with email, or even change the data without being noticed if integrity mechanisms are not in place. Confidentiality and privacy can be achieved with encryption.

The TLS protocol can be used within the SMTP protocol in order to provide the extra security that is needed [58]. TLS, more commonly known as SSL, is a popular mechanism for enhancing TCP communications with privacy and authentication. It provides a secure layer to upper layer protocols like, HTTP, SMTP and POP3. Strong confidentiality, data integrity, entity authentication and data origin authentication are security services that the TLS/SSL protocol offers [59].

Sendmail can be configured to use STARTTLS in order to support the TLS/SSL protocol. STARTTLS and SMTP AUTH are particularly useful in case of dial-up users, remote laptop users and users who connect by using wireless networks. These users are connecting from insecure networks that is why strong authentication is required. Apart from clients, mail servers can authenticate each other in order to allow or refuse relaying of emails. This can be achieved with digital certificates as well. An administrator can obtain these certificates by setting up his own Certification Authority [Section 3] or from a commercial CA. Guidelines for configuring Sendmail with STARTTLS exist in [60, 61, 62].

However, there is a disadvantage to this solution. It does not provide end to end encryption (sender to receiver). The traffic is only secured between the user and the server or between two servers. Beyond that we cannot ensure that remote administrators enforce the same mechanisms. If they do not, the message will be in plain text, readable by anyone who is able to eavesdrop on the network. Additional mechanisms should be enforced to ensure end to end security, like S/MIME and PGP [59].

5.9 Sendmail Alternatives

Sendmail is the most common mail server on the Internet. It is supported by many operating systems and is included in the distribution of most of them. However, Sendmail has a reputation of being insecure because a large number of vulnerabilities have been discovered for it.

Sendmail used to be a `suid` program, meaning that the server ran with root privileges. This helped exploiting its weaknesses more easily than other mail servers. This configuration is now changed, making sendmail more secure than it used to be. In addition to this, we should add that the developers of Sendmail have always had a rapid response to security alerts, by providing patches shortly after a vulnerability was discovered.

Nevertheless, other mail servers exist that concentrate more on security rather than anything else. Such software is Postfix (<http://www.postfix.org>) which is written by Wietse Venema, the developer of TCP Wrappers. This software was formerly known as VMailer. It was released by the end of 1998 as the IBM Secure Mailer. Documentation about the configuration of Postfix can be found in [63, 56].

Another mail server that its author claims is more reliable and secure than both Sendmail and Postfix is qmail (<http://cr.yp.to/qmail.html>). Information and documentation about qmail can be found in [64, 65, 66].

6 Web Server Security

A large number of administrators argue indeed, particularly that web services is the most important feature of the Internet. Every organisation runs a web server now. Furthermore, those servers accept connections from anyone around the world raising several security issues.

Defacing* a web site, gaining root access on the server, password or pin number sniffing, unauthorized use of resources and denial of service are common types of attacks launched against web servers.

6.1 Apache Installation

Apache software, is a fast, secure and stable web server that is used by the majority of the Internet web sites[†]. The source files of Apache can be downloaded from *<http://httpd.apache.org>*. The latest version 2.0.47 is considered to be the most secure and most stable up to now. We extract the source code in a temporary directory and compile it using the commands listed in Command 6.1.

Command 6.1 Apache installation

```
server$ cd httpd-2.0.47
server$ ./configure --prefix=/usr/local/apache \
--enable-mods-shared=all --disable-info --with-ssl \
--enable-ssl --enable-so
server$ make
server# make install
```

By using these configuration options we enable support for all available modules. This way the binary file of the server will be small, thus running faster (if a minimum number of modules are loaded). An administrator can setup its server to load only the modules needed (the fewer the better), or he can minimize the number of available modules, by customizing the compilation options [Command 6.2]. Support for SSL/TLS was also added in order to create a secure site [Section 6.5].

*Changes on the contents of the web page is order to show that a site has been compromised. This is usually the first page (*index.html*).

[†]According to a survey by Netcraft on July 2003, 63.72% of the web sites run the Apache web server (*http://news.netcraft.com/archives/web_server_survey.html*).

Command 6.2 Enabling support for specific modules only

```
server$ ./configure --prefix=/usr/local/apache \
--enable-mods-shared="access auth auth-digest usertrack \
unique-id ssl vhost-alias rewrite" --enable-so --disable-env \
--disable-cgid --disable-cgi --disable-suexec --disable-info \
--with-ssl
```

6.2 Secure Web Site Preparation

A new directory must be created in order to hold the web site. This should not be the installation directory of the server in order to differentiate the server from the web site. A better solution would be to create a new filesystem (such as `/usr/local/www/`) that will be used only for the web page storage. If the site is not changed often, this new filesystem can be mounted as `read-only` to avoid possible defacement attacks. A `read-only` filesystem can re-mounted as `read-write` only by the `root` superuser.

Two new user accounts are also needed. The first (`apache`) will be used by the server to avoid running the server as `root`. The `apache` account will not have any administrator privileges. The second account (`wwwadm`) will be used by the web master. All the pages will be owned by that user. User `apache` will not have any write access to the web space and will only have read access to the server's configuration files. User `wwwadm` will not have any access at all to the configuration files of the web server. If one of the two accounts is compromised the security of the other and the system will not be in danger. Command 6.3 lists the necessary commands.

Command 6.3 Secure web site preparation

```
server# groupadd -g 48 apache
server# groupadd -g 49 wwwadm
server# mkdir -m 711 /usr/local/www
server# useradd -u 48 -g 48 -d /usr/local/apache -s /bin/false \
-M -c "Web server admin" apache
server# useradd -u 49 -g 49 -d /usr/local/www -s /bin/false \
-M -c "Web Master" wwwadm
server# chown -R wwwadm:wwwadm /usr/local/www
server# chown -R root:root /usr/local/apache
server# chmod 700 /usr/local/apache
```

6.3 Apache Configuration

The server's configuration file is `/usr/local/apache/conf/httpd.conf`. Most of the security violations are caused by bad configuration [68]. Having said that, configuring Apache to give the minimum authorization to the web clients is essential to provide an adequate level of security. The configuration file [File 6.1 & 6.2] is broken into two parts. The security features of each will be explained.

```
# Listen on specific IP address only
Listen 10.0.0.1:80
ServerName www.example.com
HostnameLookups Off

# run the server as a non-root user
User apache
Group apache

ServerAdmin admin@example.com
ServerRoot "/usr/local/apache"
DocumentRoot "/usr/local/www"
UserDir disabled root

# hide server version
ServerTokens Prod
ServerSignature Off

# strict permissions for the files and directories
<Directory />
    Options None
    AllowOverride None
    Order Allow,Deny
    Deny from All
</Directory>
```

File 6.1: Apache configuration file `httpd.conf` part 1

Listen 10.0.0.1:80

The server will be restricted to bind only to IP address *10.0.0.1* and port number *80*. This is necessary in case multiple network interfaces exist or the server runs on an aliased interface*.

HostnameLookups Off

The server will not try to resolve any address. This improves the performance of the web server (and the DNS server) and it decreases the possibility of a spoofing attack.

User apache

Run the server as the *apache* user instead of *root*.

ServerRoot "/usr/local/apache"

Define the directory where Apache is installed. The server will search there for its configuration files.

DocumentRoot "/usr/local/www"

The web pages are stored in */usr/local/www/*. Only this directory will be available to the public.

UserDir disabled root

User *root* shall not have any personal web page. This is strongly recommended by the Apache development team [69].

ServerTokens Prod

By using this directive we hide the O/S name and the version of our Apache server. The default entry is for the server to send *Apache/2.0.47 (Unix) mod_ssl/2.0.47 OpenSSL/0.9.7b*. We changed it to send only *Apache*.

ServerSignature Off

In older versions of the Apache software this could be a leak of the version of your server. Since version 2.0.44, the version number is controlled by the **ServerTokens** directive. The signature directive can be set to **on** without any information leakage.

*An alias interface can be created by the following command:

```
ifconfig eth0:0 10.0.0.101
route add -host 10.0.0.101 dev eth0:0
The alias interface can be deactivated with the command:
route del 10.0.0.101 ; ifconfig eth0:0 down
```

```
<Directory "/usr/local/www">
# Options FollowSymLinks
  AllowOverride None
  Order allow,deny
  Allow from all
</Directory>

<Directory "/usr/local/www/CA">
  Options Indexes
  Order allow,deny
  allow from example.com
</Directory>

AccessFileName .htaccess
<Files ~ "\.ht">
  Order allow,deny
  Deny from all
</Files>
```

File 6.2: Apache configuration file `httpd.conf` part 2

`<Directory />`

This directive will enclose a group of directives that will apply to the directory / (root filesystem) and all its subdirectories. Default security options for the directories will be included inside this statement. If some of the directories need a special behaviour, it will be explicitly specified by another directive. Such entries must specify the real filesystem path and not the path listed on the web browser.

`Options None`

None of the features will be available by default. Such features may permit the execution of CGI scripts (`ExecCGI`), following of symbolic links (`FollowSymLinks`), server side includes (`Includes`) generation of directory indexes (`Indexes`) and combined features of the above. Such options increase the security threats and should be applied only when they are absolutely necessary. They must also be applied in specific directories only such as `/usr/local/www/cgi-bin/`.

`AllowOverride None`

When this directive is set to `None`, then `.htaccess` files are completely ignored. Such files are used to override default access control, user

authentication and authorization options. If this directive was not set, then each user could create an `.htaccess` inside his home page bypassing the security options set by the administrator. These files should not be available for reading to anyone. This is specified with the last entries of File 6.2.

Order Allow,Deny

This sets the ordering of access control. Any client which is not explicitly allowed, will be denied access by default. This should be set inside all `Directory` directives.

Deny from All

Control which hosts are denied access to the server. Valid entries are domain names (*bad-domain.com*), specific hostnames inside a domain (*bad-host.example.com*), IP address of a host (*10.0.2.100*), a partial IP address (*10.2*), network/netmask pair that defines a subnet (*10.1.0.0/255.255.0.0*), network/nnn CIDR specification (*10.1.0.0/16*) or even IPv6 addresses and IPv6 subnets. Multiple entries can also be specified. The default is to deny access to everyone (`All`). In order to allow access from particular addresses, the directive `Allow from` must be specified.

6.4 User Authentication

Web sites are usually accessible by anyone in the world. An administrator can minimize the access to particular web pages by applying an access control list. Such ACLs may include hostnames, domains or IP addresses as shown in the previous section. This type of access control is not user-based. It is host-based, meaning that every user in a host, that is allowed access, will be permitted to view the protected web page.

In order to extend the security level of a site, it might be necessary to enforce user-based access control. For this kind of control to take place, user authentication is a prerequisite. The Apache server has three internal mechanisms to enforce user authentication; Basic authentication, Digest authentication and SSL Client authentication.

6.4.1 Basic authentication

Basic authentication implies that the user is authenticated according to his identity in the network (IP address, domain name) or the user-id and password he submits [70].

For example, an administrator might want to setup a web page with sensitive data that must be visible only to valid users (*http://www.example.com/protected/*). The password file is the first that should be configured. Good practice is to place it in a directory that is not accessible from the web. We will create a directory named */etc/httpd/* and we will place the password file inside that directory. New users can be created using the following command.

Command 6.4 Creating new users for basic web authentication

```
server# mkdir -m 700 /etc/httpd
server# chown apache:apache /etc/httpd
server# htpasswd -cm /etc/httpd/passwd user1
server# htpasswd -m /etc/httpd/passwd user2
```

The `htpasswd(1)` program is included in the apache installation and resides in */usr/local/apache/bin/*. The argument `-c` must be used only once, when the password file does not exist in order to create it. The `-m` argument specifies that the MD5 hash function should be used to *encrypt** the password. A group file can also be created for a more flexible administration[†].

Consequently, a **Directory** entry must be inserted in the configuration file of Apache [File 6.3]. Bear in mind that the password is protected only locally on the server. The web client's browser sends the password unencrypted and it can be hijacked by anyone who manages to place a sniffer on the network. A replay attack can be easily performed after the password is gathered. This method should therefore not be used for highly sensitive data.

6.4.2 Digest authentication

Digest authentication is a more secure authentication method. The password is not sent at all from the client to the server. What is done now is that the server produces a *nonce*[‡] which is sent to the client. The client calculates the hash of the nonce and the password. The result is transmitted in cleartext. The server makes the same calculation and if they match, user is successfully authenticated [71].

*A hash function does not encrypt, although hashed passwords are often called encrypted. Other functions that can be used here are the SHA-1 and the `crypt(3)` functions (`-s`, `-d` arguments). It seems that SHA-1 does not work properly with this version of Apache.

[†]Each line of the group file should contain the group name and its members:

```
group-name: user1 user2
```

[‡]A nonce is a random number that is used only once. Nonces are often used in challenge-response protocols for strong entity authentication.

```

<Directory "/usr/local/www/protected">
  Options None
  AllowOverride None
  AuthType Basic
  AuthName "Protected Web Space"
  AuthUserFile /etc/httpd/passwd
  AuthGroupFile /etc/httpd/group
  Require group group-name
  Order deny,allow
  Deny from all
  Allow from 10.0.0
  Satisfy All
</Directory>

```

File 6.3: Web client basic authentication, `httpd.conf` part 3

New users can be created with Command 6.5. The password file is now

Command 6.5 Creating new users for digest web authentication

```

server# cd /etc/httpd
server# htdigest -c digest_passwd "Private Area" user3
server# htdigest digest_passwd "Private Area" user4

```

kept in `/etc/httpd/digest_passwd`. One more argument is now inserted in command line `htdigest(1)`, the name of the realm (Private Area), which is also included in the calculation of the hash. A protected site can be setup as it is shown in File 6.4.

```

<Directory "/usr/local/www/private">
  AuthType Digest
  AuthName "Private Area"
  AuthDigestFile /etc/httpd/digest_passwd
  AuthDigestGroupFile /etc/httpd/group
  Require valid-user
</Directory>

```

File 6.4: Web client digest authentication, `httpd.conf` part 4

A quite different client authentication method is the SSL Client authentication where the user is authenticated by using a certificate that has been provided for him. This method is presented in Section 6.5.1.

6.5 SSL/TLS Secure Web

The mechanisms analyzed so far have focused on protecting web servers from attackers, without mentioning how to protect the clients from a masqueraded or fake site*. What SSL/TLS adds to this scheme, is security services like confidentiality (privacy), data integrity and authenticity (server, client authentication). A server can be authenticated to a client if this is necessary as in most commercial sites. Furthermore, the traffic can also be encrypted to avoid information leakage to eavesdroppers. The SSL/TLS protocol combines both public-key and symmetric-key cryptography as well as hashing [59].

Our Apache server already includes support for SSL. A digital certificate must also be created which must be signed by a trusted CA. Administrators should order a certificate from companies like RSA Security or Verisign if they plan to run a commercial site. A different approach is to create and manage your own CA which will issue certificates for all of your servers†. Such a certificate has already been created in Section 3.7. Commands 6.6 must be executed to install the certificate inside the Apache configuration files.

Command 6.6 Put certificates in place

```
server# cd /usr/local/CA
server# cp wwwcert.pem /usr/local/apache/conf/ssl.crt/server.crt
server# cp wwwkey.pem /usr/local/apache/conf/ssl.key/server.key
server# cat cacert.pem > chain.crt
server# cat rootcert.pem >> chain.crt
server# mv chain.crt /usr/local/apache/conf/ssl.crt/
```

The SSL configuration is kept in `/usr/local/apache/conf/ssl.conf`. A virtual host must be created in order to setup a secure web site [File 6.5]. The server's default configuration is to support all the SSL/TLS protocols‡. We instructed the server to use only the strong and medium strength algorithms of SSLv3 and TLSv1. If export control restrictions do not allow the use of such strong algorithms, an administrator can add support for non-restricted ones. Bear in mind that these algorithms are not secure as they use smaller keys (i.e. DES-56, DES-40, RC2-56, RC2-40, RC4-56, RC4-40).

*For example a user browsing in a site that appears to be *amazon.com*, to purchase a book, enters his credit card details which finally fall into the hands of a malicious person.

†A third approach would be to use Command 3.2 to issue a self signed certificate. This is the less secure option.

‡SSL version 2, SSL version 3 and TLS version 1.

```
# General setup for the virtual host
<VirtualHost _default_:443>
    DocumentRoot "/usr/local/www"
    ServerName www.example.com:443
    ServerAdmin admin@example.com
    ErrorLog logs/error_log
    TransferLog logs/access_log

    SSLEngine on
    SSLProtocol all -SSLv2
    SSLCipherSuite HIGH:MEDIUM
    SSLCertificateFile \
        /usr/local/apache/conf/ssl.crt/server.crt
    SSLCertificateKeyFile \
        /usr/local/apache/conf/ssl.key/server.key
    SSLCertificateChainFile \
        /usr/local/apache/conf/ssl.crt/chain.crt
    SSLCARevocationFile \
        /usr/local/apache/conf/ssl.crl/ca.crl
</VirtualHost>
```

File 6.5: Secure web site configuration, `ssl.conf` part 1

We have only included support for algorithms with symmetric key size, more or equal to 128 bits, and an asymmetric key size of 1024 bits*. More information about these algorithms can be found in the Handbook of Applied Cryptography [20].

The server is now ready to be started[†]. This can be done using Command 6.7.

Command 6.7 Starting an SSL Apache server

```
openssl# /usr/local/apache/bin/apachectl startssl
```

The secure site can be viewed with a web browser that supports SSL and the address `https://www.example.com`. Because the certificate is signed by our *unknown* CA, the browser will ask for user confirmation in order to accept the web certificate. An administrator can setup a web page which will

*The stronger key exchange method supported by SSL is the *Ephemeral Diffie-Hellman* method [59]. The stronger encryption algorithms supported are AES-256 and 3DES-168.

[†]If the **ServerName** `www.example.com` in File 6.5 is not the same as the **Common Name** which was requested by Command 3.6, the server will not start.

instruct the users, how they can download and install the root certificate. If the users (manually) insert that certificate in their browser, then the secure site will be automatically reached. Although the web certificate includes a URL page from which the root certificate can be found [File B.2], browsers do not search there*.

6.5.1 SSL client authentication

Additionally, the SSL/TLS protocol supports client authentication. The client (user) must present a certificate issued by a trusted CA[†].

The user certificate has already been created in Section 3.7. The certificate and the corresponding private key is given to the user. Caution must be taken in this procedure as the private key must remain secret and protected. If an attacker manages to access the key, he would be able to masquerade as this valid user. A smart card, where data are stored in a secure manner, might be a good solution.

In addition to that, the certificate and the key must be installed in the user's browser. In order for this to be done, we must change the format of the file from PEM to PKCS#12 [Command 6.8].

Command 6.8 Creating a PKCS#12 certificate

```
server# openssl pkcs12 -in usercert.pem -inkey userkey.pem \  
-des3 -export -out usercert.p12
```

Highly sensitive data should be placed in a separate directory in order to be accessed from the web (*https://www.example.com/secured/*). The server must also be instructed to verify the user's certificate to authenticate him in that directory. The listing of File 6.6 should be placed in `ssl.conf`.

*Mozilla and Konqueror, which were the browsers tested, ignored the URL defined inside the root certificate. It is not clear if it is a bug of the browsers or a deliberate response in order to avoid attacks where fake root certificates are advertised.

[†]Our certification authority will be trusted by our web server, since we will manually insert the root certificate in the web server.

```
<Directory "/usr/local/www/secured">
  SSLVerifyClient require
  SSLVerifyDepth 2
  SSLCACertificateFile /usr/local/apache/conf/ssl.crt/chain.crt
  order allow,deny
  allow from 10.0.0
</Directory>
```

File 6.6: Web SSL client authentication, `ssl.conf` part 2

The server searches in `/usr/local/apache/conf/ssl.crt/chain.crt` to find the certification chain file of the certification authorities that have signed the user's certificate. All the certificates, up to the root CA, are verified. The `SSLVerifyDepth 2` directive, instructs the server of the depth of the certification chain. The certificate revocation list which was created with Command 3.10 is also examined by the server in case a user certificate has been revoked*.

In addition to this, normal (not protected by SSL) access must not be allowed in this directory. This is specified in `httpd.conf` [File 6.7].

```
<Directory "/usr/local/www/secured">
  order allow,deny
  deny from all
</Directory>
```

File 6.7: Deny normal access to a web page

The SSL client authentication can be combined with the previous methods of authentication (basic, digest) [72]. Furthermore certificates can only be requested if the client is browsing from the Internet, instead of the local network.

6.6 Conclusion

The threats and the security consideration of running a Web server were presented in this section. Firstly, the server must be installed and configured in a secure manner that protects itself as well as the rest of the system from an exploitation. Among the techniques used to achieve that, was the separation of the server and the web pages by creating non-privileged user

*The path of the file with the certification revocation list is specified in File 6.5.

accounts. Furthermore, the configuration was based on restricting access to options and features that are not necessarily needed.

The setup was followed by the creation of a secure web space where all traffic relies on the SSL protocol to be encrypted. The supported authentication mechanisms for both clients and servers were presented and analyzed.

The final result was that a web server is able to run without threatening the rest of the system. Additionally, advanced security features like encryption, authentication, integrity and access control are all supported in the Apache software. For all the above mentioned and for its performance as well, it is a highly recommended Web Server.

7 Firewalls

Most of the services configured in the Linux/Unix server include some form of access control. This is used to permit or deny access to users or hosts when trying to connect to the server and use network resources. Servers like the Secure Shell and the Web server include such a type of IP-based access control.

Using only the internal mechanisms that each server has for access control, is not secure enough as disadvantages appear through this procedure. An attacker is able to trace that particular services are running on a server by simply connecting to them or by port-scanning the server. If an attacker connects to a service, the server will first open a negotiation channel, lookup his access control list, whether access is allowed, and then it will deny or allow access. Even if the attacker is denied access to the service, it will be made clear to him that the service is available in that server.

Furthermore, if the service is vulnerable and the attacker knows how to exploit it, the access control mechanisms might fail to stop the attacker. Firewalls, software-based or hardware-based, add a lot to the IP-based access control. If connection to a server is denied then the server can be made invisible and completely hidden to the attacker. Additionally, services that do not have any internal mechanism enforcing access control will be protected by the firewall.

A firewall does not only protect your network and your server from remote attackers. It can protect remote hosts from your users if this is required. If your users are taking advantage of your bandwidth, they can be restricted to a certain amount of traffic per service or to a limited number of remote services.

7.1 Different Types of Firewalls

Firewalls come in many types and many security levels. The simpler firewalls are the packet filter and the circuit-level proxy firewalls. More advanced and more secure are the statefull packet filter and the application-level proxy firewall [73].

The TCP/IP packet filter examines each IP packet's header going through the firewall to decide whether the packet is going to be allowed or not. Access control is based on the source and destination IP address and port. The rules must allow for packets in both directions. Certain protocols like FTP and IRC are difficult to be supported.

The circuit-level proxy is setup on a firewall proxy system. Each packet is forwarded at this host and the rules of the firewall determine which connec-

tions are allowed. If a packet is allowed then a new connection is generated from the proxy firewall to the remote host. Access control is based on source and destination IP addresses and ports.

The statefull packet filter is similar to the packet filter but more secure. There exists a connection tracking machine that maintains a list with all the new sessions. If initial packets in one direction are allowed, then the replies are automatically allowed. The filtering rules can be simpler but tighter than those used in the packet filter. In this project, a combination of these two firewall types has been used.

Finally there is the application level proxy. This runs on a firewall proxy like the circuit-level firewall does. The client connects to the firewall and new connections are generated from the firewall to the remote server. Additionally, the access control is based on the traffic content. Filtering rules apply against viruses, exploits or other forms of malicious content. If required, user authentication can be performed on the firewall before access to services is allowed [76]. The disadvantage of an application level firewall is that large amount of processing is required per connection. Furthermore, encrypted traffic cannot be examined.

These types of firewall can either be software-based or hardware-based. The packet filters (statefull and stateless) can run on both a firewall gateway* and a single host like a server. On the contrary, the proxy firewalls cannot be run on a single host. Thus a firewall gateway is required [77].

Linux includes support for a statefull packet filter which is called IPtables. This particular filter can run on both a single host and a firewall gateway. It is distributed for free and its performance is considered one of high standards.

7.2 Firewall Policy Configuration

The firewall reads its policy rules from file `/etc/sysconfig/iptables`. There are two ways to create this file. Either the file is edited to make necessary changes or the rules are inserted directly to IPtables. The second technique will be used here. All the rules will be inserted with a command line to the firewall and when this process is finished the table will be saved into the policy file.

7.2.1 Setting the default policy rules

There are two notions on how the firewall rules should be. The first is that the default policy rule should allow any packet to pass through the

*A host configured only as a firewall. It is used to protect a network.

firewall, while explicitly defined rules must deny access to every packet that should not be permitted. This is less secure as unwanted traffic might be let through the firewall because of a configuration error. Such a mistake is difficult to trace.

A more restrictive firewall configuration is for the default policy to block everything. Then, explicitly defined rules would allow access only to what should be allowed. This is a much more secure concept as only permitted traffic actually passes through. Nevertheless, if access is falsely denied to traffic, a network service will fail to work. Unlike the previous concept, this one's mistakes are easier to trace and deal with.

The *block by default, allow some* type of access control is used in this project as it is considered to be a more secure one. The default policy rules of IPtables can be defined using Command 7.1 which block all incoming traffic.

Command 7.1 Setting the default rules

```
server# iptables -F
server# iptables -X
server# iptables -Z
server# iptables -t nat -F
server# iptables -t mangle -F
server# iptables -P INPUT DROP
```

The disadvantage of this is that no logging will be produced for what is blocked. While configuring the firewall policy, it is very useful to read the log files in order to verify that the firewall blocks only the traffic that must indeed be blocked.

A similar technique is to set default policy to **ACCEPT** instead of **DROP**, and then explicitly define which traffic is accepted and which is rejected. The last rule [Command 7.17] will block everything else in order to avoid a configuration error that allows access to unwanted traffic. The logs are kept in `/var/adm/messages`.

7.2.2 Define traffic policy

The `loopback` interface is a virtual interface installed in every Unix OS. It is used by the system and should not be blocked as it is not accessible from outside. The `loopback` traffic is accepted with Command 7.2. Furthermore, traffic coming from the internet with destination addresses used by the `loopback` interface, is blocked.

Command 7.2 Control the loopback interface

```
server# iptables -A INPUT -i lo -j ACCEPT
server# iptables -A OUTPUT -o lo -j ACCEPT
server# iptables -A INPUT -i eth0 -d 127.0.0.0/8 -j DROP
```

The firewall can be used to block various types of attacks. SYN-flood attacks, where the server is flooded with a large number of valid packets with the SYN flag set, is the most common type of attack. We will limit the inbound traffic to accept only two connections per second, after 5 connections per second have been seen [Command 7.3].

Command 7.3 SYN-flood protection

```
server# iptables -N syn-flood
server# iptables -A INPUT -p tcp --syn -j syn-flood
server# iptables -A syn-flood -p tcp -m limit --limit 2/s \
--limit-burst 5 -j RETURN
server# iptables -A syn-flood -j DROP
```

Another type of attack is used where all the tcp flags are set or none of them is set [74]. These are invalid types of packets and should therefore be blocked [Command 7.4].

Command 7.4 Drop XMAS and NULL TCP port scans

```
server# iptables -a INPUT -p tcp --tcp-flags ALL ALL -j DROP
server# iptables -a INPUT -p tcp --tcp-flags ALL NONE -j DROP
```

Other kinds of SYN attacks are performed when new packets are received with both SYN and ACK flags set, with the FIN flag set or without any SYN flag set. These suggest invalid types of new packets [75]. Furtive port scanners also use a technique where TCP packets with the RST (tcp-reset) flag are sent in order to trace open TCP ports. These packets are valid because they are used to terminate a TCP connection. A limit is placed though that will allow only 2 connections per second with the RST flag set [Command 7.5].

Spoofed packets are packets where the source address is modified. Attackers use spoofing techniques to hide their real IP address so that tracing their location is difficult. Non-routable and private IP addressing ranges are often used for these kind of attacks. The firewall should block all packets that have an invalid source IP address [Command 7.6]. Furthermore, a good practice is for the routers to block every packet coming from the internal network that uses a source IP address different than those used by that network.

Command 7.5 Extensive protection against SYN-flood and furtive port scanners

```
server# iptables -A INPUT -p tcp --tcp-flags SYN,ACK SYN,ACK \
-m state --state NEW -j LOG --log-prefix "New packet with \
both syn/ack:"
server# iptables -A INPUT -p tcp --tcp-flags SYN,ACK SYN,ACK \
-m state --state NEW -j REJECT --reject-with tcp-reset
server# iptables -A INPUT -p tcp --tcp-flags SYN,ACK,FIN FIN \
-m state --state NEW -j LOG --log-prefix "New packet with FIN:"
server# iptables -A INPUT -p tcp --tcp-flags SYN,ACK,FIN FIN \
-m state --state NEW -j REJECT --reject-with tcp-reset
server# iptables -A INPUT -p tcp ! --syn -m state \
--state NEW -j LOG --log-prefix "New packet but not syn:"
server# iptables -A INPUT -p tcp ! --syn -m state \
--state NEW -j REJECT --reject-with tcp-reset
server# iptables -A INPUT -p tcp --tcp-flags SYN,ACK,FIN,RST \
RST -m limit --limit 2/s -j ACCEPT
```

This way we protect remote sites from internal users who try to spoof their source address. If this is applied in every router around the world spoofing attacks can in fact be eliminated.

Command 7.6 Spoofing protection

```
server# iptables -A INPUT -i eth0 -s 10.0.0.1 -j DROP
server# iptables -A INPUT -i ! lo -s 127.0.0.0/8 -j DROP
server# iptables -A INPUT -i eth0 -s 172.16.0.0/12 -j DROP
server# iptables -A INPUT -i eth0 -s 192.168.0.0/16 -j DROP
server# iptables -A INPUT -i eth0 -s 224.0.0.0/4 -j DROP
server# iptables -A INPUT -i eth0 -s 240.0.0.0/5 -j DROP
```

Bear in mind that the private IP range *10.0.0.0/8* is not blocked as it should be. These addresses are being used for our examples, that is why they have not been blocked. Real sites should filter this IP range out. Other unwanted traffic can be blocked using Commands 7.7, 7.8 and 7.9.

Command 7.7 Block unwanted broadcast

```
server# iptables -A INPUT -i eth0 -d 10.0.0.255 -j DROP
server# iptables -A INPUT -p udp --dport 137:139 -j DROP
server# iptables -A INPUT -p tcp --dport 445 -j DROP
```

Command 7.8 Block identd requests

```
server# iptables -A INPUT -i eth0 -p tcp --dport 113 -j REJECT
--reject-with icmp-port-unreachable
```

Command 7.9 Protect X-resources

```
server# iptables -A INPUT -i eth0 -p tcp --dport 6000:6020 \
-j DROP
server# iptables -A INPUT -i eth0 -p tcp --dport 7100:7110 \
-j DROP
```

So far, we have blocked invalid kind of traffic or specific attacks targeting the server. However, various services must be accessible from the Internet. Such services are the Web server, the Mail server and the DNS server. Furthermore, incoming traffic that is valid should not be blocked. Such an example is a connection that the server has initiated, or traffic produced by remote hosts trying to access a publicly available service.

We enable valid incoming traffic using Commands 7.10, 7.11, 7.12, 7.13 and 7.15. Access to the SSH server is only allowed from the host *server2.example.com* with IP address *10.0.0.2*. Remote access to the server must only be allowed to a limited number of hosts. If access to the portmapper is also required, it must be limited to the internal network only [Command 7.14].

Command 7.10 Allow DNS

```
server# iptables -A INPUT -i eth0 -p udp -s 10.0.0.0/8 -d \
10.0.0.1 --dport 53 -m state --state NEW,ESTABLISHED -j ACCEPT
server# iptables -A INPUT -i eth0 -p udp -s external.DNS \
--sport 53 -m state --state ESTABLISHED -j ACCEPT
```

Command 7.11 Allow Web for everyone

```
server# iptables -A INPUT -i eth0 -p tcp --dport 80 -m state \
--state NEW,ESTABLISHED -j ACCEPT
server# iptables -A INPUT -i eth0 -p tcp --dport 443 -m state \
--state NEW,ESTABLISHED -j ACCEPT
```

Command 7.12 Allow Mail for everyone

```
server# iptables -A INPUT -i eth0 -p tcp --dport 25 -m state \
--state NEW,ESTABLISHED -j ACCEPT
```

Command 7.13 Allow SSH from admin's host only

```
server# iptables -A INPUT -i eth0 -p tcp -s 10.0.0.2 \  
--dport 22 -m state --state NEW,ESTABLISHED -j ACCEPT
```

Command 7.14 Allow internal network to access the portmapper

```
server# iptables -A INPUT -s 10.0.0.0/8 -i eth0 -p tcp \  
--dport 111 -m state --state NEW,ESTABLISHED -j ACCEPT  
server# iptables -A INPUT -s 10.0.0.0/8 -i eth0 -p udp \  
--dport 111 -m state --state NEW,ESTABLISHED -j ACCEPT
```

Command 7.15 Allow valid traffic

```
server# iptables -A INPUT -p all -m state \  
--state RELATED,ESTABLISHED -j ACCEPT
```

All the previous rules enabled filtering of TCP and UDP traffic. Apart from these protocols, the ICMP protocol must be filtered as well. Denial of service attacks exist, like the Ping of Death that may crash the server. This kind of attack is performed by sending a large number of PING requests to the victim host. Protection against this kind of attack is enabled by using Command 7.16. Custom policy rules about filtering the ICMP protocol can be found in Appendix C.2.

Command 7.16 Ping of Death protection

```
server# iptables -A INPUT -p icmp --icmp-type echo-request \  
-m limit --limit 1/s -j ACCEPT
```

Finally, as it is mentioned in Section 7.2.1, what is not explicitly allowed must be blocked. The rest of the TCP and UDP traffic will be blocked using Command 7.17. Logging is also enabled in order for other kind of attacks to be documented.

Command 7.17 Block rest of the traffic

```
server# iptables -A INPUT -i eth0 -p tcp -j LOG  
server# iptables -A INPUT -i eth0 -p udp -j LOG  
server# iptables -A INPUT -i eth0 -p tcp -j REJECT \  
--reject-with icmp-port-unreachable  
server# iptables -A INPUT -i eth0 -p udp -j REJECT \  
--reject-with icmp-port-unreachable
```

All the commands so far were used to restrict incoming traffic only. Out-

going traffic can be blocked as well, if it is required, by using `OUTPUT` instead of `INPUT` in the firewall rules. This way an administrator can restrict local users to access specific services only, like Web and Ftp.

When the firewall configuration is finished, the rules must be saved into the policy file. Each time the server reboots these will be automatically loaded. This can be done using Command 7.18.

Command 7.18 Creating and using the policy file

```
server# iptables-save > /etc/sysconfig/iptables
server# chkconfig --level 2345 iptables on
server# /etc/init.d/iptables start
```

7.3 Conclusion

Firewalls are the front line of intrusion prevention. Their role is to block certain types of network attacks as well as applying a network based access control. If a firewall is configured properly, then it will be a key component of your network security as the server and its services will be protected from remote attackers.

Bear in mind though, that most of the attacks come from the inside of the local network. The firewall should also restrict access to the hosts belonging to the internal network, otherwise attackers might take advantage of this.

Furthermore, the firewall must be supported by a proper and secure system configuration. Only the necessary services must be running in each host, while strong user authentication and access control is enforced. In case the firewall runs on a separate system which is responsible for the whole network, no other services should be run in that firewall gateway. Proxy-based firewalls can also be used in these gateways.

If it is needed, user authentication can be performed on the firewall when users from the Internet are trying to access internal services*. The user authentication can be extended to support host authentication as well. Bear in mind that firewalls apply IP-based access control only. IP addresses are easy to be manipulated thus making firewalls less restrictive than we think they are. The IPSEC protocol can be used to provide strong host-based authentication and access control, as it provides with cryptographic means for these purposes.

*IPtables does not support user authentication on the firewall. The closest feature to that, is filtering of the packets according to the user-id, the group-id or the process owner that created the packet.

In the setup that was presented, the firewall was used only for protecting the server. Additional prevention mechanisms should be enforced on the server and the local clients, in order to provide an adequate level of security. Moreover, an Intrusion Detection System [Section 8] can verify the correctness of the firewall and help in case of a successful intrusion.

Finally, it should be added that firewalls must run on all the servers and hosts which run services needing network protection. Routers should also run packet filtering firewalls, in order to protect the whole network.

The very good tutorial on IPtables [78] is recommended to anyone who uses this software. Additional resources on IPtables can be found in <http://www.linuxguruz.com/iptables/>.

8 Intrusion Detection

The main purpose of Computer Security is to protect against security breaches. Prevention is the most important part, that is why it has been covered extensively so far. Nevertheless, there is no such thing as an absolute secure system. There is always the possibility of an attacker to gain unauthorized access to your server despite all your efforts to prevent this. When this is the case detecting the attacker and his actions is necessary in order to maintain security.

Intrusion Detection Systems (IDS) are used to identify, respond and verify malicious activity targeting computers and network resources [79]. Such activities include port scanning, DoS, integrity violations*, unauthorized logins, infection of viruses, trojan horses, root kits† and sniffers. The administrator can recognize the type of the attack and whether it was successful or not by examining the IDS report. If in fact the attack was successful, the system's modifications can be examined and necessary measures can be taken to bring the server back to a secure state.

8.1 Intrusion Detection Techniques

Intrusion detection techniques are separated into three categories; Ad Hoc, Knowledge-based and Behaviour-based [80]. Ad Hoc intrusion detection is a manual procedure through which the administrator examines the system for changes and monitors for unauthorized or suspicious use of resources. It requires a very good understanding of Unix/Linux OS and Computer/Network Security. IDS software is considered to be a more common solution as it provides automation. Nevertheless, manual procedures are always used to verify the correctness of the IDS software and extend its use.

Knowledge-based IDS identify attacks by examining specific patterns of network traffic or log files that indicate suspicious behaviour [81]. These patterns are called *attack signatures*. Most of the anti-virus products use the same technique to detect viruses and worms.

Behaviour-based IDS are based on detection of anomalies without any knowledge of the attack signature [59]. A threshold is placed which defines the normal system activity. If this threshold is exceeded then an alarm is set. The response to the attack in both Knowledge-based and Behaviour-based IDS can be automated or manual.

*The modification of the password file, or another configuration file by an attacker is an integrity violation to the security of your system.

†Root kit, is software that is maliciously placed by an attacker who has successfully bypassed your security perimeter. This way he can easier regain **root** access in the future.

When an IDS looks for signatures or anomalies in network traffic it is network-based. When it looks in system and log files it is host-based. Both of these types will be used in order to maintain an adequate level of security on the server.

8.2 Ad Hoc Intrusion Detection

8.2.1 System log files

The Unix/Linux OS provides with a server, `syslogd(8)`, which is responsible for the logging of various events. Such events include login attempts, servers' activity*, authentication information and kernel messages. The log file is kept in `/var/adm/messages` and should be examined very frequently by the administrator [Section 2.3.10].

Besides the `syslogd` daemon, servers can be configured to keep their logs in different files. For instance the DNS server is configured to keep the security events in `/etc/namedb/security.log`. The Web server stores any access attempt in `/usr/local/apache/log/access_log`. Bear in mind that the system's log files and audit trails might have been modified by an attacker in order to hide his presence or destroy the evidence that suggest illegal actions.

8.2.2 Detection of invalid login records

A computer must be accessed only by authorized users and only from authorized hosts. Furthermore, a server must allow login access only to the administrators. Information about who is currently using the system is kept in `/var/run/utmp` file. This can be viewed with the command `who(1)`. This command prints the username, the login terminal, the time the user logged and the name of the host where the user connects from [Output 8.1]. Additional information like CPU time and the command line of the user's

```
5:59pm server# who
user1 pts/1 Aug 10 09:56
user2 pts/4 Aug 10 17:12
admin pts/0 Aug 10 09:56
root pts/5 Aug 10 17:43 (attacker.hostname)
```

Output 8.1: Listing logged users

*Most of the servers like the Mail and DNS server are configured to report any access to them to the `syslogd` server.

current process can be shown with the command `w(1)`.

A listing of all the users who have successfully logged in the past is also kept. This information is stored in file `/var/log/wtmp` and it can be viewed with the command `last(1)`. Suspicious entries can be found there, like a user login from unauthorized hosts. Unsuccessful login attempts are kept in file `/var/log/btmp` and they can be viewed by using the command `lastb(1)`. If these files do not exist they can be created using Command 8.1. Login attempts are also logged by the `syslogd` daemon.

Command 8.1 Creation of login records

```
server# cd /var/log
server# touch wtmp
server# touch btmp
server# chown root:utmp wtmp btmp
server# chmod 664 wtmp btmp
```

8.2.3 Detection of invalid processes

A server must only run the necessary services and programs. This should be verified by the administrator frequently otherwise an attacker might manage to run malicious software without being noticed. Any user in the system, including the administrator and the `root` user, can list all the current processes by issuing Command 8.2. Information about each process is stored in the `/proc/` pseudo-filesystem.

Command 8.2 Listing of current processes

```
server$ ps auxww | more
```

In case more details are needed, such as the files or the libraries being used by a process, the `lsuf` command can be executed. The administrator should have clean* copies of `ps` and `lsuf` stored in a `read-only` media. Nevertheless, if an attacker manages to replace the system's kernel, he is able to hide as much information as he wants. The same also applies in case he is able to load a module to the kernel. Such kind of attacks have been reported in the past.

*Trojan horses exist that make the program behave as it should but, in addition to that, they hide processes the attacker wants to run without being noticed.

8.2.4 Detection of invalid network services

Servers connected to the Internet, are accessible from remote hosts. Each service binds to a local port* to accept connections. Administrators can list all the open ports (listening and non-listening sockets) in order to discover new network services that have been installed by an attacker using Command 8.3.

Command 8.3 Listing of open sockets and ports

```
server$ netstat -a
server$ netstat -a --inet
server$ netstat -a -p
```

The latter of the above commands, reports the process id and the name of the program which each socket belongs to. A clean copy of `netstat` must also be stored by the administrator in a secure place.

In case the server runs RPC services like NFS or NIS a list of all registered programs can be found with Command 8.4.

Command 8.4 Reporting RPC and mount information

```
server$ rpcinfo -p
server$ showmount -e
```

A more intrusive approach would be to port scan the server for available services. This technique is used by attackers and it usually precedes the main attack. Nevertheless, it is often used by administrators as well, for security reasons. Such a tool is Nmap (<http://www.insecure.org/nmap>). Nessus Security Scanner (<http://www.nessus.org>) is also used for port scanning and vulnerability detection.

In case particular services are only available to hosts belonging to the internal network, port scanning must originate from a host both inside and outside of the internal network. This will verify that your access control[†] is actually taking place. Examples of Nmap's usage are presented in Output 8.2.

*The DNS server accepts connections on port 53. The Web server on port 80. The file `/etc/services` lists all the well-known port numbers assigned to each service.

[†]A firewall can apply the access control and deny access to hosts outside of your internal network.

```

client# nmap -sS -v -v -v server.example.com
Host server.example.com (10.0.0.1) appears to be up ... good.
Initiating SYN Stealth Scan against server.example.com (10.0.0.1)
Adding TCP port 22 (state open).
Adding TCP port 443 (state open).
Adding TCP port 80 (state open).
Adding TCP port 53 (state open).
The SYN Stealth Scan took 0 seconds to scan 1542 ports.
Interesting ports on server.example.com (10.0.0.1):
(The 1538 ports scanned but not shown below are in state: closed)
Port      State  Service
22tcp    open   ssh
53tcp    open   domain
80tcp    open   http
443tcp   open   https

```

Output 8.2: TCP SYN port scan on the server

8.2.5 Detection of invalid SUID files

SUID files are programs that when executed by a normal user they run with `root` privileges. They are a potential security risk and should be monitored closely [11]. The fewer SUID and SGID files there are on a system the better for security reasons.

Attackers who successfully gain access to a server, leave SUID programs (usually shells) in order to regain `root` access even if the original vulnerability is recovered. Administrators should frequently check the system for invalid SUID files [Command 8.5]. If invalid files are discovered throughout

Command 8.5 Detection of SUID and SGID files on the system

```

server# find /. -type f \( -perm -4000 -o -perm -2000 \) \
-exec ls -ld {} \;

```

this procedure, they must be immediately removed. The system should be extensively checked for other traces of an attack as the presence of invalid SUID files suggests that an attacker has successfully gained `root` access on the server. In such a case, the system should not be considered as trusted and actions should be taken to bring it back to a secure state.

8.2.6 Integrity verification

The RedHat Linux OS includes the RPM package manager. All the software is pre-compiled* and installed using RPM files which include binary versions of all the programs. The integrity of these files can be verified as the RPM database holds the hash values and the access permissions of each program. A listing of various commands that are needed for this procedure is shown in Command 8.6.

Command 8.6 Verifying the integrity of RPM software packages

```
server# rpm --query -f /bin/tcsh
server# rpm --query --filesbypkg tcsh-6.10-6
server# rpm --verify tcsh-6.10-6
server# rpm --query -pl tcsh-6.10-6.rpm
```

Bear in mind that if the software is not installed using the RPM utility (like our DNS, Mail and Web server) the integrity of the files cannot be verified with this procedure. Other means of integrity verification are required [Section 8.3.1].

8.2.7 *Spying on your users*

Spying on the local users might not be a legitimate or ethical action as it violates their rights for privacy. Nevertheless, it is a means of protection from them, as it is a fact that most of the attacks originate from people inside the organisation. Additionally, normal user accounts are used by attackers to gain super-user privileges.

The `root` user has the ability to watch all the traffic coming in and going out from the server. This is called *sniffing* or *eavesdropping*. This way anomalies on the network can be detected as well as abusive usage of network services[†]. Common tools used by administrators (and attackers) are `Tcpdump` (<http://www.tcpdump.org/>) and `Sniffit` (<http://reptile.rug.ac.be/~coder/sniffit/>).

The attackers can use SSH, which encrypts the network traffic, in order to hide their actions. However sophisticated tools like `Dsniff` (<http://naughty.monkey.org/~dugsong/dsniff/>) exist which can eavesdrop on encrypted channels. If used, extreme cautiousness should be taken, as it can perform man-in-the-middle and connection hijacking attacks.

*The administrator does not compile the source of the program. The program is compiled for a generic platform and it is automatically installed.

[†]Sniffing is also used by attackers in order to catch user passwords.

Finally programs exist that are able to spy on the user's `tty*`, if an administrator wishes to monitor the intruder's actions. Such tools are `Ttysnoop` (<http://filewatcher.org/sec/ttysnoop.html>) and `Ttywatch` (<http://people.redhat.com/johnsonm/ttywatch.html>) and they can log every command a user or an attacker executes on the system.

8.2.8 Additional traces

If an attacker gains `root` access on the system he will possibly modify various configuration options or even replace programs. The administrator should monitor crucial configuration files, like `/etc/passwd` and `/etc/shadow` where user accounts are stored, for changes. New accounts or accounts without a password should be erased or disabled.

The configuration files of all the servers like the DNS, the Web and the Mail server are important and should be examined for modifications. Changes in file and directory permissions are indications of a successful attack. Changes in `.rhosts` or invalid `su` entries in the log files are also traces of an intrusion. Additionally, the system should be checked for device files outside the `/dev/` directory and files which are not owned by anyone. These are common traces of an attack.

Command 8.7 Detecting device files and unowned files

```
server# find /. \(-type b -o -type c\) -print
server# find /. \(-nouser -o -nogroup\) -print
```

Furthermore, an attacker could have replaced a specific program or installed new to regain access more easily in the future. The system's integrity and correctness must be frequently verified otherwise it cannot be considered as a trusted one [Section 8.3.1]. Moreover these actions should take place when suspicious changes or activities are discovered.

The response to an attack depends on the type of the intrusion, the modifications that took place and the particularities of your organisation. Part V of *Practical Unix & Internet Security* [1] contains instructions on what to do if your computer's security is compromised. It is highly recommended for reading.

Whatever the case may be, the administrator should always keep a recent and uncorrupted backup of important data and configuration files. It is almost unnecessary to mention its value since this is the last resort when everything else fails. What's more, a backup allows you to see what an

*The terminal used by each user to insert commands to the system.

intruder has changed by comparing the files on the computer with the files on the backup.

8.3 Automated Host-based Intrusion Detection

As it was shown in the previous section, manual procedures can be very complex and thus require time and effort. They must take place frequently in order for security to be guaranteed. Furthermore, evidence of an attack might have been destroyed by the intruder prior to the administrator's investigations.

Automated tools provide real time detection and response to attacks. This can be done by an active monitoring of messages as they are written on log files. The Swatch (<http://swatch.sourceforge.net/>) tool does exactly so. When an attack signature is detected, the administrator is immediately notified and automated commands* may be executed to prevent the upcoming attack.

Other tools that might be useful for administrators are Tiger (<http://www.tigersecurity.org/>), a Unix security audit and intrusion detection tool, and ImSafe (<http://imsafe.sourceforge.net/>). The last one is performing anomaly detection at the process level and tries to detect various types of attacks without any prior knowledge of the exact attack signature.

8.3.1 Integrity verification

A successful intrusion is usually followed by the installation of back-door programs. These are tools that either wait for the attacker to issue commands remotely or, when executed, give root access without exploiting any vulnerability. Furthermore, an attacker can replace a valid program, like the Mail server (which accepts connections from everyone on the Internet), or a shell. When a pass-code which is known only to the attacker is given, various instructions are enabled, compromising the security of the system.

The integrity of most system files must be verified frequently. This can be easily performed by using integrity checkers like Tripwire, AIDE or Integrit. These tools store information like file permissions, ownership, file size, hash values and modification time in a database. Each time a change is detected the administrator is notified.

Tripwire can be found in two different versions. One is open source and can be found at <http://sourceforge.net/projects/tripwire> and the

*The system can be configured to put an entry on the firewall in order to deny access to the attacker's host or even disable a user account that is maliciously being used.

other is the commercial version. The open source version is quite an old one, but it provides an adequate level of integrity checking.

We download, extract and compile the source file of the open source version using Command 8.8. Two passwords are required in order to encrypt

Command 8.8 Tripwire installation

```
server$ gtar -zxvf tripwire-2.3.1-2.tar.gz
server$ cd tripwire-2.3.1-2/src/
server$ make
server$ cd ../
server$ cp install/install.* .
server# ./install.sh
```

two different secret keys*. The first key (site key) is used to protect the configuration file and the policy file. The second key (local key) protects the database and the report files. The protection disables unauthorized writing on these files. Reading is permitted by anyone who has the necessary file permissions.

The configuration file is `/etc/tripwire/tw.cfg` and the policy file is `/etc/tripwire/tw.pol`. The policy file must be customized by the administrator to reflect the needs of his server. All the system files should be protected. This includes the binaries, the SUID/SGID files, the configuration files and all directories. The administrator should first extract the configuration and policy files to a readable format, edit them in order to make the necessary changes, re-encode them and finally delete the text version of the files [Command 8.9].

Command 8.9 Tripwire configuration

```
server# twadmin --print-cfgfile > twcfg.txt
server# twadmin --print-polfile > twpol.txt
server# vi twcfg.txt
server# vi twpol.txt
server# twadmin --create-cfgfile --site-keyfile \
/etc/tripwire/site.key twcfg.txt
server# twadmin --create-polfile twpol.txt
server# rm /etc/tripwire/twcfg.txt /etc/tripwire/twpol.txt
```

Tripwire is now ready to be initialized. This should be done after the installation of the operating system which is considered to be a secure state,

*Public key cryptography is used [Section 3.1]. Tripwire uses two public keys and two secret keys to protect the configuration files and the database.

at least from an integrity point of view. This can be done using Command 8.10.

Command 8.10 Tripwire initialization

```
server# tripwire --test --email admin@example.com
server# tripwire --init -c /etc/tripwire/tw.cfg -p \
/etc/tripwire/tw.pol
```

The database is stored in `/var/lib/tripwire/server.twd` and is encrypted with the local secret key. Each time Tripwire checks the system for modification, everything is compared to the entries in the encrypted database. The verification of the system's integrity can be performed using one of the Commands 8.11 at least once daily.

Command 8.11 Integrity verification with Tripwire

```
server# tripwire --check
server# tripwire --check --twrfile report.twr
server# tripwire --check --email-report
```

An entry must also be inserted in `crond(8)` daemon for automated integrity checking [Command 8.12]. The check will be run at 03am every morning and the administrator will be notified for the results.

Command 8.12 Running automated integrity check

```
server# echo "0 3 * * * /usr/sbin/tripwire --check -s -n \
--email-report" >> /var/spool/cron/root
server# crontab /var/spool/cron/root
```

Whenever new software must be installed changes will be made to the system that will be detected by tripwire. The database should be immediately checked and updated to include changes*. The database update can be done with Command 8.13.

Command 8.13 Updating Tripwire's database

```
server# tripwire --check --interactive
server# tripwire --check --twrfile report.twr
server# tripwire --update --twrfile report.twr
```

If it is required for the policy to be updated or modified the administrator should execute Command 8.14.

*The administrator should verify that the database is only updated with changes that only he has performed on the system.

Command 8.14 Updating Tripwire's policy file

```
server# twadmin --print-polfile > twpol.txt
server# vi twpol.txt
server# tripwire --update-policy twpol.txt
server# rm twpol.txt
```

Bear in mind that the database file and the secret keys must be protected in order for integrity to be guaranteed. Although the files are encrypted, an attacker can delete them to keep his actions secret. The administrator should keep copies of the database file, the configuration file, the policy file and the keys on a **read-only** media like a cd-rom, which will not be accessible by anyone. The backup versions should be frequently compared to the versions of the files stored in the server. Restrictive permissions should be set to all the files used by tripwire [Command 8.15].

Command 8.15 Hardening file permissions for Tripwire

```
server# chmod 700 /etc/tripwire/
server# chmod 700 /var/lib/tripwire/
server# chmod 500 /usr/sbin/tripwire
server# chmod 500 /usr/sbin/twadmin
server# chmod 500 /usr/sbin/siggen
server# chmod 500 /usr/sbin/twprint
```

Further supported features of Tripwire can be found in [82].

8.4 Network-based Intrusion Detection

The network-based IDS possesses strengths that make it a needed component to any security system [83]. The traffic analysis can only be performed by a network-based IDS. Consequently, DoS and other IP-based attacks* can be detected. It provides real-time detection and it is independent of the operating system. Furthermore, it can detect attacks targeting not only the server but the whole network itself.

In order for network-based IDS to be complete for defence, they must be deployed in different locations [84]. At least one IDS is needed inside the internal network, behind the network's firewall, that will detect all the attacks targeting your servers. This is a means of testing and improving your firewall efficiently, for there is always the possibility of failure or misconfiguration when blocking an attack.

*Port scanning, SYN Flood attack, Land, Smurf, TearDrop, HTTP and Mail attacks.

A second IDS is needed outside the external firewall of the organisation. This will detect and document the number and the type of attacks originating from the Internet that target your network. Additionally, if DMZ networks have been implemented to differentiate public services like Web and Mail from the rest of the network, a network-based IDS should be placed there as well. A schematic of the IDS placement is presented in [85].

The two network-based IDS most commonly used on the Internet are Snort (<http://www.snort.org>) and RealSecure (<http://www.iss.net>). Both knowledge-based and behaviour-based techniques are supported by these tools. Snort is an open source software and is supported by a large number of operating systems. Furthermore, additional software like Blockit could be integrated with Snort. The response would be an immediate block of the intruders, using the appropriate firewall rules, whenever Snort detects an attack. Snort's Users Manual [86] is quite helpful when installing and configuring the software.

8.5 Conclusion

Intrusion detection systems are key components of good security. They come to fulfill the gaps intrusion prevention leaves. Successful and unsuccessful attacks can be detected and documented, thus making reaction and future prevention more complete.

Manual detection technique is a procedure that every system administrator or security manager should be aware of. Nevertheless, the need for continuous surveillance demands the use of automated tools and IDS. Such tools are the Host-based IDS that should be placed in each server needing to be protected. They provide real-time or near real-time detection and response. Critical files can be monitored for integrity violations, therefore the system is always kept in a secure state.

Nevertheless, network traffic cannot be examined for malicious contents by host-based IDS. The demanding requirements are met by network-based IDS. They provide real-time detection and response to network based attacks. However, difficulties may come about when they are placed on switched networks or when the traffic is encrypted, as the detection of the attack is not effective*.

Finally it should be added that both architectures of IDS have limitations. New types of attacks, variants of existing attacks or attacks launched by sophisticated coders cannot be easily detected [84]. Human interaction is most of the time necessary for proper reaction and investigation. Never-

*Host-based IDS are not impeded by switched networks or encrypted traffic.

theless, intrusion detection systems can help an organisation to improve the level of security, thus making their use a necessity.

9 Final Considerations

Reaching the end of this project, a need is felt for giving emphasis on certain aspects of computer security. Basic yet simple rules provide a sufficient level of security.

Unnecessary services must be disabled as they are a potential threat. Software that is not in use should be removed or isolated. Every administrator should also keep track of the latest security announcements provided by valid organisations. Such a helpful group is the CERT security center <http://www.cert.org> and SecurityFocus <http://www.securityfocus.com>. The latter provides with the most prominent security mailing list, the Bugtraq. Important new information concerning security vulnerabilities, fixes, documentation and research is all announced there. Keep in mind that closed security groups do exist that deal with security as well. Such groups search for vulnerabilities and the development of tools that exploits them. This kind of work is either kept secret or is revealed to the public long after its initiation.

The security patches available for software that is being used should be installed as soon as it is available. Unpatched and vulnerable services is most definite that they will be exploited by an intruder.

Strong passwords must be used wherever they are required. These must be changed frequently. In combination with strict file permissions, the basic security requirements of every system can be supported.

Finally, a backup policy should be followed to prevent accidental or malicious modification or loss of data. This should be supported by an efficient auditing and logging facility.

9.1 Further improvements

For a further improvement of this study additional topics could be investigated. Kerberos, is a protocol used for entity authentication. It can effectively and securely authenticate users and hosts and it is suited for open networks. LDAP can also cover the authentication part that is needed if users connect to multiple nodes using the same accounts.

Several types of network filesystems are available and in fact quite popular among network services. Such are the NFS, Samba, Coda and OpenAFS. Each one has different features and security needs.

IPsec, the security architecture of the IPv6 protocol, provides with a relatively large number of interesting features. Among those are the authentication and the encryption of the IP traffic which is transparent to the end

user. These are vehicles for strong access control to be applied in networked computers.

- "So what you're saying is that the network is wide open to hackers?" the boss asks.

The department Brown-Nose nods. I, however, shake my head.

Guess who he believes?

- "Well, what have you been doing about these security holes?" asks the boss, now more than a little concerned.

- "Ah..." I consider the topic carefully for almost a nano-second prior to providing my answer.

- "Not a thing."

- "But our network is wide open. The security implications are horrendous!"

- "That is correct," I say. My much maligned co-'worker' has hit the nail right on the side with his diagnosis of our situation, which I will now attempt to summarise.

- "In the unlikely event that someone manages to pick both the seven-pin tumbler locks on one of the comms room doors, bypass the alarm systems and security cameras, then open the locked FDDI cage, or alternatively, smash their way through six inches of reinforced concrete piping buried four feet under a busy suburban road, then tap into our fiber-optic cable without us knowing...then yes, we are wide open."

BOFH - The '95 Vintage

Appendix

A DNS

A.1 Zones files

```
; /etc/namedb/zone/10.0.0
; created by Kapetanakis Giannis

$TTL 86400
@ IN SOA server.example.com.
    hostmaster.server.example.com. (
        2003013103 ; Serial
        10800      ; Refresh
        3600       ; Retry
        604800    ; Expire
        86400     ); Minimum
; define the name servers
    IN      NS      server.example.com.
    IN      NS      server2.example.com.

; define our IPs
1      IN      PTR   server.example.com.
2      IN      PTR   server2.example.com.
10     IN      PTR   client.example.com.
```

File A.1: 0.0.10.in-addr.arpa zone file

The file `/etc/namedb/zone/named.ca` should contain the latest entries about the root DNS servers. It can be easily created with the Command A.1.

Command A.1 Retrieving the latest list of root DNS servers

```
server# dig @a.root-servers.net. . ns \
> /etc/namedb/zone/named.ca
```

```

; /etc/namedb/zone/example.com
; created by Kapetanakis Giannis

$TTL 86400
example.com. IN SOA server.example.com.
    hostmaster.server.example.com. (
        2003013100    ; Serial
        10800        ; Refresh
        3600         ; Retry
        604800       ; Expire
        86400        ); Minimum
; define the name servers for example.com
    IN      NS      server.example.com.
    IN      NS      server2.example.com.
    IN      A       10.0.0.1

; define the mail servers for example.com
    IN      MX      10 mx1.example.com.
    IN      MX      100 mx2.example.com.

; define sub domains
sub      IN      NS      server2

; list each node
server   IN      A       10.0.0.1
        IN      MX      10 mx1.example.com.
        IN      MX      100 mx2.example.com.

server2  IN      A       10.0.0.2
        IN      MX      10 mx1.example.com.
        IN      MX      100 mx2.example.com.

client   IN      A       10.0.0.10
        IN      MX      10 mx1.example.com.
        IN      MX      100 mx2.example.com.

; define local aliases
www      IN      CNAME   server
mail     IN      CNAME   server
pop      IN      CNAME   server
ftp      IN      CNAME   server
ns       IN      CNAME   server
ns2      IN      CNAME   server2
mx1      IN      CNAME   server
mx2      IN      CNAME   server2

```

File A.2: example.com zone file

A.2 DNS startup script

```
#!/bin/bash
#
# named: Starts, stops and controls named
# Script Author: Kapetanakis Giannis bilias@edu.physics.uoc.gr
#
# chkconfig: 35 21 80
#
# processname: /usr/local/sbin/named
# config: /etc/named.conf
# config: /etc/rndc.conf
# pidfile: /etc/namedb/named.pid
# description: DNS server
PATH=/sbin:/bin:/usr/bin:/usr/sbin:/usr/local/sbin

# Source function library.
. /etc/init.d/functions

if [ ! -x /usr/local/sbin/named ]; then
    echo "/usr/local/sbin/named does not exist"
    exit 0
fi

if [ ! -x /usr/local/sbin/rndc ]; then
    echo "/usr/local/sbin/rndc does not exist"
    exit 0
fi

if [ ! -f /etc/named.conf ]; then
    echo "Configuration file /etc/named.conf not found"
    exit 0
fi

if [ ! -f /etc/rndc.conf ]; then
    echo "/etc/rndc.conf is missing"
    exit 0
fi

RETVAL=0
prog="named"
```

File A.3: DNS startup script part 1

```
start(){
    echo -n "Starting $prog: "
    daemon /usr/local/sbin/named -u named
    RETVAL=$?
    echo
    touch /var/lock/subsys/named
    return $RETVAL
}
stop(){
    echo -n "Stopping $prog: "
    /usr/local/sbin/rndc stop
    RETVAL=$?
    echo
    rm -f /var/lock/subsys/named
    return $RETVAL
}
reload(){
    echo -n "Reloading configuration: "
    /usr/local/sbin/rndc reload
    RETVAL=$?
    echo
    return $RETVAL
}
# See how we were called.
case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    reload)
        reload
        ;;
    *)
        echo $"Usage: $0 {start|stop|reload}"
        RETVAL=1
esac

exit $RETVAL
```

File A.4: DNS startup script part 2

By using the Command A.2 you can enable the previous script.

Command A.2 Enabling a startup script

```
server# cp script /etc/init.d/named
server# chmod 700 /etc/init.d/named
server# chown root:root /etc/init.d/named
server# chkconfig --add named
server# /etc/init.d/named start
```

B Configuration Files

B.1 Virus definitions in Sendmail

```
dn1 Virus Detection
dn1 -----
LOCAL_RULESETS
HContent-Disposition: $>Check_extra_header
D{MPat}Multipart message

SCheck_extra_header
R${MPat} $* $#error $: 553 This message MAY contain the
SirCam virus

HContent-Type: $>Check_nimda_header
D{NimdaPat}multipart/related; type="multipart/alternative";
boundary="====_ABC1234567890DEF_===="

SCheck_nimda_header
R${NimdaPat} $* $#error $: 553 This message MAY contain the
Nimda virus
```

File B.1: sendmail.mc part 3

B.2 Certificate extensions in Openssl

```
[ v3_ca ] # root CA extension
basicConstraints = CA:true
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer:always

[ ca_cert ] # non-root CA extension
basicConstraints = CA:true
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer:always
crlDistributionPoints = URI:http://www.example.com/CA/root.crl
nsCaRevocationUrl = http://www.example.com/CA/root.crl
authorityInfoAccess=caIssuers;URI:http://example.com/CA/root.crt
issuerAltName = URI:https://www.example.com/foo/root.crt

[ srv_cert ] # server certificate extension
basicConstraints = CA:FALSE
nsCertType = server
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer:always
crlDistributionPoints = URI:http://www.example.com/CA/ca.crl
nsCaRevocationUrl = http://www.example.com/CA/ca.crl
authorityInfoAccess=caIssuers;URI:http://example.com/CA/ca.crt
issuerAltName = URI:https://www.example.com/foo/ca.crt

[ usr_cert ] # user certificate extension
basicConstraints = CA:FALSE
nsCertType = client, email
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer:always
crlDistributionPoints = URI:http://www.example.com/CA/ca.crl
nsCaRevocationUrl = http://www.example.com/CA/ca.crl
authorityInfoAccess=caIssuers;URI:http://example.com/CA/ca.crt
issuerAltName = URI:https://www.example.com/foo/ca.crt
```

File B.2: openssl.cnf

C Scripts

C.1 Enabling NAT

Command C.1 Enabling NAT on firewall gateway

```
server# iptables -t nat -A POSTROUTING -o eth-out -j MASQUERADE
server# iptables -A INPUT -i eth-out -m state \
--state NEW,INVALID -j DROP
server# iptables -A FORWARD -i eth-out -m state \
--state NEW,INVALID -j DROP
server# echo 1 > /proc/sys/net/ipv4/ip_forward
```

C.2 ICMP filtering

Command C.2 Allow some incoming ICMP traffic

```
server# iptables -A INPUT -p icmp --icmp-type 0 -j ACCEPT
server# iptables -A INPUT -p icmp --icmp-type 3 -j ACCEPT
server# iptables -A INPUT -p icmp --icmp-type 11 -j ACCEPT
server# iptables -A INPUT -p icmp --icmp-type 12 -j ACCEPT
server# iptables -A INPUT -p icmp --icmp-type 8 \
-s ! 10.0.0.0/8 -j DROP
```

Command C.3 Allow some outgoing ICMP traffic

```
server# iptables -A OUTPUT -o eth0 -p icmp --icmp-type 3 \
-d ! 10.0.0.0/8 -j DROP
server# iptables -A OUTPUT -p icmp --icmp-type 4 -j ACCEPT
server# iptables -A OUTPUT -p icmp --icmp-type 8 -j ACCEPT
server# iptables -A OUTPUT -p icmp --icmp-type 12 -j ACCEPT
```

Command C.4 Block unwanted ICMP traffic

```
server# iptables -A INPUT -p icmp -j DROP
server# iptables -A OUTPUT -p icmp -j DROP
```

Caution should be taken before enabling these rules, as it may prevent various services from operating correctly like incoming PING and TRACEROUTE. Although this is a security feature, other services depend on ICMP protocol as it used for network diagnosis and error transmission.

Glossary

ACL Access Control List.

A mechanism that implements access control for a system resource by enumerating the identities of the system entities that are permitted to access the resource.

CA Certification Authority.

An entity that issues digital certificates (especially X.509 certificates) and vouches for the binding between the data items in a certificate.

Capabilities A token, usually an unforgeable data value (sometimes called a "ticket") that gives the bearer or holder the right to access a system resource. Possession of the token is accepted by a system as proof that the holder has been authorized to access the resource named or indicated by the token.

cleartext Data in which the semantic information content (i.e., the meaning) is intelligible or is directly available

client A computer, a program or a user which uses services offered by servers.

CRL Certificate Revocation List. A data structure that enumerates digital certificates that have been invalidated by their issuer prior to when they were scheduled to expire.

DSA Digital Signature Algorithm.

An asymmetric cryptographic algorithm that produces a digital signature in the form of a pair of large numbers. The signature is computed using rules and parameters such that the identity of the signer and the integrity of the signed data can be verified.

FTP File Transfer Protocol.

A TCP-based, application-layer, Internet Standard protocol for moving data files from one computer to another.

Hash An algorithmic method to calculate a 'value', sometimes called the 'digest', of a document in digital form.

HMAC A keyed hash that can be based on any iterated cryptographic hash (e.g., MD5 or SHA-1), so that the cryptographic strength of HMAC depends on the properties of the selected cryptographic hash.

host A computer is usually referred as a host on the Internet. Other commonly used names are node, server, client.

IDS Intrusion Detection Systems. IDS are used to detect and respond to successful or unsuccessful attacks [Section 8].

IETF Internet Engineering Task Force.

A self-organized group of people who make contributions to the development of Internet technology. The principal body engaged in developing Internet Standards

IP Internet Protocol.

A Internet Standard protocol that moves datagrams (discrete sets of bits) from one computer to another across an internetwork but does not provide reliable delivery, flow control, sequencing, or other end-to-end services that TCP provides.

IPsec Internet Protocol security.

(1.) The name of the IETF working group that is specifying a security architecture and protocols to provide security services for Internet Protocol traffic.

(2.) A collective name for that architecture and set of protocols. (Implementation of IPsec protocols is optional for IP version 4, but mandatory for IP version 6.)

IRC Internet Relay Chat.

Basically a huge multi-user live chat facility. There are a number of major IRC servers around the world which are linked to each other. Anyone can create a channel and anything that anyone types in a given channel is seen by all others in the channel. Private channels can (and are) created for multi-person conference calls.

LDAP Lightweight Directory Access Protocol. LDAP provides access to X.500 directory services. It has advanced search capabilities, it implements a distributed model for storing information and it is optimized for fast reading [67].

MD5 MD5.

A cryptographic hash that produces a 128-bit hash result and was designed by Ron Rivest.

NFS Network File System.

NFS allows computers to mount a disk partition on a remote computer as if it was on a local hard drive.

NIS Network Information Service. NIS provides a simple network lookup service consisting of databases and processes. Such databases include files with account information like usernames, passwords and groups.

PKI Public Key Infrastructure.

The policies, legislation, facilities and relationships that create a system of trustworthy CAs.

RA Registration Authority.

An optional PKI entity (separate from the CAs) that does not sign either digital certificates or CRLs but has responsibility for recording or verifying some or all of the information (particularly the identities of subjects) needed by a CA to issue certificates and CRLs and to perform other certificate management functions.

root The account of the super-user in the Unix Operating System (OS). Who ever is using this account has advanced privileges and can perform any process without restrictions.

RSA Rivest-Shamir-Adleman.

An algorithm for asymmetric cryptography, invented in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman. RSA uses exponentiation modulo the product of two large prime numbers. The difficulty of breaking RSA is believed to be equivalent to the difficulty of factoring integers that are the product of two large prime numbers of approximately equal size.

Server A computer or a program which runs services. Other computers or users, referred as clients, connect to the server and use the provided services.

SHA-1 Secure Hash One.

A cryptographic hash function that produces a 160-bit output (hash result) for input data of any length $\leq 2^{64}$ bits

SMTP Simple Mail Transfer Protocol.

A TCP-based, application- layer, Internet Standard protocol for moving electronic mail messages from one computer to another.

sniffer A program and/or device that monitors data traveling over a network. Sniffers can be used both for legitimate network management functions and for stealing information off a network.

Spam Unwanted email sent to multiple strangers, generally for the purpose of advertising. Often Spam is made to appear as if it is sent from a server other than the one it was sent from.

SSH Secure Shell.

A program that provides secure remote access to other computers. The traffic is encrypted and authenticated.

SSL Secure Sockets Layer.

An Internet protocol (originally developed by Netscape Communications, Inc.) that uses connection-oriented end-to-end encryption to provide data confidentiality service and data integrity service for traffic between a client (often a web browser) and a server, and that can optionally provide peer entity authentication between the client and the server.

TCP Transmission Control Protocol.

An Internet Standard protocol that reliably delivers a sequence of datagrams (discrete sets of bits) from one computer to another in a computer network.

TCP/IP A synonym for "Internet Protocol Suite", in which the Transmission Control Protocol (TCP) and the Internet Protocol (IP) are important parts.

TLS Transport Layer Security.

TLS Version 1.0 is an Internet protocol based-on and very similar to SSL Version 3.0.

UDP User Datagram Protocol.

(1.) An Internet Standard protocol that provides a datagram mode of packet-switched computer communication in an internetwork.

(2.) UDP is a transport layer protocol, and it assumes that IP is the underlying protocol. UDP enables application programs to send transaction-oriented data to other programs with minimal protocol mechanism. UDP does not provide reliable delivery, flow control, sequencing, or other end-to-end services that TCP provides.

X.509 An authentication certificate scheme recommended by the International Telecommunication Union (ITU-T) which is used for SSL/TLS authentication*.

*Most of the glossary entries were taken from the Internet Security Glossary RFC 2828 [87].

References

- [1] Simson Garfinkel, Gene Spafford, Alan Schwartz: *Practical Unix & Internet Security*, O'Reilly & Associates Inc, February 2003
- [2] Walter Belgers: *UNIX Password Security*, December 1993
- [3] Check Point Software Technologies: *Minimum OS Installation Guidelines for Linux VPN-1/FireWall-1 Appliance and Linux RedHat 7.2*, Apr 2002
- [4] Mohammed J. Kabir: *RedHat Linux Security and Optimization*, Hungry Minds Inc, 2002
- [5] Arturo Guillen: *X Windows Security: How to Protect your Display*, SANS Institute, November 2001
- [6] Andrew G. Morgan: *The Linux-PAM System Administrators' Guide*, June 2002,
<http://kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam.html>
- [7] Tavis Barr, Nicolai Langfeldt, Seth Vidal, Tom McNeal: *Linux NFS-HOWTO*, August 2002
- [8] S. Kent, BBN Corp, R. Atkinson, @Home Network: *Security Architecture for the Internet Protocol*, RFC 2401, November 1998,
<http://www.ietf.org/rfc/rfc2401.txt>
- [9] US Department of Defence: *DoD Trusted Computer System Evaluation Criteria, (The Orange Book)*, DOD 5200.28-STD, December 1985
- [10] *An Introduction to Computer Security: The NIST Handbook*, NIST Special Publication 800-12, February 1996
- [11] Kevin Fenzi, Dave Wreski: *Linux Security HOWTO*, June 2002
- [12] David Elson: *Intrusion Detection on Linux*, Securityfocus Inc, May 2000
- [13] Xie Huagang: *Build a Secure System with LIDS*, October 2000
- [14] Winfried Trumper: *Summary about POSIX.1e*, February 1999,
<http://wt.xpilot.org/publications/posix.1e/>
- [15] Brian Hatch: *An Overview of LIDS*, Securityfocus Inc, October 2001

-
- [16] Susan Rajnic: *An Introduction to the NSA's Security-Enhanced Linux: SELinux*, SANS Institute, 2002
 - [17] Chris Wright, Crispin Cowan, Stephen Smalley, James Morris, Greg Kroah-Hartman: *Linux Security Module Framework*
 - [18] Chris Wright, Crispin Cowan, Stephen Smalley, James Morris, Greg Kroah-Hartman: *Linux Security Modules: General Security Support for the Linux Kernel*
 - [19] Fred Piper, Sean Murphy: *CRYPTOGRAPHY. A Very Short Introduction*, Oxford University Press Inc, 2002
 - [20] A. Menezes, P. van Oorschot, S. Vanstone: *Handbook of Applied Cryptography*, CRC Press, October 1996,
<http://www.cacr.math.uwaterloo.ca/hac/>
 - [21] R. Housley, RSA Laboratories, W. Polk, NIST, W. Ford, VeriSign, D. Solo, Citigroup: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, RFC 3280, April 2002,
<http://www.ietf.org/rfc/rfc3280.txt>
 - [22] Fred Piper: *An Introduction to Cryptography and Security Mechanisms*, Public Key Infrastructures Lecture in MSc Information Security, Royal Holloway, University of London, 2003
 - [23] Kathy Lyons-Burke, Federal Public Key Infrastructure Committee: *Federal Agency Use of Public Key Technology for Digital Signatures and Authentication*, NIST Special Publication 800-25, October 2000
 - [24] Joel Weise: *Public Key Infrastructure Overview*, SUN Blueprints, August 2001, <http://www.sun.com/blueprints>
 - [25] Verisign: *Total Cost of Ownership for Public Key Infrastructure*, February 2002
 - [26] Andrew S. Tanenbaum: *Computer Networks*, Prentice Hall, 2002
 - [27] M. Lottor: *Domain Administrators Operations Guide*, RFC 1033, 1987,
<http://www.ietf.org/rfc/rfc1033.txt>
 - [28] Internet Software Consortium: *BIND 9 Administrator Reference Manual*, 2001

-
- [29] Christoph Schuba: *Addressing Weaknesses in the Domain Name System Protocol*, 1993
- [30] Joe Stewart: *DNS Cache Poisoning - The Next Generation*, 2003
- [31] Doug Sax: *DNS Spoofing*, 2000, <http://www.sans.org>
- [32] Cricket Liu: *DNS and BIND Security*, 2002, http://www.menandmice.com/docs/dns_security_course.pdf
- [33] P. Vixie, O. Gudmundsson, D. Eastlake, B. Wellington: *Secret Key Transaction Authentication for DNS (TSIG)*, RFC 2845, 2000, <http://www.ietf.org/rfc/rfc2845.txt>
- [34] D. Eastlake: *Domain Name System Security Extensions*, RFC 2535, 1999, <http://www.ietf.org/rfc/rfc2535.txt>
- [35] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear: *Address Allocation for Private Internets*, RFC 1918, 1996, <http://www.ietf.org/rfc/rfc1918.txt>
- [36] Paul Albitz, Cricket Lui, Cricket Liu, Mike Loukides, Deborah Russell: *DNS and BIND*, O'Reilly & Associates Inc, April 2001
- [37] Internet Software Consortium: *BIND Vulnerabilities*, <http://www.isc.org/products/BIND/bind-security.html>
- [38] D. J. Bernstein: *djbdns, Domain Name System Tools*, <http://cr.yp.to/djbdns.html>
- [39] Nelson Russell: *djbdns Home Page*, <http://www.djbdns.org>
- [40] Eric Allman, Gregory Neil Shapiro, Claus ABmann, Sendmail, Inc.: *Sendmail Installation and Operation Guide*, Version 8.609.2.23
- [41] J. De Winter, Wildbear Consulting, Inc.: *SMTP Service Extension for Remote Message Queue Starting*, RFC 1985, August 1996, <http://www.ietf.org/rfc/rfc1985.txt>
- [42] R. Gellens, QUALCOMM, J. Klensin, MCI: *Message Submission*, RFC 2476, December 1998, <http://www.ietf.org/rfc/rfc2476.txt>
- [43] Mail Abuse Prevention System (MAPS): *The MAPS RBL Home Page*, <http://www.mail-abuse.org/rbl/>
- [44] ORDB: *Open Relay DataBase Home Page*, <http://ordb.org>

- [45] Bryan Costales, Eric Allman: *Sendmail*, O'Reilly & Associates Inc, December 2002
- [46] Brett Glass: *Stopping Spam and Malware with Open Source*, Presented at the O'Reilly Open Source Convention on July 27, 2001, <http://www.brettglass.com/spam/paper.html>
- [47] Fred Cohen: *Computer Viruses - Theory and Experiments*, 1984, <http://all.net/books/virus/index.html>
- [48] Fred Cohen: *Trends In Computer Virus Research*, 1991, <http://all.net/books/integ/japan.html>
- [49] Ken Thompson: *Reflections on Trusting Trust*, August 1984
- [50] Peter V. Radatti, CyberSoft, Inc.: *Computer Viruses In Unix Networks*, February 1996, <http://www.cybersoft.com/whitepapers/papers/networks.shtml>
- [51] Dieter Gollmann: *Computer Security*, John Wiley & Sons Ltd, August 2002
- [52] Jonathan B. Postel: *Simple Mail Transfer Protocol*, RFC 821, August 1982, <http://www.ietf.org/rfc/rfc821.txt>
- [53] J. Myers, Netscape Communications: *SMTP Service Extension for Authentication*, RFC 2554, March 1999, <http://www.ietf.org/rfc/rfc2554.txt>
- [54] J. Myers: *Simple Authentication and Security Layer (SASL)*, RFC 2222, October 1997, <http://www.ietf.org/rfc/rfc2222.txt>
- [55] Claus Aßmann: *SMTP AUTH in sendmail 8.10-8.12*, May 2003, <http://www.sendmail.org/~ca/email/auth.html>
- [56] Michael D. Bauer: *Building Secure Servers with Linux*, O'Reilly & Associates Inc, October 2002
- [57] joreybump.com: *SMTP AUTH with sendmail. Quick start guide for Red Hat Linux*, <http://www.joreybump.com/code/howto/smtpauth.html>
- [58] P. Hoffman, Internet Mail Consortium: *SMTP Service Extension for Secure SMTP over TLS*, RFC 2487, January 1999, <http://www.ietf.org/rfc/rfc2487.txt>

-
- [59] William Stallings: *Network Security Essentials: Applications and Standards*, Prentice Hall Inc, 2000
- [60] Claus Abmann: *SMTP STARTTLS in Sendmail/Secure Switch*, February 2003, <http://www.sendmail.org/~ca/email/starttls.html>
- [61] Jason Heiss: *Configuring Sendmail's STARTTLS (SSL) and Relaying*, October 2002, <http://www.ofb.net/~jheiss/sendmail/tlsandrelay.shtml>
- [62] Weldon Whipple: *My Experiences (So Far) with STARTTLS and Sendmail*, June 2003, <http://www.technoids.org/wstarttls.html>
- [63] Wietse Venema: *Postfix Documentation, Howtos and FAQs*, <http://www.postfix.org/docs.html>
- [64] D. J. Bernstein: *qmail: the Internet's MTA of choice*, <http://cr.yp.to/qmail.html>
- [65] *qmail Home Page*, <http://www.qmail.org>
- [66] Dave Sill: *Life with qmail*, <http://www.lifewithqmail.org>
- [67] Gerald Carter: *LDAP System Administration*, O'Reilly & Associates Inc, March 2003
- [68] *How we defaced www.apache.org, by {} and Hardbeat*, <http://www.dataloss.net/papers/how.defaced.apache.org.txt>
- [69] *Apache HTTP Server Version 2.0 Security Tips*, Apache HTTP Server Documentation Project, <http://httpd.apache.org/docs-project/>
- [70] Heinz Johner, Jouni Auer, Vitolis Bendinskas, Ng Chang Chyn, Shane Owenby, SunJong Park: *IBM HTTP Server Powered by Apache on RS/6000*, IBM redbooks, March 1999
- [71] Ben Laurie, Peter Laurie: *Apache The Definitive Guide*, O'Reilly & Associates Inc, March 1997
- [72] *Apache SSL/TLS Encryption*, Apache HTTP Server Documentation Project, <http://httpd.apache.org/docs-project/>
- [73] Antony Stone: *Network Security, Firewalls Lecture in MSc Information Security*, Royal Holloway, University of London, 2003

- [74] Rob Flickenger: *Linux Server Hacks*, O'Reilly & Associates Inc, January 2003
- [75] W. Richard Stevens: *TCP/IP Illustrated: The Protocols*, Addison Wesley, February 1994
- [76] William R. Cheswick, Steven M. Bellovin: *Firewalls and Internet Security: Repelling the Wily Hacker*, Addison Wesley, April 1994
- [77] Elizabeth D. Zwicky, Simon Cooper, D. Brent Chapman: *Building Internet Firewalls*, O'Reilly & Associates Inc, June 2000
- [78] Oskar Andreasson: *Iptables Tutorial*, v1.1.18, April 2003, <http://iptables-tutorial.frozentux.net>
- [79] Internet Security Systems (ISS): *Intrusion Detection Presentation*, Royal Holloway, University of London, March 2003
- [80] Kenny Paterson: *Network Security*, An Introduction to Intrusion Detection Lecture in MSc Information Security, Royal Holloway, University of London, 2003
- [81] Stephen Northcutt, Mark Cooper, Matt Fearnow, Karen Frederick: *Intrusion Signatures and Analysis*, New Riders Publishing, January 2001
- [82] Tripwire Inc: *Tripwire User's Guide*, <http://www.tripwire.org>
- [83] Brian Laing: *How To Guide-Implementing a Network Based Intrusion Detection System*, Internet Security Systems, 2000
- [84] Rebecca Bace, Peter Mell: *Intrusion Detection Systems*, NIST Special Publication on Intrusion Detection Systems
- [85] Scott C. Sanchez, CISSP: *IDS "Zone" Theory Diagram*, July 2000, <http://infosec.gungadin.com>
- [86] Martin Roesch, Chris Green: *Snort Users Manual*, July 2003
- [87] R. Shirey, GTE / BBN Technologies: *Internet Security Glossary*, RFC 2828, May 2000, <http://www.ietf.org/rfc/rfc2828.txt>