SEGURIDAD EN UNIX Y REDES

Versión 2.1'

Antonio Villalón Huerta

Julio, 2000 – Julio, 2002 – Enero, 2020

AUllibres

Nau Llibres - Edicions Culturals Valencianes, S.A.
Tel.: 96 360 33 36, Fax: 96 332 55 82.
C/ Periodista Badía, 10. 46010 Valencia
E-mail: nau@naullibres.com web: www.naullibres.com

Diseño de portada e interiores:
Carol López, Pablo Navarro y Artes Digitales Nau Llibres

ISBN: 978-84-18047-04-6

Dep.Legal: V-3368-2019

Imprime:
Safekat

Índice general

Índice de figuras

Índice de cuadros

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 ponerse en contacto conmigo se puede utilizar la dirección de correo electrónico que utilizo habitualmente: `toni@shutdown.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, AIX 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 demostró en las versiones iniciales de este documento, así como todas las observaciones que sobre las mismas me hizo llegar.

NOTAS A LA VERSIÓN 2.0

No hay mucho que añadir a lo dicho hace casi dos años; y es que las cosas apenas han cambiado: el panorama en España – en cuanto a seguridad se refiere – sigue siendo desolador, las empresas tecnológicas caen día a día, la investigación en materias de seguridad (si exceptuamos la Criptografía) es nula, y poco más. Sólo dar las gracias una vez más a todos los que han publicado o se han hecho eco de este documento (Kriptópolis, HispaLinux, IrisCERT, Hispasec, Asociación de Internautas...) y también a toda la gente que lo ha leido (al menos en parte ;-) y ha perdido unos minutos escribiéndome un *e–mail* con algún comentario; realmente es algo que se agradece, y aunque tarde en responder al correo, siempre trato de contestar.

Como algunos de los comentarios acerca del documento que me han llegado hablaban del 'excesivo' tamaño del mismo, en esta nueva versión he cambiado la forma de generar el fichero PDF; he convertido todas las imágenes a formato PNG y

después utilizado `pdflatex` para compilar los ficheros, habiendo modificado previamente el código mediante un sencillo *script*. Aunque aún ocupa bastante, hay que tener en cuenta que estamos hablando de unas 500 páginas de documento...

TODO
Igual que hace casi dos años, sigue en pie la intención de crear capítulos nuevos (las redes privadas virtuales es mi principal tema pendiente) y de comentar la seguridad de mecanismos como DNS, RPC, NIS o NFS... espero disponer de algo más de tiempo para poder hacerlo. Quiero también escribir más acerca de la detección de intrusos, no sé si en este documento o en uno aparte, ya que es quizás el tema que más me interesa y en lo que más trabajo actualmente. Y finalmente, en mi lista de cosas para hacer, pone **dormir** (sí, lo pone en negrita) como algo que también queda pendiente :)

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.
Versión 2.0 (Mayo´02): Capítulos dedicados a los sistemas de detección de intrusos y a los ataques remotos contra un sistema. Sustitución del capítulo referente al núcleo de algunos sistemas Unix por varios capítulos que tratan particularidades de diferentes clones con mayor detalle. Desglose del capítulo dedicado a los sistemas cortafuegos en dos, uno teórico y otro con diferentes casos prácticos de estudio. Ampliación de los capítulos dedicados a autenticación de usuarios (PAM) y a criptografía (funciones resumen). Ampliación del capítulo dedicado a políticas y normativa, que ahora pasa a denominarse *'Gestión de la seguridad'*.
Versión 2.1 (Julio´02): Alguna corrección más e inclusión de la *GNU Free Documentation License* (implica que el código fuente en TEX pasa a ser libre).
Versión 2.1′ (Diciembre´19): Auque debería haberlas, con el objetivo de respetar el original no se han realizado correcciones ni ampliaciones al documento en su versión 2.1. Únicamente se han modificado aspectos de formato, y no todos los necesarios, orientados a una impresión en papel para celebrar el vigésimo aniversario del documento. Enjoy!

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 Quartely* 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

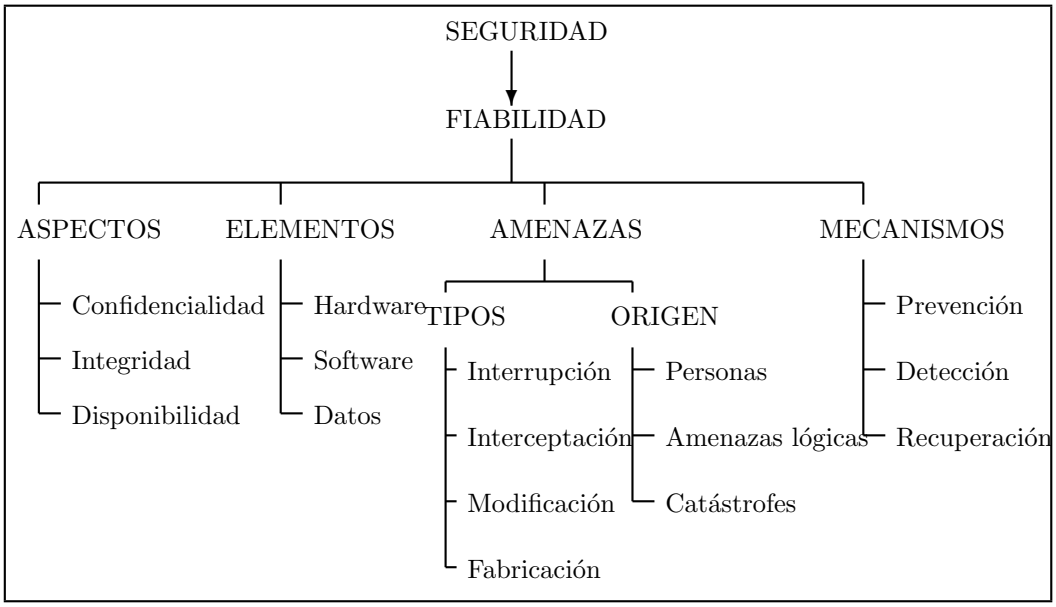

Figura 1.2: Visión global de la seguridad informática

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.

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

encontrar AT&T System V/MLS y OSF/1 (B1), Trusted Xenix[8] (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.

[8]Este 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 cuenta[1].

2.2. Protección del *hardware*

El *hardware* es frecuentemente el elemento más caro de todo sistema informático[2]. 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.

[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.

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

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 paises 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 licen-

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

ne 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 magnitud[2], 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 medidad 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

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

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

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

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 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       /        ext2        defaults   1   1
/dev/hda4       /home    ext2        defaults   1   2
none            /proc    proc        defaults   1   1
luisa:~#
```

Cuando el sistema arranque, 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

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:~#
```

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 permisos[4]; 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

[4]Esto 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.

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

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.
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.

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

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 `tcfsgenkey`, que genera una entrada para cada usuario en `/etc/tcfspasswd`:

```
rosita:~# tcfsgenkey
login: toni
password:
now we'll generate the des key.
press 10 keys:**********
Ok.
rosita:~# cat /etc/tcfspasswd
toni:2rCmyOUsM5IA=
rosita:~#
```

Una vez que un usuario tiene una entrada en `/etc/tcfspasswd` con su clave ya

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 años[1], 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:

[1] En Unix, claro.

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 seguridad[3]. 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

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

- `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 192.168.0.1
[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 192.168.0.1
[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

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 *shells-cripts* 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.

```
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, advpiomode
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, advpiomode
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@0/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@0/sd@1,0
Jan 24 18:10:03 anita icmpinfo: ICMP_Dest_Unreachable[Port] < 127.0.0.1
```

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

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.

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* propie-

Si en lugar de (o aparte de) un único directorio con todos sus ficheros y subdirectorios quisiéramos 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:

```
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

sólo de los directorios de usuario, cada día lectivo de la semana. Un *shellscript* 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")
        # Miercoles, progresiva
        FILES=‘find /export/home/ -mtime 3 -print‘
        hazback
        ;;
    "Thu")
        # Jueves, progresiva
        FILES=‘find /export/home/ -mtime 4 -print‘
        hazback
        ;;
    "Fri")
```

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

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.

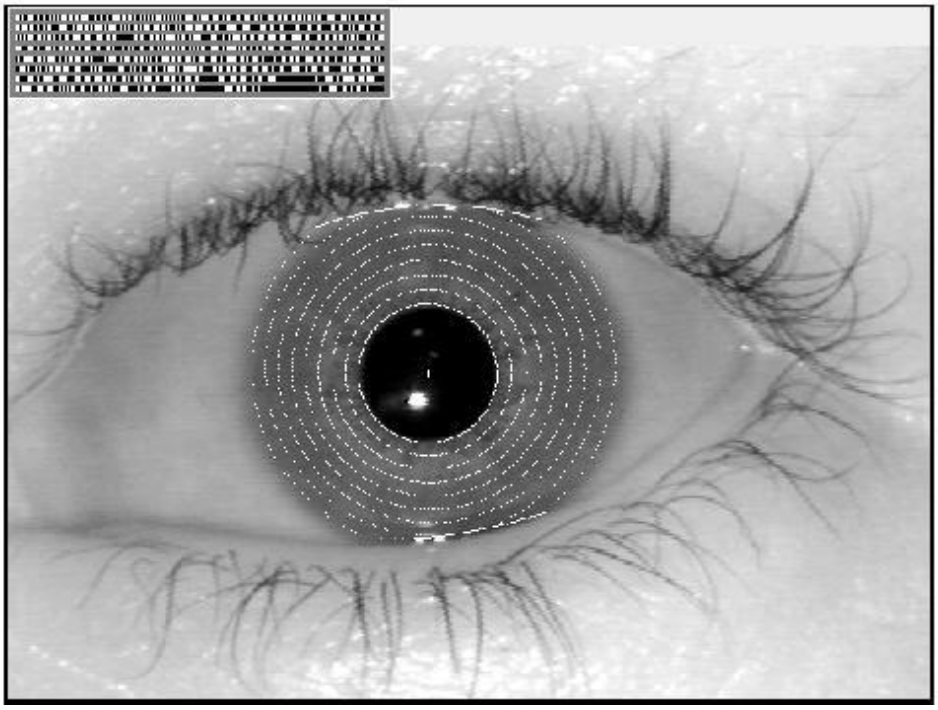

Figura 8.3: Iris humano con la extracción de su *iriscode*.

los sobre este modelo de autenticación (a diferencia de la página de EyeDentify), 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 situa 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 anali-

- `account`
  Determina si el usuario está autorizado a acceder al servicio indicado, por ejemplo si su clave ha caducado, si el acceso se produce desde una línea determinada o si se supera el número máximo de conexiones simultáneas.

- `auth`
  Determina si el usuario es autenticado correctamente, por ejemplo mediante una clave clásica de Unix o mediante un método biométrico.

- `password`
  Proporciona mecanismos para que el usuario actualice su elemento de autenticación, por ejemplo para cambiar su contraseña de acceso al sistema.

- `session`
  Define procesos a ejecutar antes o después de autenticar al usuario, como registrar el evento o activar un mecanismo de monitorización concreto.

Por su parte, el campo al que hemos etiquetado como `'control'` marca qué hacer ante el éxito o el fracaso del módulo al que afecten; los módulos PAM son apilables, esto es, podemos combinar un número indeterminado de ellos (del mismo tipo) para un único servicio de forma que si uno de ellos falla la autenticación es incorrecta, si uno de ellos es correcto no nos preocupamos del resto, si algunos son necesarios pero otros no para una correcta autenticación, etc. Se definen cuatro tipos de control:

- `required`
  El resultado del módulo ha de ser exitoso para que se proporcione acceso al servicio; si falla, el resto de módulos de la pila se ejecutan, pero sin importar su resultado el acceso será denegado.

- `requisite`
  De nuevo, el resultado del módulo ha de ser exitoso para que se proporcione acceso al servicio; en caso contrario, no se ejecutan más módulos y el acceso se deniega inmediatamente.

- `sufficient`
  Si la ejecución el módulo correspondiente tiene éxito el acceso se permite inmediatamente (sin ejecutar el resto de módulos para el mismo servicio) siempre y cuando no haya fallado antes un módulo cuyo tipo de control sea `'required'`; si la ejecución es incorrecta, no se implica necesariamente una negación de acceso.

- `optional`
  El resultado de su ejecución no es crítico para determinar el acceso al servicio requerido: si falla, pero otro módulo del mismo tipo para el servicio es exitoso, se permite el acceso. Sólo es significativo si se trata del único módulo de su tipo para un cierto servicio, en cuyo caso el acceso al servicio se permite o deniega en función de si la ejecución del módulo tiene éxito.

Parte III

Algunos sistemas Unix

Capítulo 9

Solaris

9.1. Introducción

Solaris es el nombre del actual entorno de trabajo Unix desarrollado por Sun Microsystems; con anterioridad esta compañía ofrecía SunOS, un sistema basado en BSD, pero a partir de su versión 5 el sistema fué completamente revisado, adoptó el modelo *System V* y se pasó a denominar Solaris, que es como se conoce actualmente[1]. Hasta la versión 6, el producto era conocido como *'Solaris 2.x'* – equivalentemente, *'SunOS 5.x'* –, pero desde su versión 7 se eliminó el '2.x' y simplemente se pasó a llamar *'Solaris x'*, aunque se sigue manteniendo el nombre *'SunOS 5.x'*; en la actualidad, Sun Microsystems ofrece Solaris 8, y se espera que en 2001 aparezca en el mercado Solaris 9.

Solaris es uno de los Unix más extendidos hoy en día, ya que sus posibilidades comprenden un abanico muy amplio; aunque funciona sobre arquitecturas x86 (lo que permite que cualquiera pueda instalar y utilizar el operativo en un PC casero), el principal mercado de Solaris está en las estaciones y los servidores SPARC (*Scalable Processor ARChitecture*). Dentro de esta familia de procesadores podemos encontrar todo tipo de máquinas, desde estaciones que pueden ser equivalentes en potencia a un PC (como la UltraSPARC 5) a grandes servidores (por ejemplo, los Ultra Enterprise 10000), pasando por supuesto por servidores medios, como los Ultra Enterprise 3500; esta amplia gama de máquinas hace que Solaris se utilice en todo tipo de aplicaciones, desde equipos de sobremesa o servidores *web* sencillos hasta sistemas de bases de datos de alta disponibilidad. Su uso como plataforma de aplicaciones relacionadas con la seguridad (típicamente, sistemas cortafuegos funcionando con *Firewall–1*) también está muy extendido.

Para conocer más el entorno de trabajo Solaris podemos descargar las imágenes del operativo, tanto para arquitecturas SPARC como para x86, y de forma completamente gratuita, desde la *web* corporativa de Sun Microsystems: `http://www.sun.com/`.

[1] Realmente, existió un entorno llamado Solaris 1.x, que no era más que alguna versión de SunOS 4 acompañada de OpenWindows 3.0 ([Dik99]).

Como no todo va a ser hablar bien de Unix a cualquier precio (aunque Solaris tiene muchos aspectos para hablar bien del operativo), podemos hacer un par de críticas a mecanismos relacionados con las contraseñas en Solaris. En primer lugar, quizás deja algo que desear la granularidad que nos ofrece para el envejecimiento de claves: especificar los valores máximo y mínimo en semanas a veces no es apropiado, y seguramente si Solaris nos permitiera indicar dichos valores en días podríamos definir políticas más precisas; aunque en la mayoría de ocasiones la especificación en semanas es más que suficiente, en casos concretos se echa de menos el poder indicar los días de vigencia de una contraseña. Otro aspecto que se podría mejorar en Solaris es la robustez de las claves: limitarse a rotaciones del nombre de usuario es en la actualidad algo pobre; un esquema como el ofrecido en AIX, donde se pueden especificar incluso diccionarios externos al operativo contra los que comparar las claves que un usuario elige, sería mucho más potente.

Contra el primero de estos comentarios quizás se podría decir, en defensa de Solaris, que realmente en `/etc/shadow` podemos especificar días en lugar de semanas, pero eso implicaría modificar el archivo a mano para cada usuario, algo que no suele ser recomendable en ningún sistema Unix, o bien ejecutar `/usr/bin/passwd` con las opciones apropiadas, de nuevo para todos los usuarios en cuestión. Contra el segundo también se podría argumentar que existen utilidades de terceros para reforzar las contraseñas que eligen los usuarios, o bien que no es difícil escribir un módulo de PAM para evitar que esos usuarios elijan claves triviales, pero el hecho es que Sun Microsystems por defecto no incorpora ninguno de estos mecanismos en Solaris.

9.5. El sistema de parcheado

Como en el resto de Unices, en Solaris un parche se define como un grupo de ficheros y directorios que reemplaza o actualiza ficheros y directorios existentes en el sistema para tratar de garantizar la ejecución correcta del *software* ([Mic98]); con la instalación de parches aumentamos – al menos teóricamente – la seguridad y la disponibilidad de un sistema, y es **muy recomendable** seguir una política de parcheado estricta, al menos en cuanto a parches de seguridad se refiere.

Sun Microsystems distribuye entre sus clientes con contrato de mantenimiento los parches para Solaris y el resto de sus productos vía CD-ROM cada pocas semanas, pero – mucho más rápido –, también los ofrece a través de Internet, desde `http://sunsolve.sun.com/`; aunque el acceso a ciertos parches está restringido a clientes con contrato, los relativos a la seguridad de los sistemas y los recomendados son completamente públicos.

Desde Sun Microsystems, cada parche es referenciado mediante un dos números: un *'patch id'*, que es el realmente el identificador del parche, y un número de revisión del parche, separados ambos por un guión; de esta forma cuando descargamos un parche desde *SunSolve* y lo descomprimimos, o cuando lo recibimos

Además, desde su versión 2.6 Solaris permite marcar puertos individuales como reservados, tanto UDP como TCP, y también eliminar esta restricción de puertos que previamente hayamos reservado; por defecto, aparte de los primeros 1024 puertos, Solaris define como reservados – en TCP y UDP – los puertos 2049 (`nfsd`) y 4045 (`lockd`). En el siguiente ejemplo se muestra cómo consultar los puertos marcados de esta forma, cómo añadir un alguno a la lista, y cómo eliminarlo de la misma; aunque el ejemplo se aplica a TCP, el caso de UDP es completamente análogo pero sustituyendo el nombre del dispositivo contra el que ejecutamos la orden (que sería `/dev/udp`) y la cadena `'tcp'` del nombre de cada parámetro por `'udp'`:

```
anita:/# ndd /dev/tcp tcp_extra_priv_ports
2049
4045
anita:/# ndd -set /dev/tcp tcp_extra_priv_ports_add 5000
anita:/# ndd /dev/tcp tcp_extra_priv_ports
2049
4045
5000
anita:/# ndd -set /dev/tcp tcp_extra_priv_ports_del 5000
anita:/# ndd /dev/tcp tcp_extra_priv_ports
2049
4045
anita:/#
```

Antes de finalizar este punto es importante insistir en que los cambios producidos por `'ndd'` sólo se mantienen hasta el siguiente reinicio del sistema; de esta forma, las modificaciones que hemos visto aquí se mantendrán sólo mientras la máquina no se pare, pero si esto sucede en el arranque todos los parámetros del subsistema de red tomarán sus valores por defecto. Por tanto, lo más probable es que nos interese planificar en el inicio del sistema las modificaciones estudiadas en este punto para que se ejecuten de forma automática; para conseguirlo, no tenemos más que crear el *script* correspondiente y ubicarlo en el directorio `/etc/rc2.d/` con un nombre que comience por `'S'`, con lo que hacemos que se ejecute siempre que la máquina entre en un *runlevel* 2:

```
anita:~# cat /etc/init.d/nddconf
#!/sbin/sh
#
# Configuracion segura del subsistema de red de Solaris
#
ndd -set /dev/ip ip_forwarding 0
ndd -set /dev/ip ip_strict_dst_multihoming 1
ndd -set /dev/ip ip_forward_directed_broadcasts 0
ndd -set /dev/ip ip_forward_src_routed 0
ndd -get /dev/arp arp_cleanup_interval 60000
ndd -get /dev/ip ip_ire_flush_interval 60000
ndd -get /dev/ip ip_respond_to_echo_broadcast 0
```

Capítulo 10

Linux

10.1. Introducción

A mediados de 1991 un estudiante finlandés llamado Linus Torvalds trabajaba en el diseño de un sistema operativo similar a Minix, que pudiera ejecutarse sobre plataformas Intel y compatibles, y sobre todo que fuera pequeño y barato; a raíz de un mensaje de este estudiante en `comp.os.minix`, algunas personas comenzaron a interesarse por el proyecto, y finalmente el 5 de octubre de ese año Linus Torvals hizo pública la versión 0.02 – la primera funcional – de lo que ya se denominaba Linux (*Linus´ Unix*). En esa versión, que aproximadamente utilizaron un centenar de usuarios, apenas se ofrecía soporte a *hardware* (excepto el que Linus tenía en su ordenador), no disponía de subsistema de red ni de sistema de ficheros propio, y las utilidades de espacio de usuario se podían contar con los dedos de las manos (un *shell*, un compilador, y poco más). Sin embargo, y a pesar de las duras críticas de pesos pesados en el mundo de los sistemas operativos como Andrew Tanenbaum, el proyecto era muy interesante, y poco a poco programadores de todo el mundo fueron aportando mejoras a este nuevo sistema.

A principios de 1994 apareció Linux 1.0, considerada la primera versión del operativo utilizable no sólo por *hackers* y programadores, sino por usuarios 'normales'; de las aproximadamente 10000 líneas de la versión inicial se había pasado a unas 170000, y el centenar de usuarios se había multiplicado por mil. Linux 1.0 incorporaba subsistema de red (sin duda uno de los cambios que más ha contribuido a la expansión del operativo), entorno gráfico (arrastrado de versiones anteriores) y soporte a una gama de *hardware* relativamente amplia. La popularidad del operativo crecía mes a mes – especialmente en entornos universitarios y de investigación – gracias sobre todo a su filosofía: cualquiera podía (y puede) modificar una parte del núcleo, adaptarla, mejorarla, o incorporar nuevas líneas, con la única obligación de compartir el nuevo código fuente con el resto del mundo.

Sin embargo, no fué hasta 1996, con la aparición de Linux 2.0 (que incorporaba casi medio millón de líneas de código), cuando se produjo el gran *boom* de

negativos en la funcionalidad o en la comodidad de administración: no podremos realizar *reboots* automáticos ni remotos de Unix, ya que cada vez que el sistema reinicie alguien deberá estar físicamente al lado de la máquina para teclear la clave correspondiente.

Antes de finalizar este punto dedicado a la seguridad física dentro de Linux debemos hablar de la protección ofrecida por LILO; ahora ya no se trata de algo genérico de los PCs, sino de mecanismos implantados en el cargador de Linux que sólo son aplicables a sistemas arrancados desde dicho cargador. LILO (*LInux LOader*) es un *software* que se instala en un sector de arranque – de una partición o de un *diskette* – o en el *Master Boot Record* (MBR) del disco duro y que permite de esta forma arrancar tanto Linux como otros sistemas operativos instalados en el PC.

LILO toma su configuración del archivo `/etc/lilo.conf` del sistema Linux; cada vez que modifiquemos ese archivo será necesario ejecutar la orden `/sbin/lilo` si queremos que los cambios tengan efecto en el siguiente encendido de la máquina:

```
luisa:~# /sbin/lilo
Added linux *
luisa:~#
```

Al arrancar el PC, LILO permite elegir una imagen para ser arrancada, así como especificar parámetros para el núcleo; aunque esto sea necesario para inicializar el sistema en ciertas ocasiones – principalmente cuando hay errores graves en un arranque normal –, el hecho es que los parámetros pasados a un *kernel* antes de ser arrancado pueden facilitar a un atacante un control total sobre la máquina, ya que algunos de ellos llegan incluso a ejecutar un *shell* con privilegios de `root` sin necesidad de ninguna contraseña.

En determinadas ocasiones, quizás nos interese proteger el arranque de una máquina, tanto a nivel de la elección de núcleo a arrancar como a nivel de las opciones que se pasan a dicho núcleo. Podemos habilitar desde LILO el uso de una contraseña que se solicitará antes de que LILO cargue cualquier sistema operativo instalado en el ordenador; para ello debemos hacer uso de la directiva `'password'` en `/etc/lilo.conf`:

```
luisa:~# cat /etc/lilo.conf
boot = /dev/hda
delay = 50
vga = normal
password = P,e+bqa
image = /vmlinuz
  root = /dev/hda1
  label = linux
  read-only
luisa:~#
```

Tras ejecutar `/sbin/lilo`, en el siguiente arranque de la máquina se solicitará la

Capítulo 11

AIX

11.1. Introducción

AIX (*Advanced Interactive eXecutive*) es la versión de Unix desarrollada y mantenida desde hace más de diez años por IBM; sus orígenes provienen de IBM RT System, de finales de los ochenta, y se ejecuta en las plataformas RS/6000, basadas en *chips* RISC PowerPC similares al utilizados por algunos Macintosh más o menos modernos. Este operativo, mezcla de BSD y *System V* (se suele decir que no es ni uno ni otro) es el que más características propietarias incorpora, características que no se encuentran en ninguna otra versión de Unix, por lo que a cualquier administrador le costará más adaptarse a AIX que a otros Unices. No obstante, en favor de IBM hay que decir que dispone de un excelente asistente para realizar todo tipo de tareas denominado SMIT (*System Management Interface Tool*, también ejecutado como `smitty` en modo consola): aunque por supuesto cualquier tarea también se puede ejecutar desde la línea de comandos, esto generalmente no se hace con las órdenes habituales de Unix, por lo que SMIT es una herramienta indispensable para el administrador de la máquina. AIX, a pesar de que en un principio pueda parecer el Unix más arcaico – y nadie niega que lo sea – es un entorno de trabajo fiable y sobre todo extremadamente estable, e incluso incorpora características (tanto de seguridad como de otros aspectos) que ya quisieran para sí otros sistemas Unix.

El hecho de que muchas tareas no se realicen en AIX de la misma forma en que se llevan a cabo en el resto de Unices es en parte debido al ODM (*Object Data Manager*); a diferencia de Solaris o Linux, en AIX debemos tener presente en todo momento que `vi` **no** es una herramienta para administrar el sistema, ya que los clásicos ficheros de configuración ASCII han sido sustituidos por archivos binarios, bases de datos que mantienen la configuración del sistema (aspectos como los dispositivos, los recursos del entorno, o el subsistema de comunicaciones de AIX) de forma más robusta, segura y portable que los simples ficheros de texto, al menos en opinión de los desarrolladores de IBM ([V$^+$00]).

IBM mantiene en Internet un excelente repositorio de documentación, principalmente los denominados *RedBooks*, completos manuales sobre casi cualquier tema relacionado con AIX – y otros productos de la compañía – que podamos imaginar. Están disponibles en la dirección `http://www.redbooks.ibm.com/`, y algunos de ellos son una herramienta imprescindible para cualquier administrador de sistemas. Por supuesto, también es muy recomendable instalar las páginas del manual *on–line* de AIX, algo que incomprensiblemente no se hace por defecto en una instalación estándar, y utilizar SMIT para ejecutar cualquier tarea que desde línea de órdenes dudemos de cómo hacer.

En este capítulo hablaremos de aspectos de la seguridad casi exclusivos de AIX; al ser este sistema probablemente el Unix más cerrado de todos, lo que veamos aquí difícilmente será extrapolable – en la mayoría de casos – a otros entornos como Solaris o HP–UX. En cualquier caso, es necesario para un administrador conocer los aspectos de AIX que le confieren su enorme robustez, tanto en lo referente a seguridad como en cualquier otra faceta genérica del sistema.

11.2. Seguridad física en RS/6000

El *firmware* de los sistemas RS/6000 provee de tres modos de protección física ([IBM00a]), en cierta forma similares a las que ofrecen las estaciones SPARC que comentamos al hablar de Solaris. El primero de ellos, denominado POP (*Power–On Password*) es el equivalente al modo `'full-secure'` de SPARC, y como éste impide el arranque del sistema si antes no se ha introducido la contraseña determinada como POP; por supuesto, esto evita tareas como la planificación de reinicios automáticos, por lo que en muchas ocasiones no compensa el nivel de seguridad con el de funcionalidad. Por eso existe un segundo modo de protección denominado *Unattended start mode* que permite arrancar al sistema de su dispositivo por defecto sin necesidad de que un operador teclee la contraseña POP para ello.

En el modo 'desatendido' el sistema arranca con normalidad desde el dispositivo especificado por defecto sin necesidad de ninguna clave POP (a pesar de que esta contraseña tenga que haber sido definida con anterioridad); el sistema arranca, pero la diferencia con el modo normal es que el controlador de teclado permanece bloqueado hasta el el *password* POP es introducido. De esta forma se consigue que la máquina proporcione todos sus servicios (el arranque es completo) pero que, si alguien quiere acceder a la misma desde consola, esté obligado a introducir la contraseña POP previamente definida.

Aparte del *password* POP, en las máquinas RS/6000 puede establecerse una segunda contraseña denominada PAP (*Privileged Access Password*) o *Supervisory Password*, que protege el arranque del SMS (*System Management Services*), un *firmware* que proporciona al administrador de la máquina ciertas utilidades de configuración de bajo nivel; el SMS es accesible en un determinado punto de la secuencia de arranque de la máquina, en concreto cuando el *display* marca 'FF1'

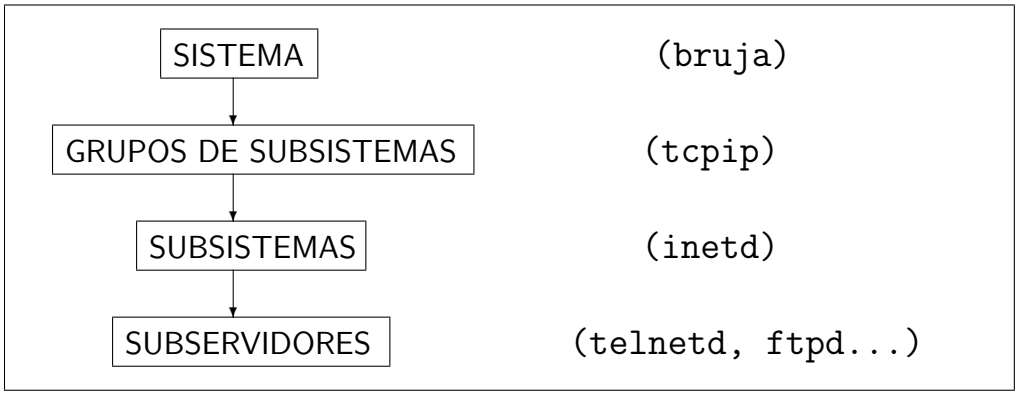

Figura 11.1: Estructura jerárquica del SRC.

```
 gated            tcpip                inoperative
 named            tcpip                inoperative
 routed           tcpip                inoperative
 rwhod            tcpip                inoperative
 iptrace          tcpip                inoperative
 timed            tcpip                inoperative
 dhcpcd           tcpip                inoperative
 dhcpsd           tcpip                inoperative
 dhcprd           tcpip                inoperative
 ndpd-host        tcpip                inoperative
 ndpd-router      tcpip                inoperative
 llbd             iforncs              inoperative
 glbd             iforncs              inoperative
 i4lmd            iforls               inoperative
 i4glbcd          iforncs              inoperative
 i4gdb            iforls               inoperative
 i4llmd           iforls               inoperative
 mrouted          tcpip                inoperative
 rsvpd            qos                  inoperative
 policyd          qos                  inoperative
 pxed             tcpip                inoperative
 binld            tcpip                inoperative
 dtsrc                                 inoperative
bruja:/#
```

Los subsistemas con funciones relacionadas entre sí forman lo que se denomina **grupos de subsistemas**, y además cada subsistema se divide en **subservidores** (simples programas o procesos), formando una estructura jerárquica como la mostrada en la figura 11.1 (figura en la que además se muestra un ejemplo – entre paréntesis – de cada categoría); como podemos ver en ella, un grupo de subsistemas es `tcpip`, que como su nombre ya adelanta es el encargado de la gestión de protocolos TCP/IP, y que incluye a subsistemas tan importantes como `inetd` o

básica, la propia IBM recomienda que el `root` de un sistema AIX sólo se autentique a través de ficheros de contraseñas locales (la entrada SYSTEM de la *stanza* del superusuario ha de tener como valor `'files'`).

Una vez que un usuario se ha autenticado correctamente y ha accedido al sistema entran en juego otra serie de directivas que nos pueden resultar interesantes de cara a nuestra seguridad. La menos importante es `umask`, que como su nombre indica define, mediante tres dígitos octales, la máscara por defecto de cada usuario; decimos que es la menos importante porque, como sucede en cualquier Unix, el usuario puede modificar ese valor por defecto siempre que lo desee, simplemente ejecutando la orden `umask` desde línea de comandos.

Dos entradas que también afectan a la seguridad de usuarios ya dentro del sistema son `su` y `sugroups`; la primera, cuyo valor puede ser `true` o `false` (o sus equivalentes), indica si los usuarios de la máquina pueden ejecutar la orden `su` para cambiar su identidad a la del usuario en cuya *stanza* se ha definido la directiva: ojo, no se trata de limitar la ejecución de `su` a un usuario concreto, sino de limitar al resto la posibiliad de convertirse en ese usuario a través de este comando. Asignar a la directiva `su` el valor de `true` puede ayudarnos a proteger ciertas cuentas especiales de la máquina que no nos interesa que sean alcanzadas de ninguna forma por el resto de usuarios. Por su parte, la segunda entrada a la que hacíamos referencia, `sugroups`, define los grupos cuyos miembros pueden (o que no pueden, si precedemos su nombre por el símbolo `'!'`) acceder mediante la orden `su` a una cuenta determinada, evidentemente si el valor de la directiva `su` de esa cuenta es verdadero.

Para finalizar este punto vamos a comentar brevemente un último grupo de directivas dentro del archivo `/etc/security/user` que definen características de los usuarios del sistema. En primer lugar, `admin` define el estado administrativo de un usuario: si es administrador o si no lo es; en caso afirmativo sólo el `root` puede modificar las características de ese usuario. Independientemente del estado administrativo de un usuario, cada uno puede ser administrador de grupos concretos (grupos que no sean administrativos, ya que en este caso, como sucede con la definición de usuarios, sólo el `root` puede modificar sus parámetros); entra entonces en juego el valor de `admgroups`, que indica los grupos (separados por comas) de los que el usuario es administrador.

Otra característica de cada usuario es la posibilidad del mismo de ejecutar tareas relacionadas con el SRC; es controlada por la directiva `daemon`, que puede tomar el valor de verdadero o falso. La entrada `registry` define la forma de gestionar la autenticación de un usuario concreto: en local (`files`) o en entornos distribuidos (NIS o DCE). Por último, `tpath` marca las características del *Trusted Path*: `nosak`, si el *Secure Attention Key* (recordemos, `Ctrl-X Ctrl-R` en AIX) no tiene efecto, `on`, si al pulsar el SAK se accede al *Trusted Path*, `always`, si siempre que un usuario accee al sistema lo hace al *Trusted Path*, y finalmente `notsh`, que desconecta al usuario al pulsar `Ctrl-X Ctrl-R`, lo que implica que nunca puede

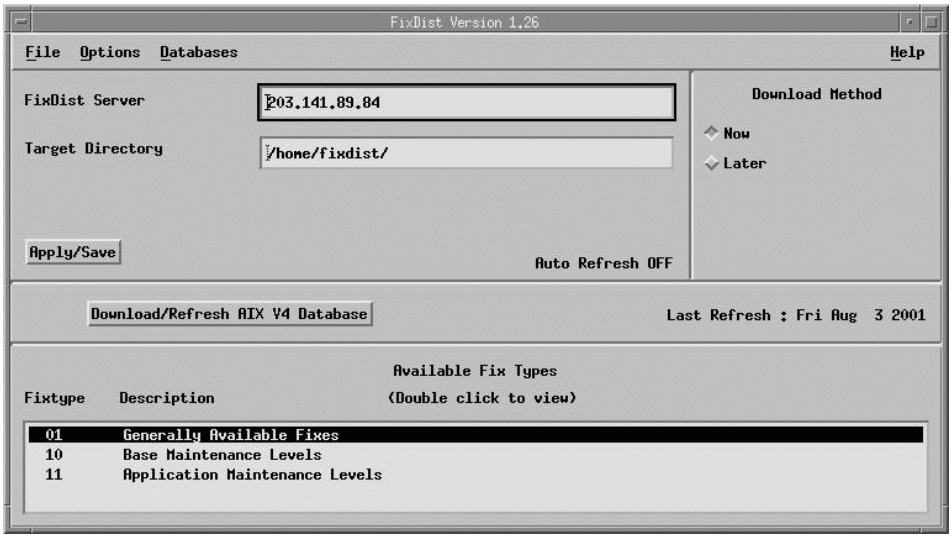

Figura 11.2: Interfaz de `fixdist` (AIX).

```
bruja:/# lslpp -l bos.net.ipsec.rte
  Fileset                      Level  State      Description
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  bos.net.ipsec.rte          4.3.3.0  COMMITTED  IP Security

Path: /etc/objrepos
  bos.net.ipsec.rte          4.3.3.0  COMMITTED  IP Security
bruja:/#
```

Un túnel define una asociación entre dos máquinas en las que se han especificado parámetros de seguridad compartidos entre los dos extremos del túnel; el hecho de que se implique a otra máquina hace que, excepto en entornos muy homogénos – sistemas AIX – o en casos concretos, no se suela utilizar este mecanismo. En cambio, el filtrado de paquetes sí que se utiliza, ya que proporciona a un entorno aislado una protección frente a otras máquinas sin tener que modificar para nada el operativo o las aplicaciones de estas: proporciona un sistema de filtrado sencillo pero efectivo en máquinas en las que por cualquier motivo – dinero, utilización, rendimiento... – no se pueden implantar otras soluciones cortafuegos más completas, como *CheckPoint Firewall–1* o *IBM SecureWay Firewall*. Por este motivo, en este punto vamos a comentar brevemente algunos aspectos del filtrado de paquetes en AIX, sin entrar en el apartado de túneles; si alguien está interesado en profundizar más en estos mecanismos de seguridad IP para AIX, puede consultar [IBM97b].

Para gestionar este sistema de filtrado podemos utilizar órdenes como `rmfilt`, `mkfilt`, `genfilt` o `lsfilt`; como siempre, es recomendable consultar las pági-

que sólo se marca un nodo por el que la trama ha de pasar y no el camino completo) pueden ser reenviados a través del *host*; su valor también ha de ser falso[3]:

```
bruja:/# no -o ipsrcroutesend=0
bruja:/# no -o ipsrcrouterecv=0
bruja:/# no -o ipsrcrouteforward=0
bruja:/# no -o ip6srcrouteforward=0
bruja:/# no -o nonlocsrcroute=0
bruja:/#
```

Respecto al *timeout* de la caché ARP del que ya hemos hablado en Solaris y Linux, en AIX este parámetro viene definido por la variable `arpt_killc`; no obstante, su valor afecta tanto a entradas ARP no utilizadas como a entradas activas, por lo que nos debemos plantear con cuidado la conveniencia de modificarlo; en cualquier caso, podemos definir en minutos el *timeout* también mediante la orden `no`:

```
bruja:/# no -o arpt_killc=1
bruja:/#
```

Además de la orden `'no'`, otro comando que nos permite configurar parámetros interesantes para nuestra seguridad a nivel del subsistema de red – especialmente si administramos un servidor NFS – es `'nfso'`, que configura y visualiza diferentes parámetros relacionados con este sistema de ficheros; su sintaxis es bastante similar a la de `'no'` (en cualquier caso, como sucede con todas las órdenes de un sistema Unix, es recomendable consultar la página de manual correspondiente), y entre los parámetros que permite configurar existen también algunos relacionados con la seguridad del sistema. Quizás el más importante es `portcheck`, que define el comportamiento del servidor ante peticiones provenientes de puertos no privilegiados: si su valor es 0 (por defecto es así) no se efectúa ningún tipo de comprobación, mientras que si es 1 se realiza el *port checking* y sólo se admiten peticiones desde puertos remotos privilegiados; por ejemplo, si queremos que esto sea así debemos ejecutar la siguiente orden (recordamos que, al igual que sucedía con `'no'`, los cambios sólo tienen efecto sobre el *kernel* que se encuentra en ejecución, por lo que si se produce un reinicio de la máquina el comportamiento será el definido por defecto):

```
bruja:/# nfso -o portcheck
portcheck= 0
bruja:/# nfso -o portcheck=1
bruja:/# nfso -o portcheck
portcheck= 1
bruja:/#
```

Existen otros parámetros configurables mediante `nfso` que nos pueden resultar interesantes para incrementar nuestra seguridad; por ejemplo, `nfs_use_reserve_`

[3]En sistemas SP/2 (*Scallable Power Parallel*) de IBM esto puede ser problemático para algunas aplicaciones, por lo que es conveniente consultar la documentación del sistema en cada caso.

Capítulo 12

HP-UX

12.1. Introducción

HP-UX es la versión de Unix desarrollada y mantenida por Hewlett–Packard desde 1983, ejecutable típicamente sobre procesadores HP PA RISC y en sus última versiones sobre Intel Itanium (arquitectura Intel de 64 bits); a pesar de estar basada ampliamente en *System V* incorpora importantes características BSD. En la actualidad la última versión del operativo es la 11.11, aunque existen numerosas instalaciones de sistemas más antiguos, especialmente HP-UX 10.x o incluso 9.x.

HP-UX es, como la mayor parte de Unices comerciales, un entorno de trabajo flexible, potente y estable, que soporta un abanico de aplicaciones que van desde simples editores de texto a complicados programas de diseño gráfico o cálculo científico, pasando por sistemas de control industrial que incluyen planificaciones de tiempo real.

Durante los últimos años Hewlett–Packard, como muchos otros fabricantes, parece haberse interesado bastante por la seguridad en general, y en concreto por los sistemas de protección para sus sistemas; prueba de ello es la gama de productos relacionados con este campo desarrollados para HP-UX, como el sistema de detección de intrusos IDS/9000 para HP-UX 11.x corriendo sobre máquinas HP-9000 o la utilidad *Security Patch Check*, similar al *PatchDiag* de Sun Microsystems. También es importante destacar las grandes mejoras en cuanto a seguridad del sistema se refiere entre HP-UX 9.x, HP-UX 10.x y muy especialmente HP-UX 11.x.

En este capítulo, como hemos hecho antes con Solaris, Linux y AIX, hablaremos de aspectos de la seguridad particulares de HP-UX, teniendo siempre presente que el resto de capítulos del documento también son aplicables a este operativo. Para conocer más HP-UX podemos consultar [Reh00] o [PP01], y si nos interesa especialmente su seguridad una obra obligatoria es [Won01]. Aparte de documentación impresa, a través de Internet podemos acceder a numerosa documentación técnica de HP-UX, en la URL `http://docs.hp.com/`; por supuesto, también son básicas

para una correcta administración las páginas de manual que se instalan con el producto.

12.2. Seguridad física en PA–RISC

Los sistemas de las series HP9000 incluyen un *firmware* muy similar a la *OpenBoot PROM* de las estaciones y servidores SPARC que se denomina PDC (*Processor Dependent Code*) y que implementa las funcionalidades dependientes del procesador; su función básica es, en el arranque de la máquina, inicializar el procesador y comprobar que su estado es correcto mediante unas pruebas llamadas POST (*Power On Self Test*). Si todo es satisfactorio el PDC carga el ISL (*Initial System Loader*) o IPL (*Initial Program Loader*) y le transfiere el control para que este pueda arrancar el sistema operativo.

Como en cualquier dispositivo *hardware* existen formas de interrumpir el proceso de arranque habitual para modificar su comportamiento; en concreto, en la familia HP9000 suele existir un intervalo de 10 segundos antes de cargar el ISL en el que sin más que pulsar la tecla ESC[1] se puede acceder al menú de arranque del equipo y, desde este, definir dispositivos de arranque alternativos o interactuar con el ISL, por ejemplo para pasar un parámetro al *kernel* de HP-UX antes de cargarlo; decimos 'suele existir' porque el intervalo durante el cual se puede interrumpir el autoarranque se respeta sólo si las variables `autoboot` y `autosearch` del ISL están activadas, ya que en caso contrario el sistema automáticamente accede al menú de arranque y no se inicia hasta que un operador no interactua con el mismo.

Si al interrumpir el proceso de arranque elegimos interactuar con el ISL llegamos a un *prompt* sencillo en el que podemos comenzar a introducir órdenes desde el cargador `hpux`, como `'hpux -v'`, que muestra la versión del ISL o `'hpux -iS'`, que inicia el operativo en modo monousuario:

```
Hard booted.

ISL Revision A.00.09  March 27, 1990

ISL> hpux -v
Secondary Loader 9000/700
Revision 1.1
@(#) Revision 1.1 Wed Dec 10 17:24:28 PST 1986

ISL>
```

Antes de acceder a este *prompt* podemos activar o resetar una variable denominada `secure`, que indica el modo de arranque seguro del equipo; si está activada

[1] Dependiendo de la serie concreta es posible que la pulsación de cualquier tecla interrumpa el proceso; esto sucede, por ejemplo, en los modelos de las series 800.

Versión	Privilegio	Descripción
9	RTPRIO	Especificación de prioridades de tiempo real
9	MLOCK	Utilización de `plock()`
9	CHOWN	*System V* `chown`
9	LOCKRDONLY	Utilización de `lockf()`
9	SETRUGID	Utilización de `setuid()` y `setgid()`
10	MPCTL	Utilización de `mpctl()`
10	RTSCHED	Utilización de `sched_setparam()`
10	SERIALIZE	Utilización de `serialize()`
11	SPUCTL	Utilización de `spuctl()`
11i	FSSTHREAD	Utilización de `fss()`
11i	PSET	Utilización de `pset_*()`

Cuadro 12.1: Privilegios de grupo en HP-UX

ellos algunos tan decisivos como la longitud mínima de una contraseña. Un esquema de este tipo resulta algo pobre actualmente, y como ya dijimos, cualquier Unix moderno debería incluir 'de serie' la posibilidad de ofrecer una granularidad más adecuada en todo lo respectivo a las claves de los usuarios. Sea como sea, el esquema seguido en HP-UX es muy similar al de Solaris en cuanto a los requisitos mínimos para un *password*: al menos seis caracteres, dos de los cuales han de ser letras y uno numérico o especial, diferencias con la contraseña anterior en al menos tres caracteres – considerando equivalentes a mayúsculas y minúsculas para este propósito –, clave diferente del nombre de usuario y cualquier rotación del mismo, etc.

Ya para finalizar este punto, y relacionado también con la gestión de usuarios en HP-UX (aunque no explícitamente con el acceso de los mismos al sistema), es necesario hablar brevemente de los privilegios de grupo; este mecanismo, introducido en HP-UX 9.0, permite asignar a un grupo ciertos privilegios de administración, distribuyendo en cierta forma el 'poder' del superusuario y rompiendo la aproximación al reparto de privilegios clásico de Unix (todo o nada). En la tabla 12.1 se muestran los privilegios de grupo soportados en HP-UX junto a la versión del operativo en la que fueron introducidos ([Spr01]).

Para asignar privilegios a un determinado grupo se utiliza la orden `setprivgrp` desde línea de comandos, y para que las modificaciones sean permanentes en el arranque de la máquina se lee el archivo `/etc/privgroup` (si existe) desde `/etc/rc` en HP-UX 9.x o de `/etc/init.d/set_prvgrp` en versiones superiores; en este fichero se indican (uno por línea) los privilegios a otorgar o eliminar a cada grupo:

```
marta:/# cat /etc/privgroup
-n CHOWN
admins CHOWN
admins SETRUGID
```

esto fuera poco, `pdfck` es incapaz de detectar la presencia de nuevos ficheros en un directorio, con lo que algo tan simple como la aparición de un archivo *setuidado* en el sistema no sería notificado por esta utilidad; incluso ciertos cambios sobre los archivos de los cuales sí tiene constancia pasan desapercibidos para ella, como la modificación de listas de control de acceso en los ficheros:

```
marta:/# lsacl /bin/ps
(bin.%,r-x)(%.sys,r-x)(%.%,r-x) /bin/ps
marta:/# chacl "toni.users+rwx" /bin/ps
marta:/# lsacl /bin/ps
(toni.users,rwx)(bin.%,r-x)(%.sys,r-x)(%.%,r-x) /bin/ps
marta:/#
```

Como vemos, la orden anterior está otorgando a un usuario un control total sobre el archivo `/bin/ps`, con todo lo que esto implica; evidentemente, si esto lo ha realizado un atacante, estamos ante una violación muy grave de nuestra seguridad. Sin embargo, `pdfck` ejecutado sobre el PDF correspondiente no reportaría ninguna anomalía, con lo que la intrusió pasaría completamente desapercibida para el administrador.

A la vista de estos problemas, parece evidente que si necesitamos un verificador de integridad 'de verdad' no podemos limitarnos a utilizar las facilidades proporcionadas por los PDFs de HP-UX; no obstante, ya que este mecanismo viene 'de serie' con el sistema, no está de más usarlo, pero sin basarnos ciegamente en sus resultados. Siempre hemos de tener en cuenta que la información de cada fichero registrada en los archivos `/system/*/pdf` se almacena igual que se está almacenando el propio archivo, en un cierto directorio de la máquina donde evidentemente el administrador puede escribir sin poblemas. Por tanto, a nadie se le escapa que si un atacante consigue el privilegio suficiente para modificar una herramienta de base (por ejemplo, `/bin/ls`) y troyanizarla, no tendrá muchos problemas en modificar también el archivo donde se ha guardado la información asociada a la misma, de forma que limitándonos a comparar ambas no nos daremos cuenta de la intrusión. Así, es una buena idea guardar, nada más instalar el sistema, una copia del directorio `/system/`, donde están los PDFs, en una unidad de sólo lectura; cuando lleguemos al tema dedicado a la detección de intrusos hablaremos con más detalle de los verificadores de integridad y la importancia de esa primera copia, sobre un sistema limpio, de toda la información que posteriormente vamos a verificar.

12.5.2. `inetd.sec(4)`

Desde hace más de diez años, HP-UX incorpora en todas sus *releases* un mecanismo de control de acceso a los servicios que el sistema ofrece a través de `inetd`; se trata de un esquema muy similar al ofrecido por *TCP Wrappers*, esto es, basado en la dirección IP de la máquina o red solicitante del servicio, y procesado **antes** de ejecutar el demonio que va a servir la petición correspondiente. De esta forma, se establece un nivel de seguridad adicional al ofrecido por cada uno de estos demonios.

Parte IV

Seguridad de la subred

Capítulo 13

El sistema de red

13.1. 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].

13.2. Algunos ficheros importantes

13.2.1. 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 aliases 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:~#
```

13.2.2. 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 *ethernet*, 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. No obstante, aún resulta útil en determinados casos, por ejemplo en cortafuegos con tarjetas *quad* donde todas las interfaces de la tarjeta tienen la misma dirección MAC.

13.2.3. 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        192.168.0.0
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.

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

xiones 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`:

```
anita:/# netstat -P tcp -f inet -a
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)

Capítulo 14

Algunos servicios y protocolos

14.1. 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 sistemas[1]. 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

[1] No 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.

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 lo servimos deste `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 órden 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 *shellscript* para conseguirlo:

```
* * * * *     kill -HUP `ps -auxw|grep inetd|grep -v grep|awk '{print $2}'`
```

14.2. 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.

14.2.1. 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 -auwwx`

[2]Es recomendable consultar la sintaxis de estos ficheros en el clon de Unix en que trabajemos, ya que puede variar entre diferentes Unices.

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.

14.2.7. 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`

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:~#
```

14.3.1. 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

```
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
        chmod 111 $1/bin
        chown root $1/bin
        if [ $OS = "AIX" ]; then
        for j in `ldd $i|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 $i|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
```

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

Capítulo 15

Cortafuegos: Conceptos teóricos

15.1. 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 15.1 (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 15.1 (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 15.1 (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

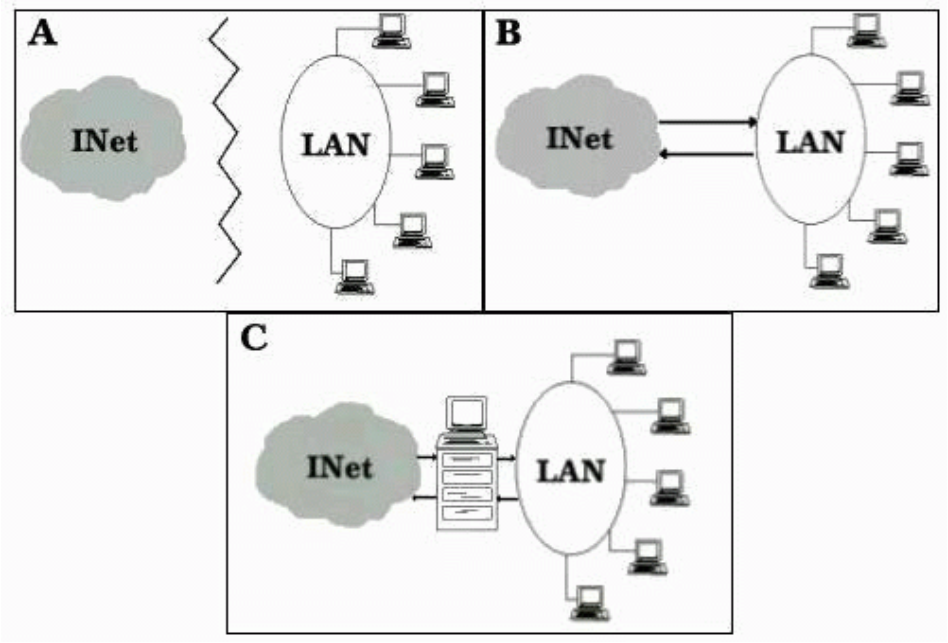

Figura 15.1: (a) Aislamiento. (b) Conexión total. (c) *Firewall* entre la zona de riesgo y el perímetro de seguridad.

so). 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]).

15.3.3. 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 leidos 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 entretener al pirata ante una posible traza de su conexión. Evidentemente, nos estamos arriesgando a que nuestro atacante descubra el

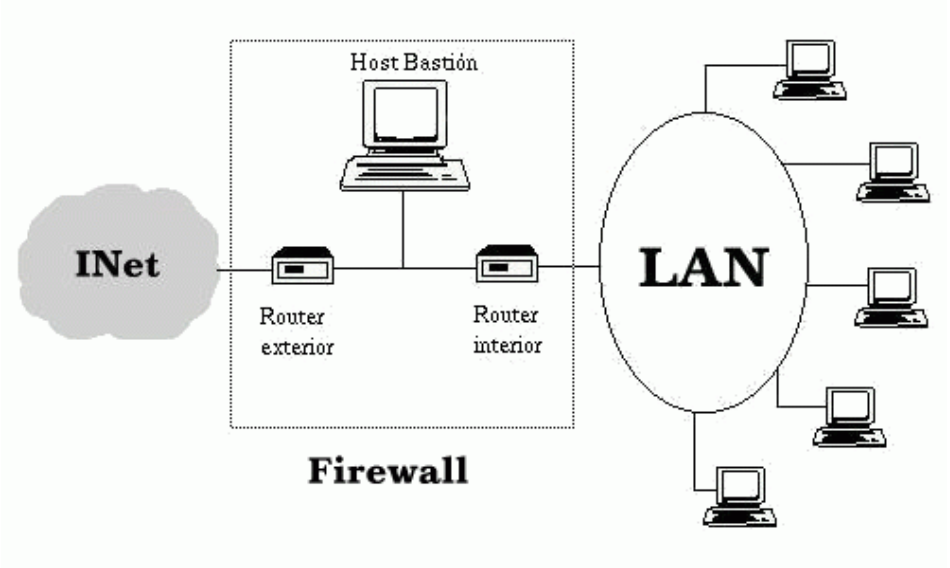

Figura 15.2: Arquitectura DMZ.

Esta arquitectura de cortafuegos elimina los puntos únicos de fallo presentes en las anteriores: antes de llegar al bastión (por definición, el sistema más vulnerable) un atacante ha de saltarse las medidas de seguridad impuestas por el enrutador externo. Si lo consigue, como hemos aislado la máquina bastión en una subred estamos reduciendo el impacto de un atacante que logre controlarlo, ya que antes de llegar a la red interna ha de comprometer también al segundo *router*; en este caso extremo (si un pirata logra comprometer el segundo *router*), la arquitectura DMZ no es mejor que un *screened host*. Por supuesto, en cualquiera de los tres casos (compromiso del *router* externo, del *host* bastión, o del *router* interno) las actividades de un pirata pueden violar nuestra seguridad, pero de forma parcial: por ejemplo, simplemente accediendo al primer enrutador puede aislar toda nuestra organización del exterior, creando una negación de servicio importante, pero esto suele ser menos grave que si lograra acceso a la red protegida.

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 abran importantes brechas de seguridad en nuestro sistema.

Capítulo 16

Cortafuegos: Casos de estudio

16.1. *Firewall-1*

16.1.1. Introducción

Quizás el cortafuegos más utilizado actualmente en Internet es *FireWall-1*, desarrollado por la empresa israelí *Check Point Software Technologies Ltd.* (`http://www.checkpoint.com/`). Este *firewall* se ejecuta sobre diferentes sistemas Unix (Solaris, AIX, Linux y HP-UX), así como sobre Windows NT y también en 'cajas negras' como las desarrolladas por Nokia, que poseen un sistema operativo propio (IPSO) basado en FreeBSD.

Quizás la característica más importante de *Firewall-1* sea que 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 16.1; 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 en el cortafuegos.

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

```
Configuration Options:
----------------------
(1)  Licenses
(2)  Administrators
(3)  GUI clients
(4)  Remote Modules
(5)  SMTP Server
(6)  SNMP Extension
(7)  Groups
(8)  IP Forwarding
(9)  Default Filter
(10) CA Keys

(11) Exit

Enter your choice (1-11) : 11

Thank You...
anita:/#
```

Como podemos ver, esta herramienta permite realizar tareas como la instalación de licencias, la planificación del *software* en el arranque de la máquina o la definición de módulos de *firewall* remotos. Aunque todo es vital para el correcto funcionamiento del cortafuegos, existen dos apartados especialmente importantes para la posterior gestión del *firewall*: la definición de administradores y la definición de estaciones que actuarán como clientes gráficos; si no definimos al menos un administrador y una estación cliente, no podremos acceder al cortafuegos para gestionarlo (evidentemente, en esta situación no estaría todo perdido, ya que siempre podemos añadir ambos elementos, así como modificar su relación, *a posteriori*).

El administrador o administradores que definamos serán los encargados de acceder a las políticas del cortafuegos a través del gestor gráfico, únicamente desde las estaciones que hayamos definido como cliente. Podemos añadir elementos a ambas listas (la de administradores y la de estaciones gráficas) ejecutando de nuevo `fwconfig` o `cpconfig`, o bien de forma más directa ejecutando la orden `fwm` (para añadir administradores) y modificando el archivo `$FWDIR/conf/gui-clients` (para añadir clientes gráficos); este archivo no es más que un fichero de texto donde se listan las direcciones IP (o los nombres DNS) de las máquinas que pueden acceder a gestionar el *firewall*:

```
anita:/# cat $FWDIR/conf/gui-clients
192.168.0.1
192.168.0.2
192.168.0.3
158.42.22.41
anita:/# fwm -p
```

Tal y como hemos definido hasta ahora nuestra política de seguridad, sólo estamos permitiendo conexiones al puerto 80 desde cualquier máquina y al puerto 22 desde la especificada; el resto del tráfico de entrada está siendo denegado gracias a la política por defecto que hemos establecido para la *chain* input (DENY). Así, tráfico como los mensajes ICMP de vuelta, o las llamadas al servicio `ident` que realizan ciertos servidores cuando se les solicita una conexión no alcanzarán su destino, lo cual puede repercutir en la funcionalidad de nuestro entorno: simplemente hemos de pensar en una comprobación rutinaria de conectividad vía `ping` o en el acceso a un servidor FTP externo que sea denegado si no se consigue la identidad del usuario remoto. Para evitar estos problemas, podemos permitir el tráfico de ciertos paquetes ICMP y el acceso al servicio `auth` (puerto 113):

```
luisa:~# /sbin/ipchains -A input -p icmp --icmp-type \
> destination-unreachable -j ACCEPT
luisa:~# /sbin/ipchains -A input -p icmp --icmp-type source-quench -j ACCEPT
luisa:~# /sbin/ipchains -A input -p icmp --icmp-type time-exceeded -j ACCEPT
luisa:~# /sbin/ipchains -A input -p icmp --icmp-type parameter-problem \
> -j ACCEPT
luisa:~# /sbin/ipchains -A input -p icmp --icmp-type echo-reply -j ACCEPT
luisa:~# /sbin/ipchains -A input -p tcp -j ACCEPT -d 158.42.22.41 113
luisa:~#
```

Como vemos, hemos ido definiendo las reglas que conforman nuestra política desde línea de comando; ya hemos comentado que toda esta configuración se pierde al detener el sistema, por lo que es necesario crear un *script* que las vuelva a generar y planificarlo para que se ejecute en el arranque de la máquina. Para ello no tenemos que escribir línea a línea la configuración vista en este punto (mejor, la configuración adecuada a nuestro entorno), o utilizar `ipchains-restore` o `iptables-restore` para leer esa configuración de un fichero; si elegimos esta opción, antes de detener al sistema hemos de ejecutar `ipchains-save` o `iptables-save` para guardar dicha política en el fichero que posteriormente leeremos. De esta forma, en la parada de la máquina hemos de ejecutar una orden similar a:

```
luisa:~# /sbin/ipchains-save >/etc/rc.d/policy
Saving 'input'.
luisa:~#
```

Mientras que en el arranque, en nuestro *script* cargaremos la política guardada al detener la máquina con una orden como:

```
luisa:~# /sbin/ipchains-restore </etc/rc.d/policy
luisa:~#
```

16.2.4. El sistema de *log*

Evidentemente, tanto `ipchains` como `iptables` están preparados para generar *logs* en el sistema: ambos permiten registrar mediante `syslogd` los paquetes que cumplan 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

16.4.2. La primera sesión con PIX *Firewall*

Si conectamos al *firewall* por consola a través de una línea serie entramos directamente sin necesidad de contraseña, en modo no privilegiado; esto lo sabemos porque nos aparece el *prompt* siguiente:

```
pixie>
```

Si en este *prompt* tecleamos la orden **'?'**, nos mostrará la ayuda disponible en el modo sin privilegios:

```
dixie> ?
enable          Enter privileged mode or change privileged mode password
pager           Control page length for pagination
quit            Disable, end configuration or logout
dixie>
```

Son pocos comandos con los que apenas se puede hacer nada; la orden **pager** nos permite ajustar el número de líneas para paginar, la orden **quit** (o **exit**) sale del *firewall*, y la orden **enable** nos pasa a modo superusuario, pidiendo la contraseña (que por defecto será `'cisco'`); cada orden del PIX se puede abreviar (por ejemplo, en lugar de `enable` podríamos teclear `ena`):

```
dixie> ena
Password: *****
dixie#
```

Como vemos, al estar en modo privilegiado, el *prompt* cambia y nos muestra una almohadilla; en este modo ya podemos reconfigurar parámetros del PIX, y tenemos más órdenes disponibles que antes:

```
dixie# ?
arp             Change or view the arp table, and set the arp timeout value
auth-prompt     Customize authentication challenge, reject or acceptance prompt
configure       Configure from terminal, floppy, or memory, clear configure
copy            Copy image from TFTP server into flash.
debug           Debug packets or ICMP tracings through the PIX Firewall.
disable         Exit from privileged mode
enable          Modify enable password
flashfs         Show or destroy filesystem information
kill            Terminate a telnet session
pager           Control page length for pagination
passwd          Change Telnet console access password
ping            Test connectivity from specified interface to <ip>
quit            Disable, end configuration or logout
reload          Halt and reload system
session         Access an internal AccessPro router console
terminal        Set terminal line parameters
who             Show active administration sessions on PIX
write           Write config to net, flash, floppy, or terminal, or erase flash
dixie#
```

Para comenzar a reconfigurar el *firewall* nos pondremos en modo configuración (desde modo privilegiado) con la orden **configure** (la 't' corresponde a `Terminal`); de nuevo, cambia el *prompt* que nos aparece en consola:

Capítulo 17

Ataques remotos

17.1. Escaneos de puertos

Una de las primeras actividades que un potencial (o no tan potencial) atacante realizará contra su objetivo será sin duda un escaneo de puertos, un *portscan*; esto le permitirá obtener en primer lugar información básica acerca de qué servicios estamos ofreciendo en nuestras máquinas y, adicionalmente, otros detalles de nuestro entorno como qué sistema operativo tenemos instalados en cada *host* o ciertas características de la arquitectura de nuestra red. Analizando qué puertos están abiertos en un sistema, el atacante puede buscar agujeros en cada uno de los servicios ofrecidos: cada puerto abierto en una máquina es una potencial puerta de entrada a la misma.

Comprobar el estado de un determinado puerto es *a priori* una tarea muy sencilla; incluso es posible llevarla a cabo desde la línea de órdenes, usando una herramienta tan genérica como `telnet`. Por ejemplo, imaginemos que queremos conocer el estado del puerto 5000 en la máquina cuya dirección IP es 192.168.0.10; si el `telnet` a dicho puerto ofrece una respuesta, entonces está abierto y escuchando peticiones:

```
anita:~$ telnet 192.168.0.10 5000
Trying 192.168.0.10...
Connected to 192.168.0.10.
Escape character is '^]'.
^D
Connection closed by foreign host.
anita:~$
```

Si por el contrario el puerto está abierto pero en él no hay ningún demonio atendiendo peticiones, la respuesta será similar a la siguiente:

```
anita:~$ telnet 192.168.0.10 5000
Trying 192.168.0.10...
telnet: Unable to connect to remote host: Connection refused
anita:~$
```

en caso de estar abierto; como en el caso de los escaneos SYN+ACK este método puede proporcionar muchos falsos positivos, por lo que tampoco se utiliza mucho hoy en día.

También en [Mai96], se propone un método de escaneo algo más complejo: el ACK. El atacante envía una trama con este *bit* activo, y si el puerto destino está abierto es muy posible que o bien el campo TTL del paquete de vuelta sea menor que el del resto de las tramas RST recibidas, o que el tamaño de ventana sea mayor que cero: como podemos ver, en este caso no basta con analizar el bit RST sino también la cabecera IP del paquete respuesta. Este método es difícil de registrar por parte de los detectores de intrusos, pero se basa en el código de red de BSD, por lo que es dependiente del operativo escaneado.

Para finalizar con la familia de *stealth scanning* vamos a hablar de dos métodos opuestos entre sí pero que se basan en una misma idea y proporcionan resultados similares: se trata de XMAS y NULL. Los primeros, también denominados escaneos 'árbol de navidad', se basan en enviar al objetivo tramas con todos los *bits* TCP (URG, ACK, PST, RST, SYN y FIN) activos; si el puerto está abierto el núcleo del sistema operativo eliminará la trama, ya que evidentemente la considera una violación del *three–way handshake*, pero si está cerrado devolverá un RST al atacante. Como antes, este método puede generar demasiados falsos positivos, y además sólo es aplicable contra máquinas Unix debido a que está basado en el código de red de BSD.

Por contra, el método opuesto al XMAS se denomina NULL *scanning*, y evidentemente consiste en enviar tramas con todos los *bits* TCP reseteados; el resultado es similar: no se devolverá ningún resultado si el puerto está abierto, y se enviará un RST si está cerrado. Aunque en principio este método sería aplicable a cualquier pila TCP/IP ([Ark99]), la implementación – incorrecta – que de la misma hacen algunos operativos (entre ellos HP/UX o IRIX) hace que en ocasiones se envien *bits* RST también desde los puertos abiertos, lo que puede proporcionar demasiados falsos positivos.

Antes de acabar el punto, vamos a hablar de otra técnica de escaneo de puertos que no se puede englobar en las tres clases de las que hemos hablado: se trata de los escaneos UDP, que al contrario de todos los comentados hasta el momento utiliza este protocolo, y no TCP, para determinar el estado de los puertos de una máquina. Al enviar un datagrama UDP a un puerto abierto este no ofrece respuesta alguna, pero si está cerrado devuelve un mensaje de error ICMP: ICMP_PORT_UNREACHABLE. Evidentemente, estos ataques son muy sencillos de detectar y evitar tanto en un sistema de detección de intrusos como en los núcleos de algunos Unices, y por si esto fuera poco debemos recordar que UDP no es un protocolo orientado a conexión (como lo es TCP), por lo que la pérdida de datagramas puede dar lugar a un elevado número de falsos positivos.

Hemos repasado las técnicas más habituales – no todas – que un atacante puede

nivel de parcheado o versiones del núcleo, como en lo referente a programas críticos encargados de ofrecer un determinado servicio (demonios, por norma general: `sendmail`, `httpd`, `pop3d`...).

De un tiempo a esta parte – en concreto, desde 1999 – se ha popularizado mucho el término 'negación de servicio distribuida' (*Distributed Denial of Service*, DDoS): en este ataque un pirata compromete en primer lugar un determinado número de máquinas y, en un determinado momento, hace que todas ellas ataquen masiva y simultaneamente al objetivo u objetivos reales enviándoles diferentes tipos de paquetes; por muy grandes que sean los recursos de la víctima, el gran número de tramas que reciben hará que tarde o temprano dichos recursos sean incapaces de ofrecer un servicio, con lo que el ataque habrá sido exitoso. Si en lugar de cientos o miles de equipos atacando a la vez lo hiciera uno sólo las posibilidades de éxito serían casi inexistentes, pero es justamente el elevado número de 'pequeños' atacantes lo que hace muy difícil evitar este tipo de negaciones de servicio.

Según el CERT ([HW01]) los ataques de negación de servicio distribuidos más habituales consisten en el envío de un gran número de paquetes a un determinado objetivo por parte de múltiples *hosts*, lo que se conoce como *packet flooding* (en función del tipo de paquetes utilizados se habla de *ping flood*, de SYN *flood*...). Defenderse de este tipo de ataques es difícil: en primer lugar, uno piensa en bloquear de alguna forma (probablemente en un cortafuegos o en un *router*) todo el tráfico proveniente de los atacantes; pero ¿qué sucede cuando tenemos miles de ordenadores atacando desde un gran número de redes diferentes? ¿Los bloqueamos uno a uno? Esto supondría un gran esfuerzo que difícilmente ayudaría, ya que lo más probable es que en el tiempo que nos cueste bloquear el tráfico de una determinada máquina, dos o tres nuevas nos comiencen a atacar. Entonces, ¿bloqueamos todo el tráfico dirigido hacia el objetivo? Si hacemos esto, estamos justamente ayudando al atacante, ya que somos nosotros mismos los que causamos una negación en el servicio a los usuarios legítimos de nuestro sistema...

Como vemos, la defensa ante una negación de servicio distribuida no es inmediata; en cualquier caso, podemos tomar ciertas medidas preventivas que nos ayudarán a limitar el alcance de uno de estos ataques (y en general de las negaciones de servicio remotas, distribuidas o no). De entrada, un correcto filtrado del tráfico dirigido a nuestras máquinas es vital para garantizar nuestra seguridad: no hay que responder a *pings* externos a nuestra red, es necesario activar el *antispoofing* en nuestros cortafuegos y en los elementos de electrónica de red que lo permitan, etc. Establecer correctamente límites a la utilización de nuestros recursos, como ya hemos visto, es también una importante medida preventiva; es posible limitar el ancho de banda dedicado a una determinada aplicación o a un protocolo, de forma que las utilizaciones por encima del margen son negadas. También podemos limitar los recursos del sistema (CPU, memoria, disco...) que puede consumir en global una determinada aplicación servidora (por ejemplo, un demonio sirviendo páginas *web*), además de restringir sus recursos por cliente simultáneo (en base, por ejemplo, a la dirección origen de ese cliente).

```
luisa:/# telnet 0 25
Trying 0.0.0.0...
Connected to 0.
Escape character is '^]'.
220 Servidor ESMTP
vrfy root
252 Cannot VRFY user; try RCPT to attempt delivery (or try finger)
quit
221 luisa closing connection
Connection closed by foreign host.
luisa:/#
```

En realidad, si un atacante quiere conocer la versión del servidor que estamos utilizando aún no lo tiene difícil, ya que simplemente ha de teclear una orden como ‘`help`’:

```
luisa:~$ telnet 0 25
Trying 0.0.0.0...
Connected to 0.
Escape character is '^]'.
220 Servidor ESMTP
help
214-This is Sendmail version 8.9.3
214-Topics:
214-    HELO    EHLO    MAIL    RCPT    DATA
214-    RSET    NOOP    QUIT    HELP    VRFY
214-    EXPN    VERB    ETRN    DSN
214-For more info use "HELP <topic>".
214-To report bugs in the implementation send email to
214-    sendmail-bugs@sendmail.org.
214-For local information send email to Postmaster at your site.
214 End of HELP info
quit
221 luisa closing connection
Connection closed by foreign host.
luisa:~$
```

Para evitar esto debemos modificar convenientemente el fichero `sendmail.hf` (en función del Unix utilizado su ubicación en la estructura de directorios cambiará) de forma que se restrinjan más los mensajes que proporciona el demonio en una sesión interactiva; para obtener información sobre este fichero, así como del resto de configuraciones de `sendmail` podemos consultar [CA97a]. Debemos tener presente que conocer ciertos datos que el programa proporciona puede facilitarle mucho la tarea a un pirata; ocultar esta información no es ni mucho menos una garantía de seguridad, ni por supuesto ha de suponer uno de los pilares de nuestra política, pero si con un par de pequeñas modificaciones conseguimos quitarnos de encima aunque sólo sea a un atacante casual, bienvenidas sean – aunque muchos

una máquina podemos obtener resultados similares a los siguientes:

```
anita:~/security/whisker$ ./whisker.pl -h luisa
-- whisker / v1.4.0 / rain forest puppy / www.wiretrip.net --

= - = - = - = - = - =
= Host: luisa
= Server: Apache/1.3.19 (Unix) PHP/4.0.4pl1 mod_ssl/2.8.2 OpenSSL/0.9.5a

+ 200 OK: HEAD /docs/
+ 200 OK: HEAD /cgi-bin/Count.cgi
+ 200 OK: HEAD /cgi-bin/textcounter.pl
+ 200 OK: HEAD /ftp/
+ 200 OK: HEAD /guestbook/
+ 200 OK: HEAD /usage/

anita:~/security/whisker$
```

Podemos ver que el servidor nos proporciona excesiva información sobre su configuración (versión, módulos, soporte SSL...), y que la herramienta ha obtenido algunos archivos y directorios que pueden resultar interesantes para un atacante: en el caso de los *CGI* no tiene más que acercarse a alguna base de datos de vulnerabilidades (especialmente recomendables son `http://www.securityfocus.com/` o `http://icat.nist.gov/`) e introducir en el buscador correspondiente el nombre del archivo para obtener información sobre los posibles problemas de seguridad que pueda presentar. El caso de los directorios es algo diferente, pero típicamente se suele tratar de nombres habituales en los servidores que contienen información que también puede resultarle útil a un potencial atacante.

¿Cómo evitar estos problemas de seguridad de los que estamos hablando? Una medida elemental es eliminar del servidor cualquier directorio o *CGI* de ejemplo que se instale por defecto; aunque generalmente los directorios (documentación, ejemplos...) no son especialmente críticos, el caso de los *CGIs* es bastante alarmante: muchos servidores incorporan programas que no son ni siquiera necesarios para el correcto funcionamiento del *software*, y que en ciertos casos – demasiados – abren enormes agujeros de seguridad, como el acceso al código fuente de algunos archivos, la lectura de ficheros fuera del `DocumentRoot`, o incluso la ejecución remota de comandos bajo la identidad del usuario con que se ejecuta el demonio servidor.

Otra medida de seguridad básica es deshabilitar el *Directory Indexing* que por defecto muchos servidores incorporan: la capacidad de obtener el listado de un directorio cuando no existe un fichero `index.html` o similar en el mismo; se trata de una medida extremadamente útil y sobre todo sencilla de implantar, ya que en muchos casos un simple `'chmod -r'` sobre el directorio en cuestión es suficiente para evitar este problema. A primera vista esta medida de protección nos puede resultar curiosa: a fin de cuentas, *a priori* todo lo que haya bajo el *Document Root*

del servidor ha de ser público, ya que para eso se ubica ahí. Evidentemente la teoría es una cosa y la práctica otra muy diferente: entre los ficheros de cualquier servidor no es extraño encontrar desde archivos de *log* – por ejemplo, del cliente FTP que los usuarios suelen usar para actualizar remotamente sus páginas, como WS_FTP.LOG – hasta paquetes TAR con el contenido de subdirectorios completos. Por supuesto, la mejor defensa contra estos ataques es evitar de alguna forma la presencia de estos archivos bajo el *Document Root*, pero en cualquier caso esto no es siempre posible, y si un atacante sabe de su existencia puede descargarlos, obteniendo en muchos casos información realmente útil para atacar al servidor (como el código de ficheros JSP, PHP, ASP... o simplemente rutas absolutas en la máquina), y una excelente forma de saber que uno de estos ficheros está ahí es justamente el *Directory Indexing*; por si esto no queda del todo claro, no tenemos más que ir a un buscador cualquiera y buscar la cadena `'Index of /admin'`, por poner un ejemplo sencillo, para hacernos una idea de la peligrosidad de este error de configuración.

Además, en cualquier servidor *web* es muy importante el usuario bajo cuya identidad se ejecuta el demonio `httpd`: ese usuario no debe ser **nunca** el `root` del sistema (evidente), pero tampoco un usuario genérico como `nobody`; se ha de tratar siempre de un usuario dedicado y sin acceso real al sistema. Por supuesto, las páginas HTML (los ficheros planos, para entendernos) **nunca** deben ser de su propiedad, y mucho menos ese usuario ha de tener permiso de escritura sobre los mismos: con un acceso de lectura (y ejecución, en caso de *CGIs*) es más que suficiente en la mayoría de los casos. Hemos de tener en cuenta que si el usuario que ejecuta el servidor puede escribir en las páginas *web*, y un pirata consigue – a través de cualquier tipo de error (configuración, diseño del demonio...) – ejecutar órdenes bajo la identidad de dicho usuario, podrá modificar las páginas *web* sin ningún problema (que no olvidemos, es lo que perseguirá la mayoría de atacantes de nuestro servidor *web*).

Igual de importante que evitar estos problemas es detectar cuando alguien trata de aprovecharlos intentando romper la seguridad de nuestros servidores; para conseguirlo no tenemos más que aplicar las técnicas de detección de intrusos que veremos en el capítulo siguiente. Una característica importante de los patrones de detección de ataques vía *web* es que no suelen generar muchos falsos positivos, por lo que la configuración de la base de datos inicial es rápida y sencilla, al menos en comparación con la detección de escaneos de puertos o la de tramas con alguna característica especial en su cabecera.

Capítulo 18

Sistemas de detección de intrusos

18.1. Introducción

A pesar de que un enfoque clásico de la seguridad de un sistema informático siempre define como principal defensa del mismo sus controles de acceso (desde una política implantada en un cortafuegos hasta unas listas de control de acceso en un *router* o en el propio sistema de ficheros de una máquina), esta visión es extremadamente simplista si no tenemos en cuenta que en muchos casos esos controles no pueden protegernos ante un ataque ([Lun90]). Por poner un ejemplo sencillo, pensemos en un *firewall* donde hemos implantado una política que deje acceder al puerto 80 de nuestros servidores *web* desde cualquier máquina de Internet; ese cortafuegos sólo comprobará si el puerto destino de una trama es el que hemos decidido para el servicio HTTP, pero seguramente no tendrá en cuenta si ese tráfico representa o no un ataque o una violación de nuestra política de seguridad: por ejemplo, no detendrá a un pirata que trate de acceder al archivo de contraseñas de una máquina aprovechando un *bug* del servidor *web*. Desde un pirata informático externo a nuestra organización a un usuario autorizado que intenta obtener privilegios que no le corresponden en un sistema, nuestro entorno de trabajo no va a estar nunca a salvo de intrusiones.

Llamaremos **intrusión** a un conjunto de acciones que intentan comprometer la integridad, confidencialidad o disponibilidad de un recurso ([HLMS90]); analizando esta definición, podemos darnos cuenta de que una intrusión no tiene por qué consistir en un acceso no autorizado a una máquina: también puede ser una negación de servicio. A los sistemas utilizados para detectar las intrusiones o los intentos de intrusión se les denomina **sistemas de detección de intrusiones** (*Intrusion Detection Systems*, IDS) o, más habitualmente – y aunque no sea la traducción literal – **sistemas de detección de intrusos**; cualquier mecanismo de seguridad con este propósito puede ser considerado un IDS, pero generalmente

o bien que el paquete no ha de ser fragmentado por un *router* intermedio (DF, *Don´t Fragment*) o bien que el paquete ha sido fragmentado y no es el último que se va a recibir (MF, *More Fragments*). Valores incorrectos de parámetros de fragmentación de los datagramas se han venido utilizando típicamente para causar importantes negaciones de servicio a los sistemas y, desde hace también un tiempo incluso para obtener la versión del operativo que se ejecuta en un determinado *host* ([Fyo98]); por ejemplo, ¿qué le sucedería al subsistema de red implantado en el núcleo de una máquina Unix si nunca recibe una trama con el *bit* MF reseteado, indicando que es el último de un paquete? ¿se quedaría permanentemente esperándola? ¿y si recibe uno que en teoría no está fragmentado pero se le indica que no es el último que va a recibir? ¿cómo respondería el núcleo del operativo en este caso? Como vemos, si en nuestras máquinas observamos ciertas combinaciones de *bits* relacionados con la fragmentación realmente tenemos motivos para – cuanto menos – sospechar que alguien trata de atacarnos.

- Dirección origen y destino.
  Las direcciones de la máquina que envía un paquete y la de la que lo va a recibir también son campos interesantes de cara a detectar intrusiones en nuestros sistemas o en nuestra red. No tenemos más que pensar el tráfico proveniente de nuestra DMZ (y que no se trate de la típica respuesta ante una petición como las que se generan al visitar páginas *web*, por poner un ejemplo) que tenga como destino nuestra red protegida: es muy posible que esos paquetes constituyan un intento de violación de nuestra política de seguridad. Otros ejemplos clásicos son las peticiones originadas desde Internet y que tienen como destino máquinas de nuestra organización que no están ofreciendo servicios directos al exterior, como un servidor de bases de datos cuyo acceso está restringido a sistemas de nuestra red.

- Puerto origen y destino.
  Los puertos origen y destino (especialmente este último) son un excelente indicativo de actividades sospechosas en nuestra red. Aparte de los intentos de acceso no autorizado a servicios de nuestros sistemas, pueden detectar actividades que también supondrán *a priori* violaciones de nuestras políticas de seguridad, como la existencia de troyanos, ciertos tipos de barridos de puertos, o la presencia de servidores no autorizados dentro de nuestra red.

- *Flags* TCP.
  Uno de los campos de una cabecera TCP contiene seis *bits* (URG, ACK, PSH, RST, SYN y FIN), cada uno de ellos con una finalidad diferente (por ejemplo, el *bit* SYN es utilizado para establecer una nueva conexión, mientras que FIN hace justo lo contrario: liberarla). Evidentemente el valor de cada uno de estos *bits* será 0 o 1, lo cual de forma aislada no suele decir mucho (ni bueno ni malo) de su emisor; no obstante, ciertas combinaciones de valores suelen ser bastante sospechosas: por ejemplo, una trama con los dos *bits* de los que hemos hablado – SYN y FIN – activados simultáneamente sería indicativa de una conexión que trata de abrirse y cerrarse al mismo tiempo. Para hacernos

desarrollado inicialmente sobre SunOS 4.1.1 (en la actualidad está portado a Solaris 2), y utiliza los registros de auditoría generados por el *C2 Basic Security Module* de este operativo. En USTAT, estos registros del C2-BSM son transformados a otros de la forma <S, A, O>, representando cada uno de ellos un evento de la forma *'el sujeto S realiza la acción A sobre el objeto O'*; a su vez, cada elemento de la terna anterior está formado por diferentes campos que permiten identificar unívocamente el evento representado. El sistema de detección utiliza además una base de datos – realmente, se trata de simples ficheros planos – formada principalmente por dos tablas, una donde se almacenan las descripciones de los diferentes estados (SDT, *State Description Table*) y otra en la que se almacenan las transiciones entre estados que denotan un potencial ataque (SAT, *Signature Action Table*). Cuando USTAT registre una sucesión determinada de eventos que representen un ataque entrará en juego el motor de decisiones, que emprenderá la acción que se le haya especificado (desde un simple mensaje en consola informando de la situación hasta acciones de respuesta automática capaces de interferir en tiempo real con la intrusión).

La tercera implementación que habíamos comentado era la basada en el uso de reglas de comparación y emparejamiento de patrones o *pattern matching* ([SG91], [KS94c]); en ella, el detector se basa en la premisa de que el sistema llega a un estado comprometido cuando recibe como entrada el patrón de la intrusión, sin importar el estado en que se encuentre en ese momento. Dicho de otra forma, simplemente especificando patrones que denoten intentos de intrusión el sistema puede ser capaz de detectar los ataques que sufre, sin importar el estado inicial en que esté cuando se produzca dicha detección, lo cual suele representar una ventaja con respecto a otros modelos de los que hemos comentado.

Actualmente muchos de los sistemas de detección de intrusos más conocidos (por poner un ejemplo, podemos citar a SNORT o *RealSecure*) están basados en el *pattern matching*. Utilizando una base de datos de patrones que denotan ataques, estos programas se dedican a examinar todo el tráfico que ven en su segmento de red y a comparar ciertas propiedades de cada trama observada con las registradas en su base de datos como potenciales ataques; si alguna de las tramas empareja con un patrón sospechoso, automáticamente se genera una alarma en el registro del sistema. En el punto 18.8.2 hablaremos con más detalle del funcionamiento de SNORT, uno de los sistemas de detección de intrusos basados en red más utilizado en entornos con requisitos de seguridad media.

Por último, tenemos que hablar de los sistemas de detección de intrusos basados en modelos ([GL91]); se trata de una aproximación conceptualmente muy similar a la basada en la transición entre estados, en el sentido que contempla los ataques como un conjunto de estados y objetivos, pero ahora se representa a los mismos como escenarios en lugar de hacerlo como transiciones entre estados. En este caso se combina la detección de usos indebidos con una deducción o un razonamiento que concluye la existencia o inexistencia de una intrusión; para ello, el sistema utiliza una base de datos de escenarios de ataques, cada uno de los cuales está

plemente no podremos decidir si se trata de verdaderas o de falsas alarmas. Esto es especialmente crítico si lanzamos respuestas automáticas contra las direcciones 'atacantes' (por ejemplo, detener todo su tráfico en nuestro *firewall*): volviendo al ejemplo de la zona desmilitarizada con servidores *web*, podemos llegar al extremo de detener a simples visitantes de nuestras páginas simplemente porque han generado falsos positivos; aunque en un entorno de alta seguridad quizás vale la pena detener muchas acciones no dañinas con tal de bloquear también algunos ataques (aunque constituiría una negación de servicio en toda regla contra los usuarios que hacen uso legítimo de nuestros sistemas), en un entorno normal de producción esto es impensable. Seguramente será más provechoso detectar y detener estos ataques por otros mecanismos ajenos al sensor.

En resumen, hemos de adaptar a nuestro entorno de trabajo, de una forma muy fina, la base de datos de patrones de posibles ataques. Quizás valga la pena perder tiempo el tiempo que sea necesario en esta parte de la implantación, ya que eso nos ahorrará después muchos análisis de falsas alarmas y, por qué negarlo, algún que otro susto; una vez tengamos todo configurado, podemos utilizar el siguiente *script* para lanzar SNORT de forma automática al arrancar el sistema (Solaris):

```
anita:~# cat /etc/init.d/snort
#!/sbin/sh
#
# Instalacion:
#       # cp <script> /etc/init.d/snort
#       # chmod 744 /etc/init.d/snort
#       # chown root:sys /etc/init.d/snort
#       # ln /etc/init.d/snort /etc/rc2.d/S99snort
#

# Directorio de log
DIRLOG=/var/log/snort
# Fichero de reglas
RULES=/usr/local/security/snort.conf
# Ejecutable
SNORT=/usr/local/security/snort
# Interfaz
IF=hme0

case "$1" in
'start')
        if [ ! -d "$DIRLOG" ]; then
                mkdir -p "$DIRLOG"
        fi
        if [ ! -r "$RULES" ]; then
                echo "No puedo leer el fichero de patrones..."
                exit -1
        fi
```

```
TCP TTL:56 TOS:0x0 ID:5040 IpLen:20 DgmLen:58 DF
***AP*** Seq: 0x91FA846  Ack: 0x5AD9A72  Win: 0x7D78  TcpLen: 20
```

Como veremos en el punto siguiente, es posible que al generar este aviso, el propio sensor decida lanzar una respuesta automática contra la dirección atacante (por ejemplo, bloquearla en el cortafuegos corporativo). Pero SNORT trabaja basándose en su base de datos de patrones de ataques; en dicha base de datos habrá una regla como la siguiente:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"WEB-CGI phf access";\
flags: A+; content:"/phf";flags: A+; nocase; reference:arachnids,128; \
reference:cve,CVE-1999-0067; )
```

A grandes rasgos, esta regla viene a decir que cuando el sensor detecte una petición al puerto 80 de nuestro servidor *web* en cuyo contenido se encuentre la cadena `'/phf'` levante la alerta correspondiente; de esta forma, cuando el atacante solicite el archivo `/cgi-bin/phf` del servidor, SNORT 'verá' dicha cadena en la petición y generará una alarma. ¿Pero qué sucede si el atacante solicita ese mismo fichero pero utilizando una petición 'ofuscada', formada por los códigos ASCII de cada carácter? Es decir, si en lugar de lanzar una petición como `'GET /cgi-bin/phf'` hace una similar a `'GET%2f%63%67%69%2d%62%69%6e%2f%70%68%66'`. La respuesta es sencilla: la petición anterior se 'decodifica' en el propio servidor *web*, o como mucho en un *proxy* intermedio. Algunos detectores de intrusos, como RealSecure, son en teoría capaces de procesar este tipo de tráfico y analizarlo normalmente en busca de patrones sospechosos, pero otros muchos (SNORT en este caso) no; así, estos últimos no verán en ningún momento la cadena `'/phf'` circulando por la red, y por tanto no generarán ninguna alarma. Parece entonces evidente que es necesario un nivel adicional en nuestro esquema de detección: justamente el compuesto por los sistemas basados en la propia máquina a proteger.

Antes hemos hablado de los tres modelos básicos de IDSes basados en máquina: verificadores de integridad, analizadores de registros y *honeypots*. Parece claro que un verificador de integridad en nuestro caso no va a resultar muy útil para detectar a ese atacante (esto no significa que no se trate de modelos necesarios y útiles en otras muchas situaciones). Tendremos por tanto que recurrir a analizadores de *logs* o a tarros de miel instalados en la máquina. Si optamos por los primeros, es sencillo construir un *shellscript* que procese los archivos generados por el servidor *web* – en nuestro caso, aunque igualmente sencillo que procesar registros del sistema o de otras aplicaciones – en busca de patrones que puedan denotar un ataque, como el acceso a determinados archivos bajo el `DocumentRoot`. Si analizamos cualquier *CGI Scanner* nos podremos hacer una idea de a qué patrones debemos estar atentos: por ejemplo, intentos de acceso a CGIs como `phf` o `printenv`, a archivos `passwd`, a ficheros fuera del `DocumentRoot`, etc.

Aunque analizar los registros generados por una aplicación en busca de ciertos patrones sospechosos es una tarea trivial, no lo es tanto el integrar ese análisis en un esquema de detección de intrusos. Seguramente procesar la salida de un `'tail -f'` del archivo de *log* correspondiente, enviando un correo electrónico al responsable de seguridad cuando se detecte una entrada sospechosa es fácil, pero por

```
print "HTTP_ACCEPT = image/gif, image/x-xbitmap, image/jpeg<br>";
print "HTTP_CONNECTION = Keep-Alive<br>";
print "REMOTE_PORT = $ENV{'REMOTE_PORT'}<br>";
print "SERVER_ADDR = $ENV{'SERVER_ADDR'}<br>";
print "HTTP_ACCEPT_LANGUAGE = en<br>";
print "SCRIPT_NAME = /cgi-bin/printenv<br>";
print "HTTP_ACCEPT_ENCODING = gzip<br>";
print "SCRIPT_FILENAME = /usr/local/httpd/cgi-bin/printenv<br>";
print "SERVER_NAME = $ENV{'SERVER_NAME'}<br>";
print "REQUEST_URI = /cgi-bin/printenv<br>";
print "HTTP_ACCEPT_CHARSET = iso-8859-1, utf-8<br>";
print "SERVER_PORT = $ENV{'SERVER_PORT'}<br>";
print "HTTP_HOST = $ENV{'HTTP_HOST'}<br>";
print "SERVER_ADMIN = webmaster<br>";

# Enviamos correo
open (MAIL, "|$mailprog $SECUREADDRESS") or die "Can't open $mailprog!\n";
print MAIL "To: $SECUREADDRESS\n";
print MAIL "From: PRINTENV Watcher <$SECUREADDRESS>\n";
print MAIL "Subject: [/CGI-BIN/PRINTENV] $ENV{'REMOTE_HOST'} $action\n\n";
print MAIL "\n";
print MAIL  "------------------------------------------------------\n";
print MAIL "Remote host: $ENV{'REMOTE_ADDR'}\n";
print MAIL "Server: $ENV{'SERVER_NAME'}\n";
print MAIL "Remote IP address: $ENV{'REMOTE_ADDR'}\n";
print MAIL "HTTP Referer: $ENV{'HTTP_REFERER'}\n";
print MAIL "Query String: $ENV{'QUERY_STRING'}\n";
print MAIL "\n------------------------------------------------------\n";
close(MAIL);
exit;
anita:/#
```

18.8.4. Estrategias de respuesta

Una pregunta importante que nos habíamos realizado con respecto a nuestro esquema de detección de intrusos (en concreto cuando hablábamos de SNORT) era qué hacer con la información obtenida del mismo; como siempre, tenemos varias posibilidades. Por un lado, podemos procesarla manualmente de forma periódica (por ejemplo cada mañana) y en función de los datos que nuestros sensores hayan recogido tomar una determinada acción: bloquear las direcciones atacantes en el *firewall* corporativo, enviar un correo de queja a los responsables de dichas direcciones (por desgracia esto no suele resultar muy útil), realizar informes para nuestros superiores, o simplemente no hacer nada. Dependerá casi por completo de la política de seguridad que sigamos o que tengamos que seguir.

Mucho más interesante que cualquiera de estas respuestas manuales es que el propio sensor sea capaz de generar respuestas automáticas ante lo que él considere un intento de ataque. Si elegimos esta opción, lo más habitual suele ser un bloqueo total de la dirección atacante en nuestro cortafuegos, unida a una notificación de cualquier tipo (SMS, *e-mail*...) a los responsables de seguridad; es siempre recomendable efectuar esta notificación (si no en tiempo real, si al menos registrando las medidas tomadas contra una determinada dirección en un *log*) por motivos evi-

Capítulo 19

Kerberos

19.1. Introducción

Durante 1983 en el M.I.T. (*Massachussetts 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 contínuamente su clave, de forma que se pierde demasiado tiempo y además la contraseña está viajando contínuamente 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).

19.2. 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

C	Cliente que solicita un servicio
S	Servidor que ofrece dicho servicio
A	Servidor de autenticación
T	Servidor de *tickets*
K_C	Clave secreta del cliente
K_S	Clave secreta del servidor
K_T	Clave secreta del servidor de *tickets*
K_{CT}	Clave de sesión entre el cliente y el servidor de *tickets*
K_{CS}	Clave de sesión entre cliente y servidor

Cuadro 19.1: Abreviaturas utilizadas.

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.

19.3. 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 19.1 se muestran las abreviaturas utilizadas, y en la figura 19.1 un resumen gráfico de este protocolo.

19.3.1. 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 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:

$$A \rightarrow C : \{K_{CT}, N\}_{K_C} , \{ticket(C,T)\}_{K_T}$$

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**.

19.3.2. 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:

$$C \rightarrow T : \{auth(C)\}_{K_{CT}} , \{ticket(C,T)\}_{K_T} , S , N$$

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:

$$T \rightarrow C : \{K_{CS}, N\}_{K_{CT}} , \{ticket(C,S)\}_{K_S}$$

C sólo podrá obtener K_{CS} si conoce la clave secreta, K_{CT}.

19.3.3. 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:

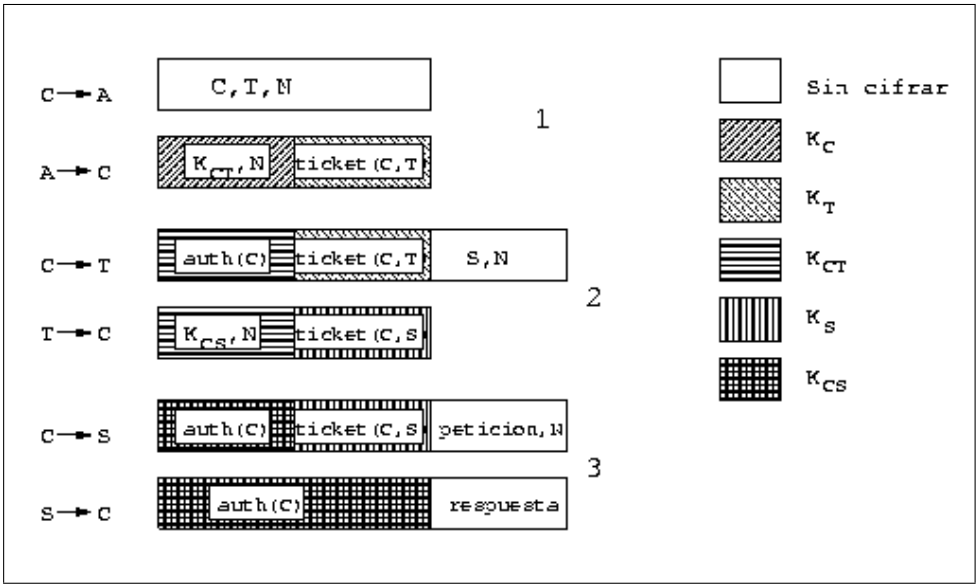

Figura 19.1: Protocolo de autenticación *Kerberos*.

$$C \rightarrow S : \{auth(C)\}_{K_{CS}} , \{ticket(C,T)\}_{K_S} , peticion , N$$

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.

19.4. 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 desee 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, intentado 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 V

Otros aspectos de la seguridad

Capítulo 20

Criptología

20.1. 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 *krypto* y *logos*, estudio de lo oculto, lo escondido) es la ciencia[1] 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 **cripto-**

[1] Hemos 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.

Veamos ahora el ejemplo contrario: somos los receptores de un mensaje del que sabemos que ha sido cifrado con la misma clave $(1, 4)$, y buscamos descifrar la cadena que nos ha sido enviada, $FVYXYW$. El valor de cada letra es

F	V	Y	X	Y	W
5	21	24	23	24	22

Tomando $f^{-1}(x) = x - 4$, tenemos el resultado

1	17	20	19	20	18
B	R	U	T	U	S

Como vemos, retornando cada número al alfabeto inglés obtenemos el texto en claro que nuestro emisor nos ha enviado: $BRUTUS$, equivalente al cifrado $FVYXYW$.

Si en el cifrado de un mensaje obtuviéramos 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 este motivo, y por otros muchos, este criptosistema es claramente vulnerable para un atacante, no ofreciendo una seguridad fiable en la transmisión de datos confidenciales.

20.5.2. El criptosistema de Vigènere

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 20.1.

La clave del sistema de cifrado de Vigenère es una palabra de k letras, $k \geq 1$, del alfabeto Z_{26} utilizado anteriormente; esta palabra es un elemento del producto cartesiano $Z_{26} \times Z_{26} \times ... \times Z_{26}$ (k veces), que es justamente el alfabeto del criptosistema de Vigenère. De esta forma, el mensaje a 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.

aplicada a cada elemento del mensaje.

Cuando el receptor del criptograma desee descifrar el mensaje recibido, ha de realizar la operación

$$m = c^d \, mod \, N$$

para obtener el texto en claro del mensaje que acaba de recibir.

El sistema RSA ha permanecido invulnerable hasta hoy, a pesar de los numerosos ataques de criptoanalistas; teóricamente es posible despejar d para obtener la clave privada, a partir de la función de descifrado, resultando

$$d = log_c m(mod(p-1))$$

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 $a^x \equiv b(n)$, 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.

20.7.2. 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

$$y \equiv a^x (mod \, p)$$

es, como hemos comentado para RSA, computacionalemente 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

$$y = a^x (mod \, p)$$

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

Una de las aplicaciones criptográficas más importante de las funciones resumen es sin duda la verificación de integridad de archivos; aunque ya hemos hablado de los verificadores de integridad tipo Tripwire en el capítulo dedicado a los sistemas de detección de intrusos, la idea es sencilla: en un sistema del que tengamos constancia que está 'limpio' (esto es, que no ha sido troyanizado o modificado de cualquier forma por un pirata) podemos generar resúmenes de todos los ficheros que consideremos clave para el correcto funcionamiento de la máquina y guardar dichos resúmenes – como ya indica su nombre, mucho más cortos que los archivos originales – en un dispositivo de sólo lectura como un CD-ROM. Periódicamente, o cuando sospechemos que la integridad de nuestro entorno ha sido violada, podemos volver a generar los resúmenes y comparar su resultado con el almacenado previamente: si no coinciden, podemos estar seguros (o casi seguros) de que el fichero ha sido modificado.

Para este tipo de aplicaciones se suele utilizar la función resumen MD5, diseñada por Ronald Rivest y que viene implementada 'de serie' en muchos clones de Unix, como Solaris o Linux (órdenes `'md5'` o `'md5sum'`):

```
luisa:~$ echo "Esto es una prueba" >/tmp/salida
sluisa:~$ md5sum /tmp/salida
3f8a62a7db3b276342d4c65dba2a5adf  /tmp/salida
luisa:~$ echo "Ahora modifico el fichero" >>/tmp/salida
luisa:~$ md5sum /tmp/salida
1f523e767e470d8f23d4378d74817825  /tmp/salida
luisa:~$
```

Otra aplicación importante de las funciones resumen es la firma digital de mensajes – documentos – y su *timestamping*; en el primer caso, como los algoritmos de firma digital suelen ser lentos, o al menos más lentos que las funciones *hash*, es habitual calcular la firma digital de un resumen del fichero original, en lugar de hacer el cálculo sobre el propio fichero (evidentemente, de tamaño mayor que su resumen). Con respecto al *timestamping*, las funciones *hash* son útiles porque permiten publicar un resumen de un documento sin publicar su contenido, lo cual permite a una parte obtener un *timestamp* de un documento sin que la autoridad de *timestamp* conozca el contenido del mismo, pero asegurándose la validez del procedimiento en caso de repudio; en ambos casos, tanto en la firma digital como en el *timestamping*, trabajar con el resumen es completamente equivalente a trabajar con el archivo original.

20.9. 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,

Capítulo 21

Algunas herramientas de seguridad

21.1. Introducción

¿Por qué utilizar herramientas de seguridad en los sistemas Unix? Ningún sistema operativo se puede considerar 'seguro' tal y como se instala por defecto[1]; 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.

[1] ¡Algunos no pueden considerarse 'seguros' nunca!

```
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/uuname
/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
```

Capítulo 22

Gestión de la seguridad

22.1. Introducción

La gestión de la seguridad de una organización puede ser – y en muchos casos es – algo infinitamente complejo, no tanto desde un punto de vista puramente técnico sino más bien desde un punto de vista organizativo; no tenemos más que pensar en una gran universidad o empresa con un número elevado de departamentos o áreas: si alguien que pertenece a uno de ellos abandona la organización, eliminar su acceso a un cierto sistema no implica ningún problema técnico (el administrador sólo ha de borrar o bloquear al usuario, algo inmediato), pero sí graves problemas organizativos: para empezar, ¿cómo se entera un administrador de sistemas que un cierto usuario, que no trabaja directamente junto a él, abandona la empresa? ¿quién decide si al usuario se le elimina directamente o se le permite el acceso a su correo durante un mes? ¿puede el personal del área de seguridad decidir bloquear el acceso a alguien de cierto 'rango' en la organización, como un directivo o un director de departamento, nada más que este abandone la misma? ¿y si resulta que es amigo del director general o el rector, y luego este se enfada? Como vemos, desde un punto de vista técnico no existe ningún escollo insalvable, pero sí que existen desde un punto de vista de la gestión de la seguridad...

Hoy en día, una entidad que trabaje con cualquier tipo de entorno informático, desde pequeñas empresas con negocios no relacionados directamente con las nuevas tecnologías hasta grandes *telcos* de ámbito internacional, está – o debería estar – preocupada por su seguridad. Y no es para menos: el número de amenazas a los entornos informáticos y de comunicaciones crece casi exponencialmente año tras año, alcanzando cotas inimaginables hace apenas una década. Y con que el futuro de la interconexión de sistemas sea tan solo la mitad de prometedor de lo que nos tratan de hacer creer, es previsible que la preocupación por la seguridad vaya en aumento conforme nuestras vidas estén más y más 'conectadas' a Internet.

Hasta hace poco esta preocupación de la que estamos hablando se centraba sobre todo en los aspectos más técnicos de la seguridad: alguien convencía a algún

- 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.

Para cubrir de forma adecuada los seis elementos anteriores, con el objetivo permanente de garantizar la seguridad corporativa, una política se suele dividir en puntos más concretos a veces llamados **normativas** (aunque las definiciones concretas de cada documento que conforma la infraestructura de nuestra política de seguridad – política, normativa, estándar, procedimiento operativo... – es algo en lo que ni los propios expertos se ponen de acuerdo). El estándar ISO 17799 ([Sta00]) define las siguientes líneas de actuación:

- Seguridad organizacional.
  Aspectos relativos a la gestión de la seguridad dentro de la organización (cooperación con elementos externos, *outsourcing*, estructura del área de seguridad...).
- Clasificación y control de activos.
  Inventario de activos y definición de sus mecanismos de control, así como etiquetado y clasificación de la información corporativa.
- Seguridad del personal.
  Formación en materias de seguridad, clausulas de confidencialidad, reporte de incidentes, monitorización de personal...
- Seguridad física y del entorno.
  Bajo este punto se engloban aspectos relativos a la seguridad física de los recintos donde se encuentran los diferentes recursos – incluyendo los humanos – de la organización y de los sistemas en sí, así como la definición de controles genéricos de seguridad.
- Gestión de comunicaciones y operaciones.
  Este es uno de los puntos más interesantes desde un punto de vista estrictamente técnico, ya que engloba aspectos de la seguridad relativos a la operación de los sistemas y telecomunicaciones, como los controles de red, la protección frente a *malware*, la gestión de copias de seguridad o el intercambio de *software* dentro de la organización.
- Controles de acceso.
  Definición y gestión de puntos de control de acceso a los recursos informáticos de la organización: contraseñas, seguridad perimetral, monitorización de accesos...

Parte VI

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 subred 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

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 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`.

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 prision 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. Iguales 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

[0]Delitos relacionados con nuevas tecnologías.

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.

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.

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.

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 integra 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 Públicas.

1. Cuando las infracciones a que se refiere el artículo 44 fuesen cometidas en ficheros de los que sean responsables las Administraciones Públicas, 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 Públicas.

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,

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:

  Â 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://ww.tis.com/Home/DataSecurityLetter.html
- Journal of Infrastructural Warfare: http://www.iwar.org/

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.

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.

[And80] James P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Co., Abril 1980.

[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.

[Ano01] Anonymous. *Maximum Linux Security: a hacker´s guide to protecting your Linux Server and Workstation.* Sams Publishing, 2001.

[ANS98] R. J. Anderson, R. M. Needham, and A. Shamir. The Steganographic File System. *Lecture Notes in Computer Science*, 1525:73–82, 1998.

[Ark99] Ofir Arkin. Network Scanning Techniques, Noviembre 1999. PubliCom Communications Solutions.

D.6. Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

D.7. Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

D.8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

D.9. Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights,