Administración de seguridad

Josep Jorba Esteve

PID_00174429

Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción

El salto tecnológico realizado desde los sistemas de escritorio aislados hasta los sistemas actuales integrados en redes locales e Internet, ha traído una nueva dificultad a las tareas habituales del administrador: el control de la seguridad de los sistemas.

La seguridad es un campo complejo, en el cual se mezclan técnicas de análisis con otras de detección o de prevención de los posibles ataques. Las técnicas a usar son tanto computacionales como relacionadas con otros campos, como el análisis de factores psicológicos, por lo que respecta al comportamiento de los usuarios del sistema o a las posibles intenciones de los atacantes.

Los ataques pueden provenir de muchas fuentes y afectar desde a una aplicación o servicio hasta a algún usuario, o a todos, o al sistema informático entero.

Los posibles ataques pueden cambiar el comportamiento de los sistemas, incluso "hacerlos caer" (es decir, inutilizarlos), o dar una falsa impresión de seguridad, que puede ser difícilmente detectable. Podemos encontrarnos con ataques de autentificación (obtener acceso por parte de programas o usuarios previamente no habilitados), escuchas (redirigir o pinchar los canales de comunicación y los datos que circulan) o sustitución (de programas, máquinas, comunicaciones o usuarios por otros, sin que se noten los cambios).

Seguridad absoluta

La seguridad absoluta no existe. Una falsa impresión de seguridad puede ser tan perjudicial como no tenerla. El área de la seguridad es muy dinámica y hay que mantener actualizados constantemente los conocimientos.

Una idea clara que hay que tener en mente es que es imposible poder conseguir una seguridad al 100 %.

Las técnicas de seguridad son un arma de doble filo, que fácilmente pueden darnos una falsa impresión de control del problema. La seguridad actual es un problema amplio, complejo y, lo que es más importante, dinámico. Nunca podemos esperar o asegurar que la seguridad esté garantizada en un determinado sistema, sino que con bastante probabilidad será una de las áreas a la cual el administrador tendrá que dedicar más tiempo y mantener actualizados sus conocimientos sobre el tema.

En este módulo examinaremos diferentes problemáticas con los que podemos encontrarnos, cómo podemos verificar y prevenir partes de la seguridad local y en entornos de red. Asimismo, examinaremos técnicas de detección de in-

trusiones y algunas herramientas básicas que nos pueden ayudar en el control de la seguridad.

También es preciso mencionar que en este módulo solo podemos hacer una mera introducción a algunos de los aspectos que intervienen en la seguridad de hoy en día. Para cualquier aprendizaje real con más detalle, se recomienda consultar la bibliografía disponible, así como los manuales asociados a los productos o herramientas comentados.

Objetivos

En los materiales didácticos de este módulo encontraréis los contenidos y las herramientas procedimentales para conseguir los objetivos siguientes:

1. Conocer los principales tipos y métodos de ataques a la seguridad de los sistemas, así como algunas contramedidas básicas y herramientas útiles para su tratamiento.
2. Saber hacer un seguimiento de la seguridad local y en red en los sistemas GNU/Linux.
3. Conocer la metodología de uso de herramientas de detección de intrusiones.
4. Saber elaborar medidas de prevención mediante filtrado de servicios o paquetes, *wrappers* y cortafuegos (*firewalls*).
5. Conocer las herramientas de seguridad disponibles, para control de accesos y permisos, a nivel de MAC (*Mandatory Access Control*).

1. Tipos y métodos de los ataques

La seguridad computacional, en administración, puede ser entendida como el proceso que ha de permitir al administrador del sistema prevenir y detectar usos no autorizados del mismo. Las medidas de prevención ayudan a parar los intentos de usuarios no autorizados (los conocidos como **intrusos**) para acceder a cualquier parte del sistema. La detección ayuda a descubrir cuándo se produjeron estos intentos o, en el caso de llevarse a cabo, establecer las barreras para que no se repitan y poder recuperar el sistema si ha sido quebrantado o comprometido.

Los intrusos (también conocidos coloquialmente como *hackers*, *crackers*, "atacantes" o "piratas") normalmente desean obtener control sobre el sistema, ya sea para causar funcionamientos erróneos, corromper el sistema o sus datos, ganar recursos en la máquina o, simplemente, para lanzar ataques contra otros sistemas y así proteger su verdadera identidad y ocultar el origen real de los ataques. También existe la posibilidad de que el ataque se produzca para examinar o robar la información del sistema, el puro espionaje de las acciones del sistema o para causar daños físicos en la máquina, ya sea formatear el disco, cambiar datos, borrar o modificar software crítico, etc.

Con respecto a los intrusos, hay que establecer algunas diferencias, que no suelen estar muy claras en los términos coloquiales. Normalmente nos referimos a **hacker** [Him01] como aquella persona con grandes conocimientos en informática, más o menos apasionada por los temas de programación y seguridad informática y que, normalmente sin finalidad malévola, utiliza sus conocimientos para protegerse a sí mismo o a terceros, introducirse en redes para demostrar sus fallos de seguridad y, en algunos casos, para reconocimiento público de sus habilidades. Un ejemplo sería la propia comunidad GNU/Linux, que debe mucho a sus *hackers*, ya que hay que entender el término *hacker* como experto en unos temas computacionales, más que como intruso que afecta a la seguridad.

Por otro lado encontraríamos a los **crackers**. Aquí es cuando se utiliza el término, de manera más o menos despectiva, hacia aquellos que utilizan sus habilidades para corromper (o destrozar) sistemas, ya sea sólo por fama propia, por motivos económicos, por ganas de causar daño o simplemente por molestar, por motivos de espionaje tecnológico, actos de ciberterrorismo, etc.

Asimismo, se habla de *hacking* o *cracking*, respectivamente, cuando nos referimos a técnicas de estudio, detección y protección de la seguridad, o por el

contrario, a técnicas destinadas a causar daño rompiendo la seguridad de los sistemas.

Desafortunadamente, obtener acceso a un sistema (ya sea desprotegido o parcialmente seguro) es bastante más fácil de lo que parece en primera instancia. Los intrusos descubren permanentemente nuevas **vulnerabilidades** (llamadas a veces "agujeros de seguridad" o *exploits*), que les permite introducirse en las diferentes capas de software (ya sean aplicaciones, servicios o partes del operativo). Así, definiremos una vulnerabilidad como el fallo de un programa (o capa de software) que permite, mediante su explotación, violar la seguridad de un sistema informático.

La complejidad cada vez mayor del software (y del hardware), hace que aumente la dificultad para testear de manera razonable la seguridad de los sistemas informáticos. El uso habitual de los sistemas GNU/Linux en red, ya sea en la propia Internet o en redes propias con tecnología TCP/IP como las *intranets*, nos lleva a exponer nuestros sistemas, como víctimas, a ataques de seguridad [Bur02, Fen02, Line].

Lo primero que hay que hacer es romper el mito de la seguridad informática: simplemente no existe. Lo que sí que podemos conseguir es un cierto grado de seguridad que nos haga sentirnos seguros dentro de ciertos parámetros. Pero como tal, esto solo es una percepción de seguridad, y como todas las percepciones, puede ser falsa y de ello podemos darnos cuenta en el último momento, cuando ya tengamos nuestros sistemas afectados y comprometidos. La conclusión lógica es que la seguridad informática exige un esfuerzo importante de constancia, realismo y aprendizaje prácticamente diario.

Tenemos que ser capaces de establecer en nuestros sistemas unas políticas de seguridad que nos permitan prevenir, identificar y reaccionar ante los posibles ataques. Y tener presente que la sensación que podamos tener de seguridad no es más que eso, una sensación. Por lo tanto, no hay que descuidar ninguna de las políticas implementadas y hay que mantenerlas al día, así como nuestros conocimientos del tema.

Amenazas

Las amenazas afectan a la confidencialidad, integridad o accesibilidad de nuestros sistemas.

Los posibles ataques son una amenaza constante a nuestros sistemas y pueden comprometer su funcionamiento, así como los datos que gestionamos. Ante todo esto, siempre tenemos que definir una cierta política de requisitos de seguridad sobre nuestros sistemas y datos. Las amenazas que podemos sufrir podrían afectar a los aspectos siguientes:

- **Confidencialidad**: la información debe ser accesible sólo a aquellos que estén autorizados; estamos respondiendo a la pregunta: ¿quién podrá acceder a la información?

- **Integridad**: la información solo podrá ser modificada por aquellos que estén autorizados: ¿qué se podrá hacer con ella?

- **Accesibilidad**: la información tiene que estar disponible para quienes la necesiten y cuando la necesiten, si están autorizados: ¿de qué manera, y cuándo se podrá acceder a ella?

Pasemos a mencionar una cierta clasificación (no exhaustiva) de los tipos de ataques habituales que podemos padecer:

- Autentificación: ataques en los que se falsifica la identidad del participante, de manera que obtiene acceso a programas o servicios a los que en un principio no tenía acceso.

- Intercepción (o escucha): mecanismo por el cual se interceptan datos, por parte de terceros, cuando estos no estaban dirigidos a ellos.

- Falsificación (o reemplazo): sustitución de algunos de los participantes, ya sea máquinas, software o datos, por otros falsos.

- Robo de recursos: uso de nuestros recursos sin autorización.

- Vandalismo: después de todo, suele ser bastante común la existencia de mecanismos que permiten interferir en el funcionamiento adecuado del sistema o de los servicios y causar molestias parciales o el paro o cancelación de recursos.

Los métodos utilizados y las técnicas precisas pueden variar mucho (es más, cada día se crean novedades) y nos obligan, como administradores, a estar en contacto permanente con el campo de la seguridad para conocer a qué nos podemos enfrentar diariamente.

Respecto a dónde se produce el ataque, hemos de tener claro qué puede hacerse o cuál será el objetivo de los métodos:

Finalidad de los ataques

Los ataques pueden tener finalidades destructivas, inhabilitadoras o de espionaje, de nuestros componentes, ya sea hardware, software o sistemas de comunicación.

- **Hardware**. A este respecto, la amenaza está directamente sobre la accesibilidad, ¿qué podrá hacer alguien que tenga acceso al hardware? En este caso normalmente necesitaremos medidas "físicas", como controles de seguridad para acceder a los locales donde estén situadas las máquinas, para evitar problemas de robo o rotura del equipo con el fin de eliminar su servicio. También puede comprometerse la confidencialidad y la integridad si el acceso físico a las máquinas permite utilizar algunos de sus dispositivos como las unidades de disco (extraibles o no), el arranque de las máquinas o el acceso a cuentas de usuario que podrían estar abiertas.

- **Software**. Si la accesibilidad se ve comprometida en un ataque, puede haber borrado o inutilización de programas, denegando el acceso. En caso de confidencialidad, puede provocar copias no autorizadas de software. En integridad podría alterarse el funcionamiento por defecto del programa, para que falle en algunas situaciones o bien para que realice tareas que puedan ser interesantes de cara al atacante, o podría simplemente comprometer la integridad de los datos de los programas: hacerlos públicos, alterarlos o simplemente robarlos.

- **Datos**, ya sean estructurados, como en los servicios de base de datos, gestión de versiones (como cvs) o simples archivos. Mediante ataques que amenacen la accesibilidad, estos datos pueden ser destruidos o eliminados, denegando así el acceso a los mismos. En el caso de la confidencialidad, estaríamos permitiendo lecturas no autorizadas y la integridad se vería afectada cuando se produjeran modificaciones o creación de nuevos datos.

- **Canal de comunicación** (en la red, por ejemplo). Para los métodos que afectan a la accesibilidad, nos puede provocar destrucción o eliminación de mensajes e impedir el acceso a la red. En confidencialidad, se puede dar la lectura y observación del tráfico de mensajes, desde o hacia la máquina. Y respecto a la integridad, podemos vernos afectados por cualquier modificación, retardo, reordenación, duplicación o falsificación de los mensajes entrantes o salientes.

1.1. Técnicas utilizadas en los ataques

Los métodos utilizados son múltiples y pueden depender de un elemento (hardware o software) o de la versión del mismo. Por lo tanto, hay que mantener actualizado el software para las correcciones de seguridad que vayan apareciendo y seguir las indicaciones del fabricante o distribuidor para proteger el elemento, además de las medidas de seguridad activa que podamos realizar.

Evolución de las técnicas de ataque

Las técnicas de los ataques son muy variadas y evolucionan constantemente en lo que a los detalles tecnológicos usados se refiere.

A pesar de ello, normalmente siempre hay técnicas o métodos "de moda", del momento actual. Algunas breves indicaciones de estas técnicas de ataque (actuales) son:

- **Explotación de agujeros (*bug exploits*)**: se trata de la explotación de agujeros o errores [Cerb, Ins, San] de un hardware, software, servicio, protocolo o del propio sistema operativo (por ejemplo, en el núcleo o *kernel*) y, normalmente de alguna de las versiones de estos en concreto. En general, cualquier elemento informático es más o menos propenso a errores en su concepción o simplemente a cosas que no se han tenido en cuenta

o previsto. Periódicamente, se descubren agujeros (a veces se denominan *holes*, *exploits* o simplemente *bugs*), que pueden ser aprovechados por un atacante para romper la seguridad de los sistemas. Suelen utilizarse técnicas de ataque genéricas, como las que se explican a continuación, o bien técnicas particulares para el elemento afectado. Cada elemento tendrá un responsable, ya sea fabricante, desarrollador, distribuidor o la comunidad GNU/Linux, de producir nuevas versiones o parches para tratar estos problemas. Nosotros, como administradores, tenemos la responsabilidad de estar informados y de mantener una política de actualización responsable para evitar los riesgos potenciales de estos ataques. En caso de que no haya soluciones disponibles, también podemos estudiar la posibilidad de utilizar alternativas al elemento, o bien inhabilitarlo hasta que tengamos soluciones.

- **Virus**: se trata de programas normalmente anexos a otros y que utilizan mecanismos de autocopia y transmisión. Son habituales los virus adjuntos a programas ejecutables, a mensajes de correo electrónico, o incorporados en documentos o programas que permiten algún lenguaje de macros (no verificado). Son quizás la mayor plaga de seguridad de hoy en día en algunos sistemas.

  Los sistemas GNU/Linux están protegidos casi totalmente contra estos mecanismos por varias razones: en los programas ejecutables, tienen un acceso muy limitado al sistema, en particular a la cuenta del usuario. Con excepción del usuario root, quien debe tener mucho cuidado con lo que ejecuta. El correo no suele utilizar lenguajes de macros no verificados (como en el caso de Outlook y Visual Basic Script en Windows, que es un agujero de entrada de virus), mientras en el caso de los documentos, estamos en condiciones parecidas, ya que no soportan lenguajes de macros no verificados (como el VBA en Microsoft Office). Aunque hay que comentar que varios aspectos de este mismo tipo comienzan a tener cierta importancia, ya que algunas de las *suites* de ofimática para GNU/Linux utilizan ya lenguajes de macros, y los clientes de correo cada vez incorporan más soporte de html empotrado con soporte de JavaScript, una de las fuentes con más problemas, como veremos en algunos apartados siguientes. Que estos problemas no hayan sido explotados (o solo mínimamente) es solo una cuestión de cuán usada es una plataforma, como GNU/Linux. En la medida en que crece su uso, también se hace más atractiva para los atacantes.

  En todo caso, habrá que prestar atención a lo que pueda pasar en un futuro, ya que podrían surgir algunos virus específicos para GNU/Linux aprovechando algunos *bugs* o *exploits*. Un punto que sí que hay que tener en cuenta es el de los sistemas de correo, ya que si bien nosotros no generaremos virus, sí que podemos llegar a transmitirlos; por ejemplo, si nuestro sistema funciona como *router* o *relay* de correo, podrían llegar mensajes con virus que podrían ser enviados a otros. Aquí se puede implementar algu-

na política de detección y filtrado de virus, si en nuestros sistemas existen entornos Windows, en los que se pueden propagar virus externos que hayamos recibido. Otra forma muy importante de problemáticas que podría entrar dentro de la categoría de virus son los mensajes basura (*spam*), que si bien no suelen ser utilizados como elementos atacantes (aunque pueden combinarse con otras técnicas, como intentos de *phising*), sí que podemos considerarlos como un problema por su "virulencia" de aparición y por el coste económico que pueden causar (pérdidas de tiempo y recursos).

- **Gusanos (*worms*)**: normalmente se trata de un tipo de programas que aprovechan algún agujero del sistema para realizar ejecuciones de código sin permiso. Suelen ser utilizados para aprovechar recursos de la máquina, como el uso de CPU, cuando se detecta que el sistema no funciona o no está en uso o, si son malintencionados, con el objetivo de robar recursos o utilizarlos para parar o bloquear el sistema. También suelen utilizar técnicas de transmisión y replicacion.

Gusano Morris

Uno de los primeros gusanos, el conocido como Gusano Morris, fue muy importante porque hizo de la seguridad un tema importante, en unos momentos en que había cierta inocencia en estos temas. Véase http://en.wikipedia.org/wiki/Morris_worm.

- **Troyanos (*trojans* o *Trojan horses*)**: programas útiles que incorporan alguna funcionalidad pero ocultan otras que son las utilizadas para obtener información del sistema o comprometerlo. Un caso particular puede ser el de los códigos de tipo móvil en aplicaciones web, como los Java, JavaScript o ActiveX; estos normalmente piden su consentimiento para ejecutarse (ActiveX en Windows) o tienen modelos limitados de lo que pueden hacer (Java, JavaScript). Pero como todo software, también pueden tener posibles agujeros, y son un método ideal para transmitir troyanos.

- **Puertas traseras (*back doors*)**: es un método de acceso a un programa escondido que puede utilizarse para otorgar acceso al sistema o a los datos manejados sin que lo sepamos. Otros efectos pueden ser cambiar la configuración del sistema o permitir la introducción de virus. El mecanismo usado puede ser desde venir incluidos en algún software común hasta en un troyano que produzca puertas traseras.

- **Bombas lógicas**: programa incrustado en otro, que comprueba que se den algunas condiciones (temporales, acciones del usuario, etc.), para activarse y emprender acciones no autorizadas.

- **Registradores de tecleo (*keyloggers*)**: se trata de un programa especial que se dedica a secuestrar las interacciones con el teclado y/o ratón del usuario. Pueden ser programas individuales o bien troyanos incorporados en otros programas. Normalmente, necesitarían introducirse en un sistema abierto al que se dispusiese de acceso (aunque cada vez más pueden venir incorporados en troyanos que se instalen). La idea es captar cualquier introducción de teclas, de manera que se capturen contraseñas (por ejemplo, las bancarias), la interacción con aplicaciones, los sitios visitados por la red, los formularios rellenados, etc.

- **Escaneo de puertos (*port scanning*)**: más que un ataque, sería un paso previo, que consistiría en la recolección de posibles objetivos. Básicamente, consiste en utilizar herramientas que permitan examinar la red en busca de máquinas con puertos abiertos, sean TCP, UDP u otros protocolos, que indican la presencia de algunos servicios. Por ejemplo, escanear máquinas buscando el puerto 80 TCP, indica la presencia de servidores web, de los cuales podemos obtener información sobre el servidor y la versión que utilizan para aprovecharnos de vulnerabilidades conocidas.

- **Husmeadores (*sniffers*)**: permiten la captura de paquetes que circulan por una red. Con las herramientas adecuadas podemos analizar comportamientos de máquinas: cuáles son servidores, cuáles son clientes, qué protocolos se utilizan y en muchos casos obtener contraseñas de servicios no seguros. En un principio, fueron muy utilizados para capturar contraseñas de telnet, rsh, rcp, ftp, etc., servicios no seguros que no tendrían que utilizarse (en su lugar deberían usarse las versiones seguras: ssh, scp, sftp). Tanto los husmeadores como los escaneadores de puertos no son necesariamente una herramienta de ataque, ya que también pueden servir para analizar nuestras redes y detectar fallos, o simplemente para analizar nuestro tráfico. Normalmente, tanto las técnicas de escaneo como las de los husmeadores suelen utilizarse por parte de un intruso con el objetivo de encontrar las vulnerabilidades del sistema, ya sea para conocer datos de un sistema desconocido (escaneadores) o para analizar la interacción interna (husmeadores).

- **Secuestro (*hijacking*)**: son técnicas que intentan colocar una máquina de manera que intercepte o reproduzca el funcionamiento de algún servicio en otra máquina a la que ha pinchado la comunicación. Suelen ser habituales los casos para correo electrónico, transferencia de ficheros o web. Por ejemplo, en el caso web, se puede capturar una sesión y reproducir lo que el usuario está haciendo, páginas visitadas, interacción con formularios, etc. En algunos casos puede ser a causa de secuestros a nivel de red, debido a que se capturan o escuchan paquetes de las comunicaciones, o puede producirse un secuestro *offline* mediante ejecución arbitraria de código de *scripts*. Por ejemplo, es común en sistemas de correo electrónico por webmail, o en aplicaciones web que incluyan el uso de html en algunas de sus ventanas, que el sistema permita inyectar código JavaScript o PHP (u otros lenguajes de *script* similares), que si no se controla o limita de forma correcta, puede provocar el robo de datos del usuario, ya sea sus identificadores de sesión o sus galletas (*cookies*) utilizadas, lo que permite al atacante reproducir o secuestrar *a posteriori* sus sesiones de correo o de aplicación web, sin necesidad de contraseña alguna, y así secuestrar la sesión y la identidad del usuario.

- **Desbordamientos (*buffer overflows*)**: técnica bastante compleja que aprovecha errores de programación en las aplicaciones. La idea básica es aprovechar desbordamientos (*overflows*) en *buffers* de la aplicación, ya sean colas,

arrays, vectores, etc. Si no se controlan los límites, un programa atacante puede generar un mensaje o dato más grande de lo esperado y provocar fallos mediante la escritura por encima de los límites permitidos por la estructura de datos, y así sobreescribir memoria adyacente. Por ejemplo, en muchas aplicaciones C con *buffers* mal escritos, en *arrays*, si sobrepasamos el límite podemos provocar una sobreescritura del código del programa, lo que causa un funcionamiento incorrecto o la caída del servicio o máquina. Es más, una variante más compleja permite incorporar en el ataque trozos de programa (compilados en C o bien *shell scripts*) que pueden permitir la ejecución de cualquier código que el atacante quiera introducir.

Algunas variaciones de esta técnica pueden aprovecharse para atacar de forma similar otras partes del código ejecutable de un programa, como por ejemplo la pila del programa (hablamos entonces de *stack overflows*) o la gestión de las peticiones dinámicas de memoria que realice el ejecutable (*heap overflows*). La alteración tanto de la pila como de la memoria dinámica tambien puede permitir al atacante la ejecución de código arbitrario añadido o la caída del ejecutable o servicio.

- **Denegación de servicio, *Denial of Service*, DoS**: este tipo de ataque provoca que la máquina caiga o que se sobrecarguen uno o más servicios, de manera que no sean utilizables. Otra técnica es la DDoS (*Distributed DoS*, DoS distribuida), que se basa en utilizar un conjunto de máquinas distribuidas para que produzcan el ataque o sobrecarga de servicio. Este tipo de ataques se suelen solucionar con actualizaciones del software, y con una correcta configuración de los parámetros de control de carga del servicio, ya que normalmente se ven afectados aquellos servicios que no fueron pensados para una carga de trabajo determinada y no se controla la saturación. Los ataques DoS y DDoS son bastante utilizados en ataques a sitios web o servidores DNS, que se ven afectados por vulnerabilidades de los servidores, por ejemplo, de versiones concretas de Apache o BIND. Otro aspecto que cabe tener en cuenta es que nuestro sistema también podría ser usado para ataques de tipo DDoS, mediante control, ya sea de a través de una puerta trasera o de un troyano que forma parte de una *bootnet* por ejemplo.

  Un ejemplo de este ataque (DoS) bastante sencillo es el conocido como SYN flood, que trata de generar paquetes TCP que abren una conexión, pero ya no hacen nada más con ella, simplemente la dejan abierta. Esto consume recursos del sistema en estructuras de datos del núcleo y recursos de conexión por red. Si se repite este ataque centenares o miles de veces, se consigue ocupar todos los recursos sin utilizarlos, de modo que cuando algunos usuarios quieran utilizar el servicio, les sea denegado porque los recursos están ocupados. Otro caso conocido es el bombardeo de correo (*mail bombing*) o, simplemente, el reenvío de correo (normalmente con emisor falso) hasta que se saturan las cuentas de correo y el sistema de correo cae o se vuelve tan lento que es inutilizable. Estos ataques son en cierta medida sencillos de realizar con las herramientas adecuadas, y no tienen

Enlaces de interés

Sobre SYN flood, véase: http://www.cert.org/advisories/CA-1996-21.html. Sobre E-mail bombing y spam, véase: http://www.cert.org/tech_tips/email_bombing_spamming.html.

una solución fácil, ya que se aprovechan del funcionamiento interno de los protocolos y servicios; en estos casos tenemos que tomar medidas de detección y control posterior.

- **Falseamiento de identidad (*spoofing*)**: las técnicas de *spoofing* engloban varios métodos (normalmente muy complejos) para falsificar tanto la información como los participantes en una transmisión (origen y/o destino). Algunos métodos de *spoofing* son los siguientes:

  – *IP spoofing*, falsificación de una máquina, permitiendo que genere tráfico falso o escuche tráfico que iba dirigido a otra máquina. Combinado con otros ataques, puede saltarse incluso protección de cortafuegos.

  – *ARP spoofing*, técnica compleja (utiliza un DDoS), que intenta falsificar las direcciones de fuentes y destinatarios de una red, mediante ataques de las cachés de ARP que poseen las máquinas, de manera que se sustituyan las direcciones reales por otras en varios puntos de una red. Esta técnica permite saltarse todo tipo de protecciones, incluidos cortafuegos, pero no es una técnica sencilla.

  – Correo electrónico, es quizás el más sencillo. Se trata de generar contenidos de correos falsos, tanto por el contenido como por la dirección de origen. En este tipo son bastante utilizadas las técnicas de lo que se denomina *ingeniería social*, que básicamente intenta engañar de una forma razonable al usuario. Un ejemplo clásico son los correos falsos del administrador del sistema, o bien del banco donde tenemos nuestra cuenta corriente, en los que se mencionan problemas con la gestión de nuestras cuentas y nos piden enviar información confidencial o la contraseña anterior para solucionarlos, o se nos pide que la contraseña se cambie por otra concreta. Sorprendentemente, esta técnica (también conocida como pesca o *phishing*) consigue engañar a un número considerable de usuarios. Incluso con ingeniera social de métodos sencillos: algún *cracker* famoso comentaba que su método preferido era el teléfono. Como ejemplo, exponemos el caso de una empresa de certificación (Verisign), de la que los *crackers* obtuvieron la firma privada de software Microsoft con solo realizar una llamada mencionando que hablaban de parte de la empresa, que se les había presentado un problema y que volvían a necesitar su clave. Resumiendo, unos niveles altos de seguridad informática se pueden ir al traste por una simple llamada telefónica o un correo que un usuario malinterprete.

Enlace de interés

Véase el caso de Microsoft en: http://www.computerworld.com/softwaretopics/os/windows/story/0,10801,59099,00.html.

- ***SQL injection***: es una técnica orientada a bases de datos, y a servidores web en particular, que se aprovecha generalmente de programación incorrecta de formularios web, en los cuales no se ha controlado correctamente la información que se proporciona. No se determina que la información de entrada sea del tipo correcto (fuertemente tipada respecto a lo que se espera) o no se controla qué tipo o caracteres literales se introducen. La técnica se aprovecha del hecho que los caracteres literales obtenidos por los formularios (por ejemplo web, aunque los ataques pueden sufrirse des-

de cualquier API que permita el acceso a una base de datos, por ejemplo php o perl) son utilizados directamente para construir las consultas (en SQL) que atacarán a una determinada base de datos (a la que en principio no se tiene acceso directo). Normalmente, si existen las vulnerabilidades y hay poco control de los formularios, se puede inyectar código SQL en el formulario, de manera que se construyan consultas SQL que proporcionen la información buscada. En casos drásticos podría obtenerse la información de seguridad (usuarios y contraseñas de la base de datos) o incluso tablas o la base de datos entera, y también podrían producirse pérdidas de información o borrados intencionados de datos. En particular, esta técnica en ambientes web puede ser grave (para la empresa proveedora de servicio, con fuertes consecuencias económicas), debido a las leyes que protegen la privacidad de datos personales, que se pueden ver comprometidos, hacerse públicos o ser vendidos a terceros si se obtienen por un ataque de este tipo. En este caso de ataque, más que una cuestión de seguridad del sistema, se trata de un problema de programación y control con tipado fuerte de los datos esperados en la aplicación, además del adecuado control de conocimiento de vulnerabilidades presentes del software usado (base de datos, servidor web, API como php, perl, etc.).

- ***Cross-site scripting*, XSS**: es otra problemática relacionada con ambientes web, en particular con alteraciones del código HTML y *scripts* cuando un usuario visualiza un determinado sitio web, que puede ser alterado dinámicamente. Se aprovecha generalmente de los errores a la hora de validar código HTML (todos los navegadores tienen más o menos problemas, por la propia definición del HTML original, que permite leer prácticamente cualquier código HTML por incorrecto que sea). En algunos casos, la utilización de vulnerabilidades puede ser directa mediante *scripts* en la página web, pero normalmente los navegadores tienen buen control de los mismos. Por otra parte, indirectamente hay técnicas que permiten insertar código de *script*, bien mediante acceso a las galletas (*cookies*) del usuario desde el navegador, bien mediante la alteración del proceso por el que se redirecciona una página web a otra. También hay técnicas mediante marcos (*frames*), que permiten redirigir el HTML que se está viendo o colgar directamente el navegador. En particular, pueden ser vulnerables los motores de búsqueda de los sitios web, que pueden permitir la ejecución de código de *scripts*. En general, son ataques con diversas técnicas complejas, pero con el objetivo de capturar información como *cookies*, que pueden ser usadas en sesiones y permitir así sustituir a una determinada persona mediante redirecciones de sitios web u obtener información suya. Otra vez desde la perspectiva de sistema, es más una cuestión de software en uso. Hay que controlar y conocer vulnerabilidades detectadas en navegadores (y aprovechar los recursos que estos ofrecen para evitar estas técnicas) y controlar el uso de software (motores de búsqueda empleados, versiones del servidor web y API utilizadas en los desarrollos).

En general, algunas recomendaciones (muy básicas) para la seguridad podrían ser:

- Controlar un factor problemático: los usuarios. Uno de los factores que puede afectar más a la seguridad es la confidencialidad de las contraseñas, y esta se ve afectada por el comportamiento de los usuarios; esto facilita a posibles atacantes las acciones desde el interior del propio sistema. La mayoría de los ataques suelen venir de dentro del sistema, es decir, una vez el atacante ya ha ganado acceso al sistema.

- Entre los usuarios, está aquel que es un poco olvidadizo (o indiscreto), que olvida la contraseña cada dos por tres, la menciona en conversaciones, la escribe en un papel que olvida o que está junto (o pegada) al ordenador o sobre la mesa de trabajo, o que simplemente la presta a otros usuarios o conocidos. Otro tipo es el que coloca contraseñas muy predecibles, ya sea su mismo identificador de usuario, su nombre, su DNI, el nombre de su novia, el de su madre, el de su perro, etc., cosas que con un mínimo de información pueden encontrarse fácilmente (más en estos momentos de auge de las redes sociales, donde es fácil obtener este tipo de información desde diferentes redes en las que participe el usuario). Otro caso son los usuarios normales con un cierto conocimiento, que colocan contraseñas válidas, pero siempre hay que tener en cuenta que hay mecanismos que pueden encontrarlas (*cracking* de passwords, husmeadores, suplantación, etc.). Hay que establecer una cierta **cultura de seguridad** entre los usuarios y, mediante diferentes técnicas, obligarles a que cambien las contraseñas, que no utilicen palabras típicas, que usen contraseñas largas (tener más de 6 o 8 cáracteres mínimo), etc. Últimamente, en muchas empresas e instituciones se está implantando la técnica de hacer firmar un contrato (o carta de compromisos) al usuario de manera que se le obliga a no divulgar la contraseña o cometer actos de vandalismo o ataques desde su cuenta (claro que esto no impide que otros lo hagan por él).

- No utilizar ni ejecutar programas de los que no podamos garantizar su origen. Normalmente, muchos distribuidores utilizan mecanismos de comprobación de firmas para verificar que los paquetes de software son tales, como por ejemplo las sumas md5 (comando `md5sum`) o la utilización de firmas GPG [Hatd] (comando `gpg`). El vendedor o distribuidor provee una suma md5 de su archivo (o imagen de CD/DVD), de modo que podemos comprobar su autenticidad. Últimamente, en las distribuciones se están usando tanto firmas para paquetes individuales como las firmas para los repositorios de paquetes, como mecanismo para asegurar la fiabilidad del proveedor.

- No utilizar usuarios privilegiados (como root) para el trabajo normal de la máquina, ya que cualquier programa (o aplicación) tendría los permisos para acceder a cualquier zona del sistema de archivos o a servicios en ejecución.

- No acceder remotamente con usuarios privilegiados ni ejecutar programas que puedan tener privilegios. Y más si no conocemos, o no hemos comprobado, los niveles de seguridad del sistema y sus conexiones. En particular, otro de los problemas actuales son las redes inalámbricas (como WiFi) inseguras, ya que, por ejemplo, algunos cifrados como el usado en redes WiFi WEP son muy débiles. No se deben usar conexiones críticas sobre redes inseguras.

- No utilizar elementos (programas o servicios) que no sabemos cómo actúan ni intentar descubrirlo a base de repetidas ejecuciones.

Estas medidas pueden ser poco productivas, pero si no hemos asegurado el sistema, no podemos tener ningún control sobre lo que puede pasar, y aun así, nadie asegura que no se pueda colar algún programa malicioso que burle la seguridad si lo ejecutamos con los permisos adecuados. En general, hay que considerar que debemos tener una vigilancia activa con todo este tipo de actividades que supongan accesos y ejecución de tareas de formas más o menos privilegiadas y un uso de mecanismos de comunicación inseguros.

1.2. Contramedidas

Respecto a las medidas que se pueden tomar sobre los tipos de ataques presentados, podemos encontrar algunas preventivas y de detección de lo que sucede en nuestros sistemas.

Veamos algunos tipos de medidas que podríamos tomar en los ámbitos de prevención y detección de intrusos (se mencionan herramientas útiles, algunas las examinaremos más adelante):

- ***Password cracking***: en ataques de fuerza bruta para romper las contraseñas suele ser habitual intentar obtener acceso por *login* de forma repetida; si se consigue entrar, la seguridad del usuario ha sido comprometida y se deja la puerta abierta a otros tipos de ataques como, por ejemplo, las puertas traseras o, simplemente, a la destrucción de la cuenta. Para prevenir este tipo de ataques, hay que reforzar la política de contraseñas, pidiendo una longitud mínima y cambios de contraseña periódicos. Una cosa que hay que evitar es el uso de palabras comunes en las contraseñas: muchos de estos ataques se hacen mediante la fuerza bruta, con un fichero de diccionario (con palabras en el idioma del usuario, términos comunes, nombres propios, argot, etc.). Este tipo de contraseñas serán las primeras en caer. También puede ser fácil obtener información del atacado, como nombres, DNI o su dirección e información procedente de sus redes sociales, y usar estos datos para probar las contraseñas. Por todo ello tampoco se recomiendan contraseñas con DNI, nombres (propios o de familiares, etc.), direcciones, nombres de mascotas, etc. Una buena elección suele ser elegir contraseñas de entre 6 y

8 caracteres como mínimo y que contengan caracteres alfabéticos, numéricos y algún carácter especial. Aunque la contraseña esté bien elegida, puede ser insegura si se utiliza en servicios no seguros. Por lo tanto, se recomienda reforzar los servicios mediante técnicas y servicios de cifrado que protejan las contraseñas y las comunicaciones. Por el contrario, debe evitarse (o no usar) todos aquellos servicios que no soporten cifrado y que, consecuentemente, sean susceptibles de ser atacados con métodos, por ejemplo, de husmeadores; entre estos podríamos incluir servicios como telnet, ftp, rsh y rlogin, entre otros.

- **Explotación de agujeros**: se debe evitar disponer de programas que no se utilicen, sean antiguos o no se actualicen (por ser obsoletos). Hay que aplicar los últimos parches y actualizaciones disponibles, tanto para las aplicaciones como para el sistema operativo, probar herramientas que detecten vulnerabilidades y mantenerse al día de las vulnerabilidades que se vayan descubriendo.

Enlaces de interés

Sobre vulnerabilidades, una buena herramienta es Nessus. Para descubrir nuevas vulnerabilidades, véase CERT en: http://www.cert.org/advisories/, sitio antiguo y http://www.us-cert.gov/cas/techalerts/index.html.

- **Virus**: se deben utilizar mecanismos o programas antivirus, sistemas de filtrado de mensajes sospechosos, evitar la ejecución de sistemas de macros (que no se puedan verificar). No hay que minimizar los posibles efectos de los virus, cada día se perfeccionan más y técnicamente es posible realizar virus simples que pueden desactivar redes en cuestión de minutos (sólo hay que observar algunos de los virus de los últimos años en ambientes Windows). En especial, si en nuestros sistemas existen entornos Windows, sería interesante la posibilidad de examinar virus que puedan afectar a estos sistemas. Existen antivirus para UNIX y Linux, como ClamAV, que evitarán que se transmitan virus a nuestros sistemas internos. En sistemas de correo basados en GNU/Linux, podemos establecer combinaciones de productos antivirus y antispam para evitar estas transmisiones, como por ejemplo utilizar ClamAV y otros productos como SpamAssasin.

Enlaces de interés

Véanse parches para el sistema operativo en: http://www.debian.org/security, http://www.redhat.com/security, http://fedoraproject.org/wiki/Security.

- **Gusanos**: hay que controlar el uso de nuestras máquinas o usuarios en horas no previstas, con patrones de comportamiento infrecuentes, así como el control del tráfico de salida y entrada.

- **Troyanos**: se debe verificar la integridad de los programas periódicamente, mediante mecanismos de suma o de firmas; detectar el tráfico anómalo de salida o entrada al sistema, y utilizar cortafuegos para bloquear tráfico sospechoso. Una versión bastante peligrosa de los troyanos la forman los *rootkits* (comentados más adelante), que realizan más de una función gracias a un conjunto variado de herramientas. Para la verificación de la integridad, podemos utilizar mecanismos de sumas, como md5 o gpg, o herramientas que automatizan este proceso, como Tripwire o AIDE.

- **Puertas traseras**: hay que obtener de los proveedores o vendedores del software la certificación de que no contiene ningún tipo de puerta trasera escondida no documentada y, por supuesto, aceptar el software provenien-

te solo de sitios que ofrezcan garantías. Cuando el software sea de terceros o de fuentes que podrían haber modificado el software original, muchos fabricantes (o distribuidores) integran algún tipo de verificación de software basado en códigos de suma o firmas digitales (tipo md5 o gpg) [Hatd]. Siempre que estas estén disponibles, sería útil verificarlas antes de proceder a la instalación del software. También puede probarse el sistema de forma intensiva, antes de colocarlo como sistema de producción. Otro problema puede consistir en la alteración del software *a posteriori*. En este caso pueden ser también útiles los sistemas de firmas o sumas para crear códigos sobre software ya instalado y controlar que no se produzcan cambios en software vital. También resultan útiles las copias de seguridad, con las que podemos hacer comparaciones para detectar cambios.

- **Bombas lógicas**: en este caso, suelen ocultarse tras activaciones por tiempo o por acciones del usuario. Podemos verificar que no existan en el sistema trabajos no interactivos introducidos de tipo `crontab`, `at` y otros procesos (por ejemplo, de tipo `nohup`), que dispongan de ejecución periódica o que estén en ejecución en segundo plano desde hace mucho tiempo (comandos `w`, `jobs`). En cualquier caso, podrían utilizarse medidas preventivas que impidieran trabajos programados no interactivos a los usuarios (`crontab`) o que solamente los permitiesen a aquellos que realmente lo necesiten.

- **Registradores de tecleo y *rootkits***: en este caso habrá algún proceso intermediario que intentará capturar nuestras pulsaciones de teclas y las almacenará o comunicará en algún lugar. Habrá que examinar situaciones donde aparezca algún proceso extraño perteneciente a nuestro usuario, o bien detectar si tenemos algún fichero abierto con el que no estemos trabajando directamente (por ejemplo, podría ser de ayuda `lsof`, ver man), o bien conexiones de red, si se tratase de un registrador de tecleo con envío externo. Para probar un funcionamiento muy básico de un registrador de tecleo muy sencillo, puede verse el comando de sistema `script` (ver man script). El otro caso, el *rootkit* (que suele incluir también algún registrador de tecleo) suele ser un paquete de unos cuantos programas con varias técnicas, y permite al atacante, una vez entra en una cuenta, utilizar diversos elementos como un registrador de tecleo, puertas traseras, troyanos (sustituyendo a comandos del sistema), etc., con tal de obtener información y puertas de entrada al sistema. Muchas veces se acompaña de programas que realizan limpieza de los registros, para eliminar las pruebas de la intrusión. Un caso particularmente peligroso lo forman los *rootkits*, que se usan o vienen en forma de módulos de núcleo, lo que les permite actuar a nivel del mismo núcleo. Una herramienta útil para verificar los rootkits es `chkrootkit`, u otras alternativas como `rkhunter`.

Enlace de interés

La herramienta chkrootkit puede encontrarse en: http://www.chkrootkit.org.

- **Escaneo de puertos**: los escaneadores suelen lanzar sobre uno o más sistemas bucles de escaneo de puertos conocidos para detectar los que quedan abiertos y aquellos servicios que están funcionando (y obtener información de las versiones de los servicios) y que podrían ser susceptibles de ataques.

- **Husmeadores**: deben evitarse intercepciones e impedir así la posibilidad de que se introduzcan escuchas. Una técnica es la construcción hardware de la red, que puede dividirse en segmentos para que el tráfico sólo circule por la zona que se va a utilizar, poner cortafuegos para unir estos segmentos y poder controlar el tráfico de entrada y salida. Usar técnicas de cifrado para que los mensajes no puedan ser leídos e interpretados por alguien que escuche la red. Para el caso tanto de escáneres como de husmeadores, podemos utilizar herramientas como `Wireshark` (antiguo `Ethereal`) y `Snort`, para realizar comprobaciones sobre nuestra red o, para el escaneo de puertos, *Nmap*. En el caso de los husmeadores, estos pueden ser detectados en la red mediante la búsqueda de máquinas en modo Ethernet promiscuo (están a la escucha de cualquier paquete que circula). Normalmente, la tarjeta de red solo captura el tráfico que va hacia ella (de tipo *broadcast* o *multicast*).

- **Secuestro**: algunas contramedidas en este caso son las siguientes: implementar mecanismos de cifrado en los servicios, requerir autentificación y, si es posible, que esta autentificación se renueve periódicamente; controlar el tráfico entrante o saliente mediante cortafuegos, y monitorizar la red para detectar flujos de tráfico sospechosos. En casos de aplicaciones web o correo electronico, limitar al máximo la utilización de códigos de *script* o, sencillamente, eliminar la posibilidad de ejecución de *scripts* internos a ventanas HTML cuando provengan de fuentes no fiables.

- **Desbordamientos**: suelen ser comunes como *bugs* o agujeros del sistema y suelen (si han sido previamente detectados) solucionarse mediante una actualización del software o paquete afectado. En todo caso, pueden observarse, gracias a los registros del sistema, situaciones extrañas de caída de servicios que deberían estar funcionando. También pueden maximizarse los controles de procesos y accesos a recursos para aislar el problema cuando se produzca en entornos de acceso controlado, como el que ofrece SELinux. Existen otras altenativas, como Grsecurity o PaX, que son parches de los núcleos oficiales de Linux para obtener protecciones adicionales en este sentido, tanto problemas de desbordamiento de *buffers*, como de pila o *heap*.

SELinux se trata en el apartado 4 de este módulo.

- **Denegación de servicio y otros como SYN flood o bombardeo de correo**: se deben tomar medidas de bloqueo de tráfico innecesario en nuestra red (por ejemplo, por medio de cortafuegos). En aquellos servicios que se pueda, habrá que controlar tamaños de *buffer*, número de clientes por atender, tiempos de espera (*timeouts*) de cierre de conexiones, capacidades del servicio, etc.

- **Falseamiento de identidad**: a) *IP spoofing*, b) *ARP spoofing*, c) correo electrónico. Estos casos necesitan un fuerte cifrado de los servicios, control por cortafuegos y mecanismos de autentificación basados en varios aspectos (por ejemplo, no basarse en la IP, si pudiera verse comprometida). Se

pueden aplicar mecanismos que controlen las sesiones establecidas y que se basen en varios parámetros de la máquina a la vez (sistema operativo, procesador, IP, dirección Ethernet, etc.). También se pueden monitorizar sistemas DNS, cachés de ARP, *spools* de correo, etc., para detectar cambios en la información que invaliden a anteriores.

- **Ingeniería social**: no es propiamente una cuestión informática, pero también es necesaria para que las personas no empeoren la seguridad. Existen diversas medidas adecuadas, como aumentar la información o educar a los usuarios (para que no divulguen datos privados ni información que pueda ofrecer pistas sobre sus contraseñas, advertirlos sobre el uso que hagan de las redes sociales, etc.) y divulgar técnicas básicas para mejorar su seguridad. También deben tenerse en cuenta otros temas de seguridad: controlar qué personal dispondrá de información crítica de seguridad y en qué condiciones puede cederla a otros. Los servicios de ayuda y mantenimiento de una empresa pueden ser un punto crítico, y debe controlarse quién posee información de seguridad, y cómo la usa.

Respecto a los usuarios finales, hay que esforzarse por mejorar su cultura de contraseñas, evitar que la dejen apuntada en cualquier sitio, a la vista de terceros o, simplemente, que la divulguen de forma indirecta (nuevamente, relacionado con el uso de redes sociales o servicios inseguros).

2. Seguridad del sistema

Ante los posibles ataques, tenemos que disponer de mecanismos de prevención, detección y recuperación de nuestros sistemas.

Para la prevención local, hay que examinar los diferentes mecanismos de autentificación y permisos de acceso a los recursos para poderlos definir correctamente, de manera que se garantice la confidencialidad y la integridad de nuestra información.

Incluso en este caso, no estaremos protegidos de atacantes que hayan ya obtenido acceso a nuestro sistema, o bien de usuarios hostiles que quieran saltarse las restricciones impuestas en el sistema.

Respecto a la seguridad en red, tenemos que garantizar que los recursos que ofrecemos (si proporcionamos unos determinados servicios) tengan los parámetros de confidencialidad necesarios (más si estos están garantizados por leyes jurídicas nacionales, el incumplimiento de las cuales puede causar graves sanciones, así como la publicidad negativa indirecta por la falta de cumplimiento). Los servicios ofrecidos no pueden ser usados por terceros no deseados, de modo que un primer paso será controlar que los servicios ofrecidos sean los que realmente queremos y que no estamos ofreciendo, además, otros servicios que no tenemos controlados. En el caso de servicios de los que nosotros somos clientes, también habrá que asegurar los mecanismos de autentificación, en el sentido de acceder a los servidores correctos y que no existan casos de suplantación de servicios o servidores (normalmente bastante difíciles de detectar a nivel de usuario).

Respecto a las aplicaciones y a los mismos servicios, además de garantizar la correcta configuración de niveles de acceso mediante permisos y la autentificación de los usuarios permitidos, tendremos que vigilar la posible explotación de fallos en el software. Cualquier aplicación, por muy bien diseñada e implementada que esté, puede tener un número más o menos alto de errores que pueden ser aprovechados para, con ciertas técnicas, saltarse las restricciones impuestas. En este caso, practicamos una política de prevención que pasa por mantener el sistema actualizado en la medida de lo posible, de manera que, o bien actualizamos ante cualquier nueva corrección, o bien somos conservadores y mantenemos aquellas versiones que sean más estables en cuestión de seguridad. Normalmente, esto significa verificar periódicamente unos cuan-

tos sitios de seguridad, de conocida solvencia, para conocer los últimos fallos detectados en el software, tanto de aplicaciones como de sistema, y las vulnerabilidades que se derivan de ellos y que podrían exponer nuestros sistemas a fallos de seguridad, ya sea localmente o por red.

3. Seguridad local

Acceso local

Diversos ataques, aunque vengan del exterior, tienen como finalidad conseguir el acceso local.

La seguridad local [Pen, Hatb] es básica para la protección del sistema [Deb, Hatc], ya que, normalmente, tras un primer intento de acceso desde la red, es la segunda barrera de protección antes de que un ataque consiga hacerse con parte del control de la máquina. Además, la mayoría de los ataques acaban haciendo uso de recursos internos del sistema.

3.1. *Bootloaders*

Respecto a la seguridad local, ya en arranque se nos pueden presentar problemas debido al acceso físico que un intruso pudiera tener a la máquina.

Uno de los problemas ya se encuentra en el propio arranque del sistema. Si el sistema puede arrancarse desde dispositivos extraíbles o desde CD/DVD, un atacante podría acceder a los datos de una partición GNU/Linux (o también en entornos Windows) solo con montar el sistema de ficheros y podría colocarse como usuario root sin necesidad de conocer ninguna contraseña. En este caso, se necesita proteger el arranque del sistema desde la BIOS, por ejemplo, protegiendo el acceso por contraseña, de manera que no se permita el arranque desde CD/DVD (por ejemplo mediante un LiveCD) o disquete, USB u otras conexiones externas.

También es razonable actualizar la BIOS, ya que también puede tener fallos de seguridad. Además, hay que tener cuidado, porque la mayoría de fabricantes de BIOS ofrecen contraseñas extras conocidas (una especie de puerta trasera), con lo cual no podemos depender de estas medidas en exclusiva.

El siguiente paso es proteger el *bootloader*, ya sea Lilo o Grub, para que el atacante no pueda modificar las opciones de arranque del núcleo o modificar directamente el arranque (caso de Grub). Cualquiera de los dos puede protegerse también por contraseñas adicionales.

En Grub, el fichero `/sbin/grub-md5-crypt` pide la contraseña y genera una suma md5 asociada. En algunas distribuciones en que no está disponible este comando, puede realizarse de manera alternativa durante el arranque: al cargar Grub, accedemos con la tecla *C* al *shell* de Grub e introducimos `md5crypt` para obtener el *hash* md5 asociado a una contraseña que propongamos.

Después, el valor obtenido se introduce en `/boot/grub/grub.conf`. Bajo la línea `timeout`, se introduce:

```
password --md5 suma-md5-calculada
```

Para LiLo se coloca, o bien una contraseña global con:

```
password=contraseña
```

o bien una en la partición que queramos:

```
image = /boot/vmlinuz-version
password = contraseña
restricted
```

En este caso `restricted` indica, además, que no se podrán cambiar los parámetros pasados al núcleo desde la línea de comandos. Hay que tener cuidado de poner el fichero `/etc/lilo.conf` protegido a solo lectura/escritura desde el root (`chmod 600`), si no cualquiera puede leer las contraseñas.

En este último punto, cabe recordar una recomendación general: nunca los ficheros con información sensible deberían estar disponibles con permisos de lectura a terceros, y siempre que sea posible se deberá evitarlos o cifrarlos. En particular para datos sensibles, seria deseable algún tipo de sistema de ficheros que permitiese el cifrado de los ficheros presentes. Por ejemplo, podemos destacar Encryptfs que ya está disponible en varias distribuciones, para el cifraje de los archivos de los directorios privados de los usuarios.

Encryptfs

Los sistemas de ficheros encriptados también son una buena opción para proteger los datos sensibles de usuarios, como por ejemplo Ecryptfs, que está disponible con soporte de núcleo desde la versión 2.6.19.

Otro tema relacionado con el arranque es la posibilidad de que alguien que tenga acceso al teclado, reinicie el sistema, ya que si se pulsa `CTRL+ALT+DEL` (ahora, en varias distribuciones, esta opción suele estar desactivada por defecto), se provoca una operación de cierre en la máquina. Este comportamiento viene definido en `/etc/inittab`, con una línea como:

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

Si se comenta, esta posibilidad de reiniciar quedará desactivada. O por el contrario, puede crearse un fichero `/etc/shutdown.allow`, que permite a ciertos usuarios poder apagar y/o reiniciar.

3.2. Contraseñas y *shadows*

Las contraseñas típicas de los sistema UNIX iniciales (y de primeras versiones GNU/Linux) estaban cifradas mediante unos algoritmos DES (pero con claves pequeñas, y una llamada de sistema se encargaba de cifrar y descifrar, en concreto `crypt`, ver su man).

Normalmente, se encontraban en el fichero `/etc/passwd`, en el segundo campo, por ejemplo:

```
user:sndb565sadsd:...
```

Pero el problema está en que este fichero es legible por cualquier usuario del sistema, con lo que un atacante podía obtener el fichero y utilizar un ataque de fuerza bruta, hasta que descifrase las contraseñas contenidas en el fichero, o bien mediante ataques de fuerza bruta un poco más inteligentes, por medio de diccionarios.

El primer paso es utilizar los ficheros `/etc/shadow`, donde se guarda ahora la contraseña. Este fichero es solo legible por el root. En este caso, en `/etc/passwd` aparece un asterisco (`*`) donde antes estaba la contraseña cifrada. Por defecto, las distribuciones de GNU/Linux actuales usan contraseñas de tipo *shadow*, a no ser que se les diga que no las usen.

Un segundo paso es cambiar el sistema de cifrado de las contraseñas por uno más complejo y difícil de romper. Ahora, tanto Fedora como Debian ofrecen contraseñas por md5; normalmente nos dejan escoger el sistema en tiempo de instalación. Hay que tener cuidado con las contraseñas md5, ya que si usamos NIS, podríamos tener algún problema, dependiendo de las versiones; si no, todos los clientes y servidores usarán md5 para sus contraseñas. Las contraseñas se pueden reconocer en `/etc/shadow` porque vienen con un prefijo `"$id$"` con `id=1`. No son los únicos algoritmos de cifrado que pueden usarse, `id=5` es sha-256 o `id=6` es sha-512. De hecho, para utilizar contraseñas cifradas más fiables, puede usarse (en algunas distribuciones) el comando `mkpasswd`, al que puede aportarse el cifrado utilizado y la contraseña a cifrar (véanse los man de `mkpasswd` y `crypt`).

Otras posibles actuaciones consisten en obligar a los usuarios a cambiar la contraseña con frecuencia (el comando `change` puede ser útil), imponer restricciones en el tamaño y el contenido de las contraseñas y validarlas con diccionarios de términos comunes.

Respecto a las herramientas, es interesante disponer de un *cracker* de contraseñas (o sea, un programa para probar y romper contraseñas), para comprobar la situación real de seguridad de las cuentas de nuestros usuarios y forzar así el cambio en las que detectemos inseguras. Dos de las más utilizadas por admi-

nistradores son `John the Ripper` y `crack`. Pueden funcionar también por diccionario, con lo cual, será interesante disponer de algún ASCII de español (pueden encontrarse en la red). Otra herramienta es `Slurpie`, que puede probar varias máquinas a la vez.

Una cuestión que debe tenerse en cuenta siempre es hacer estas pruebas sobre nuestros sistemas. No hay que olvidar que los administradores de otros sistemas (o el proveedor de acceso o ISP) tendrán sistemas de detección de intrusos habilitados y podemos ser objeto de denuncia por intentos de intrusión, ya sea ante las autoridades competentes (unidades de delitos informáticos) o en nuestro ISP para que se nos cierre el acceso. Hay que tener mucho cuidado con el uso de herramientas de seguridad, que están siempre al filo de la navaja entre ser de seguridad o de intrusión.

3.3. Bits *sticky* y *setuid*

Otro problema importante son algunos permisos especiales que son utilizados sobre ficheros o *scripts*.

El bit *sticky* se utiliza sobre todo en directorios temporales, donde queremos que en algunos grupos (a veces no relacionados), cualquier usuario pueda escribir, pero solo pueda borrar el propietario del directorio o bien el propietario del fichero que esté en el directorio. Un ejemplo clásico de este bit es el directorio temporal `/tmp`. Hay que vigilar que no haya directorios de este tipo, ya que pueden permitir que cualquiera escriba en ellos, por lo que habrá que comprobar que no haya más que los puramente necesarios como temporales. El bit se coloca mediante (`chmod +t dir`) y puede quitarse con `-t`. En un `ls` aparecerá como un directorio con permisos `drwxrwxrwt` (observad la última `t`).

El bit *setuid* permite a un usuario ejecutar (ya sea un programa binario ejecutable o un *shell script*) con los permisos de otro usuario. Esto en algún caso puede ser de utilidad, pero es potencialmente peligroso. Es el caso, por ejemplo, de programas con *setuid* de root: un usuario, aunque no tiene permisos de root, puede ejecutar un programa con *setuid* que puede disponer de permisos internos de usuario root. Esto es muy peligroso en caso de *scripts*, ya que podrían editarse y modificarse para hacer cualquier cosa. Por lo tanto, hay que tener controlados estos programas y, en caso de que no se necesite el *setuid*, eliminarlo. El bit se coloca mediante `chmod +s`, ya sea aplicándolo al propietario (se llama entonces *suid*) o al grupo (se llama bit *sgid*); puede quitarse con `-s`. En el caso de visualizar con `ls`, el fichero aparecerá con `-rwSrw-rw` (observad la `S`), si es sólo *suid*, en *sgid* la `S` aparecería tras la segunda `w`.

En caso de utilizar `chmod` con notación octal, se usan cuatro cifras, donde las tres últimas son los permisos clásicos `rwxrwxrwx` (recordad que se debe

sumar en la cifra 4 para `r`, 2 `w`, y 1 para `x`) y la primera tiene un valor para cada permiso especial que se quiera (que se suman): 4 (para *suid*), 2 (*sgid*) y 1 (para *sticky*).

Sería deseable hacer un análisis periódico del sistema de ficheros para detectar los ficheros con *suid* y *sgid* en el sistema y determinar si estos son realmente necesarios o pueden provocar problemas de seguridad.

3.4. Habilitación de *hosts*

En el sistema hay unos cuantos ficheros de configuración especiales que permiten habilitar el acceso a una serie de *hosts* para algunos servicios de red, pero cuyos errores pueden permitir atacar después la seguridad local. Nos podemos encontrar con:

- *.rhosts* de usuario: permite que un usuario pueda especificar una serie de máquinas (y usuarios) que pueden usar su cuenta mediante comandos "r" (rsh, rcp...) sin necesidad de introducir la contraseña de la cuenta. Esto es potencialmente peligroso, ya que una mala configuración del usuario podría permitir entrar a usuarios no deseados, o que un atacante (con acceso a la cuenta del usuario) cambie las direcciones en *.rhosts* para poder entrar cómodamente sin ningún control. Normalmente, no se tendría que permitir crear estos archivos, e incluso habría que borrarlos completamente y deshabilitar los comandos "r". Un análisis periódico del sistema será útil para detectar la creación de estos ficheros.

- */etc/hosts.equiv*: es exactamente lo mismo que los ficheros *.rhosts* pero a nivel de máquina, especificando qué servicios, qué usuarios y qué grupos pueden acceder sin control de contraseña a los servicios "r". Además, un error como poner en una línea de ese fichero un "+", permite el acceso a "cualquier" máquina. Normalmente, hoy en día tampoco suele existir por defecto este fichero creado, y siempre existen como alternativa a los "r" los servicios tipo ssh.

- */etc/hosts.lpd*: en el sistema de impresión LPD se utilizaba para habilitar las máquinas que podían acceder al sistema de impresión. Hay que tener especial cuidado, si no estamos sirviendo impresión, de deshabilitar completamente el acceso al sistema, y en caso de que lo estemos haciendo, restringir al máximo las máquinas que realmente hacen uso del mismo. También se puede intentar cambiar a un sistema de impresión CUPS, por ejemplo, que tiene mucho más control sobre los servicios. El sistema de impresión LPD había sido un blanco habitual de ataques de tipo gusano o de desbordamiento, y están documentados varios *bugs* importantes. Hay que estar atentos si aún utilizamos este sistema y el fichero `hosts.lpd`.

3.5. Módulos PAM

Los módulos PAM [Pen, Mor03] son un método que permite al administrador controlar cómo se realiza el proceso de autentificación de los usuarios para determinadas aplicaciones. Las aplicaciones tienen que haber sido creadas y enlazadas a las bibliotecas PAM. Básicamente, los módulos PAM son un conjunto de bibliotecas compartidas que pueden incorporarse a las aplicaciones como método para controlar la autentificación de sus usuarios. Es más, puede cambiarse el método de autentificación (mediante la configuración de los módulos PAM) sin que sea necesario cambiar la aplicación.

Los módulos PAM (las bibliotecas) suelen estar en `/lib/security` (en forma de ficheros objeto cargables dinámicamente). La configuración de PAM está presente en el directorio `/etc/pam.d`, donde aparece un fichero de configuración de PAM por cada aplicación que está usando módulos PAM. Nos encontramos con la configuración de autentificación de aplicaciones y servicios como ssh, de *login* gráfico de X Window System, como xdm, gdm, kdm, xscreensaver, etc. o, por ejemplo, del *login* del sistema (la entrada por identificador de usuario y contraseña). En las antiguas versiones de PAM, se utilizaba un archivo de configuración general (típicamente en `/etc/pam.conf`), que era donde se leía la configuración PAM si el directorio `/etc/pam.d` no existía.

Enlace de interés

Para saber más sobre los módulos PAM, se puede ver la *The Linux-PAM System Administrators Guide* en: http://www.kernel.org/pub /linux/libs/pam /Linux-PAM-html.

La línea típica de estos ficheros (en `/etc/pam.d`) tendría este formato (si se utiliza `/etc/pam.conf` habría que añadir el servicio a que pertenece como primer campo):

```
module-type control-flag module-path arguments
```

donde se especifica:

1) tipo de módulo: si es un módulo que requiere que el usuario se autentifique (`auth`) o es de acceso restringido (`account`); cosas que hay que hacer cuando el usuario entra o sale (`session`); o bien hay que actualizar la contraseña (`password`);

2) *flags* de control: especifican si es necesario (`required`), si es requisito previo (`requisite`), si es suficiente (`sufficient`) o si es opcional (`optional`). Esta es una de las posibles sintaxis, pero para este campo existe una alternativa más actual que trabaja en parejas valor y acción;

3) la ruta (*path*) del módulo;

4) argumentos que se pasan al módulo (dependen de cada módulo).

Debido a que algunos servicios necesitan diversas líneas de configuración comunes, hay posibilidad de operaciones de inclusión de definiciones comunes de otros servicios. Para ello solo hay que añadir una línea con:

```
@include servicio
```

Un pequeño ejemplo del uso de módulos PAM (en una distribución Debian), puede ser su uso en el proceso de *login* (se han listado también las líneas incluidas provenientes de otros servicios):

```
auth      requisite pam_securetty.so
auth      requisite pam_nologin.so
auth      required  pam_env.so
auth      required  pam_unix.so nullok
account   required  pam_unix.so
session   required  pam_unix.so
session   optional  pam_lastlog.so
session   optional  pam_motd.so
session   optional  pam_mail.so standard noenv
password  required  pam_unix.so nullok obscure min = 4 max = 8 md5
```

Esto especifica los módulos PAM necesarios para controlar la autentificación de usuarios en el *login* (entrada al sistema). Uno de los módulos `pam_unix.so` es el que realmente hace la verificación de la contraseña del usuario (examinando como datos ficheros `passwd`, `shadow`, etc.).

Otros módulos controlan su sesión para ver cuándo ha entrado por última vez, o guardan cuándo entra y sale (para el comando `lastlog`). También hay un módulo que se encarga de verificar si el usuario tiene correo por leer (también hay que autentificarse) y otro que controla que se cambie la contraseña (si está obligado a hacerlo en el primer *login* que haga) y que tenga de 4 a 8 letras. Puede utilizarse md5 para el cifrado de contraseñas.

En este ejemplo podríamos mejorar la seguridad de los usuarios: el `auth` y el `password` permiten contraseñas de longitud nula: es el argumento `nullok` del módulo. Esto permitiría tener usuarios con contraseñas vacías (fuente de posibles ataques). Si quitamos este argumento, ya no permitimos contraseñas vacías en el proceso de *login*. Lo mismo puede hacerse en el fichero de configuración de `passwd` (en este caso, el comando de cambio del contraseña), que también presenta el `nullok`. Otra posible acción es incrementar en ambos ficheros el tamaño máximo de las contraseñas, por ejemplo, con `max = valor`.

3.6. Alteraciones del sistema

Enlace de interés

Tenéis más información sobre AusCert UNIX checklist en: http://www.auscert.org.au/5816.

Otro problema puede ser la alteración de comandos o configuraciones básicas del sistema, mediante la introducción de troyanos o puertas trasera en el sistema, por la simple introducción de software que sustituya o modifique ligeramente el comportamiento del software de sistema.

Un caso típico es la posibilidad de forzar al usuario root para que ejecute comandos falsos de sistema; por ejemplo, si el root incluyese el "." en su variable de PATH, esto permitiría la ejecución de comandos desde su directorio actual, lo que habilitaría la colocación de archivos que sustituyesen a comandos del sistema y que serían ejecutados en primer término antes que los de sistema. El mismo proceso puede hacerse con un usuario, aunque por ser más limitados sus permisos, puede no afectar tanto al sistema, cuanto más a la propia seguridad del usuario. Otro caso típico es el de las pantallas de *login* falsas, que pueden sustituir el típico proceso de *login*, `passwd`, por un programa falso que almacene las contraseñas introducidas.

En caso de estas alteraciones, será imprescindible usar políticas de auditoría de cambios del sistema, ya sea por medio de cálculo de firmas o sumas (gpg o md5), o bien mediante algún software de control como `Tripwire` o `AIDE`. Para los troyanos podemos hacer diferentes tipos de detecciones, o bien utilizar herramientas como `chkrootkit`, si éstos viniesen de la instalación de algún *rootkit* conocido.

Enlace de interés

Tenéis más información sobre chkrootkit en: http://www.chkrootkit.org.

4. SELinux

La seguridad tradicional dentro del sistema se ha basado en las técnicas DAC (*discretionary access control*, control de acceso discrecional), donde normalmente cada programa dispone de control completo sobre los accesos a sus recursos. Si un determinado programa (o el usuario lo permite) decide hacer un acceso incorrecto (por ejemplo, dejando datos confidenciales en abierto, ya sea por desconocimiento o por mal funcionamiento), no hay nada que se lo impida. Así en DAC, un usuario tiene completo control sobre los objetos que le pertenecen y los programas que ejecuta. El programa ejecutado dispondrá de los mismos permisos que el usuario que lo está ejecutando. Así, la seguridad del sistema dependerá de las aplicaciones que se estén ejecutando y de las vulnerabilidades que estas pudiesen tener, o del software malicioso que incluyesen, y en especial afectará a los objetos (otros programas, ficheros, o recursos) a los que el usuario tenga acceso. En el caso del usuario root, esto comprometería la seguridad global del sistema.

Por otro lado, las técnicas MAC (*mandatory access control*, control de acceso obligatorio) desarrollan políticas de seguridad (definidas por el administrador) en las que el sistema tiene control completo sobre los derechos de acceso que se conceden sobre cada recurso. Por ejemplo, con permisos (de tipo UNIX) podemos dar acceso a ficheros, pero mediante políticas MAC tenemos control adicional para determinar a qué ficheros explícitamente se permite acceso por parte de un proceso, y qué nivel de acceso queremos conceder. Se fijan contextos, en los cuales se indican en qué situaciones un objeto puede acceder a otro objeto.

SELinux [Nsab] es un componente de tipo MAC reciente incluido en la rama 2.6.x del núcleo, que las distribuciones van incluyendo progresivamente: Fedora/Red Hat lo traen habilitado por defecto (aunque es posible cambiarlo durante la instalación), y en Debian es un componente opcional. SELinux implementa políticas de seguridad de tipo MAC y permite disponer de un acceso de permisos más fino que los tradicionales permisos de archivo UNIX. Por ejemplo, el administrador podría permitir que se añadiesen datos a un archivo de registro, pero no reescribirlo o truncarlo (técnicas utilizadas habitualmente por atacantes para borrar las pistas de sus accesos). En otro ejemplo, podría permitirse que programas de red se enlazaran a un puerto (o puertos) que necesitaran y, sin embargo, denegar el acceso a otros puertos (esta podría ser una técnica que permitiese controlar y limitar el acceso de algunos troyanos o puertas traseras). SELinux fue desarrollado por la agencia NSA de Estados Unidos, con aportaciones de diversas compañías para sistemas UNIX

Enlaces de interés

En los siguientes enlaces tenéis algunos recursos sobre SELinux: http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/selinux-guide, http://www.nsa.gov/research/selinux/index.shtml, http://fedoraproject.org/wiki/SELinux, http://selinux.sourceforge.net/.

y libres, como Linux y BSD. Se liberó en el año 2000 y desde entonces se ha ido integrando en diferentes distribuciones GNU/Linux.

Disponemos en SELinux de un modelo de dominio-tipo, donde cada proceso corre en un denominado contexto de seguridad y cualquier recurso (fichero, directorio, *socket*, etc.) tiene un tipo asociado con él. Hay un conjunto de reglas que indican qué acciones pueden efectuarse en cada contexto sobre cada tipo. Una ventaja de este modelo contexto-tipo es que las políticas que se definan se pueden analizar (existen herramientas) para determinar qué flujos de información están permitidos; por ejemplo, para detectar posibles vías de ataque o si la política es lo suficientemente completa para cubrir todos los accesos posibles.

Se dispone de lo que se conoce como la base de datos de políticas de SELinux (*SELinux policy database*) que controla todos los aspectos de SELinux. Determina qué contextos puede utilizar cada programa para ejecutarse y especifica a qué tipos de cada contexto puede acceder.

En SELinux cada proceso del sistema tiene un contexto compuesto de tres partes: una identidad, un rol y un dominio. La identidad es el nombre de la cuenta del usuario, o bien `system_u` para procesos de sistema o `user_u` si el usuario no dispone de políticas definidas. El rol determina qué contextos están asociados. Por ejemplo `user_r` no tiene permitido tener el contexto `sysadm_t` (dominio principal para el administrador de sistema). Así un `user_r` con identidad `user_u` no puede obtener de ninguna manera un contexto `sysadm_t`. Un contexto de seguridad se especifica siempre por esta terna de valores, como por ejemplo:

```
root:sysadm_r:sysadm_t
```

que es el contexto para el administrador del sistema, define su identidad, su rol y su contexto de seguridad.

Por ejemplo, en una máquina con SELinux activado (una Fedora en este caso) podemos ver con la opción `-Z` del `ps` los contextos asociados a los procesos:

```
# ps -ax -Z

LABEL                        PID   TTY   STAT   TIME   COMMAND
system_u:system_r:init_t     1     ?     Ss     0:00   init
system_u:system_r:kernel_t   2     ?     S      0:00   [migration/0]
system_u:system_r:kernel_t   3     ?     S      0:00   [ksoftirqd/0]
system_u:system_r:kernel_t   4     ?     S      0:00   [watchdog/0]
system_u:system_r:kernel_t   5     ?     S      0:00   [migration/1]
```

```
system_u:system_r:kernel_t      6     ?      SN   0:00  [ksoftirqd/1]
system_u:system_r:kernel_t      7     ?      S    0:00  [watchdog/1]
system_u:system_r:syslogd_t     2564  ?      Ss   0:00  syslogd -m 0
system_u:system_r:klogd_t       2567  ?      Ss   0:00  klogd -x
system_u:system_r:irqbalance_t  2579  ?      Ss   0:00  irqbalance
system_u:system_r:portmap_t     2608  ?      Ss   0:00  portmap
system_u:system_r:rpcd_t        2629  ?      Ss   0:00  rpc.statd
user_u:system_r:unconfined_t    4812  ?      Ss   0:00  /usr/libexec/gconfd-2 5
user_u:system_r:unconfined_t    4858  ?      Sl   0:00  gnome-terminal
user_u:system_r:unconfined_t    4861  ?      S    0:00  gnome-pty-helper
user_u:system_r:unconfined_t    4862  pts/0  Ss   0:00  bash
user_u:system_r:unconfined_t    4920  pts/0  S    0:01  gedit
system_u:system_r:rpcd_t        4984  ?      Ss   0:00  rpc.idmapd
system_u:system_r:gpm_t         5029  ?      Ss   0:00  gpm -m /dev/input/mice -t exps2
user_u:system_r:unconfined_t    5184  pts/0  R+   0:00  ps ax -Z
user_u:system_r:unconfined_t    5185  pts/0  D+   0:00  Bash
```

y con `ls` con la opcion `-Z` podemos ver los contextos asociados a ficheros y directorios:

```
# ls -Z
drwxr-xr-x josep josep user_u:object_r:user_home_t Desktop
drwxrwxr-x josep josep user_u:object_r:user_home_t proves
-rw-r--r-- josep josep user_u:object_r:user_home_t yum.conf
```

y desde la consola podemos conocer nuestro contexto actual con:

```
$ id -Z
user_u:system_r:unconfined_t
```

Respecto al modo de funcionamiento, SELinux presenta dos modos denominados: *permissive* y *enforcing*. En *permissive* se permiten los accesos no autorizados, pero son auditados en los registros correspondientes (normalmente directamente sobre `/var/log/messages` o bien, dependiendo de la distribución, con el uso de `audit` en `/var/log/audit/audit.log`). En *enforcing* no se permite ningún tipo de acceso que no permitan las políticas definidas. También puede desactivarse SELinux a través de su fichero de configuración (habitualmente en `/etc/selinux/config`), colocando `SELINUX=disabled`.

Hay que tener cuidado con la activación y desactivación de SELinux, en especial con el etiquetado de contextos en los ficheros, ya que en periodos en que se active o desactive pueden perderse etiquetas (puesto que el sistema no estará activo) o simplemente no ser etiquetados. Asimismo, la realización de

copias de seguridad del sistema de ficheros tiene que tener en cuenta que se conserven las etiquetas de SELinux.

Otro posible problema a tener en cuenta es el gran número de reglas de política de seguridad que pueden llegar a existir y que pueden provocar limitaciones en el control de los servicios. Ante un tipo determinado de malfuncionamiento, cabe primero determinar que precisamente no sea SELinux el que está impidiendo el funcionamiento, por una limitación demasiado estricta de seguridad (véase el subapartado 4.2 de crítica a SELinux) o a causa de opciones que no esperábamos tener activadas (pueden requerir cambiar la configuración de los booleanos, como veremos).

Con respecto a la política usada, SELinux soporta dos tipos diferentes: *targeted* y *strict*. En *targeted* la mayoría de procesos operan sin restricciones, y solo servicios (ciertos *daemons*) específicos son puestos en diferentes contextos de seguridad que son confinados a la política de seguridad. En *strict* todos los procesos son asignados a contextos de seguridad y confinados a políticas definidas, de manera que cualquier acción es controlada por las políticas definidas. En principio estos son los dos tipos de políticas definidas generalmente, pero la especificación está abierta a incluir más.

Un caso especial de política es la MLS (*multilevel security*, seguridad multinivel), que es una política multinivel de tipo *strict*. La idea es definir, dentro de la misma política, diferentes niveles de seguridad y los contextos de seguridad tienen asociado un campo adicional de nivel de acceso. Este tipo de política de seguridad (como MLS) suele ser utilizada en organizaciones gubernamentales y militares, donde hay organizaciones jerárquicas con diferentes niveles de información privilegiada, niveles de acceso general y capacidades de acción diferentes en cada nivel. Para obtener algunas certificaciones de seguridad (concedidas o avaladas por ciertas organizaciones) para los sistemas operativos, se necesita disponer de políticas de seguridad de este tipo.

Se puede definir qué tipo de política se usará en `/etc/selinux/config`, variable `SELINUXTYPE`. La política correspondiente, y su configuración, normalmente estará instalada en los directorios `/etc/selinux/SELINUXTYPE/`. Por ejemplo, suele encontrarse en el subdirectorio `policy` el fichero binario de la política compilada (que es el que se carga en el núcleo, al inicializar SELinux).

4.1. Arquitectura

La arquitectura de SELinux está compuesta de los siguientes componentes:

- Código a nivel de núcleo
- La biblioteca compartida de SELinux
- La política de seguridad (la base de datos)
- Herramientas

Examinemos algunas consideraciones con respecto a cada componente:

- El código de núcleo monitoriza la actividad del sistema, y asegura que las operaciones solicitadas estén autorizadas bajo la configuración de políticas de seguridad de SELinux actual; así, no permite las operaciones no autorizadas y normalmente genera entradas en el registro de las operaciones denegadas. El código actualmente se encuentra integrado en los núcleos 2.6.x, mientras que en los anteriores se ofrece como serie de parches.

- La mayoría de utilidades y componentes SELinux no directamente relacionados con el núcleo hacen uso de la biblioteca compartida, llamada `libselinux1.so`, que facilita una API para interaccionar con SELinux.

- La política de seguridad es la que está integrada en la base de datos de reglas de SELinux. Cuando el sistema arranca (con SELinux activado), carga el fichero binario de política, que habitualmente reside en la siguiente ruta: `/etc/security/selinux` (aunque puede variar según la distribución).

  El fichero binario de políticas se crea a partir de una compilación (vía `make`) de los ficheros fuente de políticas y algunos ficheros de configuración.

  Algunas distribuciones (como Fedora) no instalan las fuentes por defecto, que suelen encontrarse en `/etc/security/selinux/src/policy` o en `/etc/selinux`. Normalmente, estas fuentes consisten en varios grupos de información:

  – Los ficheros relacionados con la compilación, `makefile` y *scripts* asociados.

  – Ficheros de la configuración inicial, usuarios y roles asociados.

  – Ficheros de `Type-enforcement`, que contienen la mayoría de las sentencias del lenguaje de políticas asociado a un contexto particular. Hay que tener en cuenta que estos ficheros son enormes, típicamente decenas de miles de líneas. Con lo cual puede aparecer el problema de encontrar fallos o definir cambios en las políticas.

  – Ficheros que sirven para etiquetar los contextos de los ficheros y directorios durante la carga o en determinados momentos.

- Herramientas: Incluyen comandos usados para administrar y usar SELinux, versiones modificadas de comandos estándar Linux y herramientas para el análisis de políticas y el desarrollo.

Veamos, de este último punto, las herramientas típicas de que solemos disponer, algunos de los comandos principales:

Nombre	Utilización
`chcon`	Etiqueta un fichero especifico, o un conjunto de ficheros con un contexto especifico.
`checkpolicy`	Realiza diversas acciones relacionadas con las políticas, incluyendo la compilación de las políticas a binario; típicamente se invoca desde las operaciones de `makefile`.
`getenforce`	Genera mensaje con el modo actual de SELinux (*permissive* o *enforcing*). O bien desactivado si es el caso.
`getsebool`	Obtiene la lista de booleanos, o sea la lista de opciones on/off para cada contexto asociado a un servicio, u opción general del sistema.
`newrole`	Permite a un usuario la transición de un rol a otro.
`runn_init`	Utilizado para activar un servicio (start, stop), asegurándose de que se realiza en el mismo contexto que cuando se arranca automáticamente (con `init`).
`setenforce`	Cambia de modo a SELinux: `0` *permissive*, `1` *enforcing*.
`setfiles`	Etiqueta directorios y subdirectorios con los contextos adecuados; es típicamente utilizado en la configuración inicial de SELinux.
`setstatus`	Obtiene el estado del sistema con SELinux.

Por otra parte, se modifican algunos comandos GNU/Linux con funcionalidades u opciones nuevas, como por ejemplo `cp`, `mv`, `install`, `ls`, `ps`, etc. por ejemplo modificando en los ficheros la etiqueta para asociar el contexto de seguridad (como pudimos comprobar con la opción `-Z` de `ls`). `Id` se modifica para incluir la opción de mostrar el contexto actual del usuario. Y en `ps` vimos que incluía una opción para visualizar los contextos de seguridad actuales de los procesos.

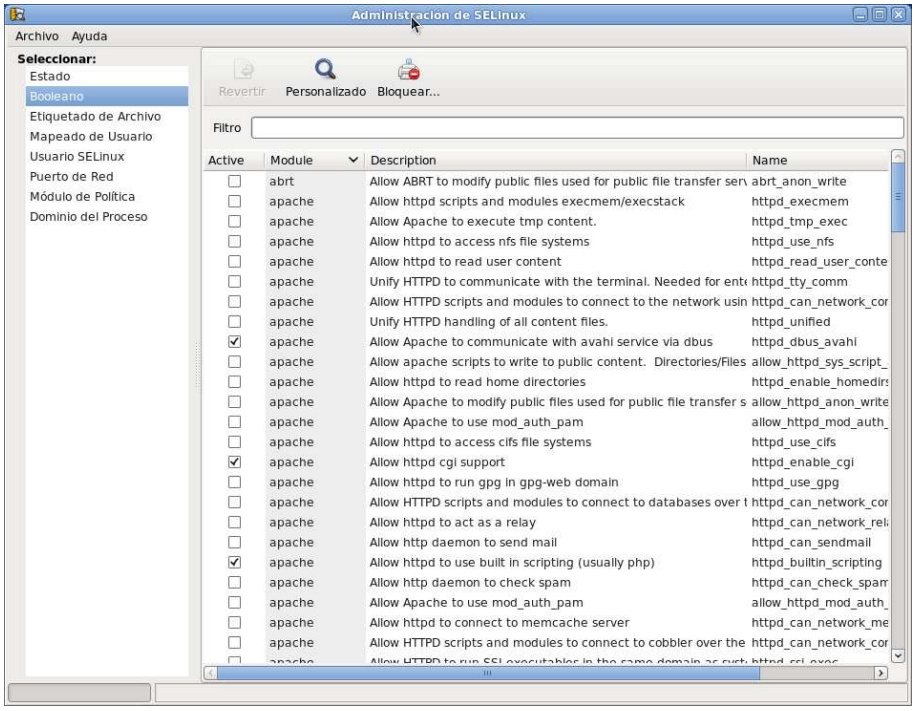
Además, se modifican algunos otros programas comunes para soportar SELinux como:

- cron: Modificado para incluir los contextos para los trabajos en ejecución por cron.
- login: Modificado para que coloque el contexto de seguridad inicial para el usuario cuando entra en el sistema.
- logrotate: Modificado para preservar el contexto de los logs cuando estén recopilados.
- pam: Modificado para colocar el contexto inicial del usuario y para usar la API SELinux y obtener acceso privilegiado a la información de contraseñas.
- ssh: Modificado para colocar el contexto inicial del usuario cuando entra en el sistema.
- Varios programas adicionales que modifican `/etc/passwd` o `/etc/shadow`.

En algunas distribuciones también se incluyen herramientas para la gestión de SELinux, como las `setools(-gui)`, que llevan varias herramientas de gestión y análisis de las políticas. Así, como alguna herramienta específica para controlar los contextos asociados a los diferentes servicios soportados por SE-

Linux en la distribución, la herramienta `system-config-securitylevel` en Fedora dispone de un apartado para la configuración de SELinux como podemos observar en la figura 1.

Figura 1. Interfaz en Fedora para configuración SELinux (examinando booleanos)



En la figura se observa la configuración de booleanos para diversos servicios y opciones genéricas, entre ellas el servidor web. También podemos obtener esta lista con `getsebool -a`, y con los comandos `setsebool`/`togglesebool` podemos activar/desactivar las opciones.

En Fedora, por ejemplo, encontramos soporte de booleanos para (entre otros): cron, ftp, httpd (apache), dns, grub, lilo, nfs, nis, cups, pam, ppd, samba, protecciones contra accesos incorrectos a la memoria de los procesos, etc.

La configuración de booleanos permite la personalización de la política SELinux en tiempo de ejecución. Los booleanos son utilizados en SELinux como valores condicionales de las reglas de la política usada, lo que permite modificaciones de la política sin necesidad de cargar una nueva política.

4.2. Crítica

Algunos administradores y expertos en seguridad han criticado especialmente a SELinux por ser demasiado complejo para configurar y administrar. Se argumenta que, por su complejidad intrínseca, incluso usuarios experimentados pueden cometer errores, lo que dejaría la configuración de SELinux no segura o inservible y el sistema, vulnerable. Esto es discutible hasta cierto punto, ya

que aunque tuviésemos SELinux mal configurado, aún seguirían activos los permisos UNIX y otras herramientas y SELinux no permitiría una operación que los permisos originales ya no permitieran. De hecho, podemos verlo como una etapa más de seguridad más estricta.

También hay factores de rendimiento que pueden verse afectados, debido al enorme tamaño de las políticas; pueden disminuir las prestaciones a causa del gran uso de memoria y al tiempo inicial de carga y, en algún caso, al procesamiento de las reglas. Cabe pensar que estamos hablando prácticamente de un sistema de más de 10.000 reglas en la política. Y aún puede ser un número mayor si escogemos una política de tipo *strict*, con la necesidad de especificar absolutamente todas las opciones a controlar. Normalmente, el procesamiento por política en formato binario y el uso de booleanos para inhabilitar reglas, permite un uso más eficiente del sistema.

Otro aspecto que suele molestar a los administradores es el problema adicional de determinar, ante un determinado malfuncionamiento, cuál es el origen o causa inicial. Es habitual que finalmente nos encontremos que el problema provenía de una configuración excesivamente restrictiva (quizás por desconocimiento del administrador) de SELinux para un determinado servicio. En última instancia, cabe señalar el amplio soporte para la seguridad que SELinux proporciona y que, como administradores, tenemos que ser conscientes de las capacidades y peligros de cualquier nueva técnica que utilicemos.

4.3. Algunas alternativas

Ante el modelo complejo de SELinux y algunas deficiencias del modelo (complejidad de politicas o mala configuración), se han producido algunas alternativas (o en algunos casos complementos) a la utilización de SELinux. Hemos de destacar en particular Grsecurity, una serie de parches para el núcleo Linux que intentan mejorar algunas deficiencias. También hay que destacar a AppArmor, que algunas distribuciones están incluyendo como alternativa a SELinux.

Grsecurity se ofrece como una serie de parches de núcleo publicados posteriormente al establecimiento de una versión estable del núcleo y se pueden obtener de su página web* para ciertas versiones de núcleo, además de paquetes ya preparados para algunas distribuciones.

*http://grsecurity.net/index.php

Enlaces de interés

Se recomienda consultar información sobre Grsecurity en http://en.wikibooks.org/wiki/Grsecurity y sobre parches complementarios como PaX en http://en.wikipedia.org/wiki/PaX, donde hay una buena descripción de los ataques más problemáticos a nivel de código ejecutable.

Grsecurity incluye diferentes prestaciones, como un modelo de acceso a recursos mediante gestión de roles (tanto locales como remotos) de los que dispone de una herramienta denominada gradm para su gestión, auditación de las operaciones internas al núcleo y prevención de código arbitrario en el mismo (de hecho hace uso de un parche de núcleo denominado PaX, que ofrece algunas de estas protecciones). Ello evita posibles errores de tipo desbordamiento de *buffer*, *heap* o *stack* (técnicas aprovechadas para exploits para el núcleo). Tam-

bién incluye notificaciones de alertas de seguridad basadas en IP del usuario. Los parches ofrecidos para el núcleo se aplican como parches al código fuente del núcleo y permiten activar durante la configuración del núcleo previa a su compilación, diferentes opciones de protección del código del núcleo relacionadas con las prestaciones mencionadas. Además, una vez el núcleo está configurado para su soporte, la herramienta gradm permite gestionar los roles y las tareas o protecciones asociadas a cada rol, y a los usuarios que incluyamos en cada uno de ellos.

Grsecurity es un buen complemento o sustituto de algunos puntos débiles de SELinux, ya que ofrece una interfaz más simple de usar e incluye algunas prestaciones de seguridad adicionales.

Enlace de interés

Puede consultarse documentación adicional referente a la administración de Grsecurity en: http://en.wikibooks.org/wiki/Grsecurity.

Otra alternativa, que está surgiendo en algunas distribuciones como Ubuntu y SuSe, es AppArmor, que permite al administrador asociar a cada programa un perfil de seguridad que restrinja las capacidades del programa. Se puede establecer un perfil individual de permisos o bien puede usarse un modo de autoaprendizaje basado en violaciones de perfil registradas y habilitadas o no por los usuarios.

Enlaces de interés

Sobre AppArmor véase: http://www.novell.com/linux/security/apparmor/. Para el caso de LSM y otros proyectos basados: http://en.wikipedia.org/wiki/Linux_Security_Modules. Véase comentarios de proyectos en: https://security.wiki.kernel.org/index.php/Projects.

Básicamente se compone de un módulo de núcleo para la gestión de perfiles de aplicaciones, módulo basado, al igual que SELinux, en LSM (*Linux Security Modules*), juntamente con diferentes utilidades. Mediante estas utilidades podemos establecer permisos de los programas para el acceso al sistema de ficheros y a los dispositivos. La mayor diferencia con SELinux, es que las políticas de seguridad no se aplican con etiquetas a los ficheros, sino respecto a políticas aplicables a los *pathnames*. AppArmor se integra en el núcleo como módulo disponible a partir de las versiones 2.6.36 del núcleo.

5. Seguridad en red

5.1. Cliente de servicios

Clientes y servicios inseguros

Como clientes de servicios, tendremos que evitar el uso de servicios inseguros.

Como clientes de servicios básicamente tenemos que asegurar que no ponemos en peligro a nuestros usuarios (o se pongan en peligro ellos solos) por usar servicios inseguros. Hay que evitar el uso de servicios que no utilicen cifrado de datos y contraseñas (ftp, telnet, correo no seguro) y utilizar en este caso técnicas de conexión cifrada, como SSH y SSL/TSL.

Otro punto importante se refiere a la posible sustitución de servidores por otros falsos, o bien a técnicas de secuestro de sesiones. En estos casos tenemos que disponer de mecanismos de autentificación robustos, que nos permitan verificar la autenticidad de los servidores (por ejemplo, SSH y SSL/TLS tienen algunos de estos mecanismos). Y también tendremos que verificar la red en busca de intrusos, potenciales peligros, intentos de intrusión fallidos o intentos de sustitución de servidores, además de políticas correctas de filtrado de paquetes mediante cortafuegos (*firewalls*), que nos permitan obtener salida a nuestros paquetes de datos válidos y usar los servidores adecuados, controlando los paquetes de entrada que recibamos como respuestas.

5.2. Servidor: inetd y xinetd

La configuración de los servicios de red [Mou01], tal como hemos visto, se realiza desde varios lugares [Ano99, Hat08, Pen]:

- En `/etc/inetd.conf` o el equivalente directorio `/etc/xinetd.d`: estos ficheros de configuración de servicios están asociados a lo que se conoce como *superservidores*, ya que controlan varios servicios hijos y sus condiciones de arranque. El servicio inetd es utilizado en Debian (aunque cada vez menos) y xinetd en Fedora (en Debian, xinetd puede ser instalado opcionalmente en sustitución de inetd).

- Servidores iniciados en arranque: según el *runlevel* tendremos una serie de servicios arrancados. El arranque se originará en el directorio asociado al *runlevel*; por ejemplo, en Debian el runlevel por defecto es el 2, los servicios arrancados serán arrancados desde el directorio `/etc/rc2.d`, seguramente con enlaces a los *scripts* contenidos en `/etc/init.d`, en los cuales se ejecutará con los parámetros `start`, `stop`, `restart`, según corresponda.

- Otros servicios de tipo RPC: asociados a llamadas remotas entre máquinas se usan, por ejemplo, en NIS y NFS. Con el comando `rpcinfo -p` puede examinarse cuáles hay.

Otros ficheros de apoyo (con información útil) son: `/etc/services`, que está formado por una lista de servicios locales o de red conocidos, junto con el nombre del protocolo (tcp, udp u otros) que se utiliza en el servicio y el puerto que utiliza; `/etc/protocols` es una lista de protocolos conocidos, y `/etc/rpc` es una lista de posibles servidores RPC, junto con los puertos (rpc) usados. Estos ficheros vienen con la distribución y son una especie de base de datos que utilizan los comandos y herramientas de red para determinar los nombres de servicios, protocolos o rpc y sus puertos asociados. Cabe destacar que son ficheros más o menos históricos, que no tienen porqué contener todas las definiciones de protocolos y servicios. Pueden asimismo buscarse diferentes listas en Internet de puertos conocidos.

Una de las primeras acciones a realizar por el administrador será deshabilitar todos aquellos servicios que no esté utilizando o no tenga previsto utilizar, en este sentido habrá que documentarse sobre la utilización de los servicios [Mou01], y qué software puede necesitarlos [Neu].

En el caso de `/etc/inetd.conf`, solo tenemos que comentar la línea del servicio que hay que deshabilitar, colocando un "#" como primer carácter en la línea.

En el otro modelo de servicios, utilizado por defecto en Fedora (y opcional en Debian), el xinetd, la configuración reside en el fichero `/etc/xinetd.conf`, donde se configuran algunos valores por defecto de control de registros y, después, la configuración de cada servicio hijo se hace a través de un fichero dentro del directorio `/etc/xinetd.d`. En cada fichero se define la información del servicio, equivalente a la que aparece en el `inetd.conf`; en este caso, para desactivar un servicio solo hay que poner una línea `disable = yes` dentro del fichero del servicio. xinetd tiene una configuración más flexible que inetd, al separar la configuración de los diferentes servicios en diferentes archivos, y tiene bastantes opciones para limitar las conexiones a un servicio, en su número o su capacidad, todo lo cual permite un mejor control del servicio y, si lo configuramos adecuadamente, podemos evitar algunos de los ataques por denegación de servicio (DoS o DDoS).

Respecto al tratamiento de los servicios de los *runlevel* desde comandos de la distribución, ya mencionamos varias herramientas en la unidad de administración local, que permitían habilitar o deshabilitar servicios. También existen herramientas gráficas como `ksysv` de KDE o el `system-config-services` y `ntsysv` en Fedora (en Debian son recomendables `sysv-rc-conf`, `rcconf` o `bum`). Y a un nivel inferior, podemos ir al nivel de *runlevel* que queramos (`/etc/rcx.d`) y desactivar los servicios que se desee cambiando las `S` o `K` ini-

ciales del *script* por otro texto: un método sería, por ejemplo, cambiar `S20ssh` por `STOP_S20ssh` y ya no arrancará; la próxima vez, cuando volvamos a necesitarlo, quitamos el prefijo y lo volvemos a tener activo. Aunque este metodo es posible, es más recomendable utilizar una serie de herramientas simples que proporcionan las distribuciones para colocar, quitar o activar un determinado servicio. Por ejemplo, comandos como, `service` y `chkconfig` en Fedora, o similares en Debian como `update-rc.d` y `invoke-rc.d`).

Otro aspecto es el cierre de servicios no seguros. Tradicionalmente, en el mundo UNIX se habían utilizado servicios de transferencia de archivos como ftp, de conexión remota como telnet y comandos de ejecución (de login o de copia) remotos, muchos de los cuales comenzaban con la letra "r" (por ejemplo, rsh, rcp, rexec, etc.). Otros peligros potenciales son los servicios finger y rwhod, que permitían obtener información desde la red de los usuarios de las máquinas; aquí el peligro estaba en la información que podía obtener un atacante y que le podía facilitar mucho el trabajo. Todos estos servicios no deberían ser usados actualmente debido a los peligros potenciales que implican. Respecto al primer grupo:

- ftp, telnet: en las transmisiones por red no cifran las contraseñas, y cualquiera puede hacerse con las contraseñas de servicios o las cuentas asociadas (por ejemplo, mediante un husmeador).

- rsh, rexec, rcp: presentan, además, el problema de que bajo algunas condiciones ni siquiera necesitan contraseñas (por ejemplo, si se hace desde sitios validados en fichero `.rhosts`), con lo cual vuelven a ser inseguros, y dejan grandes puertas abiertas.

La alternativa es usar clientes y servidores seguros que soporten el cifrado de los mensajes y la autentificación de los participantes. Hay alternativas seguras a los servidores clásicos, pero actualmente la solución más usada es mediante la utilización del paquete OpenSSH (que puede combinarse también con OpenSSL para entornos web). OpenSSH ofrece soluciones basadas en los comandos ssh, scp y sftp, permitiendo sustituir a los antiguos clientes y servidores (se utiliza un *daemon* denominado sshd). El comando ssh permite las antiguas funcionalidades de telnet, rlogin y rsh, entre otros, y scp sería el equivalente seguro de rcp, y sftp, del ftp.

Respecto a SSH, también hay que tener la precaución de usar ssh en la versión 2. La primera versión tiene algunos *exploits* conocidos; hay que tener cuidado al instalar OpenSSH, y si no necesitamos la primera versión, instalar solo soporte para ssh2 (véase la opción `Protocol` en el fichero de configuración `/etc/ssh/sshd_config`).

Además, la mayoría de servicios que dejemos activos en nuestras máquinas tendrían después que ser filtrados a través de cortafuegos, para asegurar que no fuesen utilizados o atacados por personas a los que no iban destinados.

6. Detección de intrusiones

Detección de intrusos

Los sistemas IDS nos permiten la detección a tiempo de intrusos, usando nuestros recursos o explorando nuestros sistemas en busca de fallos de seguridad.

Con los sistemas de detección de intrusos [Hat08] (IDS) se quiere dar un paso más. Una vez hayamos podido configurar más o menos correctamente nuestra seguridad, el paso siguiente consistirá en una detección y prevención activa de las intrusiones.

Los sistemas IDS crean procedimientos de escucha y generación de alertas al detectar situaciones sospechosas, es decir, buscamos síntomas de posibles accidentes de seguridad.

Hay sistemas basados en la información local que, por ejemplo, recopilan información de los registros del sistema, vigilan cambios en los ficheros de sistema o bien en las configuraciones de los servicios típicos. Otros sistemas están basados en red e intentan verificar que no se produzcan comportamientos extraños, como por ejemplo los casos de suplantación, donde hay falsificaciones de direcciones conocidas; controlan el tráfico sospechoso y posibles ataques de denegación de servicio, detectando tráfico excesivo hacia determinados servicios y controlando que no hay interfaces de red en modo promiscuo (síntoma de sniffers o capturadores de paquetes).

Ejemplos de herramientas IDS

Algunos ejemplos de herramientas IDS serían:

- Logcheck (verificación de logs),
- TripWire (estado del sistema mediante sumas md5 aplicadas a los archivos),
- AIDE (una version libre de TripWire),
- Snort (IDS de verificación de estado de una red completa).

7. Protección mediante filtrado (*TCP wrappers* y cortafuegos)

Los *TCP wrappers* [Mou01] son un software que actúa de intermediario entre las peticiones del usuario de servicio y los *daemons* de los servidores que ofrecen el servicio. Muchas de las distribuciones vienen ya con los *wrappers* activados y podemos configurar los niveles de acceso. Los *wrappers* se suelen utilizar en combinación con inetd o xinetd, de manera que protejan los servicios que ofrecen.

El *wrapper* básicamente sustituye al *daemon* (demonio) del servicio por otro que hace de intermediario (llamado tcpd, normalmente en `/usr/sbin/tcpd`). Cuando este recibe una petición, verifica el usuario y origen de la misma, para determinar si la configuración del *wrapper* del servicio permite o no utilizarlo. Además, incorpora facilidades de generar registros o bien informar por correo de los posibles intentos de acceso y después ejecuta el *daemon* adecuado asignado al servicio.

Por ejemplo, supongamos la siguiente entrada en inetd:

```
finger stream tcp nowait nobody /usr/etc/in.fingerd in.fingerd
```

Esta se cambia por:

```
finger stream tcp nowait nobody /usr/sbin/tcpd in.fingerd
```

de manera que, cuando llegue una petición, será tratada por el *daemon* tcpd que se encargará de verificar el acceso (para más detalles, véase la página *man tcpd*).

Wrappers

Los *wrappers* nos permiten controlar la seguridad mediante listas de acceso a nivel de servicios.

También existe un método alternativo de *wrapper* TCP, que consiste en que la aplicación original se compile con la biblioteca de *wrappers*. Entonces la aplicación no tiene porqué estar en inetd y podremos controlarla igual que el primer caso con la configuración que comentamos a continuación.

El sistema de *wrappers* se controla desde los ficheros `/etc/hosts.deny`, donde especificamos qué servicios denegamos y a quién, por medio de opciones, como un pequeño *shell* para guardar la información del intento, y el fichero `/etc/hosts.allow`, donde solemos colocar qué servicio vamos a utilizar, seguido de la lista de a quién dejamos entrar (más adelante, en el taller, ve-

remos un pequeño ejemplo). También existen los comandos `tcpdchk`, que comprueban la configuración de los ficheros *hosts* (véase *man hosts_access* y *hosts_options*), para comprobar que son correctos; es decir, comprueban la configuración. El otro comando útil es `tcpdmatch`, al cual damos un nombre de servicio y un posible cliente (usuario y/o *host*) y nos dice qué haría el sistema ante esta situación.

7.1. Cortafuegos

Un cortafuegos (*firewall*) es un sistema o grupo de sistemas que refuerza las políticas de control de acceso entre redes. El cortafuegos puede estar implementado en software, como una aplicación especializada ejecutándose en un computador individual, o bien puede tratarse de un dispositivo especial dedicado a proteger uno o más computadores.

En general dispondremos, o bien de la aplicación de cortafuegos para proteger una máquina concreta conectada directamente a Internet (ya sea directa o por proveedor), o bien podremos colocar en nuestra red una o varias máquinas dedicadas a esta función, de modo que protejan nuestra red interna.

Técnicamente, la mejor solución es disponer de un computador con dos o más tarjetas de red que aíslen las diferentes redes (o segmentos de red) conectadas, de manera que el software de cortafuegos en la máquina (o si fuese un hardware especial) se encargue de conectar los paquetes de las redes y determinar cuáles pueden pasar o no, y a qué red.

Este tipo de cortafuegos suele combinarse con un enrutador (*router*) para enlazar los paquetes de las diferentes redes. Otra configuración típica es la de cortafuegos hacia Internet, por ejemplo con dos tarjetas de red: en una obtenemos/proporcionamos tráfico a Internet y en la otra enviamos o proporcionamos el tráfico a nuestra red interna, pudiendo así eliminar el tráfico que no va destinado a nosotros, y también controlar el tráfico que se mueve hacia Internet, por si no queremos que se tenga acceso a algunos protocolos o bien sospechamos que hay posibilidades de fugas de información por algunos ataques. Una tercera posibilidad es la máquina individual conectada con una única tarjeta de red hacia Internet, directa o bien a través de un proveedor. En este caso, sólo queremos proteger nuestra máquina de ataques de intrusos, de tráfico no deseado o de que se vea comprometida al robo de datos.

Es decir, en estos casos podemos comprobar que el cortafuegos, dependiendo de si es software o no, de si la máquina tiene una o varias tarjetas de red o de si protege a una máquina individual o a una red, puede tener configuraciones y usos diferentes.

El cortafuegos en general permite definir al usuario una serie de políticas de acceso (cuáles son las máquinas a las que se puede conectar o las que pueden recibir información y el tipo de información a recibir) por medio del control de los puertos TCP/UDP permitidos de entrada (*incoming*) o de salida (*outcoming*). Algunas herramientas de gestión de cortafuegos vienen con una serie de políticas preconfiguradas, o solo dicen si se quiere un nivel de seguridad alto, medio o bajo, o, por último, permiten personalizar las opciones totalmente (máquinas, protocolos, puertos, etc.).

Seguridad a nivel de paquetes

Los cortafuegos permiten establecer seguridad a nivel de paquetes y conexiones de comunicación.

Otra técnica a veces relacionada es la NAT (*network address translation*, traducción de direcciones de red). Esta técnica proporciona una vía para ocultar las direcciones IP usadas en la red privada y las oculta de Internet, pero mantiene el acceso desde las máquinas. Uno de los métodos típicos es el denominado *masquerading*. Si se usa *NAT masquerading*, uno o varios dispositivos en la red pueden aparecer como una única dirección IP vistos desde fuera. Esto permite conectar varios computadores a un único dispositivo de conexión externa; por ejemplo, un caso de enrutador ADSL en el hogar permite conectar varias máquinas sin necesidad de que el proveedor nos proporcione diferentes direcciones IP. Los enrutadores ADSL suelen ofrecer algún tipo de *NAT masquerading* y también posibilidades de cortafuegos. Suele ser bastante común utilizar una combinación de ambas técnicas. En este caso, entra en juego, además de la configuración de la máquina del cortafuegos (los casos vistos antes), la configuración de la red privada interna que queremos proteger.

7.2. Netfilter: iptables

El núcleo Linux (a partir de versiones 2.4.x) proporciona un subsistema de filtrado denominado Netfilter [Net], que proporciona características de filtrado de paquetes y también NAT. Este sistema permite usar diferentes interfaces de filtrado, entre las cuales la más usada se denomina iptables. El comando principal de control es `iptables`. Anteriormente, se proporcionaba otro filtrado denominado ipchains en los núcleos 2.2 [Gre] y el sistema tenía una sintaxis diferente (aunque con similitudes). En los núcleos 2.0 se utilizaba otro sistema denominado ipfwadm. Aquí (y en los ejemplos posteriores) vamos a tratar solo con Netfilter/iptables (es decir, con las posibilidades de cortafuegos en núcleos de las versiones 2.4/2.6).

Enlace de interés

Sobre Netfilter véase: http://www.netfilter.org.

La interfaz del comando `iptables` permite realizar las diferentes tareas de configuración de las reglas que afectan al sistema de filtrado, ya sea generación de registros, acciones de pre y post enrutado de paquetes, NAT y reenvío (*forwarding*) de puertos.

Elementos de iptables

iptables aporta diferentes elementos como las tablas, *chains* y las propias reglas.

El arranque del servicio se realiza con `/etc/init.d/iptables start`, si no estaba configurado ya en el *runlevel*.

El comando `iptables -L` lista las reglas activas en ese momento en cada una de las cadenas (*chains*). Si no se han configurado previamente, suelen ser por defecto aceptar todos los paquetes de las cadenas de `INPUT` (entrada), `OUTPUT` (salida) y `FORWARD` (reenvío).

El sistema de iptables tiene como nivel superior las tablas. Cada una contiene diferentes cadenas, que a su vez contienen diferentes reglas. Las tres tablas que existen son: Filter, NAT y Mangled. La primera sirve para las propias normas de filtrado, la segunda para realizar traslación de direcciones dentro de un sistema que utilice NAT y la tercera, menos usada, sirve para especificar algunas opciones de control de los paquetes y cómo gestionarlos. Concretamente, si estamos con un sistema directamente conectado a Internet, utilizaremos, en general, tan solo la tabla Filter. Si el sistema está en una red privada que tiene que pasar por un enrutador, pasarela (*gateway*) o proxy (o combinación de estos), seguramente dispondremos de un sistema de NAT o IP *masquerading*; si estamos configurando precisamente la máquina que permite acceso externo, tendremos que tocar la tabla NAT y la Filter. Si la máquina está en un sistema de red privada, pero es una de las máquinas internas, será suficiente con la tabla Filter, a no ser que sea un servidor el que haga NAT a otro segmento de red.

Si un paquete llega al sistema, en el cortafuegos implementado por iptables, se mirará primero si existen reglas en la tabla NAT, por si hay que hacer traducciones de direcciones hacia la red interna (las direcciones normalmente no son visibles hacia el exterior); después se mirarán las reglas de la tabla Filter para decidir si se van a dejar pasar los paquetes o si no son para nosotros, y si tenemos reglas `forward` para saber hacia dónde los reenviamos. Por el contrario, cuando nuestros procesos generan paquetes, las reglas `output` de la tabla Filter controlan si los dejamos salir o no, y si hubiese sistema NAT, las reglas efectuarían la traducción de direcciones de manera que se enmascarasen. En la tabla NAT suele haber dos cadenas: `prerouting` y `postrouting`. En la primera, las reglas han de decidir si hay que hacer algún enrutamiento del paquete y cuál será la dirección de destino. En el segundo, se decide finalmente si el paquete se pasa o no hacia el interior (la red privada, por ejemplo). Y también existe una cadena `output` para el tráfico que se genere localmente de salida a la red privada, ya que `prerouting` no lo controla (para más detalles, se puede examinar *man iptables*).

A continuación comentaremos algunos aspectos y ejemplos de configuración de la tabla Filter*.

***Para las otras tablas, se puede consultar la bibliografía asociada.**

La configuración típica de la tabla Filter es de una serie de reglas que especifican qué se hace dentro de una determinada cadena, como las tres anteriores (`input`, `output` o `forward`). Normalmente, se especifica:

```
iptables -A chain -j target
```

donde `chain` es `input`, `output` o `forward` y `target`, el destino que se le va a dar al paquete que se corresponda con la regla. La opción `-A` añade la regla a las existentes.

Con esta fase de añadir reglas hay que tomar precauciones, ya que el orden importa. Hay que colocar las menos restrictivas al principio, puesto que, si primero ponemos una regla que elimine los paquetes, aunque haya otra regla, esta no será tenida en cuenta. La opción `-j` permite decidir qué haremos con los paquetes, típicamente `accept` (aceptarlos), `reject` (rechazarlos) o `drop` (simplemente perderlos). Es importante la diferencia entre `reject` y `drop`. Con el primero, rechazamos el paquete y normalmente informamos al emisario de que hemos rechazado el intento de conexión (normalmente por un paquete de tipo ICMP). Con el segundo (`drop`), simplemente perdemos el paquete como si nunca hubiese existido y no enviamos ningún tipo de respuesta. Otro `target` utilizado es `log`, para enviar el paquete al sistema de registros. Normalmente, en este caso hay dos reglas, una con el `log` y otra igual con `accept`, `drop` o `reject`, para permitir enviar al registro la información de qué paquetes han sido aceptados, rechazados o perdidos. Con las opciones de generarlos en el cortafuegos hay que controlar precisamente su uso, ya que son capaces, dependiendo del ambiente de red, de generar una enorme cantidad de información en los ficheros de registro.

Al colocar la regla, también puede usarse la opción `-I` (insertar) para indicar una posición, por ejemplo:

```
iptables -I INPUT 3 -s 10.0.0.0/8 -j ACCEPT
```

que nos dice que se coloque la regla en la cadena `input` en tercera posición; y se van a aceptar paquetes (`-j`) que provengan (con fuente, o *source*, `-s`) de la subred 10.0.0.0 con máscara de red 255.0.0.0. De forma parecida, con `-D` podemos borrar o un número de regla o la regla exacta, como se especifica a continuación, borrando la primera regla de la cadena o la regla que mencionamos:

```
iptables -D INPUT 1
iptables -D INPUT -s 10.0.0.0/8 -j ACCEPT
```

También hay reglas que permiten definir una política por defecto de los paquetes (opción `-P`); se va a hacer con todos los paquetes lo mismo. Por ejemplo, se suele colocar al inicio que se pierdan todos los paquetes por defecto, y se habilitan luego los que interesan, y muchas veces también se evita que haya reenvío de paquetes si no es necesario (si no actuamos de enrutador). Esto podría ponerse:

```
iptables -P INPUT DENY
```

```
iptables -P OUTPUT REJECT
iptables -P FORWARD REJECT
```

Todo lo cual establece unas políticas por defecto que consisten en denegar la entrada de paquetes, no permitir salir y no reenviar paquetes. Ahora se podrán añadir las reglas que conciernen a los paquetes que deseemos utilizar, diciendo qué protocolos, puertos y orígenes o destinos queremos permitir o evitar. Esto puede ser difícil, ya que tenemos que conocer todos los puertos y protocolos que utilicen nuestro software o servicios. Otra táctica sería dejar solo activos aquellos servicios que sean imprescindibles y habilitar con el cortafuegos el acceso de los servicios a las máquinas deseadas.

Algunos ejemplos de estas reglas de la tabla Filter podrían ser:

```
iptables -A INPUT -s 10.0.0.0/8 -d 192.168.1.2 -j DROP
iptables -A INPUT -p tcp --dport 113 -j REJECT --reject-with tcp-reset
iptables -I INPUT -p tcp --dport 113 -s 10.0.0.0/8 -j ACCEPT
```

Donde:

1) Perdemos los paquetes que vengan de 10.x.x.x con destino a 192.168.1.2.

2) Rechazamos los paquetes tcp con destino al puerto 113, emitiendo una respuesta de tipo tcp-reset.

3) Los mismos paquetes que en 2) pero que provengan de 10.x.x.x serán aceptados.

Respecto a los nombres de protocolos y puertos, el sistema iptables usa la información facilitada por los ficheros `/etc/services` y `/etc/protocols`, pudiéndose especificar la información (de puerto y protocolo) de forma numérica o por nombre (hay que tener cuidado, en este caso, de que la información de los ficheros sea correcta y que no hayan sido modificados, por ejemplo, por un atacante).

La configuración de iptables suele establecerse mediante llamadas consecutivas al comando `iptables` con las reglas. Esto crea un estado de reglas activas que pueden consultarse con `iptables -L`. Si deseamos salvarlas para que sean permanentes, podemos hacerlo en Fedora con:

```
/etc/init.d/iptables save
```

y se guardan en:

```
/etc/sysconfig/iptables
```

En Debian puede hacerse, si existe el anterior *script*:

```
/etc/init.d/iptables save nombre-reglas
```

Si no se da soporte directo para guardar o recuperar reglas, siempre se puede con los comandos `iptables-save` y `iptables-restore` rederigirlas a ficheros, para guardar o recuperar la configuración del cortafuegos.

En el primer caso hay que tener cuidado de que antes exista el directorio `/var/lib/iptables`, que es donde se guardan los ficheros; `nombre-reglas` será un fichero en el directorio.

Con `/etc/init.d/iptables load` podemos cargar las reglas (en Debian hay que dar el nombre del fichero de reglas o bien usar `iptables-restore`), aunque Debian soporta unos nombres por defecto de ficheros, que son `active` para las reglas normales (las que se utilizarán en un `start` del servicio) e `inactive` para las que quedarán cuando se desactive el servicio (se haga un `stop`). Otra aproximación comunmente usada es la de colocar las reglas en un fichero *script* con las llamadas iptables que se necesiten y llamarlas, por ejemplo, colocándolas en el *runlevel* necesario o con enlace hacia el *script* en `/etc/init.d`.

7.3. Paquetes para gestión de cortafuegos en las distribuciones

Respecto a herramientas de configuración más o menos automatizadas por cortafuegos, existen varias posibilidades, pero hay que tener en cuenta que no suelen ofrecer las mismas prestaciones que la configuración manual de iptables (que en la mayoría de casos sería el proceso recomendado). Algunas herramientas son:

- Lokkit: en la distribución Fedora/Red Hat, muy básico y solo permite elegir al usuario el nivel de seguridad que desea (alto, medio o bajo). Después enseña los servicios que se ven afectados y podemos dejar pasar o no al servicio cambiando la configuración por defecto. El mecanismo utilizado por debajo es iptables. Puede verse la configuración final de reglas que realiza `/etc/sysconfig/iptables` que, a su vez, es leído por el servicio iptables, que se carga en arranque o por parada o arranque mediante `/etc/init.d/iptables` con las opciones `start` o `stop`. En Debian también es posible instalarlo, pero deja la configuración de las reglas en `/etc/defaults/lokkit-l` y un *script* en `/etc/init.d/lokkit-l`. También existe una versión gráfica llamada gnome-lokkit.

- Bastille [Proa]: programa de seguridad bastante completo y didáctico, ya que nos explica paso a paso diferentes recomendaciones de seguridad y

si las queremos aplicar, así como la configuración del cortafuegos (el programa es interactivo). Funciona en varias distribuciones, tanto en Fedora como en Debian.

- fwbuilder: una herramienta que permite construir las reglas del cortafuegos de forma gráfica. Se puede usar en varios sistemas operativos (GNU/Linux tanto Fedora como Debian, OpenBSD, MacOS), con diferentes tipos de cortafuegos (iptables incluido).

- firestarter: una herramienta gráfica (Gnome) para la creación del cortafuegos. Es muy completa y prácticamente maneja todas las posibilidades de iptables, pero asimismo dispone de asistentes que facilitan la construcción intuitiva del cortafuegos. Dispone, además, de un monitor en tiempo real, para detectar las intrusiones.

Normalmente, cada uno de estos paquetes utiliza un sistema de reglas que guarda en algún fichero propio de configuración, y que suele arrancar como servicio o como ejecución de *script* en el *runlevel* por defecto. Hay que tener especial cuidado con la consistencia de las reglas iptables del cortafuegos, ya que muchas de estas utilidades no soportan sincronización bidireccional de las reglas: o bien se actualiza el cortafuegos siempre mediante la herramienta de línea o gráfica o bien con el comando `iptables`, pero si se realizan cambios de ambas maneras, estos cambios pueden no ser consistentes o puede perderse la configuración del cortafuegos. Se recomiendan copias de seguridad periódicas de la configuración del cortafuegos (en especial si este tiene cierto nivel de complejidad).

7.4. Consideraciones

Niveles de seguridad

Nunca se debe confiar en un único mecanismo o sistema de seguridad. Hay que establecer la seguridad del sistema a diferentes niveles.

Aunque dispongamos de cortafuegos bien configurados, hay que tener presente que no son una medida de seguridad absoluta, ya que hay ataques complejos que pueden saltarse el control o falsear datos que creen confusión. Además, las necesidades de conectividad modernas obligan a veces a crear software que permita el *bypass* (paso a través) de los cortafuegos:

- Tecnologías como IPP, protocolo de impresión utilizado por CUPS, o el WebDAV, protocolo de autoría y actualización de sitios web, permiten pasar (o es necesario que pasen) las configuraciones de los cortafuegos.

- A menudo se utiliza (por ejemplo, los anteriores protocolos y otros) una técnica denominada *tunneling*, que básicamente encapsula protocolos no permitidos sobre la base de otros que sí que lo están; por ejemplo, si un cortafuegos permite solo paso de tráfico http (puerto 80 por defecto), es posible escribir un cliente y un servidor (cada uno a un lado diferente del cortafuegos) que hablen cualquier protocolo conocido por ambos, pero

que en la red es transformado en un protocolo http estándar, con lo cual, el tráfico puede cruzar el cortafuegos. El `tunneling` por ssh también es utilizado para establecer diferentes tipos de túneles, para conexiones, usando los protocolos y puertos de ssh.

- Los códigos móviles por web (ActiveX, Java y JavaScript) cruzan los cortafuegos (via web) y, por lo tanto, es difícil proteger los sistemas si estos son vulnerables a los ataques contra agujeros descubiertos.

Así, aunque los cortafuegos son una muy buena solución a la mayor parte de amenazas a la seguridad, siempre pueden tener vulnerabilidades y dejar pasar tráfico que se considere válido, pero que incluya otras fuentes posibles de ataques o vulnerabilidades.

En seguridad nunca hay que considerar (y confiar en) una única solución y esperar que nos proteja absolutamente; hay que examinar los diversos problemas, plantear soluciones que los detecten a tiempo y políticas de prevención que nos curen en salud, por lo que pueda pasar.

8. Herramientas de seguridad

Algunas de las herramientas de seguridad pueden considerarse también herramientas de ataque. Por lo tanto, se recomienda probar estas herramientas sobre máquinas de nuestra red local o privada; nunca hacerlo con IP de terceros, ya que estos podrían tomarlo como intrusiones y pedirnos responsabilidades, pedírselas a nuestro ISP o avisar a las autoridades competentes para que nos investiguen o cierren nuestro acceso.

A continuación, comentamos brevemente algunas herramientas, así como algunos usos que se les puede dar:

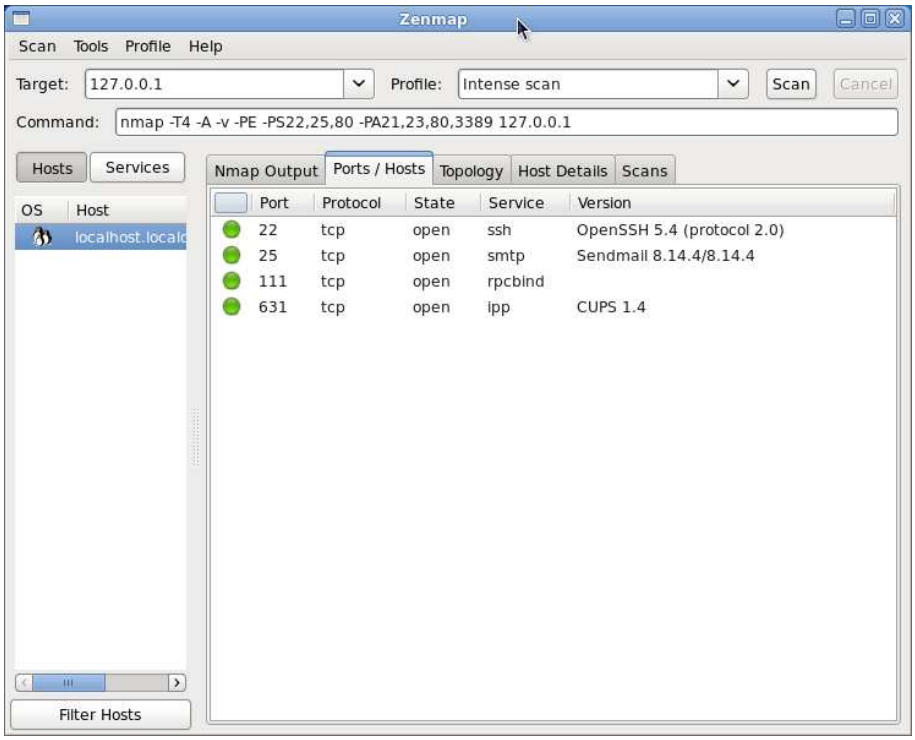
1) TripWire: esta herramienta mantiene una base de datos de sumas de comprobación de los ficheros importantes del sistema. Puede servir como sistema IDS preventivo. Nos sirve para realizar una foto del sistema, poder comparar después cualquier modificación introducida y comprobar que no haya sido corrompido por un atacante. El objetivo aquí es proteger los archivos de la propia máquina, evitar que se produzcan cambios como, por ejemplo, los que podría haber provocado un *rootkit*. Así, cuando volvamos a ejecutar la herramienta, podemos comprobar los cambios respecto a la ejecución anterior. Hay que elegir un subconjunto de archivos que sean importantes en el sistema, o fuentes de posibles ataques. Hay otras herramientas libres de funcionamiento equivalente, entre las cuales, una posibilidad es AIDE.

2) Nmap [Insb]: es una herramienta de escaneo de puertos para redes (figura 2). Puede escanear desde máquinas individuales a segmentos de red. Permite diversos tipos de escaneo de puertos, dependiendo de las protecciones que tenga el sistema. También tiene técnicas que permiten determinar el sistema operativo que usan las máquinas remotas. Puede emplear diferentes paquetes de TCP y UDP para probar las conexiones. Existen algunas interfaces gráficas para KDE y Gnome, y alguna con bastantes posibilidades como zenmap.

3) Wireshark (antes llamado Ethereal): es un analizador de protocolos y captura el tráfico de la red (actúa como husmeador). Permite visualizar el tráfico capturado, ver estadísticas y datos de los paquetes individuales y agruparlos, ya sea por origen, destino, puertos o protocolo. Puede incluso reconstruir el tráfico de una sesión entera de un protocolo TCP.

4) Snort [Sno]: es un sistema IDS que permite realizar análisis de tráfico en tiempo real y guardar registros de los mensajes. Permite realizar análisis de los

Figura 2. Interfaz gráfica para Nmap durante un análisis de servicios locales



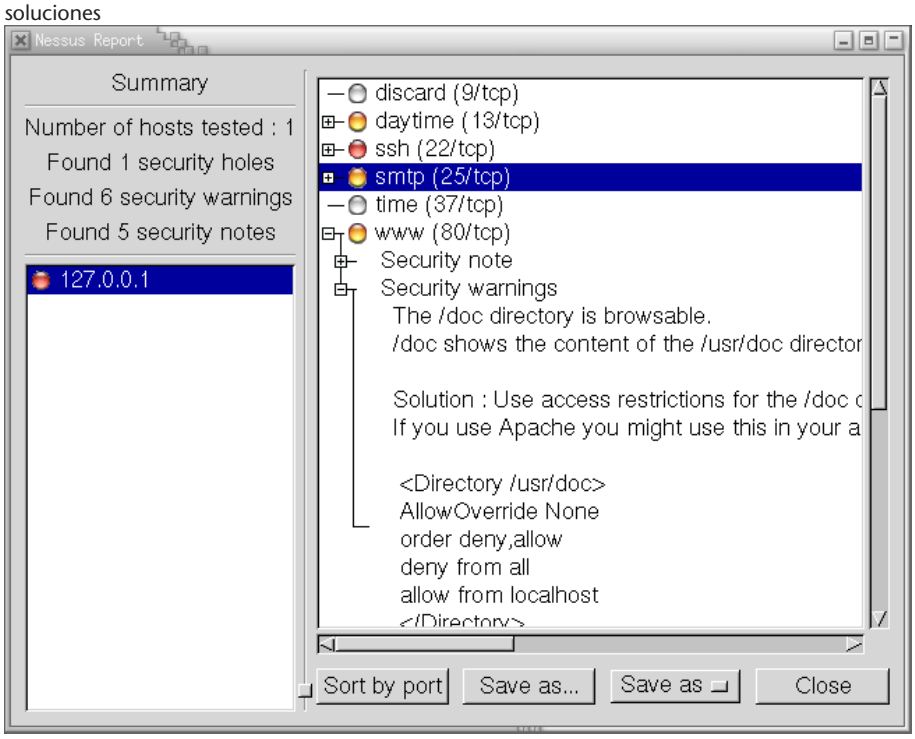
protocolos, búsquedas por patrones (protocolo, origen, destino, etc.). Puede usarse para detectar diversos tipos de ataques. Básicamente, analiza el tráfico de la red para detectar patrones que puedan corresponder a un ataque. El sistema utiliza una serie de reglas para anotar la situación (`log`), producir un aviso (`alert`) o descartar la información (`drop`).

5) Nessus [Nes]: es un detector de vulnerabilidades conocidas, que prueba diferentes técnicas de intrusión y asesora sobre cómo mejorar la seguridad para las detectadas. Es un programa modular que incluye más de 11.000 conectores (*plug-ins*) para los diferentes análisis. Utiliza una arquitectura cliente/servidor, con un cliente gráfico que muestra los resultados y el servidor que realiza las diferentes comprobaciones sobre las máquinas. Tiene capacidad para examinar redes enteras y genera los resultados en forma de informes que pueden exportarse a diferentes formatos (por ejemplo, HTML). Hasta el año 2005 Nessus 2 era una herramienta libre, pero la compañía decidió convertirla en propietaria en la versión Nessus 3. Todavía en GNU/Linux se puede encontrar y seguir utilizando Nessus 2 (figura 3), que se mantiene con licencia GPL y una serie de conectores que se van actualizando. Nessus 3 como herramienta propietaria para GNU/Linux es más potente, es ampliamente utilizada y es una de las herramientas más populares de seguridad. Normalmente se mantiene libre de coste una versión con los conectores menos actualizados que la versión de pago. Tambien ha surgido un *fork* libre a partir de Nessus 2, denominado OpenVas, que está disponible prácticamente para la mayoría de distribuciones GNU/Linux.

Enlace de interés

Para saber más sobre OpenVas tool podéis visitar su página web: http://www.openvas.org.

Figura 3. Cliente de Nessus 2 que muestra el informe de vulnerabilidades y las posibles soluciones



Podemos encontrar muchas más herramientas de seguridad disponibles. Un buen lugar a examinar es http://sectools.org, donde los creadores del Nmap mantuvieron una lista de herramientas populares votadas por los usuarios; ahora la lista es un poco antigua, pero pueden encontrase herramientas útiles.

9. Análisis de registros

Observando los ficheros de registro (*logs*) del sistema [Ano99, Fri02], podemos hacernos una idea rápida del estado global del sistema, así como de las últimas ocurrencias de sucesos, y detectar accesos (o intentos de acceso) indebidos. Pero también cabe tener presente que los registros, si ha sucedido una intrusión real, pueden haber sido limpiados o falseados. La mayoría de archivos de registro residen en el directorio `/var/log`.

Muchos de los servicios pueden poseer registros propios, que normalmente se establecen en la configuración (mediante el correspondiente fichero de configuración). La mayoría suelen utilizar las facilidades de registro incorporadas en el sistema Syslog, mediante el demonio Syslogd. Su configuración reside en `/etc/syslog.conf`. Esta configuración suele establecerse por niveles de mensajes: existen diferentes tipos de mensajes según su importancia. Normalmente suelen aparecer niveles `debug`, `info`, `err`, `notice`, `warning`, `crit`, `alert`, `emerg`, en que más o menos esta sería la ordenación de importancia de los mensajes (de menor a mayor). Normalmente, la mayoría de los mensajes se dirigen al registro `/var/log/messages`, pero puede definirse que cada tipo se distribuya a ficheros diferentes y también se puede identificar quién los ha originado: típicamente el núcleo, el correo, *news*, el sistema de autentificación, etc.

En consecuencia, cabe examinar (y en todo caso adaptar) la configuración de Syslog para conocer en qué registros podemos encontrar o generar la información. Otro punto importante es controlar su crecimiento, ya que según los que estén activos y las operaciones (y servicios) que se realicen en el sistema, los registros pueden crecer bastante. En Debian y Fedora se controla a través de `logrotated`, un demonio que se encarga periódicamente de hacer copias, y comprimirlas, de los registros más antiguos; se puede encontrar su configuración general en `/etc/logrotate.conf`, pero algunas aplicaciones hacen configuraciones específicas que se pueden encontrar en el directorio `/etc/logrotate.d`.

Comentamos en los siguientes puntos algunos ficheros de registro que habría que tener en cuenta (quizás los más utilizados):

Ficheros de registro

Hay que tener en cuenta que en la mayoría de ataques sofisticados exitosos se falsean o borran los datos de los ficheros de registro. Incluso así, dependiendo de la técnica usada, puede detectarse información útil sobre el ataque.

1) `/var/log/messages`: es el fichero de registro por defecto del demonio Syslogd, pero habría que revisar su configuración, no se hubiera dado el caso de que estuviera cambiado a otra parte o de que hubiera varios. Este fichero contiene una amplia variedad de mensajes de orígenes diversos (arranque,

diferentes demonios, servicios o el mismo núcleo); todo lo que resulte anómalo tendría que ser verificado. En caso de que haya habido una intrusión, se deberá mirar alrededor de las fechas de la intrusión en un determinado rango de tiempo, para detectar posibles intentos previos, la metodología usada, primeros avisos del sistema, etc.

2) `/var/log/utmp`: este fichero contiene información binaria para cada usuario que está actualmente activo. Es interesante para determinar quién está dentro del sistema. El comando `who` utiliza este fichero para proporcionar esta información.

3) `/var/log/wtmp`: cada vez que un usuario entra en el sistema, sale o la máquina reinicia, se guarda una entrada en este fichero. Es un fichero binario del cual el comando `last` obtiene información en que se menciona qué usuarios entraron o salieron del sistema, cuándo y dónde se originó la conexión. Puede ser útil para buscar dónde (en qué cuentas) se originó la intrusión o detectar usos de cuentas sospechosas. También existe una variación del comando, llamada `lastb`, que lista los intentos de *login* que no pudieron validarse correctamente y se usa el fichero `/var/log/btmp` (puede ser necesario crearlo si no existe). Estos mismos fallos de autentificación también suelen enviarse al registro `auth.log`. De modo parecido, el comando `lastlog` utiliza otro fichero `/var/log/lastlog` para verificar cuál fue la última conexión de cada uno de los usuarios.

4) `/var/log/secure`: suelen utilizarse en Fedora para enviar los mensajes de los *tcp wrappers* (o los cortafuegos). Cada vez que se establece una conexión a un servicio de inetd, o bien para xinetd (con su propia seguridad), se añade un mensaje de registro a este fichero. Pueden buscarse intentos de intrusos de servicios que no suelen usarse, o bien máquinas no familiares que se intentan conectar.

En el sistema de registros, otra cosa que habría que asegurar es que el directorio de registros `/var/log` solo pueda ser escrito por el usuario root (o *deamons* asociados a servicios). De otro modo, cualquier atacante podría falsificar la información de los registros. Aun así, si el atacante consigue acceso a root, puede, y suele, borrar las pista de sus accesos.

10. Taller: análisis de la seguridad mediante herramientas

Realizaremos a continuación algunos de los procesos descritos sobre un sistema Debian, para configurar mejor su seguridad.

Primero examinaremos qué ofrece nuestra máquina a la red. Para ello, utilizaremos la herramienta nmap como escaneador de puertos. Con el comando (desde root):

```
nmap -sTU -O localhost
```

obtenemos:

```
root@maquina:# nmap -sUT -O localhost
starting nmap 5.21 ( nmap.org ) 11:31 CEST
Interesting ports on localhost (127.0.0.1):

(The 3079 ports scanned but not shown below are in state: closed)
Port  State Service
9/tcp   open discard
9/udp   open discard
13/tcp  open daytime
22/tcp  open ssh
25/tcp  open smtp
37/tcp  open time
37/udp  open time
80/tcp  open http
111/tcp open sunrpc
111/udp open sunrpc
113/tcp open auth
631/tcp open ipp
728/udp open unknown
731/udp open netviewdm3
734/tcp open unknown

Remote operating system guess: Linux kernel 2.6.X
Uptime 2.011 days
Nmap run completed 1 IP address (1 host up) scanned in 9.404 seconds
```

Podemos observar que nos ha detectado un gran número de servicios abiertos (dependiendo de la máquina podría haber más: telnet, ftp, finger, etc.), tanto en protocolos tcp como udp. Algunos servicios como discard, daytime o time pueden ser útiles en alguna ocasión, pero normalmente no tendrían que estar abiertos a red, ya que se consideran inseguros. SMTP es el servicio de reenvío y enrutamiento del correo electrónico; si actuamos como *host* o servidor de correo, tendría que estar activo, pero si solo leemos y escribimos correo mediante cuentas POP3 o IMAP, no tiene porqué estarlo.

Otra manera de detectar servicios activos sería mediante la búsqueda de puertos activos a la escucha; esto puede hacerse con el comando `netstat -lut`.

El comando `nmap` también puede aplicarse con el nombre DNS o IP de la máquina; así vemos lo que se vería desde el exterior (con localhost vemos lo que puede ver la propia máquina) o, mejor incluso, podríamos utilizar una máquina de una red externa (por ejemplo, un PC cualquiera conectado a Internet) para examinar lo que verían de nuestra máquina desde el exterior. En los primeros casos no estaremos cortando las conexiones por el cortafuegos si este existiese, solo desde el exterior podremos comprobar si este efectivamente cierra los puertos esperados.

Vamos ahora a `/etc/inetd.conf` para desactivar estos servicios (si han aparecido o no en el escaneo previo, dependerá de la distribución GNU/Linux y de la configuración previa de estos servicios). Buscamos líneas como:

```
discard stream tcp nowait root internal
smtp stream tcp nowait mail /usr/sbin/exim exim -bs
```

y les colocamos un # al principio (solo para aquellos servicios que queramos desactivar y sepamos qué hacen realmente –consúltese las páginas man– o se recomiende su desactivación). Otro caso de desactivación especialmente recomendada sería el de los servicios de ftp, telnet, finger, etc. y usar `ssh` para sustituirlos.

Ahora tenemos que reiniciar inetd para que vuelva a leer la configuración que hemos cambiado: `/etc/init.d/inetd restart`.

Volvemos a nmap:

```
22/tcp  open ssh
80/tcp  open http
111/tcp open sunrpc
111/udp open sunrpc
113/tcp open auth
631/tcp open ipp
728/udp open unknown
734/tcp open unknown
```

De lo que nos queda, tenemos el servicio ssh, que queremos dejar activo, y el servidor de web, que lo pararemos de momento:

```
/etc/init.d/apache2 stop
```

ipp es el servicio de impresión asociado a CUPS. En administración local vimos que había una interfaz web de CUPS que se conectaba al puerto 631. Si

queremos tener una idea de a qué se dedica un puerto determinado, podemos mirar en `/etc/services`:

```
root@maquina:# grep 631 /etc/services
ipp 631/tcp # Internet Printing Protocol
ipp 631/udp # Internet Printing Protocol
```

Si no estamos actuando como servidor de impresión hacia el exterior, tenemos que ir a la configuración de CUPS y eliminar esa prestación (por ejemplo, colocando un *listen* 127.0.0.1:631, para que sólo escuche la máquina local) o limitar el acceso a las máquinas permitidas.

Nos aparecen también algunos puertos como desconocidos, en este caso el 728 y 734; esto indica que nmap no ha podido determinar qué servicio está asociado al puerto. Vamos a intentar comprobarlo directamente. Para ello, ejecutamos sobre el sistema el comando `netstat`, que ofrece diferentes estadísticas del sistema de red, desde paquetes enviados y recibidos, y errores, hasta lo que nos interesa, que son las conexiones abiertas y quién las usa. Intentemos buscar quién está usando los puertos desconocidos:

```
root@maquina: # netstat -anp | grep 728
udp 0 0 0.0.0.0:728 0.0.0.0:* 552/rpc.statd
```

Y si hacemos lo mismo con el 734, observamos también que quien ha abierto el puerto ha sido rpc.statd, que es un *daemon* asociado a NFS (en este caso el sistema tiene un servidor NFS). Si hacemos este mismo proceso con los puertos 111 que aparecían como `sunrpc`, observaremos que el *daemon* que hay detrás es portmap, que se usa en el sistema de llamadas RPC. El sistema de llamadas RPC (*remote procedure call*) permite utilizar el mecanismo de llamadas remotas entre dos procesos que están en diferentes máquinas. portmap es un *daemon* que se encarga de traducir las llamadas que le llegan por el puerto a los números de servicios RPC internos que se tengan, y es utilizado por diferentes servidores, como NFS, NIS y NIS+.

Los servicios RPC que se ofrecen se pueden ver con el comando `rpcinfo`:

```
root@maquina:# rpcinfo -p
programa  vers proto  puerto
100000  2 tcp 111  portmapper
100000  2 udp 111  portmapper
100024  1 udp 731  status
100024  1 tcp 734  status
391002  1 tcp 39797 sgi_fam
391002  2 tcp 39797 sgi_fam
```

donde observamos los servicios RPC con algunos de los puertos que ya se habían detectado. Otro comando que puede resultar útil es `lsof`, que, entre otras funciones, permite relacionar puertos con los servicios que los han abierto (por ejemplo: `lsof -i | grep 731`).

El *daemon* portmap es un poco crítico con la seguridad, ya que, en principio, no ofrece mecanismos de autentificación del cliente, pues se supone que se delegan en el servicio (NFS, NIS, etc.). Por lo tanto, portmap podría ser víctima de intentos de DoS/DDoS que podrían provocar errores en los servicios o hacerlos caer. Normalmente, protegeremos portmap por medio de algún tipo de *wrapper* y/o cortafuegos. Si no utilizamos, y no tenemos previsto utilizar, servicios NFS y NIS, lo mejor es desactivar completamente portmap, quitándolo del *runlevel* en que se active. También podemos pararlos momentáneamente con los *scripts* (en Debian):

```
/etc/init.d/nfs-common
/etc/init.d/nfs-kernel-server
/etc/init.d/portmap
```

dándoles el parámetro `stop` para parar los servicios RPC (en este caso NFS). En todos estos casos sólo estamos parando los servicios para la ejecución actual; si queremos desactivar realmente estos servicios, hemos de ir al nivel de ejecución concreto (*runlevel*) y desactivar los servicios (como se ha comentado en el módulo), porque si no se reiniciarán en el siguiente arranque del sistema.

A continuación, controlaremos la seguridad en base a un *wrapper* sencillo. Vamos a suponer que queremos dejar paso a través de ssh de una máquina determinada, llamémosla 1.2.3.4 (dirección IP). Vamos a cerrar portmap al exterior, ya que no tenemos NIS, y de NFS tenemos servidor pero no estamos sirviendo nada (podríamos cerrarlo, pero lo dejaremos para futuros usos). Vamos a hacer un *wrapper* (suponemos que los *TCP wrappers* están ya instalados), modificando los ficheros `hosts.deny` y `hosts.allow`. En `/etc/hosts.deny`:

```
ALL : ALL : spawn (/usr/sbin/safe_finger -l @%h \
| /usr/bin/mail -s ''%c FAILED ACCESS TO %d!!'' root) &
```

estamos denegando todos los servicios (cuidado, aquellos relacionados con inetd) (primer `all`) a todos (`all`), y la acción por tomar será averiguar quién ha pedido el servicio (solo funcionará si la maquina soporta *finger*) y en qué máquina, y vamos a enviar un correo al root informando del intento. Podríamos también escribir un fichero de registro. Ahora, en `/etc/hosts.allow`:

```
sshd: 1.2.3.4
```

habilitamos el acceso por la máquina IP 1.2.3.4 en el servidor sshd (del ssh). También podríamos colocar el acceso a portmap, solo haría falta una línea `portmap: la_ip`. Podemos colocar una lista de máquinas o bien subredes que puedan utilizar el servicio (véase `man hosts.allow`). Recordad que también tenemos los comandos `tcpdchk`, para comprobar que la configuración del *wrapper* sea correcta, y `tcpdmatch`, para simular qué pasaría con un determinado intento. Por ejemplo:

```
root@maquina:# tcpdmatch sshd 1.2.3.4

warning: sshd: no such process name in
   /etc/inetd.conf client: hostname maquina.dominio.es
client: address 1.2.3.4
server: process sshd
matched: /etc/hosts.allow line 13
access: granted
```

nos dice que sería concedido el acceso. Un detalle es que nos dice que sshd no está en el `inetd.conf` y, si lo verificamos, vemos que así es: no se activa por el servidor inetd, sino por *daemon* propio (sshd) en el *runlevel* en que estemos. Además (en Debian), este es un caso de demonio que está compilado con las bibliotecas de *wrappers* incluidas (y, por lo tanto, no necesita tcpd para funcionar). En Debian hay varios *daemons* así: ssh, portmap, in.talk, rpc.statd y rpc.mountd, entre otros. Esto permite asegurar estos *daemons* mediante *wrappers* por los ficheros *hosts* mencionados.

Otra cuestión por verificar son las conexiones actuales existentes. Con el comando `netstat -utp` podemos listar las conexiones tcp, u udp establecidas con el exterior, ya sean entrantes o salientes; así en cualquier momento podemos detectar los clientes conectados y a quién estamos conectados. Otro comando importante (de múltiples funciones) es `lsof`, que puede relacionar ficheros abiertos con procesos o conexiones por red establecidas mediante `lsof -i`, pudiéndose así detectar accesos indebidos a ficheros.

También podríamos utilizar un cortafuegos para procesos similares (o bien como mecanismo añadido). Comenzaremos viendo cómo están las reglas del cortafuegos en este momento (comando `iptables -L`):

```
root@aopcjj:# iptables -L
Chain INPUT (policy ACCEPT)
target prot opt source destination
Chain FORWARD (policy ACCEPT)
target prot opt source destination
Chain OUTPUT (policy ACCEPT)
target prot opt source destination
```

O sea, que el cortafuegos no está colocando ninguna restricción en este momento, y permite la entrada, salida y reenvío de todos los paquetes.

En este punto, podríamos añadir un cortafuegos que nos permitiese una gestión más adecuada de paquetes que recibimos y enviamos, y que sería un control previo para mejorar la seguridad. Dependiendo de nuestras necesidades, estableceríamos las reglas necesarias de modo parecido a las que comentamos en los ejemplos de cortafuegos de la unidad.

En caso de colocar activo algún cortafuegos, podemos considerar si usar este mecanismo como única garantía y quitar los *wrappers*: podría hacerse, ya que los cortafuegos (en este caso mediante iptables) ofrecen un mecanismo muy potente que nos permite seguir a un paquete por tipo, por protocolo y por lo que está haciendo en el sistema. Un buen cortafuegos podría ser más o menos suficiente, pero, por si acaso, más medidas de seguridad no vienen mal. Y en caso de que el cortafuegos no estuviese bien diseñado y dejase escapar algunos paquetes o *hosts*, el *wrapper* sería la medida, a nivel de servicio, para parar los accesos no deseados. Por poner una metáfora que se suele utilizar, si nos planteásemos nuestro sistema como la defensa de un castillo medieval, el foso y primeras murallas serían el cortafuegos, y la segunda muralla de contención, los *wrappers*.

Las siguientes medidas ya podrían venir directamente de cada servicio (web, correo, impresión, etc.), con las opciones de seguridad que ofrezca de forma adicional, ya sea mediante autentificación de sus clientes, mediante limitación de los accesos por perfiles o ACL o simplemente al ofrecer un subconjunto de funcionalidades requeridas. Veremos en diferentes módulos orientados a servicios concretos algunas de estas medidas a nivel servicio.

Resumen

En este módulo hemos examinado los conceptos básicos de seguridad aplicables a los sistemas GNU/Linux, y hemos identificado los posibles tipos de los ataques, tanto locales como a sistemas en red.

El conocimiento del proceso de los ataques nos permite tomar acciones de seguridad activa mediante herramientas de detección de intrusiones, así como de prevención de posibles situaciones problemáticas.

Sistemas como SELinux nos permiten políticas de seguridad, altamente especificadas, y nos ofrecen una amplia sintaxis de permisos, controles y prevención activa de la seguridad del sistema.

La seguridad debe examinarse tanto desde el punto de vista local al sistema, lo que incluye la seguridad física de acceso al mismo, como desde el punto de vista de los sistemas en red.

La utilización de herramientas de seguridad en las diferentes áreas de prevención, detección y actuación nos permite un control activo de seguridad, que nos puede evitar males mayores en nuestros sistemas.

También observamos las limitaciones de la seguridad de los sistemas informáticos y, en especial, las posibles sensaciones de una falsa seguridad total (difícil o imposible de obtener), que nos puede llevar a una confianza ciega, con peores resultados que no tenerla. La seguridad es un proceso activo, que necesita un seguimiento constante y participativo por parte del administrador de sistemas.

Actividades

1. Supongamos que colocamos en nuestra máquina un sitio web, por ejemplo con Apache. Nuestro sitio está pensado para diez usuarios internos, pero no controlamos este número. Más adelante nos planteamos poner en Internet este sistema, ya que creemos que puede ser útil para los clientes, y lo único que hacemos es poner el sistema con una IP pública en Internet. ¿Qué tipo de ataques podría sufrir este sistema?

2. ¿Cómo podemos detectar los ficheros con suid en nuestro sistema? ¿Qué comandos serán necesarios? ¿Y los directorios con SUID o SGID? ¿Por qué es necesario, por ejemplo, que `/usr/bin/passwd` tenga bit de SUID?

3. Los ficheros `.rhosts`, como hemos visto, son un peligro importante para la seguridad. ¿Podríamos utilizar algún método automático que comprobase su existencia periódicamente? ¿Cómo?

4. Supongamos que queremos deshabilitar un servicio del cual sabemos que tiene el *script* `/etc/init.d/servicio` que lo controla: queremos desactivarlo en todos los *runlevels* en los que se presenta. ¿Cómo encontramos los *runlevels* en los que está? (por ejemplo, buscando enlaces al *script*).

5. Examinad los servicios en activo en vuestra máquina. ¿Son todos necesarios? ¿Cómo habría que protegerlos o desactivarlos?

6. Practicad el uso de algunas de las herramientas de seguridad descritas (nmap, chkrootkit, wireshark, etc.).

7. ¿Qué reglas iptables serían necesarias para una máquina en la que solo queremos acceso por SSH desde una dirección concreta?

8. ¿Y si queremos únicamente un acceso al servidor web?

Bibliografía

[Ano99] **Anónimo**. *Maximum Linux Security: A Hacker's Guide to Protecting*.

[Aus] **CERT Australia**. *Australian CERT*. <http://www.auscert.org.au>

[Bur02] **Burgiss, H.** (2002). *Security QuickStart HOWTO for Linux*. The Linux Documentation Project.

[Cera] **CERT**. *CERT site*. <http://www.cert.org>

[Cerb] **CERT**. *CERT vulnerabilidades*. <http://www.cert.org/cert/information/sysadmin.html>

[Deb] **Debian**. *Sitio de seguridad de Debian*. <http://www.debian.org/security>

[Fbi] **Federal Bureau of Intelligence**. *Brigada del FBI para ciberdelitos*. <http://www.fbi.gov/about-us/investigate/cyber/cyber>

[Fen02] Kevin Fenzi. *Linux security HOWTO*. The Linux Documentation Project.

[Fri02] **Frisch, A.** (2002). *Essential System Administration* (3.ª ed.). O'Reilly.

[Gre] **Grennan, M.**. *Firewall and Proxy Server HOWTO*. The Linux Documentation Project.

[Hat08] **Hatch, B.** (2008). *Hacking Linux Exposed* (3.ª ed.). McGraw-Hill.

[Hatb] **Red Hat** (2003). *Red Hat 9 Security Guide*. <http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/security-guide>

[Hatc] **Red Hat** (2003). *Sitio de seguridad de Red Hat*. <http://www.redhat.com/security>

[Hatd] **Red Hat** (2003). *Utilización firmas GPG en Red Hat*. <http://www.redhat.com/docs/manuals/linux/RHL-7.3-Manual/custom-guide/ch-gnupg.html>

[Him01] **Himanen, P.** (2001). *La ética del hacker y el espíritu de la era de la información*. Destino.

[Incb] **Incidents.org**. *Vulnerabilidades Incidents*. <http://isc.incidents.org>

[Ins] **Insecure.org** (1998). *Vulnerabilidades y exploits*. <http://www.insecure.org/sploits.html>

[Insa] **Insecure.org**. *Insecure.org site*. <http://www.insecure.org>

[Insb] **Insecure.org** (2003). *Nmap*. <http://nmap.org/index.html>

[Line] **Linuxsecurity.com**. *Linux Security Reference Card*. <http://www.linuxsecurity.com/docs/QuickRefCard.pdf>

[Mor03] **Morill, D.** (2003). *Configuración de sistemas Linux*. Anaya Multimedia.

[Mou01] **Mourani, G.** (2001). *Securing and Optimizing Linux: The Ultimate Solution*. Open Network Architecture, Inc.

[Nes] **Nessus.org**. *Nessus*. <http://www.nessus.org>

[Net] **Netfilter.org.** *Proyecto Netfilter/IPtables*. <http://www.netfilter.org>

[Neu] **Neufeld, C.**. *Setting Up Your New Domain MiniHOWTO*. The Linux Documentation Project.

[Nsaa] **NSA**. *NIST site*. <http://csrc.nist.gov>

[Nsab] **NSA** (2003). *Security Enhanced Linux*. <http://www.nsa.gov/research/selinux/index.shtml>

[Pen] **Fernández-Sanguino Peña, J.** (2007). *Securing Debian Manual*. <http://www.debian.org/doc/manuals/securing-debian-howto/>

[Proa] **Bastille Project**. *Bastille*. <http://bastille-linux.sourceforge.net>

[San] **Sans**. *Top20 de vulnerabilidades*. <http://www.sans.org/top20>

[Sei] **Seifried, K.** (2002). *Securing Linux, Step by Step*. <http://seifried.org/security/os/linux/20020324-securing-linux-step-by-step.html>

[Sno] **Snort.org**. *Snort*. <http://www.snort.org>

[Usa] **The United States Department of Justice**. *División del Departamento de Justicia de Estados Unidos para el ciberdelito*. <http://www.usdoj.gov/criminal/cybercrime/>

Otras fuentes de referencia e información

[Deb] [Hatc] Los sitios de seguridad de las distribuciones.

[Pen] Imprescindible para Debian, muy buena descripción para seguir paso a paso la configuración de seguridad, [Hatb] sería equivalente para Fedora/Red Hat.

[Mou01] Excelente referencia de seguridad para Red Hat (aplicable también a Debian).

[Hat08] Libros sobre seguridad en GNU/Linux, que cubren gran número de aspectos y técnicas.

[Line] Pequeña guía (2 páginas) de seguridad.

[Sei] Guía paso a paso de identificación de los puntos clave que hay que verificar y los problemas que puedan surgir.

[Net] Proyecto Netfilter e IPTables.

[Ian] Una lista de puertos TCP/IP.

[Proa] [Sno] [Insb] [Nes] Algunas de las herramientas de seguridad más usadas.

[NSAb] Versión de Linux con miras a la seguridad, producida por la NSA. Referencia para SELinux.

[CERa][Aus][Insa][Incb] [NSAa] Sitios de organismos de seguridad.

[CERb][Ins][San] Vulnerabilidades y exploits de los diferentes sistemas operativos.

[NSAa][FBI][USA] Algunas “policías” de ciberdelitos en Estados Unidos.