2.4 SEGURIDAD EN LA RED

2.4.1 Cortafuegos bajo GNU/Linux

2.4.1.1 Introducción y componentes de un Cortafuegos

Podemos decir que un cortafuegos o *firewall* es un sistema o grupo de sistemas que hace cumplir una política de control de acceso entre dos redes. De una forma más clara, podemos definir un cortafuegos como cualquier sistema utilizado para separar, en lo que a seguridad se refiere, una máquina o subred del resto, protegiéndola así de servicios y protocolos que desde el exterior puedan suponer una amenaza a la seguridad. El cortafuegos puede ser un dispositivo de propósito específico o un software sobre un sistema operativo. En general debemos verlo como una caja con dos o más interfaces de red en la que se establecen reglas de filtrado con las que se decide si un tráfico determinado puede cursarse o no.

En este apartado introduciremos algunos conceptos básicos sobre cortafuegos, y sobre los distintos mecanismos que nos ofrece el sistema operativo GNU/Linux para implementarlos en nuestra red.

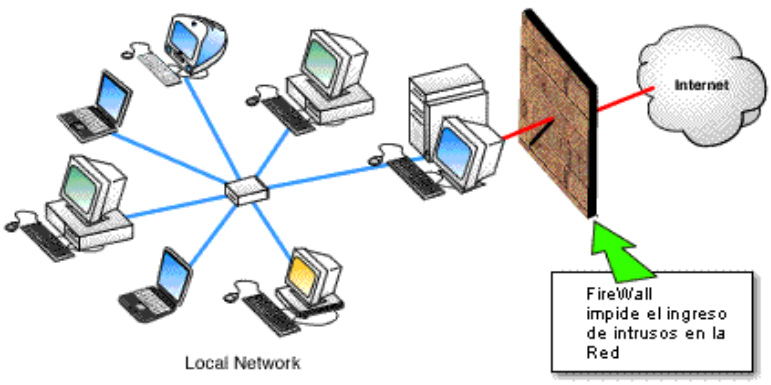
En todo cortafuegos existen tres componentes básicos para los que debemos implementar mecanismos: el filtrado de paquetes, los proxies de aplicación y la monitorización y detección de actividad sospechosa.

Filtrado de paquetes: el funcionamiento del filtrado de paquetes es muy sencillo. Básicamente consiste en analizar la cabecera de cada paquete y aceptarlo o rechazarlo según unas reglas predefinidas que hayamos establecido. A partir de las versiones 2.4 del kernel de Linux, el sistema **netfilter** y la herramienta **iptables** se encargan de realizar esta tarea (anteriormente lo hacían ipfwadm e ipchains, en los núcleos de las series 2.0 y 2.2, respectivamente). En el apartado siguiente lo comentaremos con mayor profundidad.

Veamos la siguiente tabla de reglas:

Origen	Destino	Servicio	Acción
192.168.0.0	*	80 (HTTP), 21 (FTP)	Aceptar
192.168.0.25	192.168.20.20	22 (SSH)	Aceptar
192.168.1.0	*	*	Denegar

Según esta tabla de reglas, nuestro cortafuegos permitiría ser atravesado por paquetes provenientes de la red 192.168.0.0 cuyo puerto de destino fuese el 80 o el 21, así como paquetes cuya dirección de origen fuese 192.168.0.25 y destino 192.168.20.20, haciendo uso del puerto 22 (una posible administración remota?), y denegaría cualquier paquete cuya dirección de origen fuese 192.168.1.0.



¿Qué sucedería, por ejemplo, si llegase algún paquete que no coincidiese con ninguna de estas reglas? En ese caso entraría en juego la política por defecto. En la mayoría de los casos es recomendable una política de denegación por defecto, es decir, denegar todo excepto lo que está explícitamente permitido. Aunque en muchas situaciones nos encontramos con cortafuegos que permiten todo el tráfico excepto parte que desean limitar.

Uno de los principales puntos débiles del filtrado de paquetes es que únicamente se basan en los datos de cabecera del paquete para tomar decisiones. No se analiza qué información transmite el paquete, ni si realmente se trata del tipo de transmisión que parece ser. Por tanto, no pueden determinar la validez de su contenido.

Imaginemos que en nuestra empresa únicamente disponemos de acceso a la red a través de puerto 80 (únicamente nos permiten acceder a contenido web), pero nosotros deseamos conectarnos de forma remota a nuestra casa con SSH (que analizaremos en detalle posteriormente) para ojear cómo están nuestras máquinas. Si el administrador tan sólo hubiese instalado un sistema de filtrado de paquetes, no tendríamos más que situar nuestro servicio ssh en el puerto 80, en lugar de en el 22 por defecto, y podríamos conectarnos sin ningún problema. De igual forma, una vez establecida esta conexión, como ya estamos en una máquina externa que no tiene restricciones de filtrado, podríamos conectarnos a otras máquinas y acceder a cualquier otro servicio.

Una forma de ayudar al sistema de filtrado de paquetes, y evitar muchas de estas situaciones es a través de proxies de aplicación. A continuación veremos cómo funcionan.

Proxy de Aplicación: Además del filtrado de paquetes, es habitual que los cortafuegos utilicen aplicaciones para reenviar o bloquear conexiones a servicios http, ftp, telnet, etc. A estas aplicaciones capaces de filtrar conexiones a servicios determinados se les denomina proxies, mientras que la máquina donde se ejecutan se llaman pasarela de aplicación.

Los proxies poseen una serie de ventajas de cara a incrementar nuestra seguridad. En primer lugar, permiten únicamente la utilización de servicios para los que exista un proxy. La segunda ventaja es que en el filtrado podemos basarnos en algo más que en la cabecera de los paquetes, lo que hace posible, por ejemplo, tener filtrado el servicio FTP con órdenes restringidas, de forma que nadie subiese archivos a nuestro servidor. También podemos evitar problemas como el que hablamos en el punto anterior de la conexión ssh a nuestra casa. En este caso, el proxy detectaría que aunque el tráfico circula por el puerto correspondiente al servicio web, no corresponde a una conversación en el protocolo HTTP válida, y la descartaría. Por supuesto seguiría siendo posible saltarnos esta barrera de seguridad, tunelando esta vez nuestra conexión a través de HTTP. De esta forma el proxy de aplicación pensaría que se trata de simples peticiones http estándar, y no sospecharía de nuestro tráfico. Por ejemplo, la herramienta libre httptunnel nos permitiría hacer esto con mucha comodidad.

Squid es un popular programa de software libre que permite implementar un servidor Proxy publicado bajo licencia GPL. Tiene una amplia variedad de utilidades, desde acelerar un Servidor Web, guardando en caché peticiones repetidas a DNS y otras búsquedas, Caché WEB, o la que más nos interesa para este apartado como es añadir seguridad filtrando el tráfico.

Aunque hemos visto que nuestro cortafuegos aún es vulnerable en algunos aspectos, con la puesta en marcha de los proxies de aplicación lo hemos hecho más robusto y seguro ante la gran mayoría de atacantes. Recordemos que siempre existirá alguien capaz de quebrantar la seguridad de nuestros sistemas, y que no luchamos contra él, sino contra la gran mayoría con mucho menos conocimiento.

Quizás, unas de las mayores desventajas de este mecanismo es que no existen proxies para todos los protocolos de red que conocemos, lo que hace que en muchos casos no podamos hacer uso de ellos.

Monitorización: Monitorizar la actividad de nuestro cortafuegos es algo vital para la seguridad de nuestros sistemas. Con ella obtendremos información valiosa de posibles intentos de ataque, datos de los atacantes, posibles puntos débiles en nuestros firewalls, etc.

Como veremos en el punto siguiente, iptables nos permite registrar la actividad que deseemos en cualquiera de sus reglas, de una forma intuitiva, para que posteriormente puedan ser revisadas cómodamente por el administrador.

2.4.1.2 Netfilter e Iptables

En Linux, el filtrado de paquetes está programado en el núcleo (bien como módulo o estáticamente), y se llama **netfilter**. Adicionalmente disponemos de la herramienta **iptables** para interactuar con las reglas cuanto necesitemos. Entre las principales características de netfilter, podemos destacar:

- Filtrado de Paquetes “stateless” (analiza paquetes independientes) en las versiones 4 y 6 de IP.
- Filtrado de Paquetes “statefull” (analiza conexiones establecidas) en IP v4.
- Nos permite realizar todos los tipos de traducciones de red y puertos (NAT/NATP).
- Está estructurado de una forma flexible y escalable.
- Dispone de un gran número de plugins que incrementan sus funcionalidades.
- Manejo avanzado de paquetes.

Para poder hacer uso del filtrado de paquetes, debemos previamente habilitar el soporte del núcleo, bien sea estáticamente o a través de módulos.

Ya estudiamos en los primeros temas el proceso de compilación del kernel Linux detenidamente, en esta sección sólo haremos un pequeño hincapié en ver cómo activar los modulos de netfilter necesarios. Como ya hicimos, una vez que tuviéramos las fuentes descargadas y las herramientas necesarias para llevar a cabo esta tarea, podíamos entrar en la interfaz de configuración del kernel mediante:

```
/usr/src/linux/make menuconfig
```

Desde esta interfaz podemos ir seleccionando los módulos del kernel que queramos activar o desactivar. La configuración básica para utilizar iptables es:

```
Networking --->
    [*] Networking support
        Networking options --->
         <*> Packet socket
          [*] Packet socket: mmapped IO
         <*> Unix domain sockets
          [*] TCP/IP networking
          [*] Network packet filtering (replaces ipchains) --->
                Core Netfilter Configuration --->
               <*> Netfilter Xtables support (required for ip_tables)
               <*> "limit" match support
               <*> "mac" address match support
               <*> "state" match support
         IP: Netfilter Configuration --->
        <*> Connection tracking (required for masq/NAT)
        <*> FTP protocol support
        <*> IP tables support (required for filtering/masq/NAT)
        <*> Packet filtering
```

Una vez hecho esto procederíamos a compilar e instalar el núcleo. En cualquier caso, normalmente estos módulos están activados por defecto en el kernel.

La herrramienta iptables se comunica con el módulo y le dice qué paquetes filtrar. De esta forma controlaremos el filtrado de paquetes. Iptables inserta y elimina reglas de la tabla de filtrado del núcleo. Esto quiere decir que cualquier cosa que se establezca se perderá cuando se inicie.

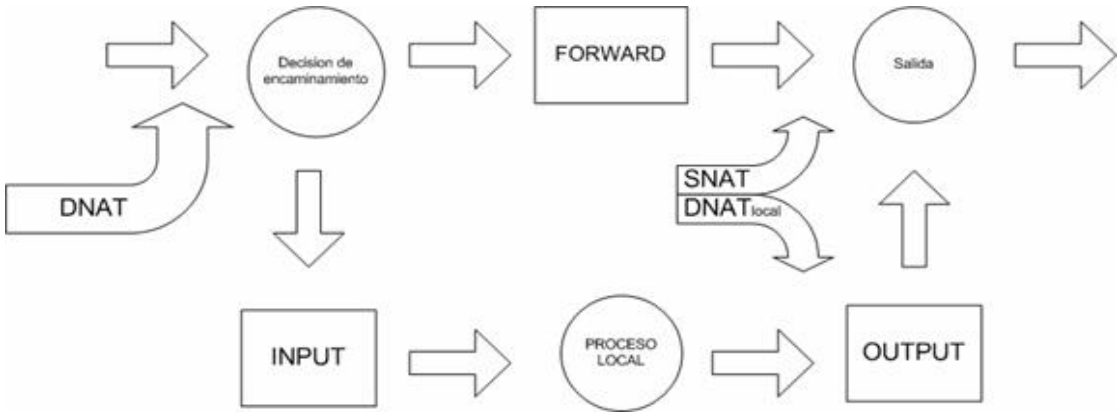
Podemos hacer “permanentes” estas reglas de dos formas:

- Haciendo uso de las herramientas *iptables-save* e *iptables-restore*
- Escribiendo un script que se lance al inicio del sistema con las reglas que se desea.

Nosotros utilizaremos este segundo método.

Cómo pasan los paquetes por los filtros

El núcleo empieza con tres listas de reglas en la tabla de filtros. Llamaremos a estas listas cadenas. Se llaman INPUT, OUTPUT y FORWARD.



Los tres rectángulos de la figura representan las tres cadenas que mencionamos arriba. Cuando un paquete alcanza un rectángulo en el diagrama, se examina esa cadena para decidir la suerte del paquete. Si la cadena dice que hay que descartar (DROP) el paquete, se destruye ahí mismo, pero si la cadena dice que hay que aceptarlo (ACCEPT), continúa su camino por el diagrama.

Como hemos visto, una cadena es una lista de reglas. Cada regla dice “si el paquete se parece a esto, entonces esto otro es lo que hay que hacer con él”. Si la regla no se ajusta al paquete se consulta la siguiente regla en la lista. Al final, si no hay más reglas por consultar, el núcleo mira la política por defecto de la cadena para decidir qué hacer. En un sistema consciente de la seguridad, esta política suele decirle al núcleo que descarte (DROP) el paquete.

Resumiendo, el proceso de la figura sería el siguiente:

- Cuando llega un paquete (digamos, por la tarjeta ethernet) el núcleo mira primero su destino. Este proceso se conoce como enrutamiento.

- Si está destinado a esa misma máquina, el paquete entra en el diagrama hacia la cadena INPUT. Si pasa de aquí, cualquier proceso que esté esperando el paquete lo recibirá.

- En caso de que nuestra máquina no sea el destinatario, si el núcleo no tiene las capacidades de reenvío activadas, o no sabe hacia dónde reenviar el paquete, lo descartará. Si está activado el reenvío, y el paquete está destinado a otra interfaz de red (de existir otra), entonces el paquete pasa directamente a la cadena FORWARD de nuestro diagrama. En ese punto, si es aceptado, saldrá de la máquina.

- Por otro lado, los paquetes que envíen programas que se ejecuten en nuestra máquina pasarán directamente por la cadena OUTPUT. Si son aceptados, continuarán su camino hacia su destino.

2.4.1.3 Configuración de Reglas de Filtrado con Iptables

Con iptables se pueden realizar muchas operaciones. A continuación vamos a detallar algunas de las operaciones que se pueden hacer sobre cadenas.

- Crear una nueva cadena (-N)
- Borrar una cadena vacía (-X)
- Cambiar la política de una cadena de registro de uso interno (-P)
- Listar las reglas de una cadena (-L)
- Con (-n) especificamos la salida de direcciones y puertos en formato numérico
- Vaciar de reglas una cadena (-F)
- Poner a cero los contadores de paquetes y bytes de todas las reglas de una cadena (-Z)
- Vaciar la tabla de información sobre NAT (-t nat -F)

```
debian:~# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Con iptables también podemos manipular las reglas de una cadena:

- Añadir una nueva regla en una cadena (-A)
- Insertar una nueva regla en alguna posición de la cadena (-I)
- Mover una regla a otra posición dentro de una cadena (-R)
- Borrar una regla de un sitio en particular de una cadena (-D)
- Borrar la primera regla que coincida con los parámetros dados en una cadena (-D)
- Especificar una acción que aplicaremos a un paquete si cumple una regla (-j). Con la opción -j DROP descartaremos paquetes y con -j ACCEPT los dejaremos pasar.

Para mayor información siempre podemos consultar la página man de iptables.

La manipulación de reglas será una de las tareas más habituales que realicemos. Lo más común es que usemos las órdenes de agregar (-A) y eliminar (-D).

Cada regla especifica un conjunto de condiciones que debe cumplir el paquete, y qué hacer si se ajusta a ella: un objetivo. Por ejemplo, podríamos querer descartar todos los paquetes ICMP que viniesen de la dirección IP 127.0.0.1.

```
ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.045 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.016 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.014 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.014 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.014/0.022/0.045/0.013 ms
```

Nuestro objetivo será por tanto DROP. La siguiente captura ilustra el proceso de la regla de iptables que deberíamos introducir:

```
iptables -A INPUT -s 127.0.0.1 -p icmp -j DROP
```

Añadimos (-A) a la cadena “INPUT”, una regla que especifica que los paquetes que vengan de 127.0.0.1 (-s 127.0.0.1) con protocolo ICMP (-p icmp) sean descartados (-j DROP).

Una vez añadida la regla probamos de nuevo a hacer el ping y observamos cómo el paquete no llega a su destino (100% packet loss).

```
ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.

--- 127.0.0.1 ping statistics ---
120 packets transmitted, 0 received, 100% packet loss, time 119024ms
```

Ahora podemos eliminar la regla de dos formas: mediante su número (sabemos que es la 1 porque es la única que hay), o repitiendo la orden, pero en lugar de utilizar “-A” (añadir), haciendo uso de “-D” (eliminar). De existir varias reglas idénticas sólo se borraría la primera.

En cualquier caso, si tuviéramos muchas reglas programadas, para saber qué número tiene asignado cada una lo haríamos con:

```
iptables -L -n --line-numbers
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    DROP       icmp --  127.0.0.1           0.0.0.0/0

Chain FORWARD (policy ACCEPT)
num  target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination
```

Finalmente, la borramos con:

```
iptables -D INPUT -s 127.0.0.1 -p icmp -j DROP
```

Como hemos comentado, iptables nos permite añadir nuevas cadenas definidas por nosotros. Pues bien, éstas pueden sernos de mucha utilidad a la hora de especificar un objetivo, ya que iptables nos permite establecer como objetivo de una regla otra cadena. Una vez que se hayan recorrido todas las reglas de la nueva cadena objetivo sin que exista coincidencia con el paquete en cuestión, se retorna de nuevo el control a la cadena que la invocó.

Además de los objetivos DROP (denegar) y ACCEPT (aceptar), existen otros dos que vienen de serie con el sistema: RETURN y QUEUE.

RETURN tiene el mismo efecto que si hubiésemos llegado al final de la cadena como comentamos en el párrafo anterior (si la regla estaba en una cadena de las que hay por defecto, se ejecutaría la política de la cadena).

QUEUE es un objetivo especial que pasa el paquete a una cola destinada al procesamiento en espacio de usuario. Es decir, con el objetivo QUEUE podemos enviar los paquetes que filtremos a otra aplicación que designemos para que sean procesados (un detector de intrusos, quizá).

Especificaciones de filtrado:

Hasta ahora hemos usado “-p” para especificar el protocolo, y “-s” para la dirección de origen. Pero podemos usar otras opciones para especificar las características de los paquetes.

- **Direcciones IP de origen y destino:** Las direcciones IP de origen (-s, --source, --src) y destino (-d, --destination, --dst) se pueden especificar de varias maneras. La más común sería utilizar el nombre completo, tal como “localhost” o “www.debian.org”. Otra forma sería especificando directamente las direcciones IP de estas máquinas.
  Además de especificar direcciones de hosts concretos, podemos describir grupos de direcciones, como 192.85.38.0/24 o 195.85.38.0/255.255.255.0. Ambas especifican cualquier dirección IP entre 192.85.38.0 y 195.85.38.255.

- **Especificar una inversión:** podemos negar en cualquier momento cualquier indicador, precediéndolo con el símbolo “!”. Por ejemplo, -s ! localhost coincidiría con cualquier paquete que no venga de localhost.

- **Especificar el protocolo:** se puede especificar el protocolo con el especificador “-p” o “--protocol”. El protocolo puede ser un número (si se conocen los valores numéricos) o un nombre en el caso especial de UTP, UDP, ICMP. No importa si lo ponemos en mayúsculas o minúsculas. El nombre del protocolo puede ir precedido de un símbolo de admiración “!”, para invertirlo, de manera que “-p ! TCP” especifica paquetes que no sean TCP.

- **Especificar la interfaz:** Las opciones “-i” (--in-interface) o “-o” (--out-interface) especifican el nombre de la interfaz de red con la que coincidir. Una interfaz es el dispositivo físico por el que entra (-i) o sale (-o) un paquete. Con el comando ifconfig podemos obtener una lista de las interfaces de red que tenemos activas en nuestro sistema.
  Los paquetes que pasan por la cadena INPUT no tienen interfaz de salida, con lo que nunca se activará una regla de esta cadena que use “-o”. De forma similar, los paquetes que atraviesan OUTPUT no tienen interfaz de entrada, de manera que ninguna regla que use “-i” en esta cadena funcionará.

Extensiones a iptables: nuevas coincidencias y objetivos:

Iptables es extensible, lo que quiere decir que se pueden extender tanto el núcleo como la herramienta iptables para proporcionar nuevas características.

- **Extensiones TCP**: Las extensiones TCP se cargan de forma automática si se especifca “-p tcp”. Esto nos proporciona las siguientes opciones:

  --tcp-flags: Nos permite filtrar dependiendo de que ciertos indicadores TCP estén activos o no. A veces puede sernos útil permitir conexiones TCP en una dirección, pero no en la otra. Por ejemplo, podemos querer permitir conexiones a un servidor www externo, pero no desde ese mismo servidor. La solución del inexperto sería bloquear los paquetes TCP que salgan del servidor. Desafortunadamente, las conexiones TCP precisan que los paquetes fluyan en ambas direcciones para poder funcionar. La solución sería bloquear sólo los paquetes que pretendan establecer una conexión. Éstos reciben el nombre de paquetes SYN (SYN activo y ACK y FIN inactivos). Rechazando estos paquetes podemos detener intentos de conexión en su inicio.

  --source-port, --sport: nos permite especificar un puerto o rango de puertos. Éstos pueden estar representados por su nombre tal como aparecen en /etc/services o de forma numérica. Al igual que con las otras opciones, podemos utilizar la negación.

  --destination-port, --dport: igual que en el caso anterior, salvo que en éste el puerto por el que filtramos es el de destino en lugar del puerto origen.

- **Extensiones UDP**: nos proporcionan las opciones --source-port y --destination-port que vimos en las extensiones TCP. Al igual que en el caso anterior, se cargan de forma automática al invocar “-p udp”.

- **Extensiones ICMP**: Nos proporciona una opción nueva, que se carga al especificar “-p icmp”.
  --icmp-type: nos permite elegir el tipo de paquete icmp de que se trata. Podemos ver una lista de los tipos icmp disponibles utilizando “-p icmp –help”.

- **Otras extensiones de coincidencia:** además de las extensiones que hemos visto existen otras, aunque para poder hacer uso de ellas tendremos que haber cargado su módulo correspondiente. De ellas podemos destacar las siguientes:

  mac: este módulo debe ser especificado de forma explícita con “-m mac” o “--match mac”. Se utiliza para coincidencias en las direcciones físicas (mac) de los paquetes entrantes, y por tanto sólo son útiles para los paquetes que pasan por las cadenas PREROUTING e INPUT. Proporciona sólo una opción:

  --mac-source: especifica la dirección por la que queremos filtrar, en formato hexadecimal separando los elementos por “:”.
  Además de las coincidencias con direcciones MAC, existen otras interesantes como “limit”, útil para restringir la tasa de coincidencias de una regla, “owner”, útil para restringir paquetes creados localmente, etc. Para más información podemos visitar www.netfilter.org.

state: este modo nos permite realizar filtrado *statefull;* o lo que es lo mismo, permite establecer reglas basadas en conexiones en vez de paquetes. Se utiliza con “-m state” o “--state ...”.
Los estados posibles son:

- NEW: el paquete no corresponde a ninguna conexión vista anteriormente.
- ESTABLISHED: el paquete corresponde a una conexión memorizada.
- RELATED: el paquete está iniciando una conexión pero relacionado con otra conexión existente.
- INVALID: el paquete no ha podido ser clasificado en alguno de los estado anteriores.

Nuevos objetivos: además de los objetivos existentes por defecto, podemos hacer uso de otros extendidos, siempre que hayamos compilado los módulos necesarios. Los más importantes son:

LOG: este módulo nos permite registrar coincidencias en cada una de las reglas en la que lo establezcamos. Nos proporciona estas opciones adicionales:

--log-level: seguido de un número o nombre de nivel. Estos niveles coinciden con los utilizados por el demonio syslogd.

--log-prefix: seguido de una cadena de hasta treinta caracteres, que corresponden a un texto que se añadirá ante cada registro, permitiendo que sean identificados unívocamente, y fácilmente reconocibles.

REJECT: este módulo tiene el mismo efecto que DROP, excepto que al remitente se le envía un mensaje de error ICMP “port unreacheable”.

2.4.1.4 Protegiendo nuestra máquina

Hasta ahora hemos analizado la sintaxis de iptables para establecer reglas de filtrado, ya es hora de aplicarlas en nuestro dispositivo Debian. Nuestro firewall no tendrá más que un conjunto de estas reglas en las que se examina el origen y el destino de los paquetes del protocolo tcp/ip.
Un ejemplo en pseudo-lenguaje del conjunto de reglas que se podrían establecer en nuestra máquina sería el siguiente:

```
Política por defecto: ACEPTAR
Todo lo que venga desde la red local al firewall ACEPTAR
Todo lo que venga desde la ip de mi casa al puerto 22 ACEPTAR
Todo lo que venga de la red y vaya al exterior ENMASCARAR
Todo lo que venga del exterior al puerto tcp/udp 1 al 1024 DENEGAR
Todo lo que venga del exterior al puerto tcp 3389 DENEGAR
```

En definitiva lo que se hace es:

- Habilita el acceso a puertos de administración a determinadas IP privilegiadas.
- Enmascara el tráfico de la red local hacia el exterior (NAT, una petición de un pc a la LAN sale al exterior con ip pública), para poder salir a Internet.
- Deniega el acceso desde el exterior a puertos de administración y a todo lo que esté entre 1 y 1024.

Hay dos maneras de implementar un firewall:

1) Política por defecto ACEPTAR: en principio todo lo que entra y sale por el firewall se acepta y sólo se denegará lo que se diga explícitamente.
2) Política por defecto DENEGAR: todo está denegado, y sólo se permitirá pasar por el firewall aquello que se permita explícitamente.

Como es obvio imaginar, la primera política facilita mucho la gestión del firewall, ya que simplemente nos tenemos que preocupar de proteger aquellos puertos o direcciones que sabemos que nos interesa; el resto no importa tanto y se deja pasar. Por ejemplo, si queremos proteger una máquina Linux, podemos hacer un *netstat -ln* (o *netstat -an*) o *nmap localhost* , saber qué puertos están abiertos, poner reglas para proteger esos puertos y ya está. ¿Para qué vamos a proteger un puerto que realmente nunca se va a abrir?.

El único problema que podemos tener es que no controlemos qué es lo que está abierto, o que en un momento dado se instale un software nuevo que abra un puerto determinado, o que no sepamos que determinados paquetes ICMP son peligrosos. Si la política por defecto es ACEPTAR y no se protege explícitamente, nos la estamos jugando un poco.

En cambio, si la política por defecto es DENEGAR, a no ser que lo permitamos explícitamente, el firewall se convierte en un auténtico MURO infranqueable. El problema es que es mucho más difícil preparar un firewall así, y hay que tener muy claro cómo funciona el sistema (sea iptables o el que sea) y qué es lo que se tiene que abrir sin caer en la tentación de empezar a meter reglas super-permisivas. Esta configuración de firewall es la recomendada.

El orden en el que se ponen las reglas del firewall es determinante. Normalmente cuando hay que decidir qué se hace con un paquete se va comparando cada regla del firewall hasta que se encuentra una que le afecta (match), y se hace lo que dicte esta regla (aceptar o denegar); después de eso no se miran más reglas para ese paquete.

Pasemos ahora a crear reglas para nuestro dispositivo Debian. Lo único que tenemos que hacer es crear un script de shell en el que se van aplicando las reglas. Los scripts de iptables suelen tener este aspecto:

```
Borrado de reglas aplicadas anteriormente (flush)
Aplicación de políticas por defecto para INPUT, OUTPUT, FORWARD
Listado de reglas de Iptables
```

Veamos un posible script para nuestro dispositivo:

```
#!/bin/sh
## SCRIPT de IPTABLES
## Ejemplo de script para proteger nuestro dispositivo Debian
## Francisco J. Rosado Recio

echo -n Aplicando Reglas de Firewall...

## FLUSH de reglas
iptables -F          # Limpia todas las cadenas
iptables -X          # Borra cadenas vacias
iptables -Z          # Pone a 0 los contadores de paquetes y bytes de todas las reglas de todas las cadenas
iptables -t nat -F   # Limpia la tabla de NAT

## Establecemos la política por defecto. Cerramos Todo. Dejamos entrar y salir lo solicitado.
iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT
itpables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -m state --state ESTABLISHED, RELATED -j ACCEPT
iptables -P FORWARD DROP

# Operar en localhost sin limitaciones
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# Permitimos ping
iptables -A INPUT -p ICMP -j ACCEPT

# Abrimos SSH desde la red local hacia nuestro dispositivo
iptables -A INPUT -s 192.168.0.0/24 -p TCP --dport 22 -j ACCEPT

# Permitimos la comunicación con el servidor DNS
iptables -A INPUT -p UDP --dport 53 -j ACCEPT
iptables -A INPUT -p TCP --dport 53 -j ACCEPT
```

```
# Permitimos la comunicación con el servidor DHCP
iptables -A INPUT -i eth0 -p UDP --dport 67:68 --sport 67:68 -j ACCEPT

# Permitimos la comunicación de nuestro punto de acceso con FreeRADIUS
iptables -A INPUT -i eth0 -p UDP -s 192.168.1.50 --dport 1812 -j ACCEPT

echo " OK . Verifique lo que se aplica con: iptables -L -n "
# Fin del script
```

El fichero se puede llamar por ejemplo firewall.sh, debemos darle permisos de ejecución:

```
chmod +x firewall.sh
```

ó

```
chmod 750 firewall.sh
```

Ahora aplicamos el script:

```
sh firewall.sh
```

Y podemos comprobar que las reglas se aplican correctamente:

```
debian:/home# iptables -L -n
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0
ACCEPT     tcp  --  192.168.0.0/24       0.0.0.0/0           tcp dpt:22
ACCEPT     udp  --  0.0.0.0/0            0.0.0.0/0           udp dpt:53
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:53
ACCEPT     udp  --  0.0.0.0/0            0.0.0.0/0           udp spts:67:68 dpts:67:68
ACCEPT     udp  --  0.0.0.0/0            0.0.0.0/0           udp dpt:1812

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
```

Sólo falta hacer que las reglas se carguen al inicio, es decir, cuando la máquina arranca. Debian tiene diferentes *runlevels* o niveles de ejecución que permiten poner el sistema en un estado definido. Por ejemplo "2" es el runlevel por omisión de Debian, "0" apaga el sistema y "1" es el modo monousuario que permite ejecutar trabajos de mantenimiento. Al entrar en un runlevel se ejecutan diferentes scripts de inicio del directorio */etc/init.d*. Por tanto es necesario copiar nuestro script firewall.sh en */etc/init.d/.* Tecleamos:

```
cp firewall.sh /etc/init.d/
```

A cada runlevel hay asociado una serie de enlaces a estos scripts de inicio que son los que se ejecutan cuando la máquina entra en dicho runlevel. Estos enlaces se encuentran en los directorios */etc/rc.?.d* donde "?" es el correspondiente número del runlevel. Los enlaces tienen nombres especiales. Si empiezan por "S" inician, si empiezan por "K" terminan el correspondiente programa. Después de la letra sigue un número de dos dígitos que determina el orden de ejecución y terminan en el nombre del script. Por ejemplo, los enlaces a nuestro script tendrá el nombre: *S20firewall.sh.* Para simplificar el trabajo de crear estos enlaces en los directorios de los runlevels existe un programa en Debian llamado *update-rc.d*.

Veámoslo:

```
update-rc.d firewall.sh defaults
```

Con la opción *defaults* se ha creado enlace para arrancar nuestro *firewall.sh* en los niveles de ejecución 2345 y pararlo en los niveles 016. Por omisión tendrá el código de secuencia 20.

Listo. Con esto tenemos asegurado nuestro dispositivo, hemos cerrado todos los puertos entrantes excepto el de ssh (22), dns (53), dhcp (67,68) y radius (1812). Además, nuestro dispositivo puede acceder a la red externa sin problemas. A medida que progresemos en nuestro proyecto e instalemos más aplicaciones y servicios sobre Debian tendremos que ir abriendo en el firewall los puertos necesarios.

Sin embargo, durante la realización de pruebas de DHCP detectamos que si no aplicábamos las reglas que habilitan el DHCP y el DNS los servicios seguían funcionando aceptando peticiones de los ordenadores de la red local.

Esto es debido a que los servicios *dhcpd* y *pdnsd* utilizan *raw sockets.* En los lenguajes de programación se disponen de funciones más o menos sencillas que realizan conexiones entre un cliente y un servidor, utilizando sockets TCP, UDP o incluso ICMP. Cuando se utilizan estos sockets, al llegar un paquete se le elimina la cabecera y se reciben sólo los datos. Sin embargo, los raw sockets permiten acceso a las cabeceras de los paquetes salientes y entrantes, lo cual permite cosas como enviar paquetes ICMP, ARP, o cualquier otro modificando cualquier flag o dato de la cabecera.

Los servicios que utilizan raw sockets reciben copias de los paquetes que se enrutan por nuestra máquina, y estas copias no pasan a través de las reglas de iptables, por eso no se filtran en nuestro cortafuegos.

Podemos ver los servicios que utilizan raw sockets en nuestra máquina con el siguiente comando:

```
netstat -lpw
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
raw        0      0 *:icmp                  *:*                     7           2540/pdnsd
raw        0      0 *:icmp                  *:*                     7           2163/dhcpd
```

Por tanto, no es necesario incluir en nuestro *firewall.sh* las siguientes reglas, ya que no tendrán efecto:

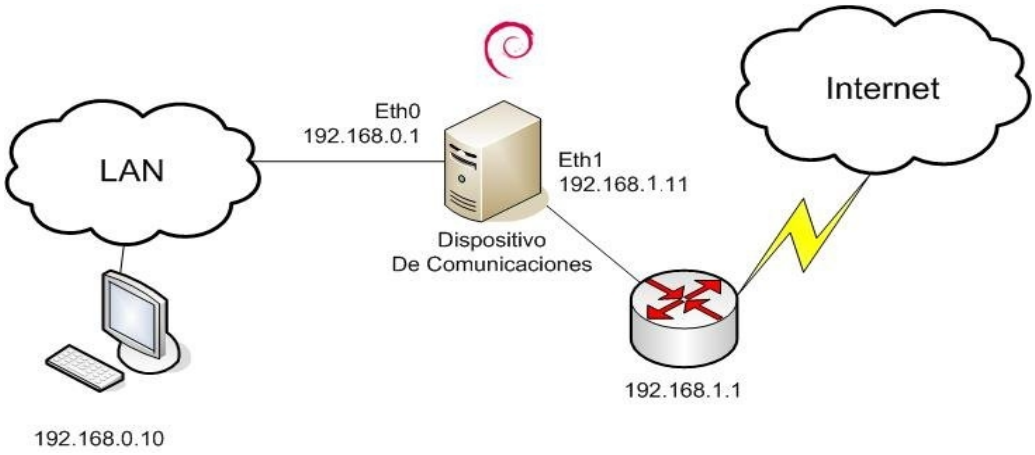
```
# Permitimos la comunicación con el servidor DNS
iptables -A INPUT -p UDP --dport 53 -j ACCEPT
iptables -A INPUT -p TCP --dport 53 -j ACCEPT

# Permitimos la comunicación con el servidor DHCP
iptables -A INPUT -i eth0 -p UDP --dport 67:68 --sport 67:68 -j ACCEPT
```

Y si realmente nos interesara eliminar estos servicios, lo haríamos en sus archivos de configuración acotando en qué interfaces escuchan peticiones y en cuáles no.

2.4.1.5 Configuración de Firewall de una LAN con salida a Internet

En el punto anterior aseguramos con el cortafuegos nuestro dispositivo de manera individual. En este apartado, además, modificaremos el script de las reglas de iptables para que nos proporcionen seguridad en el siguiente esquema de comunicaciones:



Que es el esquema que mantenemos presente durante este proyecto en el cual nuestro dispositivo desempeña la función de enrutador.

¿Qué hace falta configurar con iptables? Podríamos pensar que una regla que haga NAT hacia afuera (enmascaramiento iptables) como vimos en el capítulo anterior, en ese caso, los paquetes que partan desde la estación de trabajo hacia Internet harían dos veces NAT: una vez en el firewall Debian y otra vez en el router-módem. Entre el router y el firewall lo normal es que haya una red privada (192.168.1.1 en nuestro caso), aunque dependiendo de las necesidades en alguna situación pudiera ser que los dos tuvieran IP pública.

Ya justificamos al final del capítulo anterior que en este caso no resulta necesario realizar NAT en nuestro dispositivo Debian, por consiguiente sólo haremos routing de paquetes.

El router-modem se supone que hace un NAT completo hacia dentro (todos los puertos se redireccionan hacia nuestro dispositivo Debian), o sea que desde el exterior no se llega al router sino que de forma transparente los paquetes “chocan” contra el firewall Debian. Lo normal en este tipo de firewalls es poner la política por defecto de FORWARD en denegar (DROP).

Veamos un script para nuestro dispositivo en este caso:

```
#!/bin/sh
## SCRIPT de IPTABLES
## Ejemplo de script para firewall entre red-local e internet
## Francisco J. Rosado Recio

echo -n Aplicando Reglas de Firewall...

## FLUSH de reglas
iptables -F          # Limpia todas las cadenas
iptables -X          # Borra cadenas vacias
iptables -Z          # Pone a 0 los contadores de paquetes y bytes de todas las reglas de todas las cadenas
iptables -t nat -F   # Limpia la tabla de NAT

## Establecemos la política por defecto. Cerramos Todo. Dejamos entrar y salir lo solicitado.
iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT
itpables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -m state --state ESTABLISHED, RELATED -j ACCEPT
iptables -P FORWARD DROP
iptables -A FORWARD -m state --state ESTABLISHED, RELATED -j ACCEPT

# Enmascaramiento de la red local [Si necesitáramos hace NAT descomentaríamos esta linea]
# iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o eth1 -j MASQUERADE
```

```
# Operar en localhost sin limitaciones
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# Permitimos ping
iptables -A INPUT -p ICMP -j ACCEPT

# Aceptamos que los equipos clientes sólo puedan realizar consultas WEB
iptables -A FORWARD -s 192.168.0.0/24 -i eth0 -p tcp --dport 80 -j ACCEPT
# Aceptamos que vayan por https
iptables -A FORWARD -s 192.168.0.0/24 -i eth0 -p tcp --dport 443 -j ACCEPT

# Abrimos SSH desde la red local hacia nuestro dispositivo
iptables -A INPUT -s 192.168.0.0/24 -p TCP --dport 22 -j ACCEPT

# Permitimos la comunicación de nuestro punto de acceso con FreeRADIUS
iptables -A INPUT -i eth0 -p UDP -s 192.168.1.50 --dport 1812 -j ACCEPT

echo " OK . Verifique lo que se aplica con: iptables -L -n "
# Fin del script
```

Con este script estamos permitiendo a los equipos que estén en la red 192.168.0.0 el acceso a Internet. Por otro lado, también estamos restringiendo qué tipo de accesos van a tener: en principio se rechazan todos los servicios excepto el puerto 80 (http) y el 443 (https); de esta manera los usuarios de la red podrían solamente navegar por Internet, denegando el acceso a otras aplicaciones como Kazaa, edonkey, msn, ftp .. etc. Esta configuración es muy restrictiva, pero la más segura.

Imaginemos otro caso en el que sólamente queremos que acceda por SSH a nuestra máquina Debian el administrador de sistemas, que está sentando en un equipo de la red 192.168.0.0. Sin embargo, la dirección IP de su equipo no es siempre la misma pues la obtiene por DHCP; nos interesaría en este caso, habilitar el acceso (realizar filtros con iptables) en función de la dirección MAC del paquete.

Eso lo realizamos con la opción *-m mac –mac-source*. Para implementarlo modificamos en el script anterior la línea que permite el acceso a SSH por esta otra:

```
iptables -A INPUT -m mac --mac-source  00:11:D8:42:DE:86 -s 192.168.0.0/24 -p TCP --dport 22 -j ACCEPT
```

Donde 00:11:D8:42:DE:86 es la dirección MAC de la interfaz de red del ordenador del administrador de sistemas.

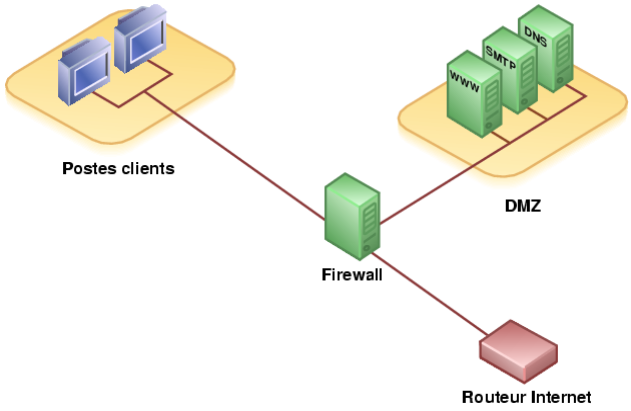
2.4.1.6 Zona Desmilitarizada (DMZ)

Ya tenemos montada la red con nuestro poderoso Firewall, pero supongamos que necesitamos exponer algún servidor a Internet (como por ejemplo un servidor WEB, un servidor de correo, etc...); en este caso se deben aceptar cualquier conexión a ellos. Para permitir estas conexiones simplemente redireccionaríamos los puertos correspondientes a las distintas máquinas con los distintos servicios (haríamos Destination NAT).

Sin embargo, esta redirección es bastante insegura. ¿Qué ocurriría si un atacante llega a controlar uno de esos servidores que tiene acceso desde el exterior y está situado dentro de nuestra red local? Pues que el firewall no serviría de gran cosa. Un atacante, que se hiciera con el control de unos de esos servidores dentro de la LAN, podría tener acceso por completo al resto de ordenadores de la red.

Lo que se recomienda hacer en esta situación es colocar esos servidores en un lugar aparte de la red, el que denominamos **DMZ** o zona desmilitarizada.

Una DMZ es una red local (una subred) que se ubica entre la red interna de una organización y una red externa, generalmente Internet. El objetivo de una DMZ es que las conexiones (sólo las necesarias) desde la red interna y la externa a la DMZ estén permitidas, mientras que las conexiones desde la DMZ sólo se permitan a la red externa –los equipos (hosts) en la DMZ no pueden conectar con la red interna.



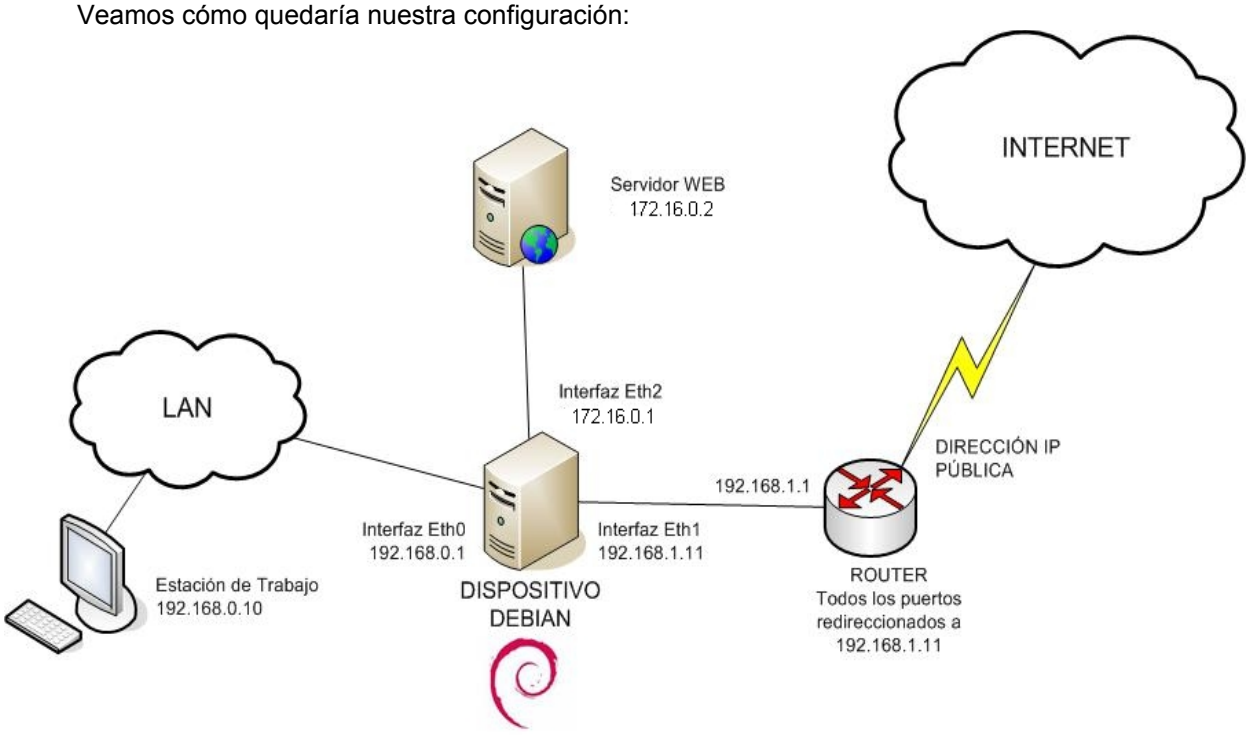
Esto permite que los equipos (hosts) de la DMZ's puedan dar servicios a la red externa a la vez que protegemos la red interna en el caso de que intrusos comprometan la seguridad de los equipos situados en la zona desmilitarizada.

Para cualquiera de la red externa que quisiera conectarse ilegalmente a la red interna, la zona desmilitarizada se convierte en un callejón sin salida. La DMZ se usa habitualmente para ubicar servidores que son necesarios que sean accedidos desde fuera, como servidores de e-mail, Web y Dns.

Las conexiones que se realizan desde la red externa hacia la DMZ se controlan generalmente utilizando port address translation (PAT). Una DMZ se crea a menudo a través de las opciones de configuración del cortafuegos, donde cada red se conecta a un puerto distinto de éste.

Para realizar una zona desmilitarizada con nuestro dispositivo Debian necesitamos una tercera tarjeta de red.

Veamos cómo quedaría nuestra configuración:



Donde le hemos dado a la DMZ la númeración de subred 172.16.0.0/24. En este tipo de Firewall hay que permitir:

- Acceso público al puerto tcp/80 del servidor activado en la DMZ.
- Acceso del servidor de la DMZ a Internet.
- Bloquear acceso de la DMZ a la red local.
- Acceso de la red local a la DMZ.
- Acceso de la red local a Internet.

¿Qué tipo de reglas son las que hay que usar para filtrar el tráfico entre la DMZ y la LAN? Sólo pueden ser las FORWARD, ya que estamos filtrando entre distintas redes, no son paquetes destinados al propio firewall.

Habilitamos en nuestra máquina Debian la nueva interfaz de red, para ello editamos el archivo */etc/network/interfaces* y añadimos:

```
auto eth2
iface eth2 inet static
        address 172.16.0.1
        netmask 255.255.255.0
        network 172.16.0.0
        broadcast 172.16.0.255
```

Comprobamos la tabla de enrutamiento :

```
debian:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     *               255.255.255.0   U     0      0        0 eth1
192.168.0.0     *               255.255.255.0   U     0      0        0 eth0
172.16.0.0      *               255.255.255.0   U     0      0        0 eth2
default         router          0.0.0.0         UG    0      0        0 eth1
```

No olvidemos también crear una ruta estática en nuestro router-módem WAN para la red 172.16.0.0:

```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     *               255.255.255.0   U     0      0        0 br0
192.168.0.0     192.168.1.11    255.255.255.0   UG    0      0        0 br0
172.16.0.0      192.168.1.11    255.255.255.0   UG    0      0        0 br0
default         84.79.128.1     0.0.0.0         UG    0      0        0 nas0
```

El script de iptables para este esquema:

```
#!/bin/sh
## SCRIPT de IPTABLES
## Ejemplo de script para firewall entre red-local e internet con DMZ
## Francisco J. Rosado Recio

echo -n Aplicando Reglas de Firewall...

## FLUSH de reglas
iptables -F             # Limpia todas las cadenas
iptables -X             # Borra cadenas vacias
iptables -Z             # Pone a 0 los contadores de paquetes y bytes de todas las reglas de todas las cadenas
iptables -t nat -F      # Limpia la tabla de NAT
```

```
## Establecemos la política por defecto. Cerramos Todo. Dejamos entrar y salir lo solicitado.
iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT
itpables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -m state --state ESTABLISHED, RELATED -j ACCEPT
iptables -P FORWARD DROP
iptables -A FORWARD -m state --state ESTABLISHED, RELATED -j ACCEPT

# Enmascaramiento de la red local y la DMZ [Si necesitáramos hace NAT descomentaríamos estas líneas]
# iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o eth1 -j MASQUERADE
# iptables -t nat -A POSTROUTING -s 172.16.0.0/24 -o eth1 -j MASQUERADE

# Todo lo que venga por el exterior y vaya al puerto 80 lo redirigimos a nuestro servidor WEB
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80 -j DNAT --to 172.16.0.2:80

# Operar en localhost sin limitaciones
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# Permitimos ping
iptables -A INPUT -p ICMP -j ACCEPT

# Aceptamos que los equipos clientes sólo puedan realizar consultas WEB (También para acceso a DMZ)
iptables -A FORWARD -s 192.168.0.0/24 -i eth0 -p tcp --dport 80 -j ACCEPT
# Aceptamos que vayan por https
iptables -A FORWARD -s 192.168.0.0/24 -i eth0 -p tcp --dport 443 -j ACCEPT

# Aceptamos que nuestro servidor pueda realizar consultas WEB
iptables -A FORWARD -s 172.16.0.0/24 -i eth2 -p tcp --dport 80 -j ACCEPT

# Abrimos SSH a la dirección mac del ordenador del administrador de sistemas
iptables -A INPUT -m mac --mac-source  00:11:D8:42:DE:86 -s 192.168.0.0/24 -p TCP –dport 22 -j
ACCEPT

# Permitimos la comunicación de nuestro punto de acceso con FreeRADIUS
iptables -A INPUT -i eth0 -p UDP -s 192.168.1.50 --dport 1812 -j ACCEPT

echo “ OK . Verifique lo que se aplica con: iptables -L -n “
# Fin del script
```

Comprobamos que el script ha funcionado perfectamente: La estación de trabajo de la LAN puede acceder y consultar páginas del servidor WEB. Sin embargo, el servidor WEB no puede hacer ni ping a la estación de trabajo, pero sí puede acceder a Internet. Por otro lado, un equipo desde Internet puede acceder al puerto 80 del servidor WEB. Hemos cumplido nuestro objetivo, nuestra máquina Debian se ha convertido en un auténtico muro infranqueable.

2.4.1.7 Bloqueando ataques básicos con Iptables

Sin embargo, a pesar de que nuestro dispositivo nos parezca infranqueable a primera vista, puede tener agujeros de seguridad y no estar preparado contra determinados ataques. En este apartado analizaremos algunos de los ataques de red más comunes que podríamos tener en nuestra máquina Linux y veremos cómo podemos evitarlos utilizando tanto iptables como el propio kernel Linux.

Ping de la muerte o ping flood: son tipos de ataques consistentes en mandar numerosos paquetes ICMP muy pesados con el fin de colapsar el sistema atacado. Para evitar este tipo de ataques convendría desactivar las respuestas de ping de nuestro dispositivo en la inferfaz externa. Por tanto, cambiamos la regla de nuestro script que teníamos por:

```
# Permitimos ping sólo desde la red local
iptables -A INPUT -i eth0 -p ICMP -j ACCEPT
```

Con esto sólo podremos hacer ping a nuestro dispositivo desde la red interna, para poder comprobar el estado del mismo.

Paquetes malformados: estos ataques pueden saturar las conexiones o causar que elementos de red u otros dispositivos conectados a la misma dejen de responder. Una regla de ejemplo de iptables para filtrar este tipo de paquetes es:

```
iptables -A FORWARD -p tcp --tcp-flags ALL NONE -j DROP
```

Donde se ha utilizado filtrado por banderas TCP. La sintaxis de este tipo de filtrado es:

--tcp-flags mascara-tcp-flags tcp-flag-activos

Donde máscara-tcp-flags y tcp-flag-activos son una lista separada por comas de los posibles valores que puede tomar el campo FLAG de un paquete TCP:

SYN, ACK, FIN, RST, URG, PSH, ALL, NONE

El comportamiento es el siguiente: se evalúan todas las banderas de *máscara-tcp-flags*, si *tcp-flags-activos* están activas, el resto de las banderas deben estar desactivadas.

En la siguiente tabla presentamos una lista de combinación de banderas TCP que se corresponden a paquetes erróneos, cualquier paquete que tuviera una combinación inválida de banderas de esta lista podría ser debido a un comportamiento malintencionado y debería ser desechado.

```
-- tcp-flags ACK,FIN FIN
-- tcp-flags ACK,PSH PSH
-- tcp-flags ACK,URG URG
-- tcp-flags FIN,RST FIN,RST
-- tcp-flags SYN,FIN SYN,FIN
-- tcp-flags SYN,RST SYN,RST
-- tcp-flags ALL ALL
-- tcp-flags ALL NONE
-- tcp-flags ALL FIN,PSH,URG
-- tcp-flags ALL SYN,FIN,PSH,URG
-- tcp-flags ALL SYN,RST,ACK,FIN,URG
```

Así que aplicaremos en nuestro firewall las siguientes reglas:

```
#Filtrado de paquetes malformados
iptables -A FORWARD -p tcp --tcp-flags ACK,FIN FIN -j DROP
iptables -A FORWARD -p tcp --tcp-flags ACK,PSH PSH -j DROP
iptables -A FORWARD -p tcp --tcp-flags ACK,URG URG -j DROP
iptables -A FORWARD -p tcp --tcp-flags FIN,RST FIN,RST -j DROP
iptables -A FORWARD -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP
iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN,RST -j DROP
iptables -A FORWARD -p tcp --tcp-flags ALL ALL -j DROP
iptables -A FORWARD -p tcp --tcp-flags ALL NONE -j DROP
iptables -A FORWARD -p tcp --tcp-flags ALL FIN,PSH,URG -j DROP
iptables -A FORWARD -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -j DROP
iptables -A FORWARD -p tcp --tcp-flags ALL SYN,RST,ACK,FIN,URG -j DROP
```

Paquetes fragmentados: son ataques parecidos a los anteriores consistentes en mandar paquetes fragmentados de tal manera que produzcan confusiones en el reensamblado en algunos sistemas operativos que presenten vulnerabilidades. Los dos ataques más conocidos relacionados con este tipo de vulnerabilidad son Teardrop y Land. En principio, los posibles ataques de denegación del servicio basados en la fragmentación de paquetes son combatidos mediante la aplicación de parches que cubren las vulnerabilidades en el momento de producirse. No obstante, con la opción -f de iptables podemos aplicar

reglas para tratar estos paquetes. La regla se cumple con el segundo fragmento y los sucesivos de un paquete fragmentado.

Con la siguiente línea desechamos todos los paquetes fragmentados:

```
#Filtrado de paquetes fragmentados
iptables -A FORWARD -f -j DROP
```

SYN-Flood: El ataque SYN Flooding se aprovecha de una vulnerabilidad en la forma en que se crean las conexiones TCP. Dichas conexiones se establecen en tres pasos (TCP Three Way Handshake):

- Desde el emisor se manda un paquete con el flag TCP SYN activado.
- El receptor responde con otro paquete, esta vez con los flags TCP SYN y ACK activos.
- Finalmente, el emisor envía otro paquete con el flag TCP ACK activado.

Durante el ataque, dicha secuencia no se llega a completar. El cliente envía el paquete SYN pero no llega nunca a responder con el paquete ACK, ocasionando que la pila TCP/IP quede a la espera de respuesta durante cierta cantidad de tiempo. Creando un número adecuado de peticiones sin respuesta se llegará a un punto en que en el servidor atacado ya no admita más conexiones y quede a la espera de forma indefinida, completamente inutilizado. Para la consecución del ataque se utilizan los llamados "inundadores SYNs", desde los cuales se configuran y establecen los puertos a atacar y las direcciones desde las cuales se ataca, siendo estas últimas ficticias y aleatorias, para evitar la detección.

Veamos un ejemplo de configuración de iptables para filtrar los paquetes SYN:

```
#Protección contra ataques SYN-Flood
iptables -N syn-flood
iptables -A FORWARD -i eth1 -p tcp --syn -j syn-flood
iptables -A syn-flood -m limit --limit 100/second --limit-burst 150 -j RETURN
iptables -A syn-flood -j DROP
```

Con la opción -N creamos una nueva cadena personalizada llamada *syn-flood.* En la siguiente línea decimos que todos los paquetes que entren por la interfaz eth1 y deban pasar por la cadena FORWARD salten a nuestra cadena específica syn-flood. En ella utilizamos una regla que utiliza el módulo *limit*, esta regla será válida ejecutando la acción RETURN hasta que el límite especificado se alcance. RETURN hace que iptables deje de seguir mirando reglas en la cadena en la que se encuentra (aquí: syn-flood). Los parámetros de limit indican:

--limit rate: tasa máxima de veces que se ejecute esta regla (ejemplo: 100 veces en un segundo)

--limit-burst number: máximo de paquetes que coincidan en un determinado rango de tiempo.

Es decir, con estas reglas se limitan el número de paquetes SYN que se reciben en un corto espacio de tiempo detectando ataques de SYN-Flood. Cuando se alcanza el límite se ejecuta la siguiente regla y se descartan paquetes para no colapsar el sistema.

UDP Flooding: la forma de llevar a cabo esta denegación de servicio consiste en explotar de forma eficiente la forma de trabajo que tiene el protocolo UDP. Mediante el envío de paquetes de forma masiva a dos o más máqinas (el protocolo UDP establece una relación directa entre máquinas) se consigue la saturación de la red en la que estén operando dichas máquinas y la caída de los recursos de los ordenadores atacados. Como el protocolo UDP no tiene mecanismos de protección frente a una posible congestión, pasado un período de tiempo, dicho protocolo se adueñará de todo el tráfico pues tiene preferencia sobre TCP.

Al igual que con los anteriores ataques, podemos prevenir el UDP Flooding escribiendo reglas en el cortafuegos que eviten el tráfico no deseado. En nuestro caso con la acción por defecto de DROP filtraremos por completo todo el tráfico UDP y permitiremos la entrada sólo a los servicios en ejecución.

Spoofing: en términos de seguridad de redes hace referencia al uso de técnicas de suplantación de identidad generalmente con usos maliciosos o de investigación. Existen diferentes tipos de spoofing dependiendo de la tecnología a la que nos refiramos, como el IP spoofing (quizás el más conocido), ARP spoofing, DNS spoofing, Web Spoofing o email spoofing, aunque en general se puede englobar dentro de spoofing cualquier tecnología de red susceptible de sufrir suplantaciones de identidad.

Como vimos, la cabecera de un paquete IP tiene un campo con la dirección origen de la máquina que lo envió. Un atacante que realice IP spoofing puede modificar esta dirección origen y parecer que está enviando paquetes desde una máquina distinta. La máquina que recibe los paquetes modificados responderá a la dirección falsificada.

Esta técnica se utiliza en ataques en los que el atacante no necesita recibir ninguna respuesta de la máquina objetivo, como los ataques de denegación de servicio (DoS). En DoS, el objetivo es inundar a la víctima enviando cantidades muy grandes de tráfico mientras que el atacante se desentiende de la respuesta de esos paquetes utilizando una dirección de origen falsificada. Además, esta dirección de origen va cambiando de manera aleatoria para dificultar la defensa de la víctima filtrando la dirección de origen.

El método de IP spoofing también puede ser utilizado por un atacante para engañar sistemas de autenticación basados en la dirección IP. Este tipo de ataques es efectivo cuando en una red existen relaciones de confianza entre máquinas, es decir, determinados servicios son accesibles dependiendo de la dirección IP que origina la petición.

Para defendernos de los ataques de IP spoofing utilizamos el filtrado de paquetes de la siguiente manera: se bloquean en nuestro cortafuegos aquellos paquetes que vienen del exterior de nuestra red local que tengan una dirección IP de origen perteneciente a un rango de direcciones utilizado en nuestra red local. Esto evita que un atacante desde fuera de la red se haga pasar por una máquina interna.
Además, también se bloquean los paquetes que salen desde la red local con una dirección IP que no pertenezca al rango de direcciones que estemos utilizando. Esto evita que un atacante de la red interna pueda realizar ataques de IP spoofing contra máquinas externas.

Vamos a ver si las reglas actuales de nuestro cortafuegos cumplen ya estas restricciones. En primer lugar analizamos qué reglas hay especificadas para los paquetes salientes

```
# Aceptamos que los equipos clientes sólo puedan realizar consultas WEB (También para acceso a DMZ)
iptables -A FORWARD -s 192.168.0.0/24 -i eth0 -p tcp --dport 80 -j ACCEPT
# Aceptamos que vayan por https
iptables -A FORWARD -s 192.168.0.0/24 -i eth0 -p tcp --dport 443 -j ACCEPT

# Aceptamos que nuestro servidor pueda realizar consultas WEB
iptables -A FORWARD -s 172.16.0.0/24 -i eth2 -p tcp --dport 80 -j ACCEPT
```

Observamos que en todas se especifica la red IP origen, por lo que ninguna dirección que no pertenezca a ese rango podrá acceder al exterior.

Por otro lado, la regla para los paquetes entrantes es la siguiente:

```
# Todo lo que venga por el exterior y vaya al puerto 80 lo redirigimos a nuestro servidor WEB
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80 -j DNAT --to 172.16.0.2:80
```

Sólo dejamos pasar peticiones WEB desde fuera haciendo *destination nat* hacia nuestro servidor web ubicado en la DMZ. Sin embargo, no especificamos desde qué direcciones de origen se pueden realizar dichos accesos. Convendría pues, introducir una regla en nuestro script (¡antes que la anterior!) que filtre aquellas direcciones de origen que pertenezcan a los rangos de direccionamiento IP de nuestras redes internas. Sería tal que así:

```
# Protección contra ataques de SPOOFING
iptables -A FORWARD -i eth1 -p all -s 192.168.0.0/24 -j DROP
iptables -A FORWARD -i eth1 -p all -s 172.16.0.0/24 -j DROP
iptables -A FORWARD -i eth1 -p all -s 127.0.0.0/8 -j DROP
```

Por último, existe una protección contra ataques de spoofing en el mismo kernel. Para activarlo, tenemos que editar el fichero */etc/network/options* o el */etc/sysctl.conf* si estamos trabajando en Debian Etch, de la misma manera que vimos en la parte de enrutamiento cuando activamos el *forwarding*. El parámetro es:

net.ipv4.conf.all.rp_filter = X

Y toma los valores:

0 – No realiza comprobaciones.
1 – Rechaza suplantaciones evidentes.
2 – Comprobación exhaustiva.

No obstante, en nuestro dispositivo ya estaba esta opción activada en 1 por defecto.

Además de ésta, hay otras variables del kernel que podemos cambiar para mejorar la seguridad. Algunas de ellas son:

tcp_syncookies: Habilita SYN Cookies que es una forma de protección contra el ataque SYN flood.
icmp_echo_ignore_all: Deniega por defecto todos los ICMP echo (pings).
icmp_echo_ignore_broadcasts. Igual que la anterior para broadcasts.
icmp_ignore_bogus_error_responses: Deniega respuestas extrañas de ICMP.

Para automatizarlas en nuestro script insertaremos las líneas:

```
# Configuración de opciones del kernel
/bin/echo “1” > /proc/sys/net/ipv4/tcp_syncookies
/bin/echo “1” > /proc/sys/net/ipv4/conf/all/rp_filter
/bin/echo “1” > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
/bin/echo “1” > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses
```

2.4.1.8 Herramientas para depurar el funcionamiento del Firewall

A continuación veremos algunos programas útiles para detectar posibles problemas en la configuración de nuestro firewall:

Tcpdump: es una herramienta en línea de comandos cuya utilidad principal es analizar el tráfico que circula por la red. Permite al usuario capturar y mostrar a tiempo real los paquetes transmitidos y recibidos en la red a la cual el ordenador está conectado. El usuario puede aplicar varios filtros para que sea más depurada la salida. Un filtro es una expresión que va detrás de las opciones y que nos permite seleccionar los paquetes que estamos buscando. En ausencia de ésta, el tcpdump volcará todo el tráfico que vea el adaptador de red seleccionado.

Tcpdump se instala mediante:

```
apt-get install tcpdump
```

Tcpdump se utiliza frecuentemente para depurar aplicaciones que utilizan la red para comunicar y para capturar y leer datos enviados por otros usuarios u ordenadores. Algunos protocolos como *telnet* y *HTTP* no cifran los datos que envían en la red. Un usuario que tiene el control de un router a través del cual circula tráfico no cifrado puede usar tcdump para lograr contraseñas u otras informaciones.

Su uso tiene la siguiente sintaxis:

tcpdump <opciones> <filtros>

Algunas opciones que podemos utilizar en la ejecución de tcpdump son:

-i <interfaz>: Para escuchar en una interfaz concreta. Si no especificamos nada escuchará por defecto en eth0.
-n: cuando estamos leyendo la red, puede que no nos interese que tcpdump intente resolver los nombres de las máquinas (pueden que no estén dadas de alta en el DNS, por motivos de seguridad, etc).
-s len: con -s len establecemos la longitud de los datos que captura tcpdump, donde len es la longitud que nos interesa. Por defecto el tcpdump sólo captura los primeros 68bytes, lo cual es útil si lo único que se quiere son las cabeceras IP, TCP o UDP, pero si estamos capturando otros protocolos se truncarían los datos y podríamos necesitar usar esta opción.

-v, -vv, -vvv: con estas opciones especificamos la cantidad de información que queramos que tcpdmp interprete.
-x: con esta opción imprimimos el contenido del paquete.
-w file: con esta opción podemos grabar la captura de datos para posteriormente leerla y analizarla. Además este tipo de ficheros de captura lo pueden leer otros analizadores como por ejemplo *Ethereal*.

El uso de **filtros** es lo más importante que nos permite hacer tcpdump. Un filtro es una expesión que va detrás de las opciones y que nos permite seleccionar los paquetes que estamos buscando. En ausencia de ésta, el tcpdump volcará todo el tráfico que vea el adaptador de red seleccionado. La expresión que se usa para definir el filtro tiene una serie de primitivas y tres posibles modificaciones a las mismas. Esta expresión será verdadera o falsa y hará que se imprima o no el paquete de datos.

Los 3 modificadores posibles son:
tipo: puede ser host, net o port. Indican respectivamente una máquina, una red completa o un puerto concreto.
dir: especifica desde o hacia dónde se va a mirar el flujo de datos. Tenemos **src** o **dst** y podemos combinarlos con **or** y **and**.
proto: en este caso es el protocolo que queremos capturar. Puede ser tcp, udp, ip, ether (en este caso captura tramas a nivel de enlace, arp), etc.

Se pueden combinar las expresiones anteriores con la ayuda de los operadores *not*, *and* y *or*, así como el uso de paréntesis. Veamos algunos ejemplos:
- Capturar todo el tráfico web: *tcpdump tcp and port 80*
- Capturar todas las peticiones DNS: *tcpdump udp and dst port 53*
- Capturar todo el tráfico excepto el WEB: *tcpdump tcp and not port 80*

A modo de ejemplo, mostramos una captura de todo el tráfico Web:

```
debian:/home# tcpdump tcp and port 80
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
14:21:40.355875 IP klecker.debian.org.www > 192.168.0.10.34098: F 3302434475:3302434475(0) ack 3442383933 win 71 <nop,nop,timestamp 1136472861 1325000>
14:21:40.394947 IP 192.168.0.10.34098 > klecker.debian.org.www: . ack 1 win 15216 <nop,nop,timestamp 1328751 1136472861>
14:21:40.446161 IP klecker.debian.org.www > 192.168.0.10.34099: F 627936784:627936784(0) ack 3456363299 win 79 <nop,nop,timestamp 1136472870 1325014>
14:21:40.482770 IP 192.168.0.10.34099 > klecker.debian.org.www: . ack 1 win 10088 <nop,nop,timestamp 1328773 1136472870>
...
```

Como vemos, todas las capturas en tcpdump tienen como primer campo una marca de tiempo. La línea general de un paquete TCP es como sigue:
src > dst:flags [dataseq ack window urgent options]

En principio, *src*,*dst* y *flags* están siempre presentes. Los otros dependiendo del tipo de conexión TCP que se trate. El significado de los parámetros es:
src: Dirección y puerto origen.
dst: Dirección y puerto destino.
flags: indica los flags de la cabecera TCP. Puede ser un . , cuyo significado es que no hay flags, o bien una combinación de S (SYN), F (FIN), P (PUSH), W(reducción de la ventana de congestión), E (ECN eco).
dataseq: el número de secuencia del primer byte de datos en este segmento TCP.
ack: el número de asentimiento. Indica el número siguiente que se espera recibir.
win: tamaño de la ventana de recepción.
urgent: existen datos urgentes.
options: indica la existencia de opciones.
De forma similar, un paquete UDP se imprimiría de la siguiente manera:
src_address.src_port > dst_address.dst_port: udp len

IPTRAF: sin duda alguna uno de los programas más prácticos para depurar el firewall es iptraf, ya que con él podemos observar si las conexiones se establecen o no; es un programa de consola que es aconsejable controlar ya que muestra en tiempo real el tráfico que atraviesa nuestra máquina con todo lujo de detalles: origen/destino de Ips y puertos, tráfico parcial o tráfico total según el interfaz de red, etc... Si vemos muchas conexiones simultáneas y nos perdemos, existe la posibilidad de aplicar filtros para captar sólo aquello que nos interesa.

Iptraf la instalamos con:

```
apt-get install iptraf
```

Y para ejecutar:

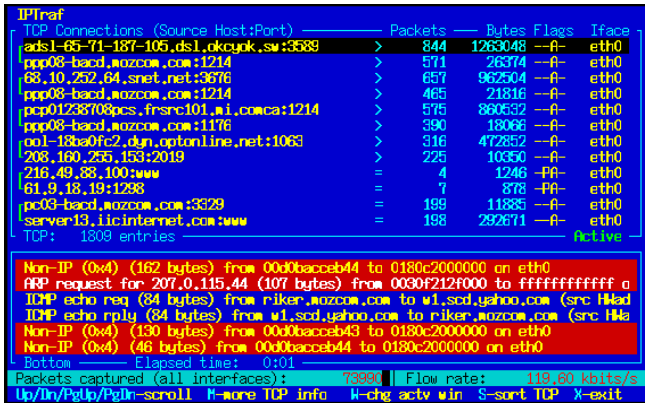
```
iptraf
```

Además de un menú de opciones a pantalla completa, Iptraf posee las siguientes características:

- Monitor de tráfico IP que muestra información del tráfico de la red.
- Estadísticas generales de las Interfaces.
- Módulo de estadísticas de LAN que descubre hosts y muestra datos sobre su actividad.
- Monitor TCP, UDP que muestra la cuenta de los paquetes de red para las conexiones de los puertos de aplicaciones.
- Utiliza el “raw socket interface” que lleva el kernel permitiendo ser usado por un amplio rango de tarjetas de red.

Con iptraf podemos comprobar si las conexiones TCP/IP se llegan a establecer o no. Una conexión tcp/ip empieza con el three-way-handshake:

- La máquina que desea conectarse a otra envía un paquete con flag SYN
- Si la otra máquina acepta, envía un SYN/ACK
- Entonces la máquina establece la conexión.
- Si el firewall está denegando la conexión, con iptraf veremos que la máquina origen sólo manda paquetes con el flag S (de SYN), y que del otro lado no sale nada. Saber usar iptraf nos ayudará mucho.



NMAP: es un programa open source que sirve para efectuar escaneos de puertos. Está orientado a la identificación de puertos abiertos en una computadora objetivo, determinando qué servicios está ejecutando la misma, e intenta determinar qué sistema operativo utiliza dicha computadora, (esta técnica es también conocida como *fingerprinting*). Ha llegado a ser una de las herramientas imprescindibles para todo administrador de sistemas, y es usado para pruebas de penetración y tareas de seguridad informática en general.

Es una herramienta de consola rápida, efectiva y con multitud de opciones. Podemos usarla desde máquinas ajenas a nuestra red para comprobar si realmente el firewall está filtrando correctamente y en cierta manera para hacernos una idea de qué “visión” pueden tener los atacantes de nuestro sistema.

Nmap es a menudo confundido como una herramienta para verificación de vulnerabilidades como nessus. Nmap es difícilmente detectable, ha sido creado para evadir los sistemas de detección de intrusos (IDS) e interfiere lo menos posible con las operaciones normales de las redes y de las computadoras que son analizadas.

Instalamos nmap mediante:

```
apt-get install nmap
```

Tal y como hemos comentado el uso de Nmap es muy sencillo, por ejemplo, para averiguar los puertos que tiene abierto nuestro servidor WEB además del puerto 80, desde un equipo que conectáramos en la red DMZ (172.16.0.0/24) que tenga instalado nmap, ejecutamos:

```
nmap 172.16.0.2

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2007-08-07 21:15 CEST
Interesting ports on 172.16.0.2:
(The 1662 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
80/tcp   open  http
111/tcp  open  rpcbind
992/tcp  open  telnets
2000/tcp open  callbook
3306/tcp open  mysql

Nmap finished: 1 IP address (1 host up) scanned in 0.199 seconds
```

Hemos comprobado que además del 80, tiene otros puertos abiertos que podrían ser puntos clave en la seguridad del equipo. Si ahora hacemos un rastreo de puertos desde el exterior (al otro lado de nuestro firewall), bastará con ejecutar (desde un equipo de la red 192.168.1.0/24 que tenga nmap instalado):

```
nmap 192.168.1.11

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2007-08-07 21:28 CEST
Interesting ports on 192.168.1.11:
(The 1660 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
80/tcp  open  http

Nmap finished: 1 IP address (1 host up) scanned in 0.816 seconds
```

Como era de esperar, observamos que NMAP sólo encuentra abierto el puerto 80 de nuestro servidor WEB.

Por último, vamos a hacer la prueba desde un equipo de la red interna (192.168.0.0/24):

```
nmap 192.168.0.1

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2007-08-07 21:43 CEST
Interesting ports on 192.168.0.1:
(The 1660 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
22/tcp  open  ssh
53/tcp  open  domain

Nmap finished: 1 IP address (1 host up) scanned in 0.813 seconds
```

Observamos que tenemos abierto el puerto correspondiente al servicio dns y ssh. ¿Pero dónde está el radius y el dhcp? La respuesta es que por defecto nmap hace un escaneo de puertos TCP. Para hacer un barrido de los puertos udp tecleamos:

```
nmap -sU 192.168.0.1

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2007-08-07 22:30 CEST
Interesting ports on 192.168.0.1:
(The 1472 ports scanned but not shown below are in state: closed)
PORT     STATE         SERVICE
53/udp   open|filtered domain
67/udp   open|filtered dhcpserver
1812/udp open|filtered radius

Nmap finished: 1 IP address (1 host up) scanned in 2.086 seconds
```

Existen muchas más opciones y alternativas, por lo que es más recomendable acceder a la documentación incluída con nmap, así como a la página man del mismo. Estos escaneos de máquinas y redes suelen dejar huellas de su ejecución en los registros (logs) de las máquinas escaneadas (por ejemplo, en */var/log/messages*), por lo que es interesante el utilizar alguno de los modos de escaneos invisibles tales como *-sF, -sX, -sN Stealh FIN, Xmas, o Null scan*, de forma que se evita finalizar la negociación TCP, evitando al mismo tiempo el comentado registro en los ficheros de logs.

Incluido con nmap se encuentra **nmapfe** que se un front-end gráfico, que permite ejecutar nmap usando el ratón. Indicar que existen otros front-ends gráficos para facilitar aún más el uso de esta potente aplicación (KNmap, LNmapFE, QNMap, Kmap, Web-NMap, vnmap, ...).

Resumiendo, Nmap es una herramienta ideal para verificar/auditar el firewall, tal como dice uno de los banners de Nmap, "*Audit your network security before the bad guys do*".

SHELL: en el propio script del firewall podemos añadir algunas opciones para descubrir fallos de sintaxis en las reglas. Imaginemos que tenemos un firewall de 40 líneas y una de ellas falla cuando ejecutamos el script. ¿Cuál es?. Es probable que el mensaje de error no aclare lo suficiente, por eso se puede añadir algo al final de cada regla:

```
..
iptables -A INPUT -s 192.168.1.0 -j ACCEPT && echo "regla 21 ok"
iptables -A INPUT -s 213.62.89.145 -j ACCEPT && echo "regla 22 ok"
...
```

Si la regla se ejecuta bien se mostrará el mensaje de ok.

2.4.2 Sistemas de detección / protección de Intrusiones (IDS/IPS)

2.4.2.1 Introducción a los IDS

Un sistema de detección de intrusos (o **IDS** de sus siglas en inglés *Intrusion Detection System*) es una herramienta de seguridad que se utiliza para monitorizar los eventos que ocurren en un sistema informático con el objetivo de localizar posibles intentos de intrusión.

El IDS suele tener sensores virtuales (por ejemplo, un sniffer de red) con los que el núcleo del IDS puede obtener datos externos (generalmente sobre el tráfico de red). El IDS detecta, gracias a dichos sensores, anomalías que pueden ser indicio de la presencia de ataques o falsas alarmas.

El funcionamiento de estas herramientas se basa en el análisis pormenorizado del tráfico de red, el cual al entrar al analizador es comparado con firmas de ataques conocidos, o comportamientos sospechosos, como puede ser el escaneo de puertos, paquetes malformados, etc. El IDS no sólo analiza las cabeceras de los paquetes sino también analiza el contenido de éstos.

En ocasiones se distingue entre lo que se denomina actividad anómala y actividad intrusiva. Existen actividades intrusivas pero no anómalas (falsos negativos) que por el hecho de no escaparse de unos patrones de comportamiento normal, el IDS no las detecta. Por otro lado existen actividades anómalas que no son realmente intrusivas (falsos positivos), éstas las detecta el IDS pero después de un análisis exhaustivo de la alerta se llega a la conclusión de que no se trata de ningún intento de intrusión. Estas últimas son las más comunes en estos entornos y para ello se estima un período de ajuste hasta eliminar un porcentaje elevado de estos falsos positivos. Con ello se consigue una optimización del sistema.

Normalmente esta herramienta se integra con un firewall. El detector de intrusos es incapaz de detener los ataques por sí solo, excepto los que trabajan conjuntamente en un dispositivo de puerta de enlace con funcionalidad de firewall, convirtiéndose en una herramienta muy poderosa ya que se une la inteligencia del IDS y el poder de bloqueo del firewall, al ser el punto donde forzosamente deben pasar los paquetes y pueden ser bloqueados antes de penetrar en la red. Además, existen IDS que pueden aplicar políticas en remoto a otros firewalls.

Los IDS suelen disponer de una base de datos de “firmas” de ataques conocidos. Dichas firmas permiten al IDS distinguir entre el uso normal del PC y el uso fraudulento, y/o entre el tráfico normal de la red y el tráfico que puede ser resultado de un ataque o intento del mismo. El buen funcionamiento de un sistema de estas características no sólo depende de una buena instalación y configuración sino de que la base de datos en la que tenemos los patrones esté actualizada.

Existen tres tipos de sistemas de detección de intrusos:

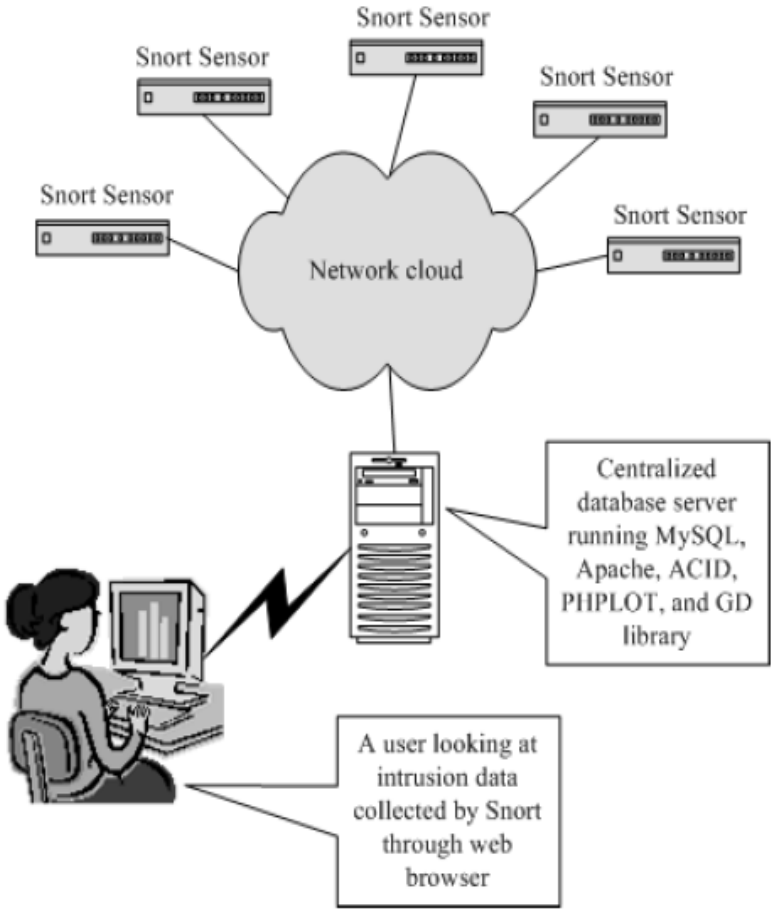
- **HIDS (Host IDS)**: su función es analizar cambios a nivel de S.O., es decir, controlar todos aquellos parámetros y archivos que están disponibles a nivel del sistema operativo, y que suelen ser modificados cuando tiene lugar un ataque. La ventaja es que la carga de procesado es poca. Sin embargo, requieren más tiempo de gestión y configuración que otros IDS ya que cada host donde lo implantemos tendrá una configuración específica.

- **NIDS (Network IDS)**: es un IDS basado en red, detectando ataques a todo el segmento de la red. Su interfaz debe funcionar en modo promiscuo capturando así todo el tráfico de la red.

- **DIDS (Distributed IDS)**: sistema basado en la arquitectura cliente-servidor compuesto por una serie de NIDS (IDS de redes) que actúan como sensores centralizando la información de posibles ataques en una unidad central que puede almacenar o recuperar los datos de una base de datos centralizada. La ventaja es que en cada NIDS se puede fijar unas reglas de control especializándose para cada segmento de red. Es esta estructura es habitual utilizar redes privadas virtuales (VPN), pues se utilizan para conseguir privacidad entre los sensores y el servidor central.

Otra clasificación es **sistemas pasivos** y **sistemas activos**: en un sistema pasivo, el sensor detecta una posible intrusión, almacena la información y manda una señal de alerta que se almacena en una base de datos. En un sistema reactivo, el IDS responde a la actividad sospechosa reprogramando el cortafuegos para que bloquee tráfico que proviene de la red del atacante.

En el siguiente dibujo observamos un ejemplo de arquitectura con múltiples dispositivos que corren snort y registran sus alarmas en una base de datos centralizada:



Dependiendo de la topología de la red, será necesario considerar el lugar de colocación del IDS en un punto o en otro. Si la red está segmentada con hubs (capa 1 del modelo OSI) no hay problema en analizar todo el tráfico de la red realizando una conexión a cualquier puerto. En cambio, si se utiliza un switch (capa 2 del modelo OSI), es necesario conectar el IDS a un puerto SPAN (Switch Port Analiser) para poder analizar todo el tráfico de esta red.

Si colocamos el IDS antes del cortafuegos capturaremos todo el tráfico de entrada y salida de nuestra red. La colocación detrás del cortafuegos monitorizará todo el tráfico que no sea detectado y parado por el firewall o cortaguegos, por lo que será considerado como malicioso en un alto porcentaje de los casos. En nuestro caso colocaremos el IDS en la misma máquina que el cortafuegos, así actúan en paralelo, es decir, el firewall detecta los paquetes y el IDS los analizará.

Pese a la automatización de los sistemas de detección, los costes de mantenimiento son extremadamente elevados ya que es necesario destinar muchos recursos humanos para visualizar y gestionar las alertas, así como para distinguir entre los ataques reales y los denominados falsos positivos. Pocas empresas están dispuestas a contratar personal para analizar logs o controlar patrones de tráfico de red. No obstante, es estrictamente necesario disponer de personal especializado para dicho análisis ya que un sistema automatizado no puede determinar si la alerta detectada es realmente una alerta o un falso positivo. El IDS no debe generar una cantidad elevada de falsos positivos o de logs ya que llegaría un momento en que nadie se preocuparía de comprobar las alertas emitidas por el detector.

2.4.2.2 Introducción a Snort

Snort es un sniffer de paquetes y un IDS basado en red (NIDS), monitoriza todo un dominio de colisión. Implementa un motor de detección de ataques y barrido de puertos que permite registrar, alertar y responder ante cualquier anomalía previamente definida como patrones que corresponden a ataques, barridos, intentos de aprovechar alguna vulnerabilidad, análisis de protocolos, etc. Todo esto en tiempo real. Así mismo existen herramientas de terceros para mostrar informes en tiempo real (ACID) o para convertirlo en un Sistema Detector y Preventor de Intrusos.

Este IDS implementa un lenguaje de creación de reglas flexibles, potente y sencillo. Durante su instalación ya nos provee de cientos de filtros o reglas para backdoor, ataques ddos, finger, ftp, ataques web, CGI, escaneos nmap...

Puede funcionar como sniffer (podemos ver en consola y en tiempo real qué ocurre en nuestra red, todo nuestro tráfico), registro de paquetes (permite guardar en un archivo los logs para su posterior análisis, un análisis offline) o como un IDS normal. cuando un paquete coincide con algún patrón establecido en las reglas de configuración, se registra. Así se sabe cuando, de dońde y cómo se produjo el ataque.

Aún cuando la herramienta *tcpdump* que vimos en la sección anterior es considerada una herramienta de auditoría muy útil, no se considera un verdadero IDS puesto que no analiza ni señala paquetes por anomalías. Tcpdump imprime toda la información de paquetes a la salida en pantalla o a un archivo de registro sin ningún tipo de análisis. Un verdadero IDS analiza los paquetes, marca las transmisiones que sean potencialmente maliciosas y las almacena en un registro formateado, así, Snort utiliza la librería estándar libpcap y tcpdump como registro exhaustivo de los paquetes capturados.

Snort está disponible bajo licencia GPL, gratuito y funciona bajo plataformas Windows y UNIX/Linux. Dispone de una gran cantidad de filtros o patrones ya predefinidos, así como actualizaciones constantes ante casos de ataques, barridos o vulnerabilidades que vayan siendo detectadas a través de los distintos boletines de seguridad.

La característica más apreciada de Snort, además de su funcionalidad, es su subsistema flexible de firmas de ataques. Snort tiene una base de datos de ataques que se está actualizando constantemente y a la cual se pueden añadir nuevas reglas manualmente o actualizarlas a través de Internet. Los usuarios pueden crear 'firmas' basadas en las características de los nuevos ataques de red y enviarlas a la lista de correo de firmas de Snort, para que así todos los usuarios de Snort se puedan beneficiar. Esta ética de comunidad y compartir ha convertido a Snort en uno de los IDS basados en red más populares, actualizados y robustos.

Para instalar snort:

```
apt-get install snort
```

2.4.2.3 Trabajando con Snort

Una vez instalado snort hemos de configurarlo modificando el archivo *snort.conf*, estableciendo las variables oportunas de acuerdo con la configuración de nuestra red. Estudiaremos este fichero de configuración más adelante en esta sección. De momento vamos a mostrar paso a paso de forma gradual en dificultad el uso de Snort.

Para ejecutar Snort como un simple *Sniffer* de paquetes escribiremos:

```
snort -v -i eth1
```

Si queremos almacenar las capturas de todos los paquetes que recibimos tendremos que hacer lo siguiente:

```
snort -v -l ./log -i eth1
```

Con -l especificamos un directorio de *logging*.

Podemos añadir, además de las opciones, una serie de filtros para optimizar los resultados obtenidos. Estos filtros se añadirán en el mismo formato que usa el programa Tcpdump y que vimos en la sección anterior.

Si en lugar de almacenar todos los paquetes queremos aplicar una serie de reglas utilizaremos la opción -c:

```
snort -v -l /log -i eth1 -c /etc/snort/snort.conf
```

El modo detección de intrusos de red se activa añadiendo a la línea de comandos de snort la opción **-c snort.conf**. En este archivo, *snort.conf*, se guarda toda la configuración de las reglas, preprocesadores y otras configuraciones necesarias para el funcionamiento en modo NIDS. Con la opción **-D** indicará a snort que corra como un servicio.

Snort creará, aparte de la estructura de directorios, un archivo *alert* dentro de */var/log/snort* donde almacenará las alertas generadas. Hay varias maneras de configurar la salida de snort, es decir, las alertas, el modo en que se almacenarán éstas en el archivo alert. Existen distintos modos:

Fast: el modo Alerta Rápida nos devolverá información sobre: tiempo, mensaje de la alerta, clasificación, prioridad de la alerta, IP y puerto de origen y destino.

```
snort -A fast -dev -l /log -i eth1 -c /etc/snort/snort.conf
```

Full: El modo de alerta completa nos devolverá información sobre: tiempo, mensaje de alerta, clasificación, prioridad de la alerta, IP y puerto de origen/destino e información completa de las cabeceras de los paquetes registrados.

```
snort -A full -dev -l /log -i eth1 -c /etc/snort/snort.conf
```

Después veremos cómo generar alertas de otra forma y cómo almacenarlas en una base de datos.

Una vez que tenemos snort funcionando, nos interesaría saber si realmente está capturando datos y registrando actividad sobre intrusiones. El siguiente comando nos genera algunas alertas que podremos ver en la consola o bien en el archivo */var/log/snort/alert* si tenemos a snort en background.

```
ping -n -r -b 192.168.1.1 -p “7569643d3028726f6f74290a” -c3
PATTERN: 0x7569643d3028726f6f74290a
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=128 time=5.05ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=128 time=0.173ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=128 time=0.172ms

--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
```

Si se generan las alertas correctamente indica que Snort está funcionando bien. Observamos que en el fichero /var/log/snort/alert aparecen entradas del tipo:

```
[**] [1:498:6] ATTACK-RESPONSES id check returned root [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
08/17-14:06:05.526999 192.168.1.1 -> 192.168.1.1
ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF
Type:8 Code:0 ID:64265 Seq:1 ECHO

...
```

El patrón “7569643d3028726f6f74290a” es igual a “uid=0(root)” que es el patrón que genera la alerta. La opción -c3 hace que se envíen 3 paquetes.

Snort dispone de muchas opciones en línea de comandos que son útiles para ejecutarlo en diferentes situaciones. Aunque ya hemos visto algunas de ellas, presentamos una tabla resumen a continuación con las más comunes:

Opción	Descripción
-A	Como hemos visto establece el modo de alerta, como “fast” o “full”. Esta opción se utiliza para presentar la información de las alertas en pantalla en lugar de los ficheros de log.
-b	Esta opción es para registrar los paquetes en el formato de tcpdump.
-c	Es la opción más común. Especifica la ubicación del fichero snort.conf.
-D	Para que snort corra en background.
-i	Esta opción hace que snort escuche en una interfaz en concreto.
-l	Especifica el directorio donde snort registrará los mensajes. La opción por defecto es /var/log/snort.
-T	Opción para encontrar errores en los archivos de configuración.

La generación de alertas de Snort está basada en reglas que se le aplican a los paquetes. Snort es capaz de analizar tanto en las capas 3 y 4 de TCP/IP como en la capa de aplicación. Las reglas de Snort que generan estas alertas utilizan una sintaxis fácil de entender, la mayoría de ellas se escriben en una única línea, que se encuentra en el fichero de configuración que suele ser *snort.conf*, aunque se suelen utilizar múltiples ficheros que se incluyen desde este fichero de configuración principal.

Una regla de ejemplo (pésima porque no aporta nada de información sobre intrusiones, y además sobrecargaría el archivo de logs) podría ser:

```
alert ip any any -> any any (msg: “IP Packet detected”;)
```

Esta regla generá un mensaje de alerta por cada paquete IP capturado. La sintaxis es:
alert: especifica que la línea es una regla de alerta.
ip: la reglá será aplicada a todos los paquetes IP.
any: el primero es la dirección IP origen y el segundo el puerto origen, aquí se aplicarán a todos los paquetes. El tercero y el cuarto son la dirección y el puerto destino, respectivamente.
-> : es la dirección del paquete.
La última parte son las opciones de la regla y contiene un mensaje que será grabado con la alerta.

Es decir, la regla tiene una cabecera tal que así:

Action	Protocol	Address	Port	Direction	Address	Port

Action puede ser:
pass: ignora el paquete.
log: registra el paquete
alert: genera una alerta.
activate: activa otra regla para chequear otras condiciones.
dynamic: son reglas que se aplican con un activate previo.
Direction puede ser “-> “, “<-” ó “<>”

Y después de esta cabecera vendrían las opciones de la regla entre paréntesis, algunas de ellas pueden ser:
msg: “mensaje”: registra la alarma con el mensaje especificado.
config classification: name,description,priority: establece clasificaciones a las reglas, otorgándole prioridades.
content:encuentra patrones de datos dentro de la parte de datos de un paquete. El

problema de este análisis es que es bastante costoso computacionalmente.

dsize: utilizado para detectar distintos tamaños en los paquetes.

flags: utilizado para detectar los distintos flags que van en la cabecera de los paquetes TCP.

react: sirve para terminar una sesión o bloquear algún servicio.

resp: envía paquetes al receptor. Puede servir para anular conexiones maliciosas.

...

El siguiente paso es configurar snort editando su fichero de configuración */etc/snort/snort.conf*. El fichero se divide en 4 partes fundamentales:

1. Configuración de las variables de red para nuestro entorno:
   En esta sección asignaremos valores a las variables tales como la red que monitorizará Snort en busca de ataques (HOME_NET), los servidores DNS (DNS_SERVERS), servidores smtp (SMTP_SERVERS), etc.

   Por ejemplo, definiremos la variable HOME_NET de la siguiente manera:

   var HOME_NET 192.168.0.0/24

   Más tarde, podríamos utilizar esta variable en nuestras reglas de la siguiente manera:

```
alert ip any any -> $HOME_NET any (ipopts: lsrr; msg: “Loose source routing attempt”; sid: 100001;)
```

   Como podemos ver, utilizar variables nos facilita la tarea de modificar todas las reglas que tengamos escritas, modificando únicamente el valor de nuestra variable.
   También es posible especificar varias redes separadas por comas:

   var HOME_NET [192.168.0.0/24, 192.168.10.0/24]

   La palabra “any” también puede ser una variable. Por ejemplo, para chequear paquetes que provengan desde cualquier interfaz:

   var EXTERNAL_NET any

   Se recomienda establecer las variables de aquellos servicios que tengamos activos, de esta manera, Snort sólo monitorizará ataques en aquellos servicios que tengamos en lugar de todo el tráfico capturado. ¿Para qué chequear ataques HTTP si no tenemos un servidor WEB?

   Existe también la posibilidad de utilizar directivas que se especifican con “config”, que nos permiten configurar aspectos generales de Snort. Estas directivas pueden utilizarse para modificar parámetros en línea de comandos. Su sintaxis es la siguiente:

   config directive_name [: value]

   Por ejemplo, en lugar de utilizar la opción -D en línea de comandos podríamos utilizar:

   config daemon

2. Configuración de los preprocesadores:
   Los preprocesadores permiten operar sobre los paquetes capturados antes de que sean procesados por el motor de detección de Snort.
   El formato general es el siguiente:

   preprocessor <preprocessor_name> [: <configuration_options>]

   A continuación mostramos un ejemplo de desfragmentación IP utilizando el preprocesador frag2:

   preprocessor frag2

   Otro ejemplo utilizando el preprocesador stream4 con un argumento para detectar escáneres de puertos:

   preprocesor stream4: detect_scans

   Tanto frag2 como stream4 son preprocesadores definidos. También podríamos crear nuestros propios preprocesadores si fuéramos programadores.
   Para empezar, podemos dejar los valores por defecto que vienen.

3. Configurar los plug-ins de salida:
   Los plugings de salida nos permiten elegir de entre una gran variedad de formatos de salida. En el snort.conf vienen todas comentadas y con ejemplos.
   El formato general es el siguiente:
   output <output_module_name> [: <configuration_options>]

   Por ejemplo, si quisiéramos grabar mensajes de log en una base de datos MYSQL, podríamos configurar un módulo de salida que contiene el nombre de la base de datos, la dirección del servidor donde se encuentra la base de datos, el nombre de usuario y contraseña:
   output database: alert, mysql, user=fran password=pfc dbname=snort host=localhost

4. Personalizar las reglas:
   Este es el último paso, si queremos deshabilitar o agregar algún grupo de reglas sólo tenemos que comentar o añadir el include correspondiente del archivo de reglas.

   La sintaxis es tal que así:
   include misreglas.rules

Las firmas de ataques que usa snort para generar alertas se almacenan en el directorio */etc/snort/rules*. Estas firmas se encuentran en los archivos *.rule y el nombre del archivo hace referencia al tipo de alertas que contienen:

```
attack-responses.rules  icmp-info.rules   p2p.rules       telnet.rules
backdoor.rules          icmp.rules        policy.rules    tftp.rules
bad-traffic.rules       imap.rules        pop2.rules      virus.rules
chat.rules              info.rules        pop3.rules      web-attacks.rules
ddos.rules              local.rules       porn.rules      web-cgi.rules
deleted.rules           misc.rules        rpc.rules       web-client.rules
dns.rules               multimedia.rules  rservices.rules  web-coldfusion.rules
dos.rules               mysql.rules       scan.rules      web-frontpage.rules
experimental.rules      netbios.rules     shellcode.rules  web-iis.rules
exploit.rules           nntp.rules        smtp.rules      web-misc.rules
finger.rules            oracle.rules      snmp.rules      web-php.rules
ftp.rules               other-ids.rules   sql.rules       x11.rules
```

Por ejemplo, todas las reglas relacionadas con ataques X-Windows están en el fichero X11.rules, que tiene un aspecto tal que así:

```
# (C) Copyright 2001-2004, Martin Roesch, Brian Caswell, et al.
# All rights reserved.
# $Id: x11.rules,v 1.19 2004/07/23 20:15:44 bcm Exp $
#----------
# X11 RULES
#----------

alert tcp $EXTERNAL_NET any -> $HOME_NET 6000 (msg:"X11 MIT Magic Cookie detected";
flow:established;      content:      "MIT-MAGIC-COOKIE-1";      reference:arachnids,396;
classtype:attempted-user; sid:1225; rev:4;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 6000 (msg:"X11 xopen"; flow:established;
content: "l|00 0b 00 00 00 00 00 00 00 00|"; reference:arachnids,395;
classtype:unknown; sid:1226; rev:4;)
```

De forma similar, cada fichero contiene un listado de reglas para una clase particular de ataques. El archivo *dns.rules* contiene todas las reglas relacionadas con ataques sobre servidores DNS, el archivo *telnet.rules* contiene todas las reglas relacionadas con ataques en el puerto de telnet, y así sucesivamente.

El archivo *local.rules* no contiene ninguna regla. Está pensado para que el administrador de Snort escriba ahí sus reglas personalizadas. Sin embargo, podríamos usar cualquier fichero nuevo tan sólo escribiendo un include en *snort.conf*.

Analicemos a continuación en detalle una de esas reglas para que tengamos una visión de cómo son las reglas que se utilizan en los sistemas de producción. Por ejemplo, una extraída de *telnet.rules*:

```
alert tcp $TELNET_SERVERS 23 -> $EXTERNAL_NET any (msg:"TELNET Attempted SU from wrong
group";    flow:    from_server,established;    content:"to    su    root";    nocase;
classtype:attempted-admin; sid:715; rev:6;)
```

Esta regla genera una alerta aplicada a paquetes TCP que tiene las siguientes características:

- $TELNET_SERVERS es una variable definida en el archivo *snort.conf* y es nuestra lista de servidores telnet.
- Se analiza el puerto 23, la regla sólo chequea las respuesta del servidor.
- Como vimos, $EXTERNAL_NET es una variable definida en snort.conf y lista todas las redes que están fuera de nuestra red interna. Esta regla sólo aplicará a sesiones que se generen desde el exterior de nuestra red interna.
- La palabra flow se utiliza para aplicar esta regla únicamente en conexiones establecidas.
- La palabra content significa que la alerta será generada cuando el paquete contenga la cadena “to su root”.
- Nocase indica que se ignorarán mayúsculas o minúsculas.
- Classtype es para asignar una prioridad a la regla.
- El identificador de la regla es 715.
- La palabra rev es para indicar la revisión de la regla.

Hay una extensa lista de reglas en los demás archivos .rules de Snort. De forma general, una regla que estuviera bien escrita y ordenada debería tener un mensaje (msg), una clasificación, un número de identidad para identificar la regla, y si la vulnerabilidad es conocida, una referencia a una URL donde se pueda encontrar más información sobre el ataque. Por último, la revisión para llevar las actualizaciones de la regla.

Sería conveniente que fuéramos actualizando estas reglas periódicamente. Snort.org se actualiza frecuentemente y va incorporando cambios ante nuevas vulnerabilidades haciendo a Snort más robusto. **Oinkmaster** es una herramienta escrita en Perl que descarga, modifica y pone en funcionamiento archivos de reglas de *snort.org*.

Descargaremos oinkmaster de sourceforge. La razón de descargar el script desde su página en lugar de utilizar los repositorios de Debian es que recientemente la página de *snort.org* requiere darse de alta como usuario para poder acceder a las actualizaciones de las reglas, y la versión de oinkmaster que se encuentra actualmente en los repositorios no está preparada para esta funcionalidad:

```
cd /usr/src/
wget http://mesh.dl.sourceforge.net/sourceforge/oinkmaster/oinkmaster-2.0.tar.gz
tar -xvzf oinkmaster-2.0.tar.gz
```

Para que funcione debemos colocar los archivos en su lugar correcto:

```
cp /usr/src/oinkmaster-2.0/oinkmaster.conf /etc/
cp /usr/src/oinkmaster-2.0/oinkmaster.pl /usr/bin/
```

Después de habernos dado de alta como usuario en la página de snort.org, iniciamos una sesión en la web y pedimos un código de activación, obtenemos:

3f44d12b1d2d07854be854c6dcc80fb7e2845f1e

Que introducimos en el archivo de configuración de oinkmaster. Editamos *oinkmaster.conf,* descomentamos la línea siguiente y sustituímos <oinkcode> por nuestro código:

```
# Example for Snort 2.4
url = http://www.snort.org/pub-bin/oinkmaster.cgi/<oinkcode>/snortrules-snapshot-2.4.tar.gz
```

Si hubiéramos modificado las reglas por defecto de snort y hubiéramos hecho reglas a medida, podríamos decirle a oinkmaster que no actualizara estos archivos concretos para no perder los cambios.

Finalmente, ejecutamos el script perl mediante.

```
oinkmaster.pl -o /etc/snort/rules/
```

La herramienta nos da información detallada sobre las acciones que tienen lugar durante el proceso.

A medida que vamos teniendo algunas alarmas resulta tediosa la monitorización del archivo de logs. Necesitamos, por tanto, un sistema eficiente de registro de alarmas, esto es, una base de datos donde se almacene la información de las alarmas de manera ordenada. Instalaremos una base de datos MySQL en nuestro dispositivo con snort.

```
apt-get install mysql-server
```

Para ello, necesitamos tener snort compilado con soporte para mysql. Podemos instalar el siguiente paquete que ya trae el binario de snort con esta opción:

```
apt-get install snort-mysql
```

Una vez que tenemos el servidor de MySQL corriendo debemos crear la base de datos para snort. Podemos crearla utilizando el usuario root y darle todos los privilegios al usuario snort.

El cliente mysql es un programa que se utiliza para conectarse al servidor de bases de datos MySQL.

En primer lugar, cambiamos la contraseña de root:

```
mysqladmin -u root password pfc
```

Nos conectamos al servidor de mysql y creamos la base de datos para snort:

```
mysql -u root -p
create database snort;
```

La opción -u especfica el usuario y con la opción -p le damos el password en la siguiente línea. Utilizar el usuario root para acceder a la base de datos no es recomendado. Para este propósito, se recomienda crear un nuevo usuario que llamaremos snort. El siguiente comando crea un usuario llamado snort y le asigna privilegios a todas las tablas de la base de datos snort:

```
grant all on snort.* to snort@localhost identified by 'password_snort';
```

El siguiente comando recarga los privilegios desde las tablas de concesiones en la base de datos mysql.

```
flush privileges;
```

Una vez que tenemos creada la base de datos e snort, necesitamos crear sus tablas. Afortunadamente, en lugar de ir una por una podemos utilizar el script *create_mysql* que nos creará todas las tablas de manera automática. Dicho esquema se encuentra en:

/usr/share/doc/snort-mysql/create_mysql.gz

```
gunzip /usr/share/doc/snort-mysql/create_mysql.gz
```

Y el comando siguiente utiliza este script para crear las bases:

```
mysql -u snort -p snort < /usr/share/doc/snort-mysql/create_mysql
```

Donde snort es el nombre de la base de datos y la última parte < *.../create_mysql* especifica un archivo del cual el cliente de mysql leerá comandos.

Si queremos echarle un vistazo a cómo son las tablas, podemos verlo mediante:

```
mysql -u snort -p
use snort;
show tables;
```

Ahora necesitamos editar el archivo de configuración snort.conf para que deje los registros en la base de datos:

```
output database: log, mysql, user=snort password=password_snort dbname=snort host=localhost
```

Ya que hemos configurado Snort lo reiniciamos mediante:

```
snort -c /etc/snort/snort.conf
```

Cuando arrancamos snort después de la configuración de la base de datos, el mensaje de inicio nos muestra qué base de datos está siendo utilizada:

```
database: compiled support for ( mysql )
database: configured to use mysql
database: user = snort
database: password is set
database: database name = snort
database: host = localhost
database: sensor name = 192.168.1.11
database: sensor id = 1
database: schema version = 106
database: using the "log" facility
```

Después de haber configurado la base de datos, nos interesaría comprobar si los mensajes de alertas están siendo grabados en las tablas. Para ver que esto es así, vamos a introducir una regla de ejemplo:

```
alert ip any any -> any 30 (msg: Actividad en el puerto 30";)
```

Ejecutamos Snort:

```
snort -i eth1 -c /etc/snort/snort.conf
```

Y generamos alguna actividad en el puerto citado desde otra computadora:

```
telnet 192.168.1.11 30
```

Si nos conectamos a la base de datos mysql podemos ver las capturas mediante:

```
mysql> select * from tcphdr;
+-----+-----+-----------+-----------+----------+---------+---------+---------+-----------+---------+----------+---------+
| sid | cid | tcp_sport | tcp_dport | tcp_seq  | tcp_ack | tcp_off | tcp_res | tcp_flags | tcp_win | tcp_csum | tcp_urp |
+-----+-----+-----------+-----------+----------+---------+---------+---------+-----------+---------+----------+---------+
|   2 |   5 |      4851 |        30 | 36354123 |       0 |      10 |       0 |         2 |    5840 |    57485 |       0 |
+-----+-----+-----------+-----------+----------+---------+---------+---------+-----------+---------+----------+---------+
1 row in set (0.00 sec)
```

ACID (Analysis Console for Intrusion Databases) es una herramienta para presentar los datos de Snort utilizando un interfaz Web. Está escrito en PHP. Trabaja con Snort y bases de datos como MySQL. Lo utilizaremos para ver de forma más amigable los datos de las alertas que Snort almacena en su base de datos.

Se instala de la siguiente manera:

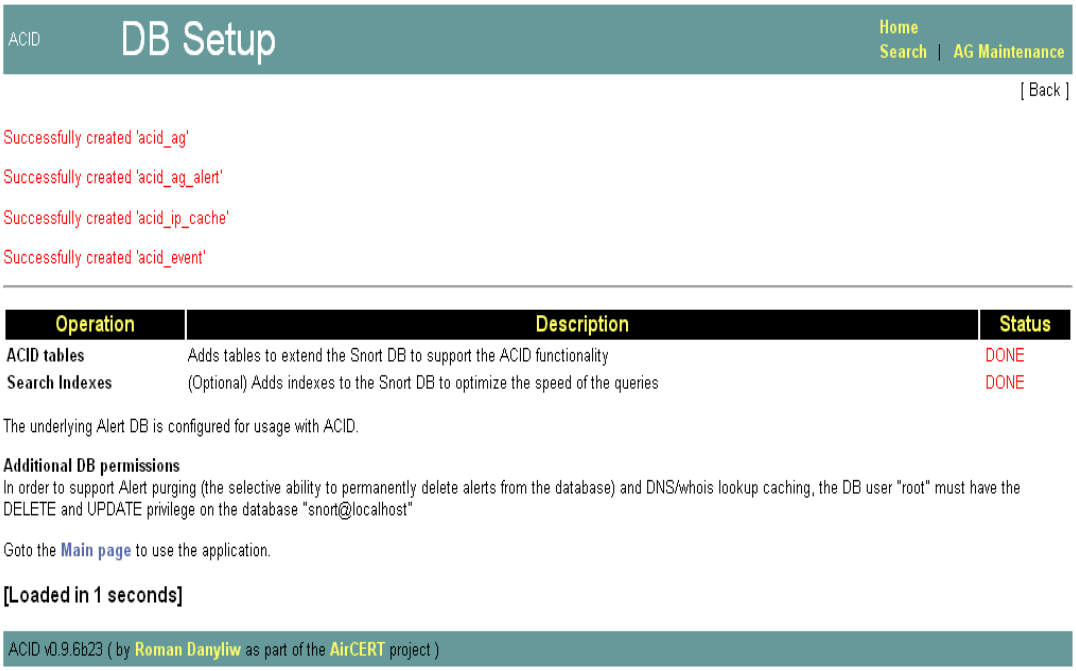
```
apt-get install acidbase
```

Para acceder al interfaz ACID desde un ordenador interno a la red teclearemos en cualquier navegador WEB la dirección de la interfaz ethernet interna de nuestro dispositivo:

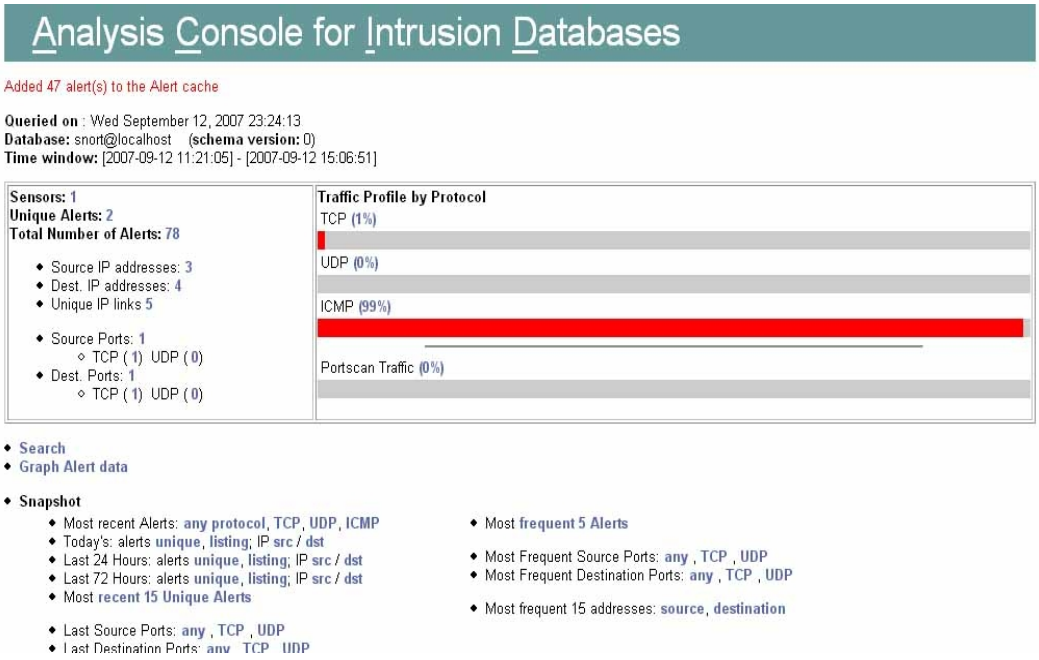
```
http://192.168.0.1/acid/
```

La primera vez que ejecutamos ACID, necesitaremos hacer algunas tareas de configuración. Haremos click en el botón “Create ACID AG” para que ACID cree una tabla en la base datos necesaria para que funcione bien con Snort.

Después de haber hecho esto se nos presenta una pantalla tal que así:



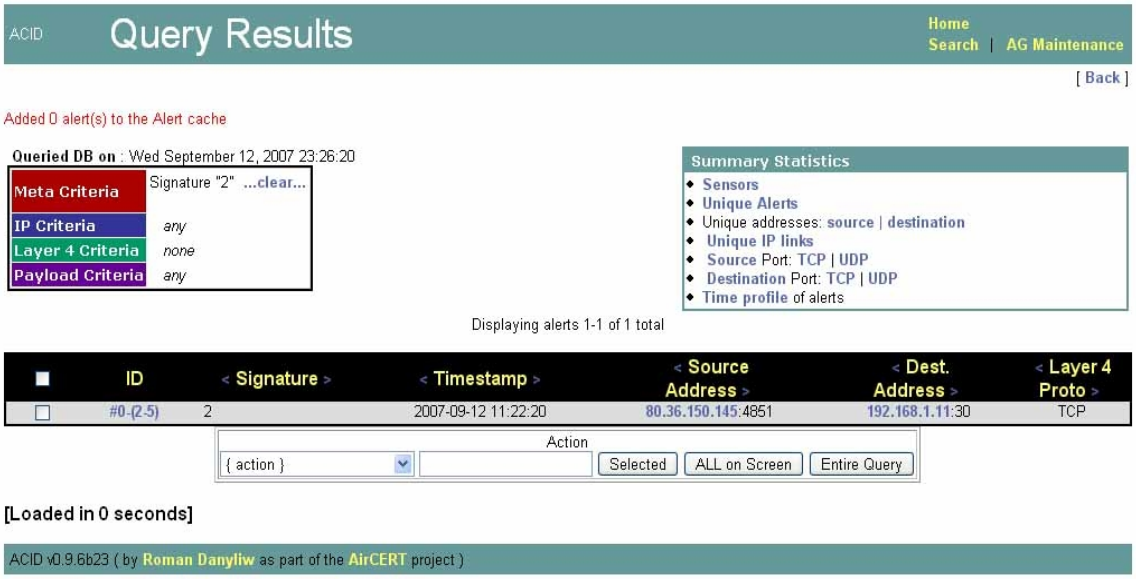
La página principal de ACID proporciona información sobre los porcentajes de tráfico cursados hasta el momento y el número de alarmas encontradas.



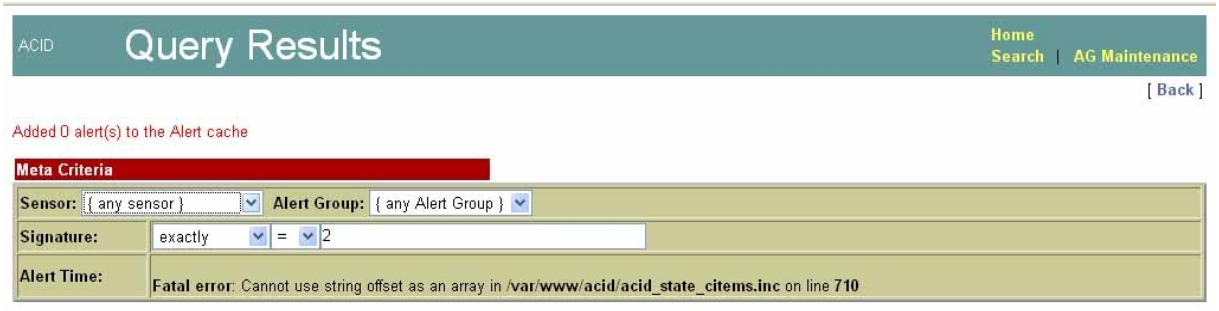
Desde esta página podemos acceder a otras que nos muestren información sobre:

- Lista de sensores que registran información en la base de datos de Snort.
- Número de alertas.
- La dirección IP de origen que ha generado las alertas, para ver quién está intentando atacar nuestro dispositivo.
- Dirección IP destino, así como puertos origen y destino.
- Alertas relacionadas con un protocolo particular: TCP, UDP, ICMP ...
- Las alertas más frecuentes.

En la siguiente imagen vemos en detalle la captura de la alerta generada anteriormente sobre el puerto 30, la misma que vimos directamente en la base de datos MySQL.



Otro aspecto importante de ACID es que podemos realizar búsquedas de los datos capturados basándonos en un sensor en particular, la hora exacta en la que se produce la alerta, las direcciones IP origen y destino, protocolos, etc...



Otra opción interesante es la posibilidad de enviar reportes a nuestro correo electrónico desde ACID, tanto detallado, como resúmenes. A continuación mostramos la información que hemos recibido por correo electrónico sobre la alerta del puerto 30.

De: ACID alert <acid@debian>
Asunto: ACID Incident report
Generated by ACID v0.9.6b23 on Wed, 12 Sep 2007 12:44:40 +0200

#2-5| [2007-09-12 11:22:20] “direccion_IP_publica”:4851 -> 192.168.1.11:30 2

Sin embargo, nos podría ser de utilidad que Snort nos mandase un correo automáticamente cuando detectase una intrusión. Para ello haremos uso de la herramienta **syslog-ng,** que es un demonio que gestiona dónde y cómo registra los logs del sistema, e incluso puede mandarlos a un puerto, a otro servidor, etc..., es infinitamente más flexible que el syslog "común". Permite aplicar filtros, clasificar de acuerdo a distintos orígenes y enviar a diferentes destinos los logs.

Instalaremos syslog-ng mediante:

```
apt-get install syslog-ng
```

Syslog-ng reemplaza al *syslog* anterior, pero la configuración del syslog es la misma.

Syslog-ng necesita un fichero de configuración para funcionar: *etc/syslog-ng/syslog-ng.conf*. Está formado por las secciones *source*, *filter* y *destination*. Los filtros describen como syslog-ng debería manejar los mensajes que recibe de las diversas fuentes. Usamos filtros para ordenar los mensajes y pasarlos a los destinos apropiados. Los destinos especifican dónde y por qué medios un mensaje debe ser redirigido y procesado.

Con el siguiente Script podemos enviarnos un correo electrónico

```
#!/bin/bash
while read input;
do
echo $input | mail -s "Alerta de Snort" nuestradireccion@dominio.com
done
```

Donde *mail* es un cliente de correo electrónico, pero sólo funciona en local, es decir, necesitaremos tener instalado un servidor MTA (Mail Transfer Agent) como Postfix para que el envío del correo pueda realizarse:

```
apt-get install mailx
apt-get install postfix
```

En nuestro caso, nos interesaría utilizar un servidor de correo externo en lugar de usar el propio dispositivo para realizar esta función. De hecho, podríamos utilizar un servidor de correo electrónico que tuvieramos en la DMZ de nuestro esquema, y utilizar dicho servidor para enviar el correo saliente a la dirección del administrador de la red avisando de la alerta. Como sólo necesitamos enviar un correo a través de una pasarela o relay, programas del estilo de *postfix* o *sendmail* son demasiado pesados para realizar esta tarea. La forma más sencilla de utilizar un servicio de pasarela es mediante la instalación de *ssmtp*.

```
apt-get install ssmtp
```

Una vez que ha sido instalado, podemos configurarlo mediante un fichero de sólo cuatro líneas ubicado en: */etc/ssmtp/ssmtp.conf.*

```
root=nuestradirecciondecorreo@ejemplo.com
mailhub=mail.ejemplo.com
rewriteDomain=ejemplo.com
hostname=_HOSTNAME_
```

Debemos asegurarnos de que se utiliza una dirección real para root. Se debe introducir nuestra pasarela de correo en lugar de mail.ejemplo.com (la dirección IP de nuestro servidor de correo ubicado en la DMZ). Ejecutar ssmtp en lugar de mail de esta forma permite que el script nos envíe el correo a través del MTA remoto.

Pero nosotros en lugar de syslog-ng utilizaremos **Swatch**. Swatch es una excelente herramienta de monitorización de logs escrita en Perl, que además de poder avisar alertas por correo, puede resumir todas las alertas en un único correo en lugar de sobrecargar la cuenta del administrador enviando un email por cada alerta. También tiene otras opciones como ejecutar las tareas en función de la hora del día.

Lo instalaremos con:

```
apt-get install swatch
```

Swatch no arranca sin un fichero de configuración. Cuando un usuario ejecuta el proceso, Swatch busca el archivo *.swatchrc* en el directorio home del usuario. Podemos especificar un archivo concreto con la opción -c.

Este fichero consiste en describir una cadena de caracteres que se buscará en el archivo de logs, seguido de la acción que se ejecutará cuando se encuentre dicha cadena. Las cadenas se especifican de la siguiente manera:

```
watchfor /ATTACK-RESPONSES/
```

Esto buscará la cadena ATTACK-RESPONSES en nuestro archivo de logs de alertas de Snort. A continuación, tenemos que decirle qué hará Swatch cuando lo encuentre. Las opciones existentes son:

echo	Envía el texto de la línea encontrada a la salida estándar.
bell [n]	Hace sonar un beep en el pc seguida de n, donde es el número de veces que repite dicho beep.
exec command	Ejecuta un comando.
mail [addresses=...],[subject=...]	Envía la alerta por correo.
throttle hours: minutes:seconds	La opción throttle limita el número de veces que una acción específica se ejecuta para un patrón encontrado.
when=day_of_week:hour_of_day	Limita la ejecución a un número de veces.

A continuación mostramos un fichero .swatchrc de ejemplo que hemos creado en nuestro dispositivo. Buscaremos los patrones Priority que se registran con las alertas, y dependiendo de la prioridad de la alerta le asignaremos una acción u otra.

```
watchfor /Priority\: 1/
echo
mail addresses=Nuestrocorreo@nuestrodominio.com,subject=Snort_Alert,when=2.6:8-17
# Envía un correo a nuestro administrador durante las horas de trabajo
```

Otro ejemplo podría ser:

```
watchfor  /Priority\: 1/
echo
mail addresses=Nuestrocorreo@nuestrodominio.com,subject=Snort_Alert
throttle 0:10:0
# Envía un correo a nuestro administrador a cualquier hora, la opción throttle ejecutará la opción ante una alerta encontrada cada 10 minutos.

watchfor /Priority\: 2/
echo
bell 5
# Si se encuentra una alarma de prioridad 2 el PC hará beep 5 veces.
```

Para comprobar que este fichero funciona correctamente ejecutaremos Swatch indicándole el fichero que debe monitorizar. Las opciones en línea de comandos de Swatch son las siguientes:

Opción	**Descripción**
-c	Especifica un archivo de configuración distinto al /.swatchrc.
-t	Le dice a Swatch en qué fichero buscará los patrones.

-daemon	Arranca Swatch en modo demonio.
-p command	Si queremos que Swatch busque cadenas en la salida de un comando en lugar de en un archivo de logs.
-f filename	Se usa esta opción si queremos utilizar Swatch para leer un fichero que ya ha sido escrito en lugar de uno que se escribe en tiempo real.

En nuestro caso arrancaremos Swatch mediante:

```
swatch -c /home/swatchrc -t /var/log/snort/alert --daemon
```

Y listo, en nuestro correo veremos aparecer alarmas de esta forma:

```
De root. <root@debian>
asunto: Snort Alert
Texto:
[**] [1:0:0] Priority 1 [**]
```

2.4.2.4 Otras Herramientas de Seguridad

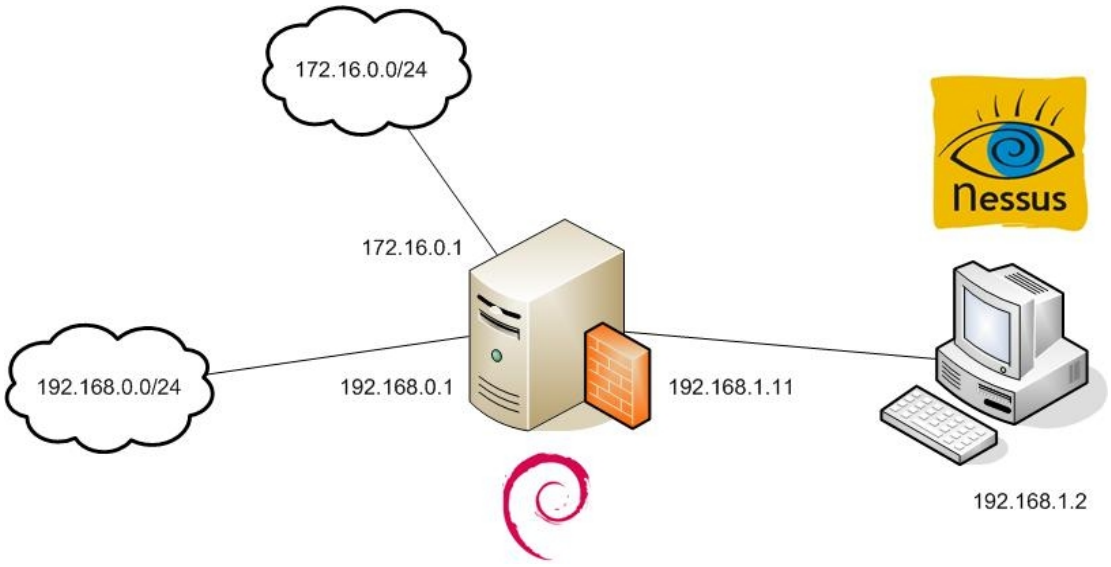
A continuación veremos algunas herramientas útiles para nuestro dispositivo Debian relacionadas con el tema de la seguridad:

Vulnerability Scanner NESSUS: Nessus es un programa de escaneo de seguridad en diversos sistemas operativos. Consiste en *nessusd,* el daemon Nessus, que realiza el escaneo en el sistema operativo, y *nessus*, el cliente (basado en consola o gráfico) que muestra el avance y reporte de los escaneos. Desde consola *nessus* puede ser programado para hacer escaneos programados con cron.

En operación normal, *nessus* comienza escaneando los puertos con namp o con su propio escaneador de puertos para buscar los puertos abiertos y después intentar varios exploits para atacarlos. Las pruebas de vulnerabilidad, disponibles como una larga lista de plug-ins, están escritas en NASL (Nessus Attack Scripting Language o Lenguaje de Scripting de Ataque Nessus), un lenguaje de scripting optimizado para interacciones personalizadas en redes.

Opcionalmente, los resultados del escaneo pueden ser exportados en reportes con varios formatos, como texto plano, XML, HTML, y LaTEX. Los resultados también pueden ser guardados en una base de conocimiento para referencia en futuros escaneos de vulnerabilidades. Algunas de las pruebas de vulnerabilidades de Nessus pueden causar que los servicios o sistemas operativos se corrompan y caigan. El usuario puede evitar esto desactivando “unsafe test” (pruebas no seguras) antes de escanear.

Para auditar nuestro dispositivo Debian, instalaremos tanto el servidor como el cliente de Nessus en nuestra máquina de escritorio con Ubuntu, tal como mostramos a continuación:




```
apt-get install nessusd
```

El archivo de configuración se encuentra en /etc/nessus/nessusd.conf y podremos especificar cosas como el rango de puertos en los que se realizará el escaneo, etc.. Veamos algunas líneas del mismo:

```
# Nessus Security Scanner, Debian default configuration file
#
# Empty lines and those starting with '#' are ignored.

# Directory where plug-ins are to be found
plugins_folder = /var/lib/nessus/plugins

....
```

```
# File that contains rules database that apply to all users
rules = /etc/nessus/nessusd.rules

# Remote file that the plugins will try to read:
test_file = /etc/passwd

# Range of the ports that nmap will scan
port_range = 1-15000

...
```

En nuestro caso lo dejaremos por defecto y ejecutaremos el servidor mediante:

```
nessusd -D
```

Además, necesitaremos crear un usuario y un password para identificar al cliente cuando se conecte al demonio nessusd, para ello procedemos haciendo:

```
nessus-adduser
```

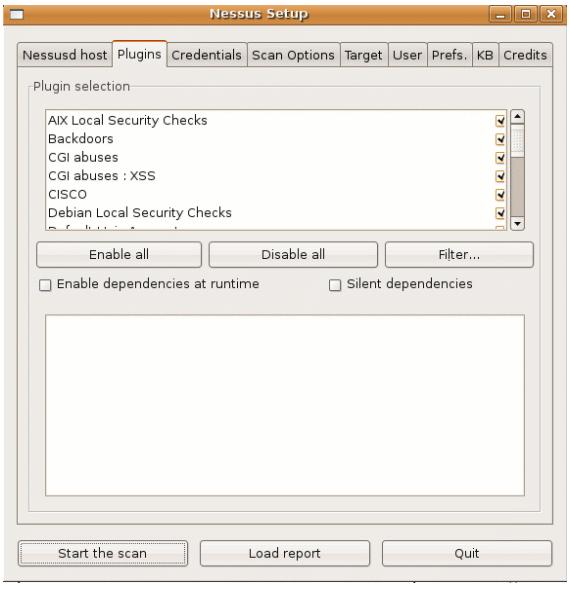
Ahora desde la misma máquina con Ubuntu instalaremos y ejecutaremos el cliente haciendo:

```
apt-get install nessus
```

Y abriremos la interfaz simplemente escribiendo:

```
nessus
```

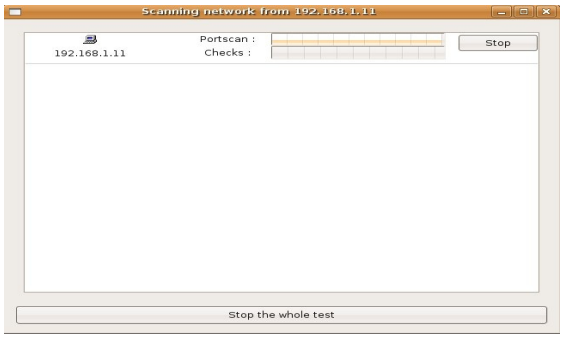
El aspecto que presenta el interfaz es tal que así:



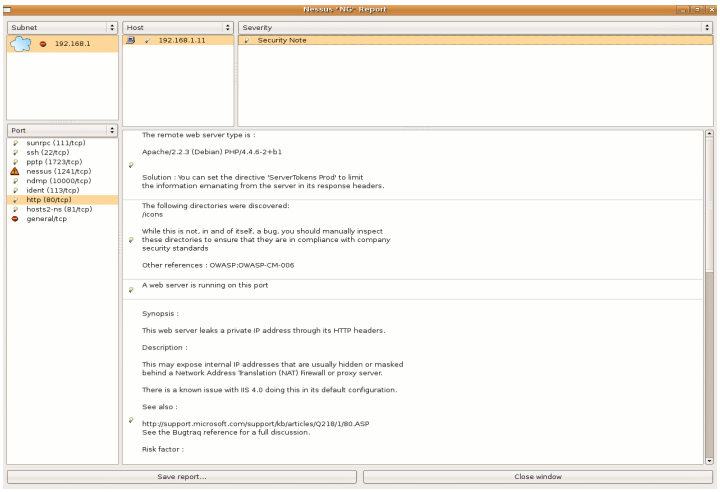
La pestaña de Plug-ins son los ataques que se pueden seleccionar para comprobar las vulnerabilidades. En Scan Options podemos especificar el rango de puertos a escanear así como el programa escáner de puertos a utilizar (nmap o uno propio de nessus).

En Target Selection indicamos la dirección IP de la máquina a escanear (192.168.1.11). En la pestaña nessusd host se indican los datos que se han añadido en la creación de usuario (login/password, etc) y conectarse al servidor de nessus.

Una vez especificadas las opciones (las dejaremos por defecto la mayoría) comenzamos el análisis:



Cuando finaliza Nessus nos muestra una pantalla con los resultados obtenidos como la que mostramos a continuación, sugiriéndonos actividades que podemos realizar para mejorar la seguridad de nuestro sistema.



En nuestro caso particular, simplemente obtenemos dos avisos de información sobre un servidor ssh encontrado en el puerto 22/TCP y otro de PPTP en el puerto 1723. Nessus no nos detecta ninguna vulnerabilidad. Más datos sobre el análisis son mostrados a continuación:

```
Nassus version: 2.2.6
Plug in version: 200511080815
Type of plug in feed: GPL only
Scanner IP: 192.168.1.2
Port scanner: nessus_tcp_scanner
Port range: 1-15000
...
```

Si por contra, hubiésemos realizado la misma prueba desactivando el cortafuegos iptables, Nessus detectaría vulnerabilidades en nuestro sistema. Por ejemplo, detectaría que tenemos un servidor Apache escuchando en el puerto 80 (el servidor apache que es utilizado por el front-end ACID de Snort).

Como esta situación no es la habitual, es decir, siempre tendremos iptables activado y además no queremos tener acceso a ACID desde el exterior de nuestra red, no nos centraremos en reforzar nuestro sistema ante esas vulnerabilidades, que podrían corregirse por ejemplo actualizando a la última versión de Apache y activar distintos mecanismos de seguridad en la configuración del mismo.

Tripwire: Una forma cómoda de detectar ataques locales (y también de red) en nuestro sistema es ejecutar un programa que verifique la integridad de la información almacenada en los ficheros, como Tripwire. El programa Tripwire ejecuta varios cheksums de todos los archivos binarios importantes y ficheros de configuración y los compara con una base de datos con valores de referencia aceptados como buenos. Así se detecta cualquier cambio en los ficheros.

Cuando alguien logra acceder a nuestro sistema no deseará perder la oportunidad de seguir utilizándolo, por lo que probablemente instalará algo llamado **rootkit**. Estos *rootkit,* vistos de una manera muy sencilla, son un conjunto de archivos destinados a reemplazar programas del sistema, creando versiones modificadas para que el administrador no descubra la presencia de procesos extraños, o logs de acceso de usuarios desconocidos. Por ejemplo, instalan un programa en lugar del pstree original, de modo que cuando se ejecute pstree, no se mostrarán todos los procesos que se están ejecutando. Como puede verse si el administrador no puede confiar en su propio sistema, será muy difícil determinar si éste ha sido comprometido.

Tripwire puede ser una de las mejores herramientas para detectar intrusos, con él al monitorizar cómo se modifican los ficheros del sistema podemos discernir lo que es actividad de un cracker y lo que es actividad normal del sistema.

Para instalar Tripwire en nuestra máquina Debian:

```
apt-get install tripwire
```

Durante la instalación se nos piden claves de administración, para tener permiso para poder generar la base de datos de los checksum. Posteriormente, ejecutaremos:

```
tripwire -m i
....
Wrote database file: /var/lib/tripwire/debian.twd
The database was successfully generated.
```

Y con ésto tendremos la base de datos generada. Cada vez que queramos comprobar el sistema de archivos teclearemos:

```
tripwire -m c
*** Processing Unix File System ***
Performing integrity check...
```

Y al final veremos un reporte que nos indicará si alguno de nuestros archivos ha cambiado desde la creación de la base de datos.

Chkrootkit: (Check Rootkit) es un programa informático para consola común en sistemas operativos GNU/Linux que permite localizar rootkits conocidos, realizando múltiples pruebas en las que busca entre los ficheros binarios modificados por dicho rootkit.

Como hemos comentado anteriormente, un rootkit es una herramienta, o un grupo de ellas (como ps, netstat, w o passwd) usadas para esconder los procesos y archivos que permiten al intruso mantener el acceso al sistema, a menudo con fines maliciosos. De este modo, el intruso podría mantener el control del sistema con privilegios de superusuario, pero quedando oculto a los ojos de los usuarios y administradores. Los rootkits son habitualmente considerados troyanos.

Básicamente chkrootkit hace múltiples comprobaciones para detectar todo tipo de rootkits y ficheros

maliciosos. Los nuevos rootkits pueden específicamente intentar detectar y comprometer copias del programa chkrootkit o tomar otras medidas para evitar la detección por ellas.

AIDE: (Entorno Avanzado de Detección de Intrusiones) es una libre alternativa a Tripwire. Genera una base de datos que puede ser usada para verificar la integridad de los archivos en el servidor. Puede usar una gran cantidad de algoritmos de verificación para asegurar que los archivos no han sido alterados. Aide se distribuye bajo licencia GPL.

PORTSENTRY: Portsentry es una herramienta de seguridad que permite monitorizar puertos, con el objetivo de detectar intentos de escaneo del sistema. Posee un mecanismo que permite bloquear no sólo el paquete, sino también el host, del cual proviene el paquete dado.

SWATCH: Como vimos antes al estudiar Snort, el WATCHer (ó SWATCH) utiliza archivos de registro generados por *syslog* para alertar a los administradores de las anomalías, basándose en los archivos de configuración del usuario. SWATCH fue diseñado para registrar cualquier evento que el usuario desee añadir en el archivo de configuración; sin embargo, ha sido adoptado ampliamente como un IDS basado en host.

LIDS: El Sistema de Detección de Intrusos Linux (Linux Intrusion Detection System, LIDS) es un parche del Kernel y una herramienta de administración que también puede controlar la modificación de archivos a través de las listas de control de acceso (ACLs) y proteger procesos y archivos, hasta del superusuario o root.

2.4.3 Redes Privadas Virtuales bajo GNU/Linux

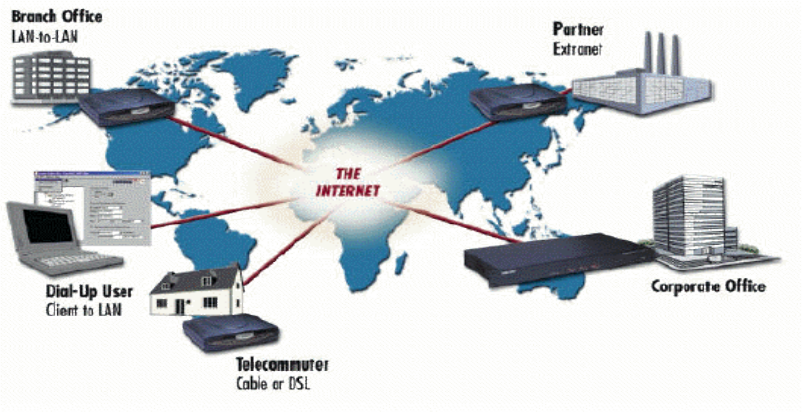
2.4.3.1 Introducción a las Redes Privadas Virtuales (VPN)

VPN es una tecnología de red que permite una extensión de la red local sobre una red pública o no controlada, como por ejemplo Internet.

Los ejemplos más comunes son la posibilidad de conectar dos o más sucursales de una empresa utilizando como vínculo Internet, permitir a los miembros del equipo de soporte técnico la conexión desde su casa al centro de cómputo, o que un usuario pueda acceder a su equipo doméstico desde un sitio remoto, como por ejemplo un hotel. Todo esto utilizando la infraestructura de Internet.

Para hacerlo posible de manera segura es necesario proveer los medios para garantizar la autenticación, integridad y confidencialidad de toda la comunicación:

- **Autenticación y autorización**: ¿Quién está al otro lado? Usuario/equipo y qué nivel de acceso debe tener.
- **Integridad:** La garantía de que los datos enviados no han sido alterados. Para ello se utiliza un método de comparación (Hash). Los algoritmos comunes de comparación son *Message Digest (MD)* y *Secure Hash Algorithm (SHA).*
- **Confidencialidad:** dado que los datos viajan a través de un medio potencialmente hostil como Internet, los mismos son susceptibles de intercepción, por lo que es fundamental el cifrado de los mismos. De este modo, la información no debe poder ser interpretada por nadie más que los destinatarios de la misma. Se hace uso de algoritmos de cifrado como *Digital Encryption Standard (DES), Triple DES(3DES) y Advanced Encryption Standard (AES).*
- **No repudio:** un mensaje tiene que ir firmado, y el que lo firma no puede negar que el mensaje lo envió él.

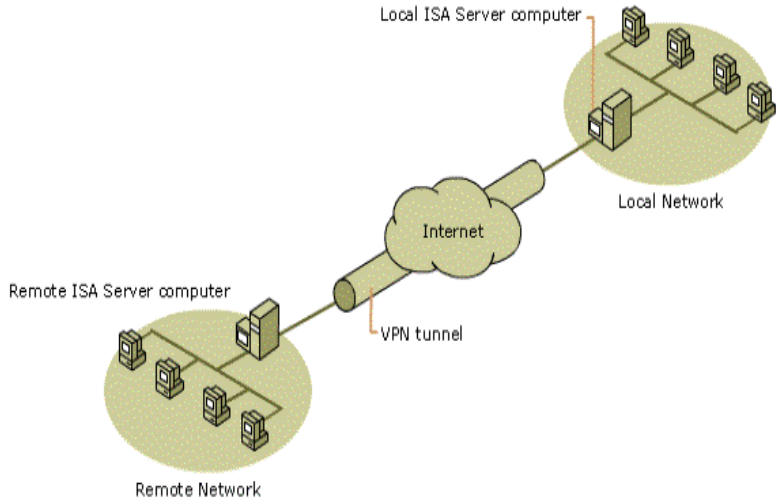


Básicamente existen tres arquitecturas de conexión VPN:

- **VPN de acceso remoto:** este es quizás el modelo más usado actualmente y consiste en usuarios o proveedores que se conectan con la empresa desde sitios remotos (oficinas comerciales, domicilios, hotel, aeropuertos, etcétera) utilizando Internet como acceso. Una vez autenticados tienen un nivel de acceso muy similar al que tienen en la red local de la empresa.
- **VPN punto a punto:** este esquema se utiliza para conectar oficinas remotas con la sede central de organización. El servidor VPN, que posee un vínculo permanente a Internet, acepta las conexiones vía Internet provenientes de los sitios y establece el túnel VPN.
- **VPN interna:** este esquema es el menos difundido pero uno de los más poderosos para utilizar dentro de la empresa. Es una variante del tipo “acceso remoto” pero, en vez de utilizar Internet como medio de conexión, emplea la misma red de área local (LAN) de la empresa. Sirve para aislar zonas y servicios de la red interna. Esta capacidad lo hace muy conveniente para mejorar las prestaciones de seguridad de las redes inalámbricas (WiFi). Un ejemplo muy clásico es un servidor con información sensible, como las nóminas de los sueldos, ubicado detrás de un equipo VPN, el cual provee autenticación adicional más el agregado del cifrado, haciendo posible que sólo el personal de RRHH habilitado pueda acceder.

Tunneling: Internet se construyó desde un principio como un medio inseguro. Muchos de los protocolos utilizados hoy en día para transferir datos de una máquina a otra a través de la red carecen de algún tipo de cifrado, o medio de seguridad, que evite que nuestras comunicaciones puedan ser interceptadas y espiadas. HTTP, FTP, POP3 y otros muchos protocolos ampliamente usados, utilizan comunicaciones que viajan en claro a través de la red. Esto supone un grave problema en todas aquellas situaciones en las que queremos transferir entre máquinas información sensible, como pueda ser una cuenta de usuario, y no tengamos un control absoluto sobre la red, a fin de evitar que alguien pueda interceptar nuestra comunicación por medio de la técnica de "man in the middle".

El problema de los protocolos que envían sus datos en claro, es decir, sin cifrarlos, es que cualquier persona que tenga acceso físico a la red en la que se sitúan nuestras máquinas puede ver dichos datos. Es tan simple como utilizar un sniffer que, básicamente, es una herramienta que pone nuestra tarjeta de red en modo promíscuo (modo en el que las tarjetas de red operan aceptando todos los paquetes que circulan por la red a la que se conectan, sean o no para esa tarjeta).



De este modo, alguien que conecte su máquina a una red y arranque un sniffer recibirá y podrá analizar, por tanto, todos los paquetes que circulen por dicha red. Si alguno de esos paquetes pertenece a un protocolo que envía sus comunicaciones en claro, y contiene información sensible, dicha información se verá comprometida.

Si por el contrario, ciframos nuestras comunicaciones con un sistema que permita entenderse sólo a las dos máquinas que son partícipes de la comunicación, cualquiera que intercepte desde una tercera máquina nuestros paquetes, no podrá hacer nada con ellos, al no poder descifrar los datos.

Una forma de evitar el problema que nos atañe a nivel de aplicación, sin dejar por ello de utilizar todos aquellos protocolos que carezcan de medios de cifrado, es usar una útil técnica llamada **tunneling**. Básicamente, esta técnica consiste en abrir conexiones entre dos máquinas por medio de un protocolo seguro, como puede ser SSH (Secure Shell), a través de las cuales realizaremos las transferencias inseguras, que pasarán de este modo a ser seguras.

De esta analogía viene el nombre de la técnica, siendo la conexión segura (en este caso de SSH) el túnel por el cual enviamos nuestros datos para que nadie más, aparte de los interlocutores que se sitúan a cada extremo del túnel, pueda verlos. Ni que decir tiene, que este tipo de técnica requiere de forma imprescindible que tengamos una cuenta de acceso seguro en la máquina con la que nos queremos comunicar.

Otra forma de evitar el problema anteriormente comentado es a nivel de red o incluso a niveles más bajos. El protocolo estándar es el IPSec, pero también tenemos PPTP, L2F, L2TP, SSL/TLS, etc. Cada uno con sus ventajas y desventajas en cuanto a seguridad, facilidad, mantenimiento y tipos de clientes soportados.

2.4.3.2 IPSec (Internet Protocol Security)

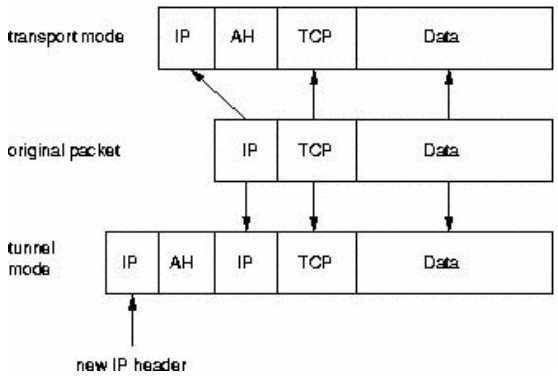
IPSec es una extensión al protocolo IP que añade cifrado fuerte para permitir servicios de autenticación, integridad, confidencialidad y no repudio y, de esta manera, asegurar las comunicaciones a través de dicho protocolo. Inicialmente fue desarrollado para usarse con el nuevo estándar IPv6, aunque posteriormente se adaptó a IPv4.

IPSec actúa a nivel de capa de red, protegiendo y autenticando los paquetes IP entre los equipos participantes en la comunidad IPSec. No está ligado a ningún algoritmo de cifrado o autenticación, tecnología de claves o algoritmos de seguridad específico. Es más, IPSec es un marco de estándares que permite que cualquier nuevo algoritmo sea introducido sin necesitar de cambiar los estándares.

IPSec está formado por un conjunto de protocolos de cifrado para asegurar flujos de paquetes de datos e intercambiar claves. Añade dos campos en la cabecera de los paquetes IP.

- **Encapsulating Security Payload (ESP)**, que provee autenticación, confidencialidad de datos e integridad del mensaje.
- **Authentication Header (AH)**, que provee de autenticación e integridad de datos, pero no de confidencialidad.

Existen dos modos de operación de IPSec: modo transporte y modo túnel. En modo túnel el datagrama IP se encapsula completamente dentro de un nuevo datagrama IP que emplea el protocolo IPSec. En modo transporte IPSec sólo maneja la carga del datagrama IP, insertándose la cabecera IPSec entre la cabecera IP y la cabecera del protocolo de capas superiores.



Para proteger la **integridad** de los datagramas IP, los protocolos IPSec emplean códigos de autenticación de mensaje basados en resúmenes (HMAC - Hash Message Authentication Codes). Para el cálculo de estos HMAC los protocolos emplean algoritmos de resumen como MD5 y SHA. El HMAC se incluye en la cabecera del protocolo IPSec y el receptor del paquete puede comprobar el HMAC si tiene acceso a la clave secreta.

Para proteger la **confidencialidad** de lo datagramas IP, los protocolos IPSec emplean algoritmos estándar de cifrado simétrico. El estándar IPsec exige la implementación de NULL y DES. En la actualidad se suelen emplear algoritmos más fuertes: 3DES, AES y Blowfish.

Para protegerse contra ataques por **denegación de servicio**, los protocolos IPSec emplean ventanas deslizantes. Cada paquete recibe un número de secuencia y sólo se acepta su recepción si el número de paquete se encuentra dentro de la ventana o es posterior. Los paquetes anteriores son descartados inmediatamente. Esta es una medida de protección eficaz contra ataques por repetición de mensajes en los que el atacante almacena los paquetes originales y los reproduce posteriormente.

Para que los participantes de una comunicación puedan encapsular y desencapsular los paquetes IPSec, se necesitan mecanismos para almacenar las claves secretas, algoritmos y direcciones IP involucradas en la comunicación. Todos estos parámetros se almacenan en asociaciones de seguridad (SA - Security Associations). Las asociaciones de seguridad, a su vez, se almacenan en bases de datos de asociaciones de seguridad (SAD - Security Asocciation Databases).

Cada asociación de seguridad define los siguientes parámetros:

- Dirección IP origen y destino de la cabecera IPSec resultante. Estas son las direcciones IP de los participantes de la comunicación IPSec que protegen los paquetes.
- Protocolo IPSec (AH o ESP).
- El algoritmo y clave secreta empleados por el protocolo IPSec.
- Índice de parámetro de seguridad (SPI - Security Parameter Index). Es un número de 32 bits que identifica la asociación de seguridad.

Algunas implementaciones de la base de datos de asociaciones de seguridad permiten almacenar más parámetros:

- Modo IPSec (túnel o transporte)
- Tamaño de la ventana deslizante para protegerse de ataques por repetición.
- Tiempo de vida de una asociación de seguridad.

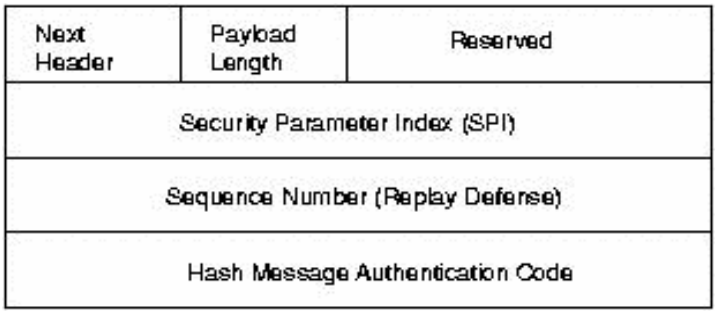
En una asociación de seguridad se definen las direcciones IP de origen y destino de la comunicación. Por ello, mediante una única SA sólo se puede proteger un sentido del tráfico en una comunicación IPSec full duplex. Para proteger ambos sentidos de la comunicación, IPSec necesita de dos asociaciones de seguridad unidireccionales.

Las claves secretas y algoritmos de cifrado deben compartirse entre todos los participantes de la VPN. Uno de los problemas críticos a los que se enfrenta el administrador de sistemas es el intercambio de claves: ¿cómo intercambiar claves simétricas cuando aún no se ha establecido ningún tipo de cifrado?

Para resolver este problema se desarrolló el protocolo de intercambio de claves por Internet (**IKE** - **Internet Key Exchange Protocol**). Este protocolo autentica a los participantes en una primera fase. En una segunda fase se negocian las asociaciones de seguridad y se escogen las claves secretas simétricas a través de un intercambio de claves Diffie Hellmann. Posteriormente lo veremos con más detalle. El protocolo IKE se ocupa incluso de renovar periódicamente las claves para asegurar su confidencialidad.

Veamos a continuación con más detalle los campos que el protocolo IPSec incorpora en las cabeceras:

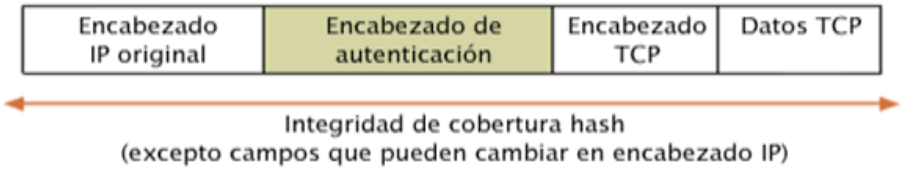
AH – Cabecera de autenticación: El protocolo AH protege la integridad del datagrama IP. Para conseguirlo, el protocolo AH calcula una HMAC basada en la clave secreta, el contenido del paquete y las partes inmutables de la cabecera IP (como son las direcciones IP). Tras esto, añade la cabecera AH al paquete. La cabecera AH se muestra en la siguiente figura:



Este HMAC protege la integridad de los paquetes ya que sólo los miembros de la comunicación que conozcan la clave secreta pueden crear y comprobar HMACs.

AH proporciona autenticación de origen de datos, integridad de datos y protección antieco para todo el paquete (tanto el encabezado IP, como la carga de datos que transporta el paquete), excepto los campos del encabezado IP que pueden cambiar en tránsito. AH no proporciona confidencialidad de datos, es decir, no cifra los datos. Los datos pueden leerse pero están protegidos contra modificaciones e imitaciones.

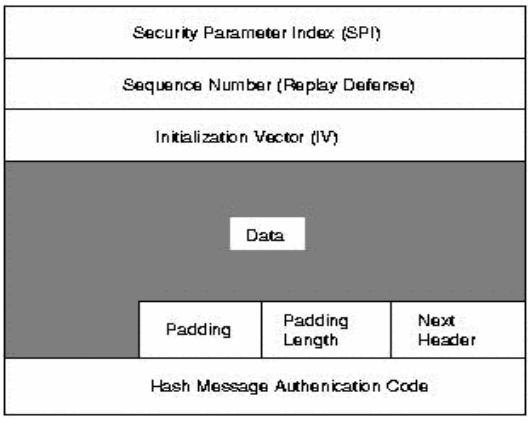
Como el protocolo AH protege la cabecera IP incluyendo las partes inmutables de la cabecera IP como las direcciones IP, el protocolo AH no permite NAT. NAT reemplaza la dirección IP de la cabecera por una IP diferente, tras el intercambio, la HMAC ya no es válida. La extensión a IPsec NAT-transversal implementa métodos que evitan esta restricción. En la práctica, casi nadie utiliza AH.



ESP - Carga de Seguridad Encapsulada: el protocolo ESP puede asegurar la integridad de los datos del paquete empleando una HMAC y la confidencialidad empleando cifrado. La cabecera ESP se genera y añade al paquete tras cifrarlo y calcular su HMAC.

ESP proporciona autenticación de origen de datos, integridad de datos, protección antieco y la opción de confidencialidad sólo para la carga IP. ESP en modo de transporte no protege todo el paquete con una suma de comprobación de cifrado. El encabezado IP no está protegido.

Esta HMAC sólo tiene en cuenta la carga del paquete: la cabecera IP no se incluye dentro de su proceso de cálculo. El uso de NAT, por lo tanto, no rompe el protocolo ESP. Sin embargo, en la mayoría de los casos, NAT aún no es compatible en combinación con IPSec.



El protocolo IKE: el protocolo IKE resuelve el problema más importante del establecimiento de comunicaciones seguras: la autenticación de los participantes y el intercambio de claves simétricas.Emplea el puerto 500 UDP para su comunicación.

El protocolo IKE funciona en dos fases. La autenticación de los participantes en la primera fase suele basarse en claves compartidas con anterioridad (PSK - Pre-shared keys), claves RSA y certificados X.509. En la segunda fase, el protocolo IKE intercambia propuestas de asociaciones de seguridad y negocia asociaciones de seguridad.

Existen algunas implementaciones de IPSec disponibles para Linux:

- **FreeS/wan:** esta fue la primera implementación de IPSec para Linux. Sin embargo, FreeS/wan ya no es un proyecto en desarrollo. Continuó en otros dos proyectos: Openswan y strongSwan.

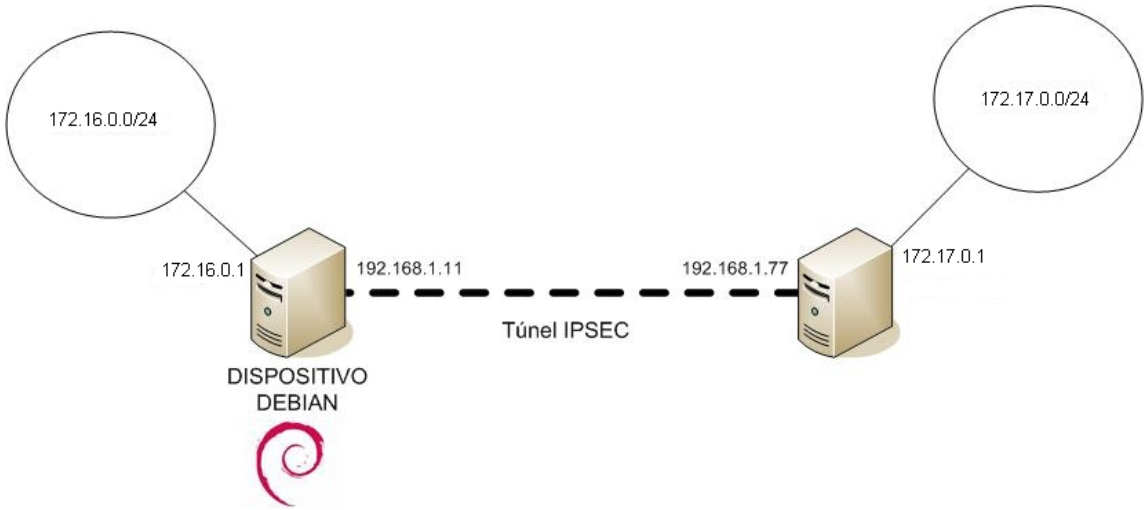
- **Openswan:** es una implementación open source de IPSec para linux. Es un código hijo del proyecto FreeS/Wan, realizado por el equipo de trabajo que formó la compañía Xelerance.

- **StrongSwan:** es también otra continuación del proyecto FreeS/WAN. Su principal autor es Andreas Steffen, el creador del parche sobre certificados X.509 para FreeS/Wan.

- **ipsec-tools**: está basado en KAME (implementación original de IPSec para BSD) y son unas herramientas utilizadas por defecto en muchas distribuciones. Su demonio de IKE se llama racoon.

2.4.3.3 Configurando VPN Punto a punto IPSec con Openswan

En este punto configuraremos una conexión VPN punto a punto con la herramienta Openswan. Para emular dicho escenario en nuestro laboratorio realizaremos el siguiente montaje con nuestras máquinas:



Para ello necesitamos dos gateways con Linux, Openswan instalado en ambos, y que detrás de ellos haya dos subredes IP que no se solapen. Para instalar Openswan desde la consola como *root* ejecutaremos:

```
apt-get install openswan
```

Además del paquete openswan, se instalarán varios más de los cuales depende. Es probable que si no teníamos instalado el paquete ca-certificates nos consulte acerca de confiar en ciertas Autoridades Certificantes, las respuestas son indistintas para nuestro caso.

Cuando le toque el turno de configuración a openswan nos preguntará por: Encriptación oportunista: permite formar túneles seguros sin demasiada administración en ambos extremos, esta opción no es necesaria para nuestro ejemplo, así que la desactivaremos. Creación de un certificado autofirmado: también descartaremos esta opción pues un certificado autofirmado no proporciona las garantías de seguridad que un certificado firmado por una CA sí proporciona.

El archivo */etc/ipsec.secrets* contiene tanto nuestras llaves RSA (privadas) como los PSK de otros nodos, las públicas de otros nodos no se especifican aquí. A continuación creamos una llave en cada servidor ejecutando:

```
ipsec newhostkey --output - >> /etc/ipsec.secrets
```

El archivo de configuración principal es el */etc/ipsec.conf* y en éste se declara un extremo como ***izquierdo*** y el otro como ***derecho***, quién es cada cual es indistinto y además es determinado en la ejecución. Esto permite tener una misma configuración en ambos extremos sin tener que hacer modificaciones a los ficheros de configuración. Para nuestro ejemplo usaremos el servidor en 192.168.1.11 como izquierdo y el situado en 192.168.1.77 como derecho.

Ahora determinamos la llave pública de cada extremo ejecutando en cada máquina:

```
debian:/etc# ipsec showhostkey --left
      # RSA 2192 bits   debian   Sun Apr  8 16:16:38 2007
      leftrsasigkey=0sAQPS/WzGT+/SVTSrQ5CNX+5NCrXLgNb3uMC1Il1JUDrfTyPRgrVSTdJM753wGHq
OEHA2gLwTgFsePtp9p2L/9r5cxoir6TK8B4z90Kml5Fs+Mv1IfqAn3wTa5FxVojVuOpZ5Dydtqgv9Cfmj+66Fb
oj639EuvLeMYhg8sv/44T4BKoMKaEbx8EEcOkPUCToehtfLQGOGnaHRaly74R7ZgQnmBA19vWnXaRd3k
y9YSEO67uxflAUWxcSIgQLHzQSFdeb7dTsRTL9WIzGWAj+pcJbSUu6B1l4qrIB8mCLdIdqn+SX0AKlYDwX
rOUnyZSVKUEOb3mVIKXSFXjvAFw1vpFg4Cc0A78J799tPpqj7tQU5vdtV
```

Que es la que hemos creado con el comando ipsec newhostkey. Es indiferente escribir la opción left o right.

El fichero */etc/ipsec.conf* está dividido en secciones.

```
### /etc/ipsec.conf ###
version 2.0     # conforms to second version of ipsec.conf specification
# basic configuration
config setup
     # Debug-logging controls:  "none" for (almost) none, "all" for lots.
     # klipsdebug=all
     # plutodebug=dns

#conn %default va aqui', antes de los include

#Disable Opportunistic Encryption
include /etc/ipsec.d/examples/no_oe.conf
```

Las secciones que empiezan con la palabra *conn* contienen una especificación de conexión, definiendo una conexión de red utilizando IPSec. Los parámetros que acompañan a conn en líneas separadas son:

left y right: la dirección IP del extremo izquierdo y derecho del túnel VPN.
leftsubnet y rightsubnet: la subred donde se encuentra el extremo izquierdo del túnel VPN.
rightrsasigkey y leftrsasigkey: son las claves públicas de ambos extremos.
auto=add: carga la definición de la conexión automáticamente.

Después editamos en ambos extremos el archivo /etc/ipsec.conf y agregamos al final:

```
conn mivpn
      left=192.168.1.11
      leftsubnet=172.172.0.0/24
      leftrsasigkey=0sAQPS/WzGT+/SVTSrQ5CNX+5NCrXLgNb3uMC1ll1JUDrfTyPRgrVSTdJ ...
      right=192.168.1.77
      rightsubnet=172.171.0.0/24
      rightrsasigkey=0sAQONgP5eT3jkR07YqOrSgYQprvgby5Sdqm3zwiI1yANSppfC9aGvEb$ ....
      auto=add
```

Que son los datos que definen el enlace. “mivpn” es un identificador de la conexión. “left” y “right” representan las máquinas que actuarán de “gateway” en nuestra conexión IPSec y “leftsubnet” y “rightsubnet” son las subredes que quedan enlazadas mediante el túnel.

Copiaremos dicha información en ambos gateways. Con esto ya tenemos todos los datos necesarios para echar a andar nuestro túnel VPN.

Los comandos necesarios para arrancar OpenSwan son los siguientes:

ipsec setup --start	Arranca el pluto daemon y setup asociado
ipsec auto --status	Muestra status de las conexiones VPN
ipsec auto --add nombre_conexión ipsec auto --delete nombre_conexión ipsec auto --replace nombre_conexión	Agregan/eliminan conexiones al demonio pluto que se está ejecutando, este comando se utiliza cuando se agregó una sección “conn” al fichero /etc/ipsec.conf y no se desea, o no se puede bajar el servicio ipsec para cargar la nueva conexión
ipsec auto –up nombre_conexión	Se le indica a pluto que arme el túnel nombre_conexión

Pluto es el nombre del demonio que gestiona las claves en Openswan.
Reiniciaremos Openswan en ambos servidores mediante:

```
/etc/init.d/ipsec restart
```

Y a continuación activamos la conexión “mivpn” mediante:

```
ipsec auto --add mivpn
```

Y probamos que levante manualmente:

```
debian:~# ipsec auto --up mivpn
104 "mivpn" #1: STATE_MAIN_I1: initiate
106 "mivpn" #1: STATE_MAIN_I2: sent MI2, expecting MR2
108 "mivpn" #1: STATE_MAIN_I3: sent MI3, expecting MR3
004 "mivpn" #1: STATE_MAIN_I4: ISAKMP SA established
112 "mivpn" #2: STATE_QUICK_I1: initiate
004 "mivpn" #2: STATE_QUICK_I2: sent QI2, IPsec SA established {ESP=>0xa02992a2 <0x8425bea1}
```

Con esto ya hemos establecido la asociación de seguridad (SA) IPSec. Ahora vamos a asegurarnos que nuestra conexión funciona, para ello hacemos un ping desde la máquina 172.16.0.2 hacia 172.17.0.1 y capturamos simultáneamente el tráfico en la interfaz eth1 de que nuestra máquina debian (192.168.1.11):

```
tcpdump -i eth1
```

Observamos:

```
20:20:05.554089 IP 192.168.1.11 > 192.168.1.77: ESP(spi=0xb16fec78,seq=0x7)
20:20:05.557435 IP 192.168.1.77 > 192.168.1.11: ESP(spi=0x5a16aa85,seq=0x7)
...
```

Es decir, paquetes ESP (Encapsulating Security Payload) moviendoše entre los dos gateways con la misma frecuencia que nuestro ping.

Para hacer que nuestro túnel comience al arrancar, cambiaremos en el fichero */etc/ipsec.conf* la línea *auto=add* por *auto=start*.

Mencionar que las pruebas las hemos realizado sin firewall activado. Si lo tuviéramos activado tendríamos que habilitar conexiones entrantes en los puertos 50 y 51, además del puerto udp 500:

```
iptables -A INPUT -p TCP --dport 50 -j ACCEPT
iptables -A INPUT -p TCP --dport 41 -j ACCEPT
iptables -A INPUT -p UDP --dport 5000 -j ACCEPT
```

Así mismo, a los paquetes que pasan por el túnel no debemos hacerle NAT; en nuestro caso no tenemos NAT activado en nuestro dispositivo, pero si lo tuviéramos, podríamos sustituir la regla que teníamos:

```
iptables -t nat -A POSTROUTING -s 172.16.0.0/24 -o eth1 -j ACCEPT
```

Por esta otra:

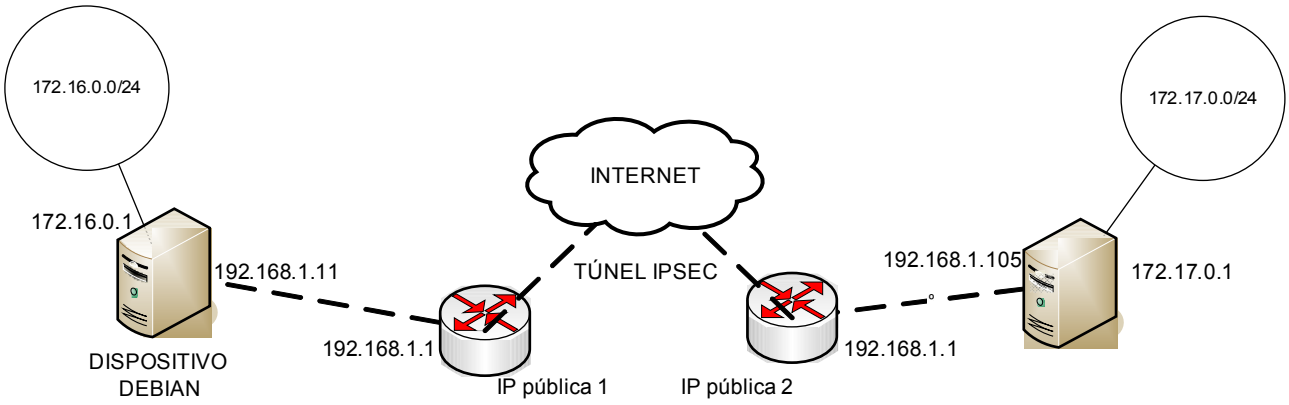
```
iptables -t nat -A POSTROUTING -s 172.16.0.0/24 -d ! 172.17.0.0 -o eth1 -j ACCEPT
```

De tal forma que a los paquetes que viajen hacia la otra red por el túnel no se les aplicará traducción de direcciones. Si tuviéramos algún dispositivo intermedio como algún encaminador que realizase traducción de direcciones NAT el enlace no podría establecerse correctamente, pues al encapsular los paquetes originales se perdería información relativa a las direcciones IP correctas de los nodos.

Para evitar este problema la solución es utilizar NAT traversal con Openswan. NAT Traversal es un método para encapsular los paquetes ESP de IPSec en paquetes UDP para facilitar el paso a través de Routers y Firewalls que emplean NAT. NAT Traversal ha sido publicado por el IETF en una serie de RFCs:

RFC 3947: el estándar oficial de NAT-T.
RFC 3715: establece requisitos para la RFC de NAT-T.
RFC 3948: encapsulando paquetes ESP de IPSec dentro de UDP.

Un esquema de red más complejo que probamos en nuestro laboratorio fue el siguiente:



Se trata de unir dos redes remotamente mediante un túnel IPSec a través de Internet. En los extremos tenemos un router ADSL seguido de nuestros dispositivos GNU/Linux con Openswan instalado. En esta configuración tenemos que abrir el puerto de NAT traversal, 4500 en UDP:

```
iptables -A INPUT -p UDP --dport 4500
```

En este caso para probar otro método de autenticación en lugar de utilizar las claves RSA, utilizaremos una clave compartirda entre ambos extremos o PSK (PreShared Key).

El archivo de configuración */etc/ipsec.conf* nos queda por tanto tal que así:

```
config setup
        nat_traversal=yes
        interfaces=”ipsec0=eth0”

conn mivpn
        left=192.168.1.11
        leftid=192.168.1.11
        leftnexthop=192.168.1.1
        leftsubnet=172.16.0.0/24
        right=IP_publica2
        rightid=192.168.1.105
        rightsubnet=192.168.1.0/24
        righnexthop=192.168.1.1
        authby=secret
        auto=add
```

Siendo análogo en el otro extremo, cambiando la dirección pública y del dispositivo de acuerdo con el esquema de red anterior. *Leftid* y *rightid* designan cómo los participantes deben de ser especificados en la autenticación extremo a extremo, es decir, la dirección IP de los extremos a la hora de autenticarse. *rightnexthop* y *leftnexthop* es la dirección IP del siguiente nodo del dispositivo GNU/Linux hacia la red pública.

Con *authby=secret* activamos una autenticación de secreto compartido. Habrá que editar el archivo ipsec.secrets y añadir nuestra clave PSK:

```
192.168.1.11 any : PSK “prueba_clave”
```

Una vez hecho esto reiniciamos ipsec y establecemos el túnel:

```
/etc/init.d/ipsec restart
ipsec auto --add mivpn
ipsec auto --up mivpn
```

Podemos ver el estatus de la conexión con:

```
Ipsec auto --status
```

Vemos el túnel establecido:

```
000 #2: "mivpn" STATE_QUICK_I2 (sent QI2, IPsec SA established); EVENT_SA_REPLACE in 27678s;
newest IPSEC; eroute owner
000 #2: "mivpn" esp.1de74679@IP_publica2 esp.9411d5a8@192.168.1.11 tun.0@ IP_publica2
tun.0@192.168.1.11
000 #1: "mivpn" STATE_MAIN_I4 (ISAKMP SA established); EVENT_SA_REPLACE in 2743s; newest
ISAKMP
```

Y con una prueba de ping comprobamos que tenemos conectividad entre las redes.

2.4.3.4 PPTP (Point-To-Point Tunneling Protocol)

Point-To-Point Tunneling Protocol (PPTP) es un protocolo de red creado por Microsoft que permite la realización de transferencias seguras desde clientes remotos a servidores emplazados en redes privadas, empleando para ello tanto líneas telefónicas conmutadas como Internet.

PPTP encapsula los paquetes PPP en datagramas IP. Una vez que los datagramas llegan al servidor PPTP, son desensamblados con el fin de obtener el paquete PPP y desencriptados de acuerdo al protocolo de red transmitido. Por el momento, PPTP únicamente soporta los protocolos de red IP, IPX y NetBEUI. El protocolo PPTP especifica además una serie de mensajes de control con el fin de establecer, mantener y destruir el túnel PPTP. Estos mensajes son transmitidos en paquetes de control en el interior de segmentos TCP. De este modo, los paquetes de control almacenan el mensaje de control PPTP, la cabecera TCP, la cabecera IP y los trailers apropiados.

La autenticación remota de clientes PPTP es realizada empleando los mismos métodos de autenticación utilizados por cualquier otro tipo de servidor de acceso remoto (RAS). En el caso de Microsoft, la autenticación utilizada para el acceso a los RAS soporta los protocolos CHAP(Challenge-Handshake Authentication Protocol), MS-CHAP, y PAP (Password Authentication Protocol).. Los accesos a los recursos NTFS o a cualquier otro tipo, precisa de los permisos adecuados, para lo cual resulta recomendable utilizar el sistema de ficheros NTFS para los recursos de ficheros a los que deben acceder los clientes PPTP.

En cuanto a la encriptación de datos, PPTP utiliza MPPE, encriptación de secreto compartido en el cual sólo los extremos de la conexión comparten la clave. Dicha clave es generada empleando el estándar RSA RC-4 a partir del password del usuario. La longitud de dicha clave puede ser 128 bits (para usuarios de Estados Unidos y Canadá) o 40 bits (para el resto de usuarios).

Por último, PPTP puede ser utilizado conjuntamente con cortafuegos y routers. La dificultad básica de ejecutar PPTP a través de un cortafuegos es que PPTP consiste en dos conexiones que se ejecutan a través de protocolos muy diversos. La conexión de control de PPTP se ejecuta a través del TCP vía acceso al puerto 1723 y los datos se envían sobre los paquetes UDP utilizando el protocolo 47 de IP. Los paquetes de GRE-UDP son paquetes UDP con el puerto 47 como destino.

2.4.3.5 Configurando un servidor PPTP

En esta sección del documento configuramos un servidor VPN bajo GNU/Linux utilizando el protocolo PPTP. Para la gestión de Redes Privadas Virtuales con PPTP existen diferentes herramientas, obviamente libres, pero sin duda el más usado es PPTPD (Point-to-Point Tunneling Protocol Daemon http://www.poptop.org) que oficia de servidor VPN y puede ser accesible tanto por clientes que corran Windows como también GNU/Linux. Los clientes GNU/Linux utilizan el programa pptp-linux, que nos permite acceder a servidores corriendo GNU/Linux, Windows NT o 2000 Server.

Comenzaremos instalando el paquete *pptpd:*

```
apt-get install pptpd
```

En primer lugar configuramos el servidor de VPN sin requerir cifrado MPPE. Nuestra máquina Debian usará dos interfaces: una pública donde recibe las peticiones y otra privada (192.168.0.1). El demonio pptpd se queda escuchando al puerto 1723 para peticiones de clientes. Una vez recibida una petición pasará el control de autenticación al demonio pppd.

El fichero de configuración */etc/pptpd.config* mostrará una apariencia similar a:

```
option /etc/ppp/pptpd-options
localip 192.168.0.234-238,192.168.0.245
remoteip 192.168.1.234-238,192.168.1.245
```

localip: IP que tendrá el servidor vpn (IP del túnel), esto puede ser un rango, por lo cual asignará (en el servidor) una IP por cada túnel que se genere. A veces para mejor control es mejor que el server tenga la misma IP siempre.

remoteip: rango de IPs que se asignan a los usuarios que se conecten, en caso de proveer una sola IP, permitirá un cliente por vez. Es importante tener en cuenta la cantidad de conexiones que se van a permitir, porque si se requiere asignar direcciones IP estáticas a usuarios (esto se hace en el */etc/ppp/chap-secrets*) las IPs deberán estar contempladas en el rango.

Ponemos una IP privada de la red del servidor de Linux como IP remota, será transparente para los equipos de la red. Podemos hacerlo así debido a que tenemos la opción proxyarp en /etc/ppp/pptp-options y en el cliente. Así el servidor pptp hará de servidor proxy, y aún sin estar en la misma red local será transparente al sistema operativo como si lo estuviera.

Ahora debemos editar el archivo */etc/ppp/pptp-options* que ha de tener como mínimo las siguientes opciones:

```
# Autenticación

# Nombre del sistema local para propósitos de autenticación
name pptpd

# Sistema de encriptación que se utilizará, en este caso PAP
require-pap

# Networking and Routing
# Si pppd actúa como un servidor para clientes de Microsoft Windows, esta opción permite
# a pppd prover uno o dos direcciones de servidores DNS a los clientes. Aquí especificamos
# un dns primario:
ms-dns 80.58.61.250

# La siguiente línea añade una entrada a la tabla ARP con la dirección IP del extremo y su dirección
# Ethernet. Esto tendrá el efecto de aparecer a los demás sistemas como estar en la misma red.
Proxyarp
```

```
# No sustituye la ruta por defecto en Debian
nodedefaultroute
```

y con esto quedaría configurado los servicios PPTPD, ahora pasamos a dar de alta los usuarios VPN. La configuración de usuarios se hace en el archivo pap-secrets o chap-secrets. No es necesario que el usuario que se conecte por VPN tenga que ser usuario de nuestro sistema Debian. La primera columna de estos ficheros corresponde al nombre de usuario (para autenticar dominios de windows, se debe poner el nombre de dominio más dos barras invertidas antes del usuario). En la segunda columna va el nombre del servidor (puede ir solamente un asterisco asumiento que es él mismo donde está corriendo pptp). En la tercera columna va el password. En la cuarta se fija la IP, en caso de asignar estáticamente una a un usuario, de lo contrario, con un asterisco (*) el servidor le dará a ese usuario una IP arbitraria que va a estar comprendida en el rango específico en /etc/pptpd.conf. Veamos un ejemplo:

```
# client   server      secret     IP address
fran          *           pfc          *
```

Habrá tantos usuarios especificados como conexiones se permitan realizar (no se pueden loguear dos usuarios al mismo tiempo).

Como tenemos activado iptables, debemos agregar algunas reglas para permitir que se establezca el túnel, los puertos que nos interesan son el 1723 y el 47. Añadimos, por tanto, a nuestro script *firewall.sh* las siguientes líneas:

```
iptables -A INPUT -p tcp –dport 1723 -j ACCEPT
iptables -A INPUT -p 47 -j ACCEPT
iptables -A OUTPUT -p tcp –sport 1723 -j ACCEPT
iptables -A OUTPUT -p 47 -j ACCEPT
```

Ahora sólo resta por levantar el demonio pptpd del siguiente modo:

```
/etc/init.d/pptpd restart
```

El servidor PPTPD se pondrá automáticamente en background escuchando por el puerto TCP 1723 en espera de conexiones entrantes.

Ahora bien, si queremos activar la encriptación MPPE debemos tener un kernel de Linux superior o igual a la versión 2.6.15-rc1. Si no es el caso tendremos que parchear y recompilar nuestro kernel para que soporte MPPE. En cualquier caso, actualmente la versión etch de Debian ya incluye 2.6.18 y solventamos este problema, además de trabajar con un kernel con más funcionalidades.

Por tanto, después de repetir los pasos anteriores en nuestra máquina Debian Etch, cargamos el módulo y comprobamos que el soporte MPPE está activado:

```
modprobe ppp-compress-18 && echo success
```

Obteniendo success. Ahora activaremos la encriptación de contraseña y datos en el archivo */etc/ppp/pptp-options:*

```
# Encriptación:
# La siguiente línea requiere que el extremo se autentique utilizando MS-CHAPv2 [Microsoft
# Challenge Handshake Authentication Protocol, version 2 ] authentication.
Require-mschap-v2
# La siguientes líneas requieren MPPE 128-bit y 40-bits
require-mppe-128
require-mppe-40
```

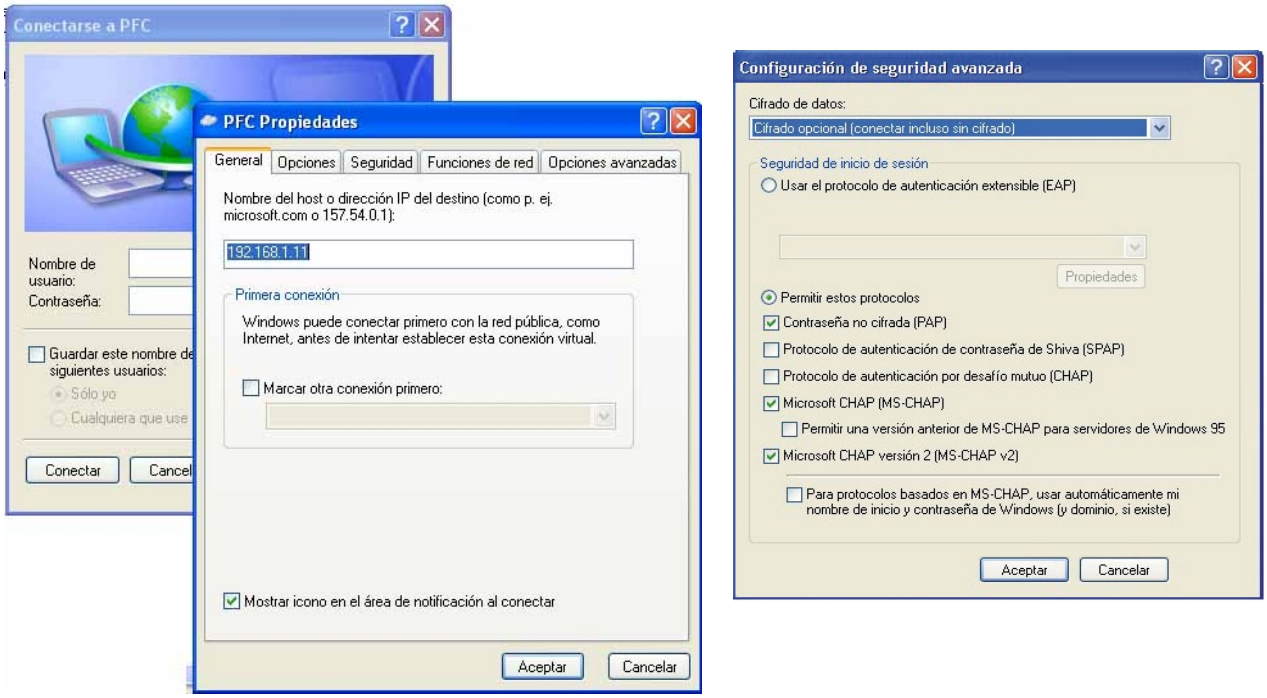
Después de esto desde nuestro cliente activaremos el cifrado de la contraseña y el de datos y podremos acceder a nuestro servidor Linux de forma más segura.

2.4.3.6 Configurando un cliente en GNU/Linux y Windows de PPTP

Para configurar un cliente desde **Windows** tendremos que hacer lo siguiente:

1. Hacemos click con el botón derecho en “Mis sitios de Red”
2. Doble-click en “Nueva Conexión”
3. Marcamos “Conectar a una red privada a través de Internet” y hacemos siguiente.
4. Nos pide la IP o el nombre del servidor pptp. Lo ponemos y “Siguiente”
5. Ponemos un nombre para la conexión.
6. Por último tecleamos el nombre de usuario que pusimos en pap-secrets ó chap-secrets.

Para conectarnos a un servidor con cifrado activado presionamos en “*Conectar*” (Por defecto viene el cifrado activado).



Si por el contrario el cifrado está desactivado tendremos que darle a “*Propiedades*”, pestaña “*Seguridad*”. “*Avanzada (Configuración personalizada)*” y hacemos click en “*Configuración*”. En la combobox de arriba seleccionamos “*Cifrado opcional (conectar incluso sin cifrado)*” y abajo permitimos “*Contraseña no cifrada PAP*” y aceptamos. Una vez hecho esto ya nos podremos conectar.

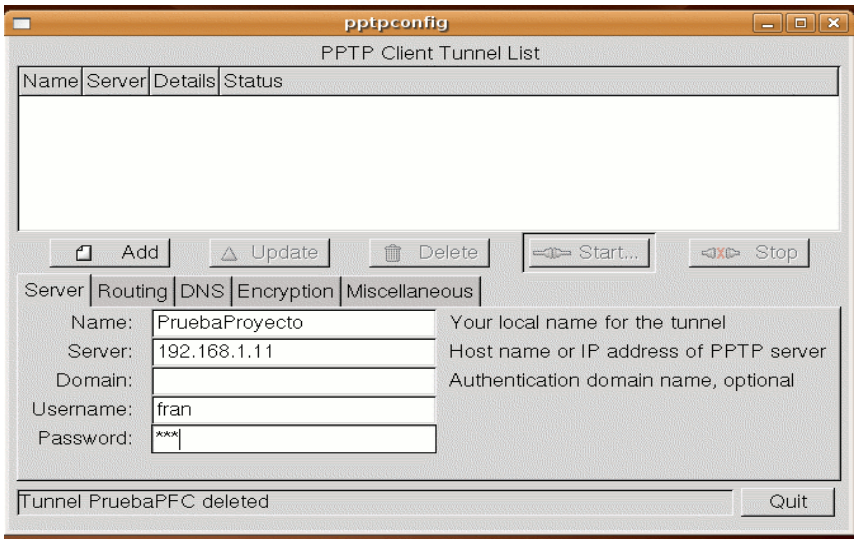
Para el cliente en **GNU/Linux** tendremos que instalar los paquetes pptp-linux y pptpd. Si bien podemos configurar el enlace mediante ficheros de texto ya que estamos trabajando desde una máquina con escritorio (en este caso, una máquina con Ubuntu) instalaremos un Frontend para pptp-linux muy intuitivo. Para ello modificamos el fichero de fuentes de la instalación */etc/apt/sources.list*:

```
#James Cameron’s PPTP GUI packaging
deb http://quozl.netrek.org/pptp/pptpconfig ./
```

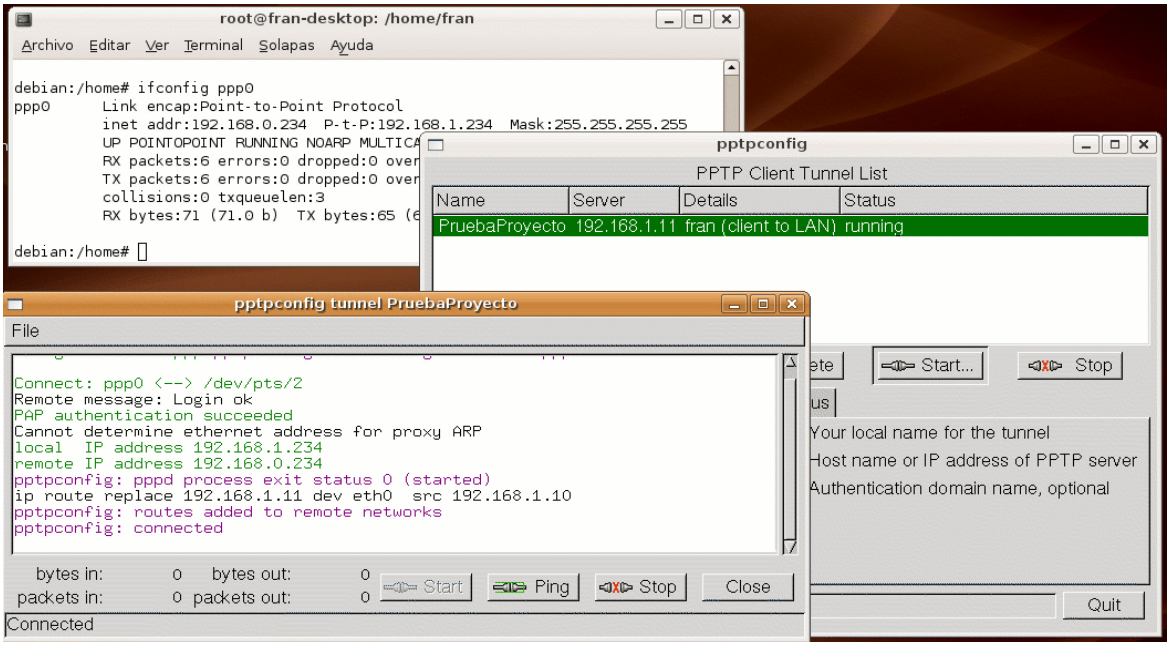
Actualizamos la lista de paquetes e instalamos el interfaz de cliente PPTP.

```
apt-get update
apt-get install pptpconfig
```

Ejecutamos *pptpconfig* como root (o sudo en su defecto):



Rellenamos las diferentes pestañas con información del enlace y pulsamos add para guardar la configuración y start para conectarnos. En el caso de obtener algún error podemos ver información de depuración avanzada activando la opción *Enable connection debugging facilities* en la pestaña *Miscellaneus.*



En la captura anterior observamos cómo se establece correctamente el túnel. Si ejecutamos ifconfig vemos que tenemos una interfaz ppp0 con la dirección IP que nos ha dado el servidor dentro de su rango de dirección para los clientes.

2.4.4 Control Remoto Seguro (Secure Shell)
2.4.4.1 Introducción e instalación de SSH

SSH (Secure Shell) es un programa de login remoto que permite una transmisión segura de cualquier tipo de datos: passwords, inicios de sesión, ficheros, sesiones gráficas remotas, comandos de administración, etc. Permite manejar por completo el ordenador mediante un intérprete de comandos, y también puede redirigir el tráfico X para poder ejecutar programas gráficos si tenemos un servidor X arrancado.

Además de la conexión con otras máquinas, SSH nos permite copiar datos de forma segura (tanto ficheros sueltos como simular sesiones FTP cifradas), gestionar claves RSA para no tener que escribir las claves al conectar a las máquinas (como veremos posteriormente) y pasar los datos de cualquier otra aplicación por un canal seguro tunelizado mediante SSH.

SSH trabaja de forma similar a como se hace con telnet. La diferencia principal es que SSH usa técnicas de cifrado que hacen que la información que viaja por el medio de comunicación vaya de manera no legible, y ninguna tercera persona pueda descubrir el usuario y contraseña de la conexión, ni lo que se escribe durante toda la sesión; aunque es posible atacar este tipo de sistemas por medio de ataques de REPLAY y manipular así la información entre destinos. Un ataque de REPLAY es una forma de ataque de red, el cual pretende capturar información y posteriormente reenviarla con el objetivo de falsificar la identidad de uno de los lados.

La primera versión del protocolo SSH fue desarrollado por un programador finlandés, Tatu Ylönen, que publicó su trabajo bajo una licencia libre. Pronto el éxito de su programa lo llevó a patentar la marca registrada “SSH” y crear una empresa con fines comerciales. Las siguientes versiones dejaron de ser libres y se permitió su empleo únicamente para usos no comerciales. Los desarrolladores de OpenBSD se dieron cuenta de que su sistema operativo, centrado en la criptografía y seguridad, se quedaba “cojo” sin SSH. Además, las encuestas afirmaban que lo primero que la mayoría de usuarios de OpenBSD añadía después de instalar el sistema era SSH. Con estas ideas a la cabeza y bastantes buenas intenciones, surgió el proyecto OpenSSH, cuyo principal objetivo consistió en desarrollar una implementación totalmente libre de SSH. Los primeros pasos de este proyecto estuvieron centrados en utilizar solamente código libre y portable. Todos estos esfuerzos no fueron vistos con tan buenos ojos por el desarrollador inicial de SSH, que veía como su proyecto comercial se iba al traste. Después de varias demandas judiciales y muchas discusiones, OpenSSH pudo mantener el “SSH” en su nombre y seguir con el trabajo que estaban realizando.

Existen en la actualidad dos versiones: SSH1 y SSH2. Es ampliamente aconsejable el uso de SSH2 (incorpora varias mejoras y es más seguro). Por lo tanto, en el resto del texto nos referiremos siempre a éste último.

Al conectarse un cliente de SSH con el servidor, se realizan los siguientes pasos:

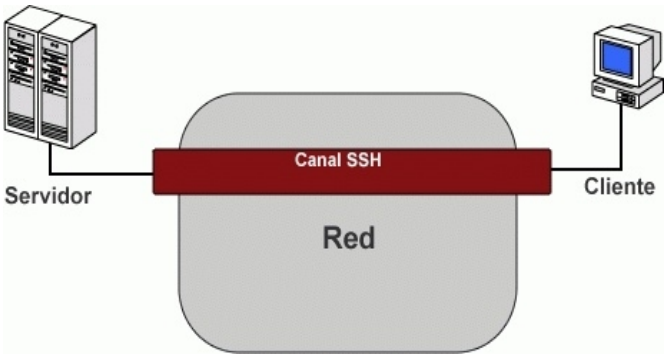
1. El cliente abre una conexión TCP al puerto 22 del host servidor.
2. El cliente y el servidor acuerdan la versión del protocolo a utilizar, de acuerdo a su configuración y capacidades.
3. El servidor posee un par de claves públicas/privada de RSA (llamadas claves de host). El servidor envía al cliente su clave pública.
4. El cliente compara la clave pública de host recibida con la que tiene almacenada, para verificar su autenticidad. Si no lo conociera previamente, pide confirmación al usuario para aceptarla como válida.
5. El cliente genera una clave de sesión aleatoria y selecciona un algoritmo de cifrado simétrico.
6. El cliente envía un mensaje conteniendo la clave de sesión y el algoritmo seleccionado, cifrado con la clave pública de host y del servidor usando el algoritmo RSA.
7. En adelante, para el resto de la comunicación se utilizará el algoritmo de cifrado simétrico seleccionado y clave compartida de sesión.
8. Luego se realiza autenticación del usuario. Aquí pueden usarse distintos mecanismos. Más adelante analizaremos los más importantes.
9. Finalmente se inicia la sesión, por lo general, interactiva.

El método más simple (y común) de autenticación es la autenticación con contraseña. El cliente solicita al usuario el ingreso de una contraseña (password) y la misma es enviada al servidor, el cual validará la misma utilizando los mecanismos configurados en el sistema (generalmente, utilizará la autenticación del sistema Linux para validar la contraseña contra el contenido del archivo /etc/shadow, de la misma manera que lo hace el login)

Las desventajas de este sistema son varias:
El usuario debe tipear su contraseña cada vez que se conecta al servidor.
La contraseña del usuario es enviada hacia el servidor.

Otra forma es la autenticación con clave pública. Para poder llevarse a cabo, el usuario debe tener un par de de claves públicas/privada, y la clave pública debe estar almacenada en el servidor. Luego de establecida la sesión, el servidor genera un número aleatorio llamado “desafío” (challenge) que es cifrado con la clave pública del usuario usando RSA ó DSA (la primera es para ssh versión 1 y la DSA para protocolo ssh de versión 2). El texto cifrado es enviado al cliente, que debe descifrarlo con la clave privada correspondiente y devolverlo al servidor, demostrando de esta manera que el usuario es quién dice ser.

Las ventajas de este método respecto del anterior son:
El usuario no debe tipear (ni recordar) ninguna contraseña.
Ninguna contraseña secreta (en este caso, la clave pública del usuario) es enviada al servidor.



Para instalar SSH en Debian ejecutaremos el siguiente comando:

```
apt-get install ssh
```

Una vez hecho esto, veremos cómo se generan las claves pública/privada de forma automática y se inicia el servicio sshd. A partir de estos momentos podremos acceder de forma remota a nuestra máquina haciendo uso de un cliente ssh, disponible prácticamente para cualquier plataforma.

Para ello tenemos que asegurarnos que en nuestro firewall tenemos escrita la regla que abre el puerto 22 de nuestro dispositivo. En nuestro caso dejábamos acceder sólo al equipo que tenía la MAC del administrado de sistemas y perteneciente a la red local:

```
iptables -A INPUT -m mac --mac-source  00:11:D8:42:DE:86 -s 192.168.0.0/24 -p TCP –dport 22 -j ACCEPT
```

Los ficheros de configuración de ssh se encuentran en */etc/ssh*, de los cuales, *sshd_config* será el encargado de la configuración del servidor. En este fichero podremos configurar diversos parámetros como el puerto en el que escuchará el demonio, el tipo de autenticación que usaremos, si permitiremos o no que el usuario root se conecte al servicio, etc.

Analizaremos las opciones más relevantes del fichero **sshd_config**:

port: esta opción permite especificar el puerto TCP que utilizará el servidor. El valor usual es 22.
protocol: versión del protocolo a utilizar. Usaremos solamente el valor 2.
HostKey: Clave privada de RSA o DSA del host. Normalmente las claves de host son generadas en el momento de la instalación de OpenSSH, y el valor de esta opción es */etc/ssh/ssh_host_rsa_key.*

PubkeyAuthentication: Si el valor de esta opción es yes, entonces se permite la autenticación de usuarios mediante clave pública.
AuthorizedKeysFile: Mediante esta opción se indica al servidor en dónde están almacenadas las claves públicas de los usuarios. El valor por defecto es *%h/.ssh/authorized_keys*, e indica que deben buscarse en el archivo *authorized_keys*, del directorio .ssh dentro del directorio home de cada usuario.
PasswordAuthentication: Si el valor de esta opción es yes, se permite la autenticación de usuarios mediante contraseñas.
X11Forwarding: Si el valor de esta opción es yes, se habilita el reenvío de X11 a través de la conexión SSH.

Por otro lado, en el archivo **ssh_config**, se describe la configuración del cliente SSH. Las opciones más importantes son:

Host: esta opción actúa como un divisor de sección. Puede repetirse varias veces en el archivo de configuración. Su valor, que puede ser una lista de patrones, determina que las opciones subsiguientes sean aplicadas a las conexiones realizadas a los hosts en cuestión. El valor * significa todos los hosts.
Port: es el puerto de TCP que utiliza el servidor (normalmente, 22).
protocol: es la versión del protocolo utilizada (se recomienda solamente 2).
PubkeyAuthentication: autenticación mediante clave pública (se recomienda yes).
IdentifyFile: archivo que contiene la clave pública.
PasswordAuthentication: Autenticación mediante contraseñas (yes o no).
StrictHostKeyChecking: define qué hará el cliente al conectarse a un host del cual no se dispone de su clave pública. El valor no hace que sea rechazada la clave del servidor (y por lo tanto, se aborte la conexión), el valor yes hace que se acepte automáticamente la clave recibida, y el valor ask hace que se pida confirmación al usuario.
Ciphers: Algoritmos de cifrado simétrico soportados para su uso durante la sesión.
ForwardX11: Reenvío de aplicaciones X11 (los posibles valores son yes o no)

2.4.4.2 Trabajando con SSH

Una vez hayamos realizado la instalación y configuración podremos conectarnos a nuestra máquina de forma remota. Para ello utilizaremos el cliente ssh que instalamos anteriormente. Su sintaxis básica es:

ssh [usuario@]host

Si el nombre de usuario en el host remoto (servidor) coincide con el nombre de usuario en el host local (cliente), podemos omitirlo. Por ejemplo, el siguiente comando:

root# ssh servidor.midominio

intentará iniciar una sesión interactiva (shell) en el host remoto servidor.midominio usando el usuario root, en tanto que:

root# ssh fran@servidor.midominio

intentará iniciar una sesión interactiva en el host remoto *servidor.midominio* usando el usuario *fran*.

Otra forma de especificar el usuario es con el parámetro '-l'. Veamos un ejemplo en nuestra máquina Debian:

```
root@fran-desktop:/# ssh -l root 192.168.1.35
The authenticity of host '192.168.1.35 (192.168.1.35)' can't be established.
RSA key fingerprint is b0:b7:42:95:8e:9f:cc:b9:a7:73:31:e3:9f:b2:8c:0b.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.35' (RSA) to the list of known hosts.
Password:
```

Podemos observar como lo primero que nos aparece es si deseamos proseguir con la conexión pese a que no ha sido posible verificar la autenticidad de la máquina. Hemos de tener cuidado con esto, ya que podríamos estar conectando a una máquina no deseada (un posible intruso intermediario) y nuestras

claves serían interceptadas. Por tanto, aceptaremos la conexión si sabemos casi con seguridad que es la máquina a la que queremos acceder, o bien si previamente sabemos su clave pública y coincide con la especificada.

La primera vez que nos conectamos via SSH desde otro equipo tarda un poco más porque se tiene que escribir en el archivo */home/nuestro_usuario/.ssh/known_host* (o bien */root/.ssh/known_host* si somos root) una entrada de la clave pública para la IP del host destino.

Las subsiguientes veces que nos conectemos compara la clave gpg de la IP con la que tiene guardada en el archivo *known_host*, en caso de variar supone que el ordenador es distinto, por ejemplo porque el ordenador servidor ssh ha sido formateado o porque se ha apagado y sustituido por otro al que se le ha asignado la misma IP. Ese es el error **man-in-the-middle** y significa: *"el ordenador cuya IP es esa no es el mismo que yo conocía, puede ser que alguien se esté haciendo pasar por él"* y no nos deja volvernos a conectar salvo que borremos la entrada de esa IP en el archivo *known_host*.

Una vez establecida la conexión, dispondremos de una shell remota en la que podremos realizar nuestras tareas de administración habituales, con la tranquilidad que nos proporcionan las comunicaciones encriptadas. Al finalizar, tecleando exit (o presionando CTRL+D), se cerrará la sesión remota.

Si al intentar una nueva conexión a la dirección IP anterior el cliente recibe una clave pública diferente de la que tiene almacenada en known_hosts, (imaginemos que hemos cambiado la IP por otra de otra máquina que también tiene un servidor de SSH) nos aparecerá el siguiente mensaje de advertencia y se abortará la conexión:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
f8:76:e5:f9:17:6d:b9:ea:52:96:33:64:f8:1e:7e:1d.
Please contact your system administrator.
Add correct host key in /root/.ssh/known_hosts to get rid of this message.
Offending key in /root/.ssh/known_hosts:1
RSA host key for 192.168.1.35 has changed and you have requested strict checking.
Host key verification failed.
```

Otra utilidad de ssh es la transferencia de ficheros segura. En muchas ocasiones deseamos transferir una información sensible, que no debería ser leída por terceras personas o caer en manos ajenas. SSH nos puede ayudar en esto, ya que mediante el comando **scp**, nos permite copiar un fichero a un destino remoto a través de un canal seguro, sometido a una fuerte encriptación.

La sintaxis del comando scp es:

```
scp fichero usuario@maquina:/ruta/de/destino
```

Además de estas utilidades, SSH también nos ofrece la posibilidad de crear túneles seguros cifrados, muy útiles para asegurar otros protocolos, redireccionamiento de puertos, etc.

Siempre que conectemos a un ordenador remoto con SSH nos va a pedir el password de acceso para asegurarse de que tenemos permisos. Hay una forma de evitar que nos pida el password siempre haciendo que el ordenador remoto confíe en nosotros. Para ello hemos de generar un par de claves RSA que se usarán en la relación de confianza:

```
debian:~/.ssh# ssh-keygen -t dsa
```

El programa ssh-keygen permite generar y administrar claves para SSH. El comando ingresado sirve para crear un par de claves pública/privada de RSA (con una longitud por defecto de 1024 bits). El programa responderá con lo siguiente:

```
Generating public/private dsa key pair.
Enter file in which to save the key (/root/.ssh/id_dsa):
```

Nos solicita que ingresemos el nombre del archivo en donde se almacenará la clave privada, proponiéndonos si somos root el archivo /root/.ssh/id_rsa, lo cual concuerda con la configuración del cliente SSH. Presionando Enter se acepta el valor por defecto. Luego nos solicitará una frase clave:

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

Finalmente nos informa de:

```
Your identification has been saved in /root/.ssh/id_dsa.
Your public key has been saved in /root/.ssh/id_dsa.pub.
The key fingerprint is:
ea:e1:71:bb:58:26:61:2b:1d:b2:4d:b3:f9:ae:0d:ff root@debian
```

Una vez generada la copiamos al usuario del ordenador remoto con el que queremos mantener la relación de confianza usando el comando *ssh-copy-id*. Este es un ejemplo del uso con la salida del programa:

```
root@fran-desktop:/home/fran# ssh-copy-id root@192.168.1.11
Password:
Now try logging into the machine, with "ssh 'root@192.168.1.11'", and check in:

  .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting
```

ssh-copy-id es un script que se conecta a la máquina y copia el archivo (también es posible indicarlo con la opción -i) en */root/.ssh/authorized_keys*, y ajusta los permisos a los valores adecuados. Si no se dispone del programa *ssh-copy-id* se puede realizar una copia manual a la máquina remota del fichero conteniendo la clave pública (por ejemplo usando scp) y añadiendo su contenido al fichero */root/.ssh/authorized_keys*.

Este es el contenido del fichero *authorized_keys*:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEA/HmNKUqO9BLeGyKJmdBIac+j4Gzf6wUKIDhNvTDGn9IbTrA8e0Z
sHGZ0xmU25ILshu4XhPyL7a8U6+StXRKS3u5Yd ... cCfSvHNpLsiBqjl/EJZZPw== root@192.168.1.10
```

Después de haber hecho esto, ingresamos el comando:

```
ssh 192.168.1.11
```

El servidor nuevamente envía su clave pública RSA, la cual es comparada con la almacenada en *known_hosts*, y si coincide el proceso continúa. El cliente de SSH, al encontrar el archivo */root/.shh/id_rsa*, primero intentará la autenticación con clave pública. El servidor le enviará el *challenge* cifrado con la clave pública encontrada en */root/.ssh/authorized_keys* y el cliente deberá devolverla descifrada (usando la clave */root/.ssh/id_rsa*). Si esto re realiza correctamente, se iniciará la sesión remota. Si la autenticación con clave pública hubiera fallado, el cliente intentará con la autenticación con contraseña descrita anteriormente.

Como acabamos de ver, la utilización de una frase clave implicará que al intentar conectar con el servidor, el cliente la solicitará al usuario, debiendo éste teclearla. Para evitar esta molestia, se utiliza el *ssh-agent*. Este programa se ejecuta como un demonio y permite almacenar las claves privadas (descifradas) para su posterior uso por el cliente de SSH.

En una sesión de shell, podemos ejecutar el ssh-agent de la siguiente forma:

```
$ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-shqQfD2257/agent.2257; export SSH_AUTH_SOCK;
SSH_AGENT_PID=2258; export SSH_AGENT_PID;
echo Agent pid 2258;
```

Cuando el agente arranca, muestra información por pantalla que podemos usar para configurar nuestras variables del shell. El programa ssh usa la variable de entorno SSH_AUTH_SOCK para saber cómo contactar con el demonio de *ssh-agent* que estamos corriendo. Por tanto, después de iniciar *ssh-agent*, deberíamos configurar esas variables de entorno tal como nos lo indica. La manera más fácil de hacerlo es utilizando la función *eval* de bash, que ejecuta el comando (especificado en el argumento) y estableces las variables sucesivas.

```
eval `ssh-agent`
Agent pid 2244
```

Tras tener activado ssh-agent. Debemos agregar nuestra clave privada de RSA. Para ello usamos el comando ssh-add.

```
$ssh-add /root/.ssh/id_rsa
Need passphrase for /root/.ssh/id_rsa
Enter passphrase for /root/.ssh/id_rsa
Identity added: /root/.ssh/id_rsa (/root/.ssh/id_rsa)
```

Nos pregunta la contraseña y luego, siempre y cuando nos encontremos en el mismo shell en donde hemos iniciado el ssh-agent, al ejecutar ssh éste le solicitará al ssh-agent nuestra clave privada, con lo cual no deberemos introducir la frase clave.El agente sólo dura mientras esté activa la consola. O sea que si la cerramos y la volvemos a abrir ya no está activo el agente y nos volverá a pedir el password cada vez.

Para listar las claves que tenemos en el ssh-agente hacemos:

```
$ssh-add -l
2048 f0:eb:52:8c:74:8e:a1:51:82:05:a8:88:b4:29:3a:ae /root/.ssh/id_rsa (RSA)
```

Podemos borrar claves con la opción *ssh-add -d.* Necesitaríamos borrar claves cuando los servidores a los que nos conectamos las cambian, dejan de servir o bien porque nos podría interesar por razones de seguridad sólo recordar la clave durante un cierto tiempo (ej: una hora). Para esta última situación tenemos la opción *ssh-agent -t seconds*, donde seconds es el tiempo de vida que tendrán nuestras claves.

Una vez que tenemos totalmente operativo un servidor SSH en nuestro dispositivo Debian, podría interesarnos conectarnos a él para gestionarlo desde una máquina con S.O. Windows. Un buen cliente de libre distribución es *putty*. Disponible en:

http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html