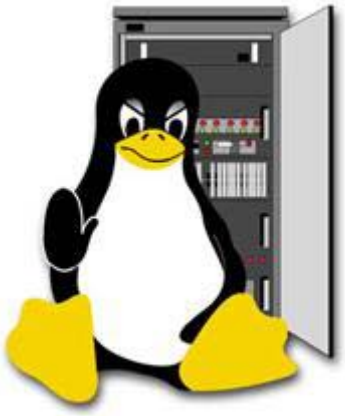
FI – UPV 2002

Interconexión de Redes
Bloque II - Seguridad

El módulo Netfilter de Linux: iptables



David Pascual Serral.

ÍNDICE

1. Introducción

¿Qué es un *firewall*? La traducción más acertada de este término inglés al idioma español es la palabra *cortafuegos*. Vemos cual es la definición en el DRAE[1].

> **<<Cortafuego o cortafuegos.** (De *cortar* y *fuego*). m. *Agr.* Vereda ancha que se deja en los sembrados y montes para que no se propaguen los incendios. || **2.** *Arq.* Pared toda de fábrica, sin madera alguna, y de un grueso competente, que se eleva desde la parte inferior del edificio hasta más arriba del caballete, con el fin de que, si hay fuego en un lado, no se pueda este comunicar al otro.**>>**

Estas dos definiciones ya introducen una idea muy aproximada al significado de la palabra *firewall* en términos informáticos, en ellas, aparecen términos como *lado*, lo que implica la existencia de dos o más partes y *comunicar* lo que pone de manifiesto que las partes están conectadas.

Bien, pues casi sin analogías, eso es un *firewall* en términos de sistemas computacionales. Un cortafuegos, es un sistema informático, simple o compuesto que actúa como punto de conexión segura entre otros dos o más sistemas informáticos.

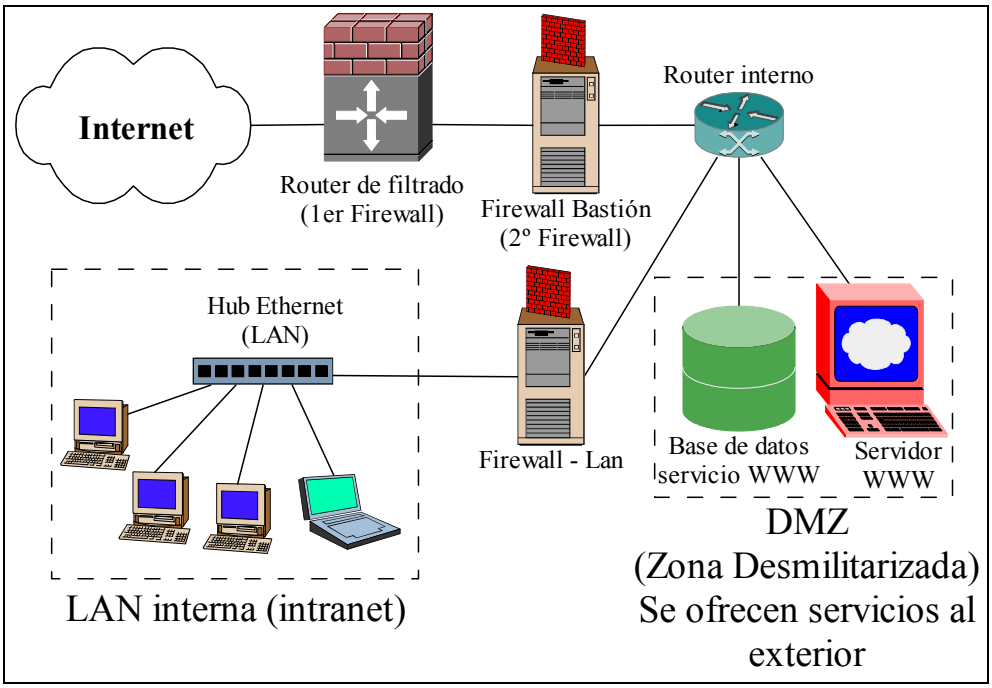


Figura 1 Sistema complejo con diversos Cortafuegos

Descendiendo en la abstracción, un cortafuego se sitúa entre dos o más redes con la intención de hacer cumplir unas determinadas directivas de seguridad sobre la comunicación entre ellas. Por ello, un cortafuego, puede ser desde un simple router, un PC[2] antiguo o una subred de servidores SOLARIS.

---

[1] Diccionario de la Real Academia de la lengua Española. http://buscon.rae.es/drae/drae.htm

[2] Por ejemplo, una red Ethernet donde podemos incluir un PC "antiguo", como un intel 486 o 386 con 2 o más tarjetas Ethernet, sin HD, y configurado para arrancar con un disquete de MS-DOS en el que incluimos alguna utilidad configurable de filtrado a nivel 2 (capa de MAC y trama Ethernet) haciendo uso por ejemplo de los packet-drivers.

Como vemos el término cortafuegos, puede concretarse de manera muy distinta según los mecanismos utilizados para su construcción, la familia de protocolos que reconoce (desde el estándar universal TCP/IP, hasta aproximaciones de corte comercial como NETBEUI de Microsoft, o IPX/SPX de Novell Netware), el nivel de la pila sobre el que actúa para un mismo protocolo (ver figura), y como no la arquitectura de red subyacente.

2. Cortafuegos de filtrado de paquetes, IPFW

El documento presente, no pretende ser un estudio sobre la infinidad de familias y variedades de los *firewalls*, seremos menos abstractos, y nos centraremos en el estudio de un modelo de cortafuegos mas concreto, la familia de los *firewalls* de filtrado de paquetes sobre la pila de protocolos TCP/IP, los llamados IPFW.

Los IPFW funcionan como indican las dos primeras letras sobre paquetes IP, es decir, en el nivel de transporte y red de TCP/IP.

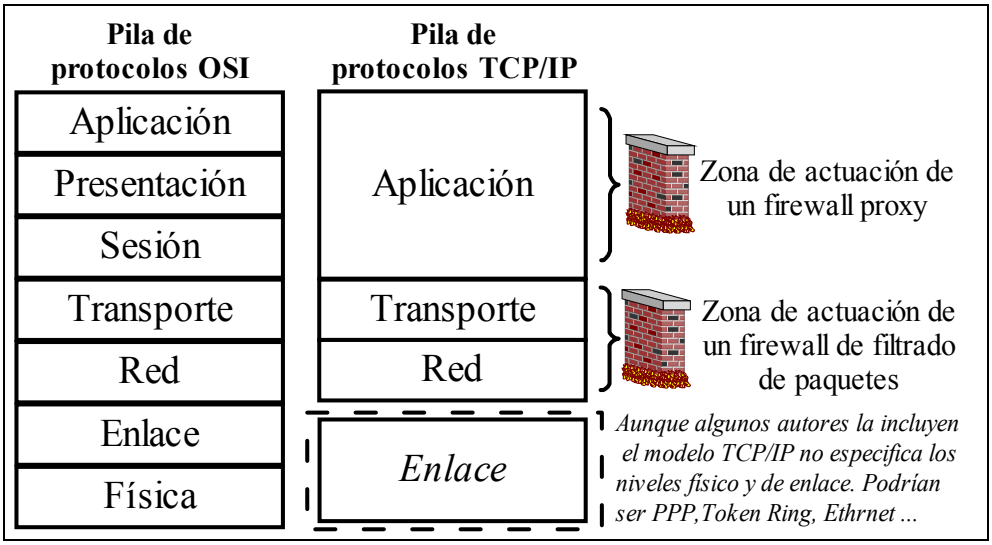


Figura 2 Niveles ISO/OSI vs TCP/IP

Los cortafuegos de filtrado de paquetes IP, suelen implementarse dentro del sistema operativo y funcionan en las capas de transporte y red, trabajando sobre la información de las cabeceras de los paquetes IP, es decir, que no analizarán el área de datos[3], sino que únicamente utilizan la información que puede obtenerse de una cabecera IP.

Como ya habíamos dicho, habitualmente, un cortafuego se sitúa entre dos o más redes, lo que implica que tiene al menos dos interfaces de red. A pesar de que el cortafuegos separa dos redes cualquiera entre si, lo habitual, es que separe redes propietarias distintos. Es decir, aunque pueden utilizarse cortafuegos dentro de una misma red, filtrando las comunicaciones entres las distintas subredes internas, lo habitual, es que el cortafuego forme parte de una red propia[4] y sea él quien vigile las comunicaciones con otra red o redes ajenas en las que no se confía[5], por ejemplo y sobre todo Internet.

[3] Por ejemplo, un cortafuegos de filtrado de paquetes, NO puede evitar que un usuario mande desde su equipo de trabajo un mensaje de correo con las cuentas de la empresa a la competencia, lo más que podría sería evitar que esa estación acceda al servidor de correo, con lo que no podría mandar correo a nadie.

[4] De hecho, un único ordenador con un MODEM y un firewall configurado, ya representa dicha separación, aunque la red interna la forma sólo él y por tanto NO tendrá dos interfaces de red, sino una sola.

[5] Por si no se ha comentado, los firewalls, son principalmente una herramienta de seguridad y prevención ante ataques externos.

Es decir, un IPFW funciona filtrando las comunicaciones en ambos sentidos entre su interfaz interna (la que lo conecta a su red) y la externa. El mecanismo de funcionamiento de para realizar este filtrado es a través de una lista de reglas.

Las reglas, pueden ser de dos tipos aceptación y rechazo, aunque en realidad, éste último se descompone en dos "subtipos", por lo que se suele hablar incluyendo la especialización, es decir, en términos de *aceptación*, *rechazo* y *denegación*[6]. La lista de reglas de entrada (del exterior hacia la red) es totalmente independiente de la lista de reglas de filtrado de salida (de la red hacia el exterior). Las distintas listas de reglas se llaman cadenas (chains).

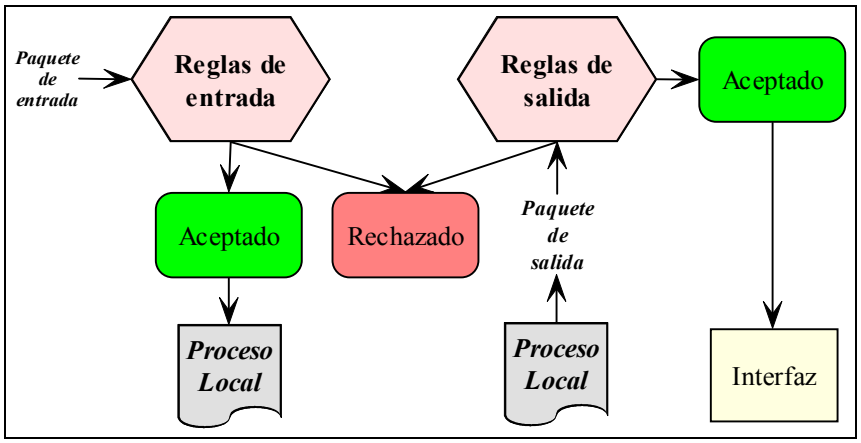


Figura 3 Flujo de un paquete en un FW

Hemos hablado de aceptación, rechazo y denegación. Los dos últimos son bastante similares, la diferencia radica en lo siguiente, cuando un IPFW rechaza una petición externa, envía una respuesta negativa diciendo que no acepta la comunicación, por el contrario, cuando un firewall descarta una petición, no envía ningún tipo de respuesta, es decir, que el agente externo que intentó establecer contacto, no sabrá ni siquiera si la máquina existía o estaba encendida.

Una vez definido un IPFW, podemos comenzar a estudiar las técnicas y políticas de creación de las cadenas. Hemos comentado que las cadenas eran listas de reglas por las que pasaban los paquetes IP, pero ¿debemos definir las reglas para todos los posibles paquetes IP? Obviamente, eso es inabordable, no podemos definir un regla de aceptación o rechazo para cada característica de cada paquete proveniente de cada una de las IP. La solución, pasa por hacer uso de "comodines" o "máscaras" para conseguir reglas generales. Esta manera de actuar, implica la primera decisión: *La regla por defecto*. Podemos establecer dos criterios:

- Aceptar TODO por defecto y especificar aquello que no deseemos.
- Rechazar TODO por defecto y especificar aquello que deseemos.

Hoy en día, si se quiere llevar una política de seguridad mínimamente seria, la primera propuesta es inviable, de hecho, muchos sistemas de IPFW ni siquiera la permiten, y sólo se puede trabajar con la segunda propuesta. Así que, esa será la política que elijamos nosotros.

Para poder asegurar un lugar mediante cortafuegos, deberemos estudiar reglas de entrada y de salida para las interfaz(es) interna(s) y externa(s). Sin embargo, la parte más importante y donde se realiza el mayor esfuerzo en la elaboración de las reglas, es en la cadena de entrada de la interfaz externa.

[6] En iptables, un IPFW que será el principal objeto de estudio del trabajo, se corresponden con los argumentos ACCEPT, REJECT y DROP, respectivamente.

2.1. Filtrado de paquetes entrantes en un IPFW.

Como ya hemos comentado, un IPFW basa su filtrado en unas reglas que atienden a distintas condiciones sobre la información que aparece en una cabecera IP, en realidad, además de la cabecera IP, también se utiliza información de las cabeceras de los protocolos inmediatamente superiores, es decir TCP y UDP. Esto implica que podemos filtrar basándonos en la dirección de origen, la dirección destino, el puerto origen, el puerto destino, el BIT de estado de TCP, etc…

Intentaremos dar unas pinceladas generales sobre reglas que interesa que se cumplan para no crear agujeros de seguridad en el IPFW.

- *Filtrado de dirección remota*: La dirección remota es lo único que identifica al remitente del paquete. Deberemos impedir la entrada de paquetes cuya dirección remota pertenezca a redes o hosts "problemáticos". Mediante este filtrado, disminuimos las posibilidades de ataques de *ip spoofing*. En este tipo de agresiones, el atacante suplanta su dirección IP por otra, para dificultar las posibilidades de suplantación, en la interfaz externa siempre denegaremos:
    - La dirección IP propia de la interfaz de red externa.
    - Direcciones IP privadas, de clase A (10.0.0.0 hasta 10.255.255.255), de clase B (172.16.0.0 hasta 172.31.255.255) y de clase C (192.168.0.0 hasta 192.168.255.255).
    - Direcciones IP de multidifusión de clase D (224.0.0.0 239.255.255.255).
    - Direcciones reservadas de clase E (240.0.0.0 hasta 247.255.255.255).
    - Direcciones de loopback (127.0.0.0 hasta 127.255.255.255).

- *Filtrado de dirección destino local*: Por defecto la interfaz de red ignora los paquetes de red que no se dirigen a ella, a excepción de las difusiones. Si no se va a hacer uso de ellas, conviene denegarlas.

- *Filtrado de puerto de origen remoto y destino loca*: Se configurarán en función de los servicios de la red, tanto los locales que se exporten al exterior (por ejemplo un servidor web) como los externos que utilice (por ejemplo, un cliente de correo conectándose al servidor POP3 ajeno a nuestra red).

- *Filtrado del estado de la conexión TCP*: Para los paquetes entrantes procedentes de servidores remotos, se exigirá que el BIT de AWK está activo (normalmente, siempre ocurre así puesto que responden a peticiones de un cliente local). Respecto a los paquetes IP entrantes desde clientes remotos hacia servicios locales que se exportan, dependerá de las necesidades de dichos servicios[7].

- *Sondeos y exploraciones de puertos*: Son relativamente habituales en nuestros días. No conviene ser extremista, es decir, no ser excesivamente muy paranoico ni pasar absolutamente del tema. En realidad la mayoría de ataques y/o escaneos no los realizan *hackers* o *crackers*, suelen llevarlos a cabo los que en esta terminología se denominan *lamers*, es decir, gente con pocos o nulos conocimientos, que utiliza herramientas de otro para intentar acceder o averiguar datos de un sistema. Si el firewall lo permite, la mejor opción es auditarlos (registrar cuando y desde donde ocurren en un fichero histórico) para poder estudiar sus características (frecuencia, origen, puertos afectados, etc...). Con todos esos datos en la mano es más sencillo determinar si se trata de alguien inexperto, o si realmente se adivina un ataque serio en ciernes.

[7] Por ejemplo, generalmente se desea que un servidor WEB puede ser accesible por todo el mundo. Sin embargo, puede que deseemos que otros servicios como el de FTP o el servicio seguro SSH sólo estén disponible para hosts concretos.

- *Ataque DOS (Denial of Service)*: Este tipo de ataques tal y como su nombre indica, intentan inutilizar uno o varios servicios de los ofrecidos por nuestra red. Para conseguirlo, pueden explotar bugs que afecten a la red o las máquinas (por ejemplo, el conocido WinNuke, aprovechaba un fallo del protocolo NETBEUI y los servicios SMB de recursos compartidos, de Microsoft y forzaba un reinicio en máquinas con el SO Windows 95), o intentar saturar un servicio por sobrecarga, bien sea aprovechando un mayor ancho de banda o capacidad de proceso por parte del atacante, o bien porque varios atacantes se ponen de acuerdo y aúnan sus recursos (por ejemplo, se puede saturar un servidor SMTP cuando varios de sus usuarios hacen *spawn* de correo con mensajes tamaño considerable). No es posible defenderse totalmente de ellos. La mejor manera de evitarlos es estar informado sobre los últimos agujeros de seguridad. Ejemplo de estos ataques son la inundación SYN TCP, inundación ping, bombas de redirección ICMP, etc… Si desea obtener más información sobre el tema puede consultar la bibliografía, aunque la mejor manera de estar al día sobre los últimos *exploits* o *bugs* de seguridad que se descubren y sus soluciones, les suscribirse a alguna lista de seguridad. En la bibliografía también se indican algunas de ellas.

Este trabajo, no tiene como principal objetivo el diseño de cortafuegos propiamente, sino describir la familia de IPFW y explicar la configuración de en un caso concreto (IPTables). A pesar de ello, si debemos insistir en algunos conceptos.

El cortafuegos, debe evitar accesos externos a servicios que se ofrecen al ámbito local o interno, y/o asegurar el correcto funcionamiento e integridad de servicios que se ofrecen al exterior protegiéndolos debidamente. Pero independientemente de la configuración del cortafuegos, es muy importante vigilar que los servicios son seguros (por ejemplo, manteniendo siempre parcheados lo posibles bugs que surjan), que están correctamente configurados, y que se utilizan. ¿Para qué tener en marcha un servicio que no se utiliza y que pueda poner en riesgo la seguridad de toda la red?

3. Introducción a *iptables*.

Bien, como ya se ha comentado, el objetivo del documento, es centrarse en el estudio de una herramienta concreta. Concretamente, el sistema a estudiar, será la herramienta de cortafuegos **iptables** sobre un sistema operativo Linux, concretamente el RedHat 7.3.

¿Por qué Linux? Como todo sistema operativo UNIX tiene entre sus objetivos la seguridad, además es FREEWARE.

¿Por qué Red Hat 7.3? La versión 7.3, es la última[8] del sistema operativo Red Hat, el kernel que lleva es la versión 2.4.9, que soporta IPTables. Conozco RedHat porque es la distribución con la que suelo trabajar. De cualquier forma, en lo que a la configuración de IPTables concierne, las diferencias son prácticamente inexistentes entre los distintos Linux. Lo único necesario es que la distribución Linux elegida cuente con una versión de Kernel superior[9] a la 2.4.

Iptables es como se conoce al módulo *Netfilter*, la herramienta estándar actual de cortafuegos bajo el sistema operativo l[10] estándar Linux de cortafuegos.

[8] La distribución RedHat 7.3 fue lanzada durante la composición de este documento.
[9] En realidad, es suficiente con kernels compatibles con NETFILTER. Es decir, recompilando un kernel de la serie 2.3.15. OJO: Como antes, no puedo afirmar que en futuras versiones no se sustituya IPTables por una nueva herramienta. Es decir, que si en el momento de leer esto, transcurre el año 2010, dudo que el documento le aporte nada útil.
[10] Como siempre, el término actual en este mundo informático, no puede alejarse más de unos pocos meses a partir de la fecha de elaboración del documento.

3.1. Un poco de historia.

La primera pila IP para Linux la desarrolló Ross Biro (NET-1). Al mismo tiempo, Orest Zborowski y Laurence Culthane trabajaban en la API sockets y en los controladores SLIP. Sin embargo, la verdadera integración de Linux a la red llegó de la mano de Alan Cox con NET-2 y NET-3. Esta última, es la actual pila de protocolos TCP/IP en la que además de Cox participaron muchos otros como Donnald Becker, etc…

El sistema operativo Linux, ha contado con herramientas de filtrado de paquetes (IPFW) incorporadas en su núcleo desde la versión del kernel 1.1.

Esta primera versión con filtrado, contaba con una adaptación de la herramienta *ipfw* del sistema operativo BSD llevada a cabo Alan Cox por el año 94.

El holandés Jos Vos junto a otras personas, mejoró el sistema de filtrado para las series 2.0 del kernel, e introdujo la utilidad de configuración *ipfwadm*.

A mediados de 1998, de la mano de Michael Neuling y Rusty Russell aparece la herramienta *ipchains*, incorporada en los kernels de la serie 2.2 y que todavía hoy es utilizada en gran parte de los sistemas Linux, aunque sólo se asegurará su compatibilidad en el núcleo hasta el año 2003. ¿Por qué solo hasta el 2003?

La respuesta llega a mediados de 1999. Cuando de nuevo Rusty Russell aparece en escena con una nueva herramienta de filtrado *iptables*. Como lo fue ipchains sobre ipfw, iptables es una modificación que permite la construcción de reglas más precisas y un mejor aprovechamiento de los recursos.

3.2. Novedades de iptables. NAT.

Además de realizar un mejor aprovechamiento de los recursos del sistema, la principal novedad del módulo *netfilter-iptables,* es la integración de las herramientas de filtrado (el cortafuegos propiamente dicho), de *NAT* y de manipulación (*MANGLING*).

Aunque lo trataremos con más profundidad posteriormente, creo que debemos deternos un momento para explicar lo que es NAT. NAT es el acrónimo de *Network Addres Translation*. Lo que hace NAT básicamente es alterar las cabeceras de los paquetes IP, (principalmente las direcciones) y mantener un "registro de entrada y salida" de los paquetes modificados, para poder alterar los paquetes respuesta de igual forma. Las aplicaciones de NAT son muchísimas, aunque actualmente, la aplicación más conocida del NAT, es lo que se conoce como enmascaramiento o *masquerading*. El enmascaramiento, se utiliza principalmente para dos cosas:

- *La conexión de varios equipos a través de una sola dirección IP.* Como ejemplos, las pequeñas LAN domésticas, un CyberCafé, y en general cualquier red con más equipos que IPs. En lugar de instalar un servidor proxy y configurar las distintas aplicaciones para que hagan uso del mismo, se instala NAT, y no hay necesidad de configurar las aplicaciones, ya que al trabajar en un nivel más bajo de la pila, resulta totalmente transparente.

No debe confundirse el NAT con un "proxy", aunque tengan aplicaciones parecidas. NAT no es un proxy, los proxies trabajan en un nivel superior de la pila de protocolos, bien en el nivel de TCP/UDP o incluso en el nivel de aplicación, lo que implica que los clientes deben configurarse para hacer uso del servicio. Por el contrario, NAT, al igual que el filtrado, trabaja en un nivel más bajo, a nivel IP, por lo que es "transparente". ¿Por qué se confunden muchas veces el NAT y un proxy? La respuesta puede deberse a que tanto el uso de un proxy como el uso de NAT se emplean actualmente[11] para "compartir una conexión a Internet".

[11] Tanto la definición de Proxy como el protocolo NAT, contemplan muchas más posibilidades que el simple "compartir" de la conexión a Internet. Pero, un usuario normal de un PC, es lo único con lo que lo suele asociarlos, si es que ha oído hablar de ellos.

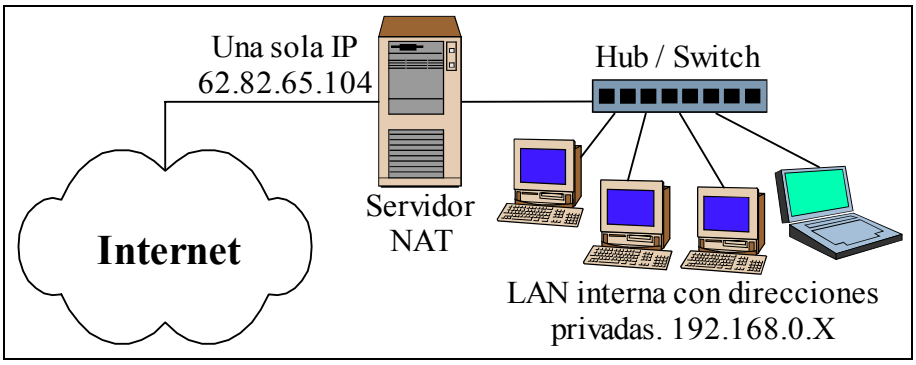


Figura 4 Ejemplo de NAT

- *El balanceo de carga.* Para explicarlo, lo mejor es ofrecer un ejemplo. Imaginemos que tenemos una máquina en la que corre un servidor HTTP, una de las páginas que ofrece el servidor, tiene un número de visitas diarias tremendamente elevado, (por ejemplo www.millonesdevisitas.zzz). El servidor, está conectado a Internet con un ancho de banda suficientemente importante y está disponible para todo el mundo. El problema, es que dada la sobrecarga de peticiones a las que se ve sometido el servidor, la máquina se sobrecarga y las las páginas se sirven con mucha lentitud. ¿Cómo resolverlo? O compramos un servidor más potente, o utilizamos NAT. La IP que tenía la máquina servidor y que estaba asociada a la URL www.millonesdevisitas.zzz, se le asignaría a una máquina que actuaría de radware (balanceador) y repartiría la carga entre varias máquinas (que formarían parte por ejemplo de una LAN Ethernet) cuyos servidores HTTP estuviesen configurados de forma similar (sirviesen todos las mismas páginas). La carga de los servidores quedaría repartida y además se facilitaría la redundancia de existencia, los servidores HTTP que alojan webs importantes o con un gran número de visitas utilizan este sistema de balanceo para que el tiempo de respuesta de las peticiones sea bajo y mantener el servicio si cae alguno de los servidores.

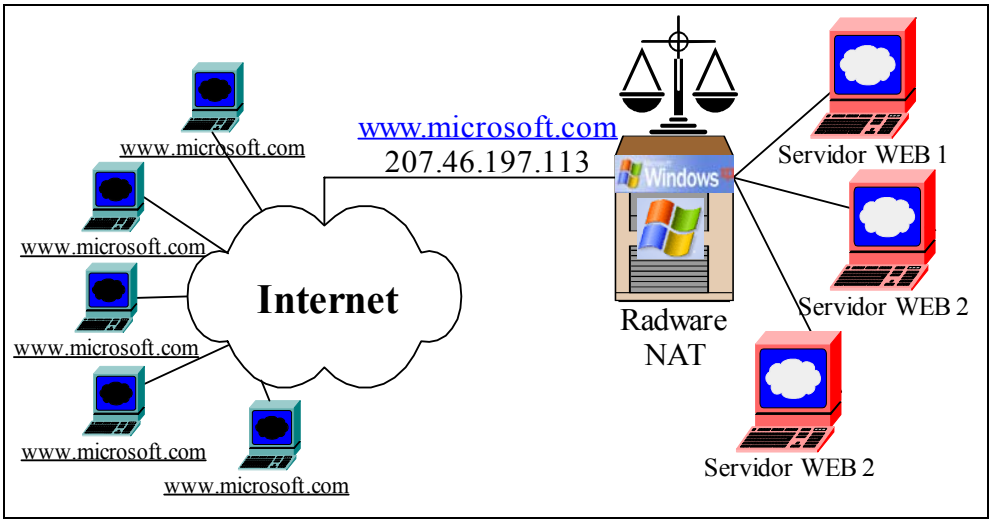


Figura 5 Ejemplo[12] de un sistema de balanceo de carga HTTP ☺

[12] Desconozco si algunos de los sistemas operativos Windows facilitan este tipo de configuraciones, así que voy a suponer que si (xDDDD). Si no fuese así, podría hacerse uso de Proxies software (como wingate, winproxy…). No obstante, este tipo de balanceo se realiza habitualmente con Routers/Switches que incorporan NAT y políticas de balanceo.

El filtrado de paquetes y *NAT* están ampliamente ligados, ya que comos ya hemos ambos servicios constituyen el módulo Netfilter y se configuran haciendo uso de la herramienta *iptables*. Antes de *iptables*, la herramienta Linux utilizada para configurar y ofrecer servicios NAT era *ipmasqadm*. A diferencia de *ipchains / iptables* cuyo manejo es conceptualmente similar, entre *ipmasqadm* e *iptables*, existen diferencias de uso.

4. Instalación de iptables.

Lo primero que debemos hacer para instalar iptables, es conseguirlo. Podemos descargar iptables de http://www.netfilter.org/. En cualquier distribución actual se instala por defecto y el kernel "precompilado" soporta la configuración, a pesar de eso, vamos a describir brevemente el proceso necesario para una instalación.

4.1. Parámetros del kernel.

Como ya hemos dicho, para poder utilizar iptables, necesitamos un kernel superior a la versión 2.3.15. Para configurarlo correctamente deberemos recompilar el mismo (*make config*) e incluir distintas opciones dentro del kernel, o bien habilitar la posibilidad de que puedan cargarse posteriormente como módulos. En la figura ofrecida a continuación, están descritas brevemente las opciones más importantes:

CONFIG_PACKET: Permite a ciertos programas trabajar directamente con la interfaz de red.
CONFIG_NETFILTER: Opción necesaria para utilizar la máquina como cortafuegos y/o router.
CONFIG_IP_NF_FILTER: Habilita el uso de la tabla FILTER.
CONFIG_IP_NF_NAT: Habilita el uso de la tabla de NAT.
CONFIG_IP_NF_IPTABLES: Opción necesaria para utilizar iptables.
CONFIG_IP_NF_CONNTRACK: Opción necesaria para usar NAT y enmascaramiento (seguimiento de conexiones).
CONFIG_IP_NF_TARGET_MASQUERADE: opción necesaria para trabajar con NAT cuando la IP de conexión a Internet es dinámica.
CONFIG_IP_NF_FTP: Opción necesaria para poder hacer seguimiento de conexiones a servidores ftp a través del cortafuegos.
CONFIG_IP_NF_MATCH_LIMIT: Esta opción permite limitar en el tiempo el número de paquetes que casan con una cierta regla. Se utiliza para limitar ataques DOS por sobrecarga.
CONFIG_IP_NF_MATCH_MAC: Permite el uso de direcciones MAC (Ethernet) en las reglas de filtrado.
CONFIG_IP_NF_MATCH_STATE: Es una de las principales novedades respecto a ipchains, pues permite hacer filtrado a partir del estado de una conexión TCP (por ejemplo ESTABLISHED…).
CONFIG_IP_NF_MATCH_OWNER: Es un módulo recientemente incorporado a iptables, con el podemos filtrar paquetes en función del usuario que es dueño (UID).
CONFIG_IP_NF_TARGET_LOG: Permite registrar en ficheros .log el disparo de las reglas. Se utiliza para observar situaciones extrañas en la configuración o posibles ataques.

Figura 6 Distintas opciones del kernel.

Hay más opciones además de las citadas, por ejemplo, CONFIG_IP_NF_COMPAT_IPCHAINS, habilita compatibilidad con reglas de ipchains. Por ello, se dice que la herramienta *iptables* es extensible, es decir, que tanto iptables como el kernel, están preparados para soportar la definición de nuevos criterios de evaluación en las condiciones de las reglas o nuevas acciones a efectuar con los paquetes. Estas extensiones, pueden ser implementadas por nosotros, o bien podemos descargar alguna extensión pública y utilizarla[13]. Lo normal es optar por la segunda opción, aunque, los gurús comentan que los fuentes de iptables son bastante claros.

[13] Como siempre en el mundo Linux, cuando las distintas extensiones públicas demuestran su utilidad se incorporan a la instalación por defecto de iptables/netfilter, por lo que muchos de estos módulos se soportan actualmente en la instalación por defecto.

4.2. Instalando iptables.

Lo primero que debemos hacer tras descargar el paquete iptables, es descomprimirlo. La última versión estable de iptables durante la elaboración de éste documento es la 1.2.5. Para descomprimirla, utilizaremos la orden:

```
bzip2 -cd iptables-1.2.5.tar.bz2 | tar -xvf
```

Con ello, tendremos el paquete descomprimido en un directorio `iptables-1.2.3/`, para llevar a cabo la instalación, conviene consultar el fichero `iptables-1.2.3/INSTALL`, donde se indica como instalarlo, será algo muy similar a esto:

```
cd /root/iptables-1.2.2
make KERNEL_DIR=/usr/src/linux BINDIR=/usr/bin LIBDIR=/usr/lib MANDIR=/usr/man
make install KERNEL_DIR=/usr/src/linux BINDIR=/usr/bin LIBDIR=/usr/lib MANDIR=/usr/man
```

En las distribuciones RedHat superiores a la 7.1, el Kernel que se instala por defecto viene configurado. Además también se incluye el paquete de iptables y si se selecciona en la instalación, aparece configurado. El único problema, es que puede que ipchains este habilitado también. Para deshabilitar del arranque del sistema evitando que no cause conflictos con iptables basta ejecutar[14]:

```
chkconfig --level 0123456 ipchains off; service ipchains stop
```

Finalmente, para arrancar el servicio iptables en el arranque del sistema ejecutaremos el comando siguiente:

```
hkconfig --level 235 iptables on; service iptables start
```

Naturalmente, no hay ninguna regla activa. Las reglas creadas con el comando `iptables`[15] se almacenan solamente en RAM, es decir, que si reiniciamos el sistema tras haber configurado varias reglas de iptables, éstas se perderán y tendremos que volver a teclearlas. Para incluir las reglas al inicio del sistema, podemos hacer varias cosas. La primera, editar el archivo script `/etc/rc.d/init.d/iptables`, éste script se ejecutará cada vez que se inicie el servicio iptables, que hemos configurado antes para iniciarse con el sistema. Otra opción, es introducir las reglas mediante el comando `iptables` y cuando el cortafuegos funcione como es debido, salvarlas en el fichero `/etc/sysconfig/iptables`. Es decir, introducimos las reglas de filtrado, y cuando el cortafuego funcione como esperamos, ejecutamos:

```
/sbin/service iptables save
```

Esto hace que el script de inicio de iptables (rc.d) ejecute el programa `/sbin/iptables-save` y escriba la configuración actual de iptables en el fichero `/etc/sysconfig/iptables`. Este fichero debería ser de sólo lectura para el usuario `root`, para que las reglas de filtrado de paquetes no sean visibles por el resto de los usuarios. La próxima vez que se inicie el sistema, el script de inicio de iptables volverá a aplicar las reglas guardadas en `/etc/sysconfig/iptables` usando el comando `/sbin/iptables-restore`.

Finalmente, podemos desinstalar el paquete de ipchains con el comando:

```
rpm -e ipchains
```

---

[14] Para los partidarios de las herramientas gráficas, pueden utilizar `serviceconf` o `ksysv`.

[15] En los apartados siguientes veremos la sintaxis del comando `iptables`, utilizado para configurar el módulo netfilter.

5. Estructura y funcionamiento de iptables.

Comentamos al principio del documento, que el módulo netfilter, integraba tres posibilidades en el manejo de los paquetes, cada una de esas posibilidades, se corresponde con una tabla donde se aplican las reglas. Con la opción `iptables -t tabla`, especificamos la tabla sobre la que queremos trabajar. Estas tablas son *filter*, *nat* y *mangling*. Veamos que podemos hacer sobre cada una de ellas:

- ***nat**:* La tabla *nat* se utiliza para configurar el protocolo de Network Address Translation. Cuando un flujo de paquetes (una conexión TCP) atraviesa la tabla, el primer paquete es admitido, el resto, son automáticamente identificados como parte del flujo de ese primer paquete y de manera automática se llevan a cabo sobre ellos las operaciones NAT o de enmascaramiento. Esta es la razón por la cual, no se lleva a cabo ningún tipo de filtrado en esta tabla. La tabla de *nat* tiene tres *chains* o cadenas sobre las que podemos añadir reglas. La cadena PREROUTING se utiliza para alterar los paquetes tan pronto llegan al cortafuegos (DNAT o NAT del destino). La cadena OUTPUT, se utiliza para alterar los paquetes generados localmente en el cortafuegos, antes de tomar ninguna decisión de enrutado. Finalmente tenemos la cadena POSTROUTING para alterar los paquetes que acaban de dejar el cortafuegos (SNAT o NAT en el origen).
- ***mangle**:* La tabla de *mangling* o "manipulación", permite manipular otros elementos de los paquetes, como el TTL, el TOS, etc…, ha excepción del NAT, que se realiza en la otra tabla. La funcionalidad de esta tabla está en expansión y aunque potencialmente puede ser muy valiosa, no tiene demasiada utilidad (salvo a hackers). Consta de dos cadenas, PREROUTING y OUTPUT.
- ***filter**:* Esta es la reina de la casa. En la tabla *filter*, se llevan a cabo la funcionalidad principal de iptables, el filtrado de paquetes. Como las anteriores, consta de varias chains predefinidas, en este caso INPUT, FORWARD y OUTPUT. La primera hace referencia a los paquetes entrantes cuyo destino es el propio cortafuegos. La segunda se emplea para decidir que hacer con los paquetes que llegan al cortafuegos y tienen como destino otro host, así podemos decidir si encaminarlos o no. Por último, la cadena OUTPUT se utiliza para filtrar paquetes generados en el propio host con destinos externos.

Cuando un paquete entra en el cortafuegos, lo hace a través de alguna interfaz (tarjeta de red, MODEM…). El paquete se dirige al Kernel, entrando en las distintas cadenas de las tablas sólo si procede. En la figura que ofrecemos posteriormente, puede observarse el esquema general del procesado de paquetes en iptables. En la tabla siguiente podemos ver también ejemplos de las desventuras de los paquetes en su tránsito por el módulo netfilter del kernel.

Paso	Tabla	Cadena	Comentario
1			En el aire, por ejemplo Internet
2			Llega a la interfaz de red, por ejemplo eth0
3	MANGLE	PREROUTING	Si casa con alguna de las reglas de *mangling*, se actúa según indique la acción.
4	NAT	PREROUTING	Si casa con alguna de las reglas de *NAT*, se actúa según indique la acción.
5			Decisión de enrutado
6	FILTER	INPUT	Se procede a realizar el filtrado del tráfico con destino local.
7			Aplicación local (cliente o servidor)

Tabla 1 Pasos en el Kernel de un paquete externo con destino en el propio firewall.

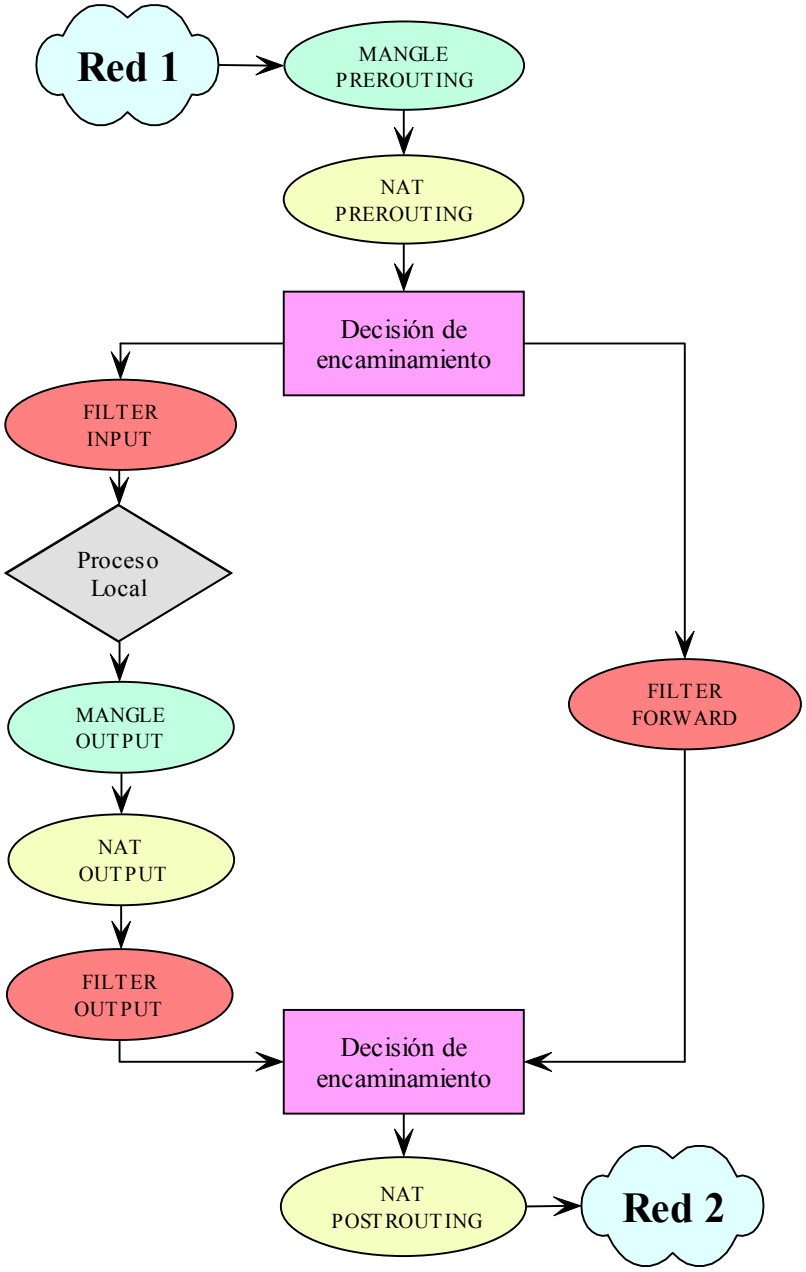


Figura 7 Procesado de paquetes en netfilter-iptables

Podemos comentar algunas curiosidades del esquema anterior. Partamos del supuesto de una conexión compartida a Internet. Supongamos que la RED1 es una red local con IPs privadas, y la RED2 es Internet, para que se accediese a servicios externos de Internet desde la RED1 (por ejemplo WWW), deberemos utilizar NAT, ¿Cómo funciona? Un paquete con origen en 192.168.0.2 (IP de un PC de la LAN), tiene como destino el puerto 80 de la IP 158.42.180.64 (www.redes.upv.es). El paquete llega a la interfaz interna del cortafuegos, pasa por la tabla de *mangling* sin recibir ninguna modificación, atraviesa la tabla de NAT prerouting sin modificarse, se observa que el destino es externo, por lo que se encamina a través de la tabla filter y de la cadena FORWARD, suponiendo que atraviesa todas sus reglas (es decir, que no se impide que los usuarios de la LAN accedan a servidores HTTP externos), se llega de nuevo a la decisión de encaminamiento y tras ello, el paquete atraviesa la tabla NAT y la cadena POSTROUTING, es aquí donde se modifica la dirección origen del paquete para que coincida con la IP pública del cortafuegos, a partir de este momento, los paquetes (flujo) de esa conexión, van marcados por la tabla NAT que lo gestiona de forma autónoma.

6. El comando `iptables`.

En este apartado, vamos a presentar el uso del comando `iptables`, que es la herramienta para crear las reglas de nuestro cortafuegos. A pesar de la descripción que vamos a ofrecer, como ocurre siempre en el mundo Linux, no hay que olvidar nuca la utilidad del comando `man`, si se desea un conocimiento exhaustivo de todas las opciones de la herramienta *iptables* lo mejor es consultar el manual. No sólo eso, además en el mundo Linux, funcionan las maravillosas HOWTOs, la HOWTO de *iptables* ha sido llevada a cabo por el propio Rusty Russell, lo que es una garantía de su calidad. Así que, ante cualquier duda que surja, lo mejor es acudir a esas fuentes.

iptables -[ADC] chain rule-specification [options]
iptables -[RI] chain rulenum rule-specification [options]
iptables -D chain rulenum [options]
iptables -[LFZ] [chain] [options]
iptables -[NX] chain
iptables -P chain target [options]
iptables -E old-chain-name new-chain-name

Figura 8 Opciones de *iptables* a través del comando *man*.

Como ya habíamos dicho, iptables, funciona mediante tres tablas, a su vez, cada una de esas tablas, tiene definidas unas "*chains*" o cadenas. Cada una de estas chains se compone de una lista de reglas de filtrado. Cada regla no es más que un par condición/acción sobre atributos del paquete IP. El paquete, irá pasando secuencialmente por cada una de las reglas de la *chain* hasta encajar en el patrón de alguna de ellas. Cuando esto ocurra, el paquete se tratará según indique la acción de la regla con cuya condición el paquete ha hecho matching[16]. Si tras recorrer toda la lista, el paquete no encaja con ninguna de las reglas, se ejecutará la acción por defecto asociada a esa *chain*.

Ahora, podemos adentrarnos ligeramente en las opciones de la herramienta *iptables*. Primero, veremos como manipular las *chains*. La utilización del comando, presenta siempre el siguiente patrón:

```
iptables [-t tabla] comando [match] [objetivos/saltos]
```

Las tablas son siempre una de las tres siguientes *filter*, *nat*, o *forward.* Si no se indica tabla alguna, por defecto, nos referimos a la tabla *filter*.

Para añadir y manipular cadenas utilizaremos siempre los comandos siguientes:

- *`iptables -N`*: Crear una nueva chain o cadena vacía (sin reglas).
- *`iptables -X`*: Elimina una chain que esté vacía (a excepción de las tres internas)
- *`iptables -F`*: Vacía una chain. Es decir, elimina todas las reglas de una chain.
- *`iptables -P`*: Cambia la política por defecto de una chain.
- *`iptables -L`*: Lista las reglas de una chain.
- *`iptables -Z`*: Pone a cero las variables de auditoría de una chain (número de paquetes, de bytes, etc…)

[16] El recorrido por la lista de reglas es secuencial, por lo que es muy importante el orden en el cual coloquemos las reglas. Haremos hincapié en ello más adelante.

Con los comandos siguientes conseguimos manejar las reglas de esas cadenas:

- *iptables –A*: Inserta al final de una cadena una nueva regla.
- *iptables –I*: Inserta al comienzo de una chain una nueva regla.
- *iptables –R*: Reemplaza una regla de una chain.
- *iptables –D*: Elimina una regla de una chain (podemos indicar el orden o su condición).

Por último, sólo queda mostrar como podemos establecer las condiciones y acciones sobre cada regla, es decir como establecer las condiciones de *match* entre paquetes y reglas:

- *iptables –s*: Indica un dominio o IP (rango de IPs) de origen sobre el que se evalúa la condición de la regla.
- *iptables –d*: Indica un dominio o IP (rango de IPs) de destino el que se evalúa la condición de la regla.
- *iptables –i (--in-interface)*: Indica la interfaz de entrada sobre la cual se evalúa la condición de la regla.
- *iptables –o (--out-interface)*: Indica la interfaz de salida sobre la cual se evalúa la condición de la regla.
- *iptables –p*: Especifica el protocolo del datagrama que concordará con esta regla. Los nombres válidos de protocolos son *tcp, udp, icmp*, o un número, si se conoce el número del protocolo de IP. Cada protocolo lleva asociados sus propios modificadores a traves de las extensiones correspondientes:
  - Extensiones de TCP:
    - *--sport*: Especifica el puerto que debe utilizar el origen del datagrama para concordar con esta regla. Se pueden especificar los puertos en la forma de un rango, especificando los límites inferior y superior con los dos punto ":" como delimitador. Por ejemplo, 20:25 describe todos los puertos que van desde el 20 hasta el 25 incluyendo ambos. De nuevo, el signo "!" puede utilizarse para negar los valores.
    - *--dport*: Igual que la opción anterior pero para el puerto destino.
    - *--tcp-flags*: Especifica mediante una máscara si los bits indicadores de TCP del datagrama concuerden con ella. La máscara es una lista separada por comas de los indicadores que deben examinarse en la comprobación.(SYN, ACK, FIN, RST, URG, PSH, ALL o NONE).
    - *--syn*: La regla casa con los datagramas cuyo bit SYN valga 1 y cuyos bits ACK y FIN valgan ambos 0. Esta opción es una abreviatura de: `--tcp-flags SYN,RST,ACK SYN`
  - Extensiones UDP:
    - *--sport*: Como en TCP, especifica el puerto que debe utilizar el origen del datagrama para concordar con esta regla.
    - *--dport*: Igual que la opción anterior pero para el puerto destino.
  - Extensiones ICMP:
    - *--icmp-type*: Puede especificarse el tipo de mesaje ICMP tanto por su número como por los siguientes identificadores: `echo-request`, `echo-reply`, `source-quench`, `time-exceeded`, `destination-unreachable`, `network-unreachable`, `host-unreachable`, `protocol-unreachable`, y `port-unreachable`.
  - Extensiones MAC:
    - *--mac-source*: Se especifica la dirección MAC (ethernet) Sólo tiene sentido en la cadena INPUT y FORWARD.
- *iptables –f*: Cuando se fracciona un datagrama porque supera el MTU de la red, podemos utilizar esta opción para especificar acciones sobre el segundo y restantes fragmentos del datagrama.
- *iptables ¡*: Invierte el valor lógico de la condición de la regla.

Existen más posibilidades de filtrado, y como ya hemos dicho, podemos diseñar las propias por la facilidad de extensión de iptables. Estas extensiones, son librerías compartidas que normalmente están en `/usr/local/lib/iptables/`, aunque, según distribuciones, pueden aparecer en `/lib/iptables/` o en `/usr/lib/iptables/`. Sirviéndonos de ellas podemos incluir filtrado sobre multitud de nuevas opciones o realizar acciones tremendamente exóticas con los paquetes filtrados (por ejemplo, el módulo REJECT incluye la nueva acción REJECT, cuyo efecto es similar al de DROP exceptuando los mensajes de error ICMP que si son tratados), incluso podemos listar el número de reglas similares que se dispararán por minuto, para evitar ataques DOS. Como siempre, a medida que las extensiones demuestran su utilidad, se añaden a la distribución estándar de iptables, y si no hay ninguna que nos ofrezca lo que buscamos, podemos hacerla nosotros.

7. Un caso real: Una Home-LAN.

El manejo de *iptables* es sencillo, aunque explicarlo mediante la simple narración puede hacerlo parecer más complicado. La mejor manera de entenderlo es a través de su aplicación a un caso real. Para ello, supongamos que vamos a configurar un cortafuegos de filtrado de paquetes para la red de la figura siguiente.

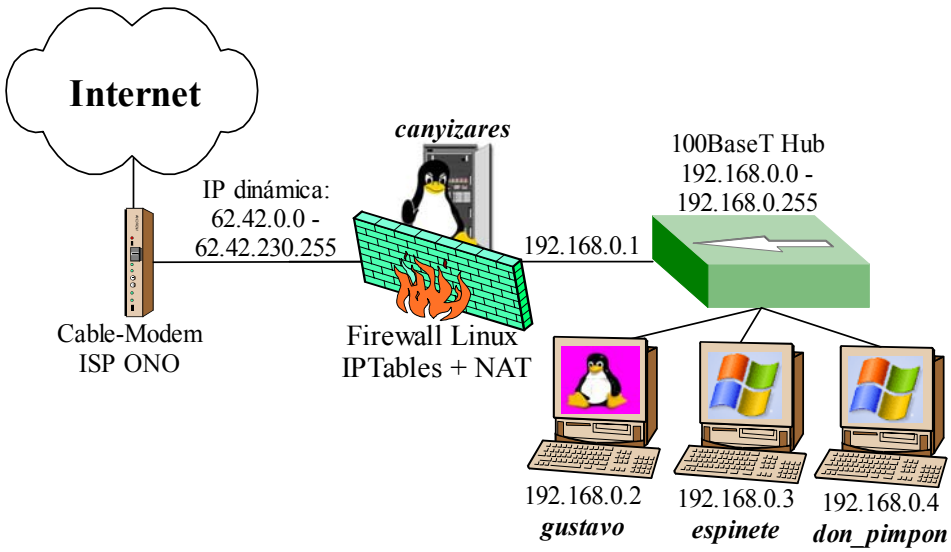


Figura 9 Home-LAN de ejemplo a configurar.

En principio, las tres maquinas de la red son simples estaciones de trabajo, que tal vez exportan algún directorio por SMB dentro de la propia red local. Por lo que debemos de preocuparnos principalmente de que tengan salida a Internet los principales servicios, clientes HTTP, FTP, etc. Por contra, en *canyizares* además de actuar de cortafuegos, queremos que corran servicios exportados tanto a la LAN como a Internet, por ejemplo un servidor HTTP, un servidor FTP y un servidor SSH para poder conectarnos remotamente a la máquina.

Además, deberemos resolver el problema del servicio DNS para la red local. La opción más sencilla, es configurar las máquinas de la red para que utilicen el mismo servidor DNS que la máquina cortafuegos (*canyizares*), es decir, los DNS del ISP (en este caso ONO). La otra alternativa, es ejecutar un servidor DNS en el cortafuegos y configurar las máquinas de la red local para que lo utilicen. Naturalmente, el servicio DNS que correrá en canyizares, será sólo un servicio de caché. Esta segunda configuración, es mas compleja, aunque ofrece ventajas respecto a la primera porque reduce el de peticiones tráfico DNS hacia Internet, ya que se resuelven localmente salvo cuando no están en caché, por esta misma causa, el tiempo medio de servicio en la resolución de un nombre de maquina desciende ligeramente.

Supondremos que la red interna es confiable, es decir, que de existir alguna amenaza, ésta vendrá desde Internet, no desde dentro. Tenemos que tener en cuenta que la IP proporcionada por ONO es dinámica, esto puede traernos pequeños quebraderos de cabeza en el caso que deseemos (como es nuestro caso, ya que corremos servicios en el cortafuegos) filtrar utilizando nuestra dirección IP pública. Hay diversas soluciones, pero dado que mi ISP ofrece cierta "estabilidad" en la duración[17] de las IP (algunas me han durado meses). La solución más sencilla, es obtener la IP mediante la ejecución de:

```
EXT_IP=`ifconfig $EXT_IF | grep inet | cut -d : -f 2 | cut -d \ -f 1`
```

O de algún método equivalente como:

```
EXT_IP="`ifconfig $EXT_IF | grep 'inet addr' | awk '{print $2}' | sed -e 's/.*://'`"
```

Resuelto estros problemas, sólo queda estructurar el script en función de nuestras necesidades.

Por ejemplo, para permitir accesos al servidor HTTP, FTP o SSH de *canyizares* desde cualquier lugar, bastaría con:

```
#Servicio FTP
iptables -t filter -A INPUT -p TCP -s 0/0 --dport 21 -j allowed
#Servicio SSH
iptables -t filter -A INPUT -p TCP -s 0/0 --dport 22 -j allowed
#Servicio HTTP
iptables -t filter -A INPUT -p TCP -s 0/0 --dport 80 -j allowed
```

Si el servidor HTTP se ejecutase en *gustavo* en lugar de en *canyizares*, deberíamos emplear DNAT.

```
#Servicio HTTP dentro de la LAN. Se exporta al exterior
iptables -t nat -A PREROUTING -i $EXT_IF -p tcp --dport 80 -j \
      DNAT --to-destination 192.168.0.2
```

Tras ver la sintaxis de iptables y comprender como es su funcionamiento, el configurar un cortafuegos a medida es una cuestión relativamente sencilla. A media que nos volvemos "paranoicos" o necesitamos una mayor seguridad y a la vez el poder exportar más servicios, es cuando las cosas se van complicando.

El script de configuración de un cortafuegos para la red local de la figura anterior, se presenta como el primer anexo del documento. El cortafuegos presentado, es relativamente seguro y bastante sencillo y aun así son cerca de 300 líneas de código.

Si no se desea una seguridad o perfilado extremo, actualmente existen herramientas gráficas que permiten generar scripts generales de configuración, a pesar de que puede que no excesivamente eficientes, pueden servir de base para "retocarlo" después a mano y evitar teclear demasiado.

[17] Si el ISP cambia nuestra IP de forma muy frecuente, tal vez interese utilizar alguna utilidad como `pump`, `DHCPCD` o `dhclient`, programas que permiten ejecutar un script cada vez que se produce un cambio de IP.

8. Anexo: Script de iptables para una Home-Lan

```
#!/bin/bash
# Script de configuración del Firewall $IPTABLES para Kernels 2.4.x
#------------------------------------------------------------------------------
#------------------------------------------------------------------------------
#Colores para la información que ofreceremos por pantalla.
        rojo="${esc}[31m"
        verde="${esc}[32m"
        azul="${esc}[34m"
echo "${rojo}"
echo "-----------------------------------------------"
echo "${azul}
echo "---- Script de configuración de iptables ----"
echo "${rojo}"
echo "-----------------------------------------------"

#------------------------------------------------------------------------------
#------------------------------------------------------------------------------
# Configuración de algunas variables.
echo "${azul}"
echo -n " ---- Configurando Variables ---- "
KERNEL=`uname -a|cut -f3 -d " "`
LAN_IP_RANGE="192.168.0.0/24" #IP's permitidas en la LAN 192.168.0.[0...255]
LAN_IP="192.168.0.1" #IP de la interfaz de la LAN.
ANYIP="0/0" #Cualquier IP.
LAN_BCAST_ADRESS="192.168.0.255" # Dirección bradcast.
LO_IF="lo" #Interfaz de loopback
LO_IP="127.0.0.1" # LocalHost IP 127.0.0.1
EXT_IF="eth0" # Interfaz de red Externa (Conexión a Internet)
INT_IF="eth1" # Interfaz de red Interna (Conexión a la LAN)
IPTABLES="/usr/sbin/iptables" #"whereis iptables" si se desconoce el PATH.
IPPRIVADACLASE_A="10.0.0.0/8"
IPPRIVADACLASE_B="172.16.0.0/12"
IPPRIVADACLASE_C="192.168.0.0/16"
CLASE_D_MULTICAST="224.0.0.0/4"
CLASE_E_RESERVADAS="240.0.0.0/5"

EXT_IP="`ifconfig $EXT_IF|grep 'inet addr'|awk '{print $2}'|sed -e's/.*://'`"

echo "${verde}"
echo "  [.] Versión del Kernel : $KERNEL"
echo "  [.] Localización iptables : $IPTABLES"
echo "  [.] Interfaz externa : $EXT_IF"
echo "  [.] Dirección IP externa : $EXT_IP"
echo "  [.] Interfaz LAN : $INT_IF"
echo "  [.] Dirección IP LAN : $LAN_IP"
echo "¡Hecho!"
#------------------------------------------------------------------------------
#------------------------------------------------------------------------------
# Cargamos los módulos de iptables que necesitamos
echo "${azul}"
echo -n " ---- Cargando módulos requeridos de NetFiler ----"
/sbin/depmod -a
/sbin/modprobe ip_tables
/sbin/modprobe ip_conntrack
/sbin/modprobe iptable_filter
/sbin/modprobe iptable_mangle
/sbin/modprobe iptable_nat
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_limit
/sbin/modprobe ipt_state
/sbin/modprobe ipt_MASQUERADE
#/sbin/modprobe ipt_owner
#/sbin/modprobe ipt_REJECT
#/sbin/modprobe ip_conntrack_ftp
#/sbin/modprobe ip_conntrack_irc
```

```
echo "${verde}"
#echo "lsmod"
echo "¡Hecho!"
#------------------------------------------------------------------------
#------------------------------------------------------------------------
echo "{azul}
echo -n "---- Estableciendo parametros del Kernel en /proc/sys/net/ipv4 ----"

# Activamos la opción de FORWARDING. La opción es CRÍTICA si el firewall
# tiene más de una interfaz de red y queremos que encamine tráfico entre
ellas.
echo "1" > /proc/sys/net/ipv4/ip_forward

# Habilitamos el la opción de verificación de dirección
for f in /proc/sys/net/ipv4/conf/*/rp_filter; do
  echo "1" > $f
done

# Activamos la opción de sync cookies. Con ello, nos protejemos ante el
# ataque DOS de bomba SYNC
echo "1" > /proc/sys/net/ipv4/tcp_syncookies

# Activamos la opción de protección de mansajes ICMP, con ello evitamos
# ataques DOS de bomba ICMP
echo "1" > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses

# IMPORTANTE: SÓLO PARA USUARIOS CON IP DINÁMICA. (La mayoría)
# Si tu ISP proporciona IP's dinámicas en tu conexión SLIP, PPP, o DHCP,
# debes ejecutar la línea siguientes.
echo "1" > /proc/sys/net/ipv4/ip_dynaddr

# DESHABILITAMOS la aceptación de paquetes cuyas ruta haya sido
# prefijada en el origen (source routed packets).
for f in /proc/sys/net/ipv4/conf/*/accept_source_route; do
  echo "0" > $f
done

# DESHABILITAMOS la aceptación de paquetes ICMP del tipo
# Redirect Acceptance
for f in /proc/sys/net/ipv4/conf/*/accept_redirects; do
    echo "0" > $f
done

# DESHABILITAMOS la posibilidad de enviar paquetes ICMP del tipo
# Redirect Acceptance.
for f in /proc/sys/net/ipv4/conf/*/send_redirects; do
    echo "0" > $f
done

#DESHABILITAMOS la posibilidad de enviar mensajes de redirección
for f in /proc/sys/net/ipv4/conf/*/send_redirects; do
    echo "0" > $f
done

echo "${verde}"
echo -n " [.] Parámetros del Kernel establecidos. "
echo "¡Hecho!"
#------------------------------------------------------------------------
#----------------------------- INICIALIZACIÓN ---------------------------
# Inicializamos el módulo Netfilter
echo "${azul}"
echo -n "... Inicializando NetFilter ... "

# Eliminamos las reglas de TODAS las cadenas de cada tabla
$IPTABLES -t filter --flush
$IPTABLES -t nat --flush
$IPTABLES -t mangle --flush
# Con las TODAS las cadenas vacías, dejamos sólo las cadenas por defecto
```

```
$IPTABLES -t filter --delete-chain
$IPTABLES -t nat --delete-chain
$IPTABLES -t mangle --delete-chain

# Fijamos las políticas por defecto para el filtrado.
$IPTABLES -t filter --policy INPUT DROP
$IPTABLES -t filter --policy OUTPUT DROP
$IPTABLES -t filter --policy FORWARD DROP

echo "${verde}"
echo -n " [.] Tablas y cadenas vaciadas. Establecidas políticas por defecto"
echo "¡Hecho!"
#-------------------------------------------------------------------------
#------------------ Construcción de las reglas de filtrado ----------------
echo "${azul}"
echo -n ".... Comienza la inserción de reglas de cortafuefos .... "
# Comenzamos la inserción de reglas en la tabla filter.
# Creamos cadenas para los paquetes no validos
$IPTABLES -N bad_tcp_packets
$IPTABLES -N bad_tcp_input_packets
# Creamos cadenas separados para los protocolos ICMP, TCP y UDP
$IPTABLES -t filter -N allowed
$IPTABLES -t filter -N icmp_packets
$IPTABLES -t filter -N tcp_packets
$IPTABLES -t filter -N udpincoming_packets

# --- Cadenas bad_tcp_packets y bad_tcp_input_packets
# Los paquetes TCP entrantes no presentarán situaciones extrañas
# en sus BITS (evitaremos ataques DOS)
$IPTABLES -t filter -A bad_tcp_input_packets -p tcp --tcp-flags \
      ALL NONE -j DROP
$IPTABLES -t filter -A bad_tcp_input_packets -p tcp --tcp-flags \
      SYN,FIN SYN,FIN -j DROP
$IPTABLES -t filter -A bad_tcp_input_packets -p tcp --tcp-flags \
      SYN,RST SYN,RST -j DROP
$IPTABLES -t filter -A bad_tcp_input_packets -p tcp --tcp-flags \
      FIN,RST FIN,RST -j DROP
$IPTABLES -t filter -A bad_tcp_input_packets -p tcp --tcp-flags \
      ACK,FIN FIN -j DROP
$IPTABLES -t filter -A bad_tcp_input_packets -p tcp --tcp-flags \
      ACK,PSH PSH -j DROP
$IPTABLES -t filter -A bad_tcp_input_packets -p tcp --tcp-flags \
      ACK,URG URG -j DROP

$IPTABLES -t filter -A bad_tcp_packets -p tcp ! --syn -m state --state \
    NEW -j LOG --log-prefix "filter bad_tcp_packets Posible ataque SYN:"
$IPTABLES -t filter -A bad_tcp_packets -p tcp ! --syn -m state --state \
      NEW -j DROP

# --- Cadena allowed
$IPTABLES -t filter -A allowed -p tcp --syn -j ACCEPT
$IPTABLES -t filter -A allowed -p tcp -m state --state ESTABLISHED, \
    RELATED -j ACCEPT
$IPTABLES -t filter -A allowed -p tcp -j DROP

# --- Cadena icmp_packets
#Permitimos la salida y entrada de PINGs
$IPTABLES -t filter -A icmp_packets -p icmp -s $ANYIP --icmp-type 8 \
    -j ACCEPT
$IPTABLES -t filter -A icmp_packets -p icmp -s $ANYIP --icmp-type 11 \
    -j ACCEPT

# --- Cadena tcp_packets
#Servicio FTP
$IPTABLES -t filter -A tcp_packets -p tcp -s $ANYIP --dport 21 -j allowed
#Servicio SSH
$IPTABLES -t filter -A tcp_packets -p tcp -s $ANYIP --dport 22 -j allowed
#Servicio HTTP
```

```
$IPTABLES -t filter -A tcp_packets -p tcp -s $ANYIP --dport 80 -j allowed
#Servicio IDENTD
$IPTABLES -t filter -A tcp_packets -p tcp -s $ANYIP --dport 113 -j allowed

# --- Cadena udpincoming_packets
#Servicio DNS caché para la red local.
$IPTABLES -t filter -A udpincoming_packets -p udp -s $LAN_IP_RANGE \
    --source-port 53 -j ACCEPT

# --- Cadena INPUT.
# Todo paquete entrante, deberá ser bueno.
$IPTABLES -t filter -A INPUT -p tcp -j bad_tcp_packets
# Rechazamos paquetes entrantes por la interfaz de red Externa con IPs de
# origen extrañas (anti spoofing)
$IPTABLES -t filter -A INPUT  -i $EXT_IF -s $EXT_IP -j DROP
$IPTABLES -t filter -A INPUT  -i $EXT_IF -s $IPPRIVADACLASE_A -j DROP
$IPTABLES -t filter -A INPUT  -i $EXT_IF -s $IPPRIVADACLASE_B -j DROP
$IPTABLES -t filter -A INPUT  -i $EXT_IF -s $IPPRIVADACLASE_C -j DROP
$IPTABLES -t filter -A INPUT  -i $EXT_IF -s $CLASE_D_MULTICAST -j DROP
$IPTABLES -t filter -A INPUT  -i $EXT_IF -s $CLASE_E_RESERVADAS -j DROP

# Los paquetes que provienen de Internet deben ser alguno de los definidos
$IPTABLES -t filter -A INPUT -p icmp -i $EXT_IF -j icmp_packets
$IPTABLES -t filter -A INPUT -p tcp -i $EXT_IF -j tcp_packets
$IPTABLES -t filter -A INPUT -p udp -i $EXT_IF -j udpincoming_packets
# Reglas especiales
#Mensajes a la dirección de broadcast desde la LAN hacia la LAN
$IPTABLES -t filter -A INPUT -p ALL -i $INT_IF -d $LAN_BCAST_ADRESS -j ACCEPT
#Permitimos conexión de Loopback.
$IPTABLES -t filter -A INPUT -p ALL -i $LO_IF -s $LO_IP -j ACCEPT
$IPTABLES -t filter -A INPUT -p ALL -i $LO_IF -s $INT_IP -j ACCEPT
$IPTABLES -t filter -A INPUT -p ALL -i $LO_IF -s $EXT_IP -j ACCEPT
#Peticiones de la red local
$IPTABLES -t filter -A INPUT -p ALL -i $INT_IF -s $LAN_IP_RANGE -j ACCEPT
$IPTABLES -t filter -A INPUT -p ALL -d $EXT_IP -m state --state ESTABLISHED, \
    RELATED -j ACCEPT
#Registramos un máximo de 3 paquetes por minuto que no casen con las
#reglas anteriores
$IPTABLES -A INPUT -m limit --limit 3/minute --limit-burst 3 -j \
      LOG --log-level DEBUG --log-prefix "filter INPUT paquete perdido: "

# --- Cadena FORWARD.
# Solo encaminamos paquetes válidos
$IPTABLES -t filter -A FORWARD -p tcp -j bad_tcp_packets
# Aceptamos ecaminar los paquetes porcedentes de la LAN
$IPTABLES -t filter -A FORWARD -i $INT_IF -j ACCEPT
$IPTABLES -t filter -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

#Registramos un máximo de 3 paquetes por minuto que no casen con las
#reglas anteriores
$IPTABLES -A FORWARD -m limit --limit 3/minute --limit-burst 3 -j \
      LOG --log-level DEBUG --log-prefix "filter FORWARD paquete perdido: "

# --- Cadena OUTPUT.
# Solo permitimos paquetes válidos
$IPTABLES -t filter -A OUTPUT -p tcp -j bad_tcp_packets
# Permitimos paquetes salientes con origen en el host
$IPTABLES -t filter -A OUTPUT -p ALL -s $LO_IP -j ACCEPT
$IPTABLES -t filter -A OUTPUT -p ALL -s $INT_IP -j ACCEPT
$IPTABLES -t filter -A OUTPUT -p ALL -s $EXT_IP -j ACCEPT

#Registramos un máximo de 3 paquetes por minuto que no caden con las
#reglas anteriores
$IPTABLES -A OUTPUT -m limit --limit 3/minute --limit-burst 3 -j \
      LOG --log-level DEBUG --log-prefix "filter OUTPUT paquete perdido: "

echo "${verde}"
echo -n " [.] Política de filtrado establecida"
```

```
echo "¡Hecho!"
#-------------------------------------------------------------------------
#------------------- Construcción de las reglas de NAT -------------------
# Habilitamos NAT, TODO el tráfico interno es enmascarado cuando sale al
# exterior
echo "${azul}"
echo -n ".... Configuración de NAT .... "
#REM: Utilizamos MASQUERADE porque la IP es dinámica
$IPTABLES -t nat -A POSTROUTING -s $LAN_IP_RANGE -o $EXT_IF -j MASQUERADE
echo "${verde}"
echo -n " [.] Política de enmascaramiento establecida"
echo "¡Hecho!"
```

9. Anexo: Diferencias entre iptables e ipchains.

En un primer momento, *ipchains* e *iptables* parecen ser bastante similares. Al fin y al cabo ambos son métodos de filtrado de paquetes usando cadenas o reglas que operan con el kernel de Linux para decidir no sólo qué paquetes se permite entrar o salir, sino también qué hacer con los paquetes que cumplen determinadas reglas, donde *iptables* ofrece un método mucho más extensible de filtrado de paquetes, proporcionando al administrador un nivel de control mucho más refinado sin tener que aumentar la complejidad del sistema entero.

Más concretamente, los usuarios que se encuentren cómodos con ipchains deberían tener cuidado con las siguientes diferencias significativas entre ipchains e iptables antes de utilizar iptables:

- Bajo *iptables*, cada paquete filtrado se procesa únicamente usando las reglas de una cadena, en lugar de hacerse con múltiples. Es decir, un paquete FORWARD que llega al sistema usando ipchains tendrá que pasar por las cadenas INPUT, FORWARD, y OUTPUT para llegar a su destino. Con iptables, el paquete tan solo se envía a la cadena INPUT si su destino es el sistema local y tan solo los envía a la cadena OUTPUT si el sistema local es quien genera los paquetes. Por esta razón, deberá estar seguro de situar la regla destinada a interceptar un paquete en particular en la cadena adecuada que será la que vea el paquete. La principal ventaja es que tendrá un control más refinado sobre la disposición de cada paquete. Si está intentando bloquear el acceso a un sitio web en particular, ahora es posible bloquear los intentos de acceso desde clientes que están en máquinas que utilicen nuestro servidor como pasarela (gateway). Una regla OUTPUT que deniegue el acceso no prevendrá más el acceso a las máquinas que utilicen nuestro servidor como pasarela.
- El objetivo DENY ha sido cambiado por DROP. En ipchains, los paquetes que cumplían una regla en una cadena podían ser redirigidos a un objetivo DENY, que dejaba caer el paquete en silencio. Este objetivo deberá cambiarse a DROP con iptables para obtener el mismo resultado.
- El orden es importante al poner opciones en una regla de una cadena. Anteriormente, con ipchains, no era muy importante cómo se ordenasen las opciones de las reglas a la hora de escribirla. El comando iptables es un poco más reticente sobre el lugar que ocupan las diferentes opciones. Por ejemplo, ahora deberemos especificar el puerto origen y destino después del protocolo (ICMP, TCP, o UDP) que vayamos a utilizar en una regla de una cadena.
- Cuando especificamos las interfaces de red que vamos a usar en una regla, deberemos utilizar sólo interfaces de entrada (opción `-i`) con cadenas INPUT o FORWARD y las de salida (opción `-o`) con cadenas FORWARD o OUTPUT. Esto es necesario debido al hecho de que las cadenas OUTPUT no se utilizan más con las interfaces de entrada, y las cadenas INPUT no son vistas por los paquetes que se mueven hacia las interfaces de salida.

10. Bibliografía

Publicaciones:

Firewalls Linux: guía avanzada. Robert L. Ziegler. Ed. Prentice-Hall, D.L. 2000.
Networking Linux. A practical Guide to TCP/IP. Pat Eyler. © 2001 by New Riders Publishing.
Iptables tutorial. Completo tutorial de Oskar Andreasson, que nos explica y complementa con ejemplos el howto y faq del netfilter: http://people.unix-fu.org/andreasson/index.html
Packet-filtering-HOWTO. Rusty Russells. GPL.
NAT-HOWTO. Rusty Russells. GPL.
Netfilter-hacking-HOWTO. Rusty Russells. GPL.

Direcciones de Internet con información de iptables:

Pagina Principal de Netfilter/Iptables.
http://netfilter.samba.org | http://netfilter.filewatcher.org.

Excelente página con gran cantidad de información sobre iptables y netfilter.
http://www.linuxguruz.org/iptables/.

Charla en el IRC de uninet sobre el netfilter.
http://6fevu.uninet.edu/text/laforge.html, http://6fevu.uninet.edu/text/laforge1.html
http://6fevu.uninet.edu/text/laforge2.html, http://6fevu.uninet.edu/text/laforge22.html
http://6fevu.uninet.edu/text/redes22.html.

Distintos tutoriales sobre iptables.
http://www.glug.es/sections.php?op=viewarticle&artid=1
http://www.linux-mag.com/cgi-bin/printer.pl?issue=2000-01&article=bestdefense
http://pinehead.com/articles.php?view=371.

Direcciones de Internet con información sobre Linux:
http://bulmalug.net/
http://lucas.hispalinux.es/
http://www.europe.redhat.com/documentation/
http://www.google.com/intl/es/