El "COMO" de ipchains e iptables

Traducido por Francisco J. Sucunza para www.cortafuegos.org

El "COMO" de ipchains e iptables: Traducido por Francisco J. Sucunza para www.cortafuegos.org

Table of Contents

1. Introducción 8
1.1. ¿Qué? 8
1.2. ¿Por qué? 8
1.3. ¿Cómo? 8
1.4. ¿Dónde? 9
2. Fundamentos sobre el Filtrado de Paquetes 10
2.1. ¿Qué? 10
2.2. ¿Por qué? 10
2.3. ¿Cómo? 11
2.3.1. Un núcleo con filtrado de paquetes. 12
2.3.2. ipchains 12
2.3.3. Hacer que las reglas sean permanentes 13
3. ¡Estoy confundido! Enrutamiento, enmascaramiento, port forwarding, ipautofw 15
4. La Guia de "tres lineas de Rusty" de Masquerading 16
4.1. Publicidad Gratuita: Reglas WatchGuard. 16
4.2. Configuraciones comunes de cortafuegos. 16
4.2.1. Red Privada: Proxies Tradicionales. 16
4.2.2. Red Privada: Proxies Transparentes 18
4.2.3. Red Privada: Enmascaramiento (Masquerading). 19
4.2.4. Red Pública. 21
4.2.5. Servicios internos limitados 22
4.3. Más información de Masquerading 23
4.4. Cadenas de filtrado (IP firewalling chains). 23
4.5. Cómo atraviesan los filtros los paquetes. 23
4.5.1. Usando ipchains 26
4.5.2. Qué verás cuando enciendas el ordenador. 27
4.5.3. Operaciones en una sola regla. 27
4.5.4. Especificaciones de filtrado 29
4.5.4.1. Especificando direcciones IP orígen y destino 29

4.5.4.2. Especificando Inversión (negaciones). 30
4.5.4.3. Especificando Protocolo 30
4.5.4.3.1. Especificando puertos UDP y TCP. 30
4.5.4.3.2. Especificando ICMP tipo & código. 31
4.5.4.4. Especificando una interfaz. 32
4.5.4.5. Especificando sólo paquetes SYN TCP. 32
4.5.4.6. Manejando Fragmentos. 33
4.5.5. Efectos secundarios del filtrado. 34
4.5.5.1. Especificando un objetivo. 35
4.5.5.2. "logeando" paquetes. 38
4.5.5.3. Manipulando el Tipo de servicio (Type Of Service) 40
4.5.5.4. Marcando un paquete. (no es mi culpa que suene así) 41
4.5.5.5. Operaciones en una cadena entera. 41
4.5.5.6. Crear una nueva cadena. 42
4.5.5.7. Borrar una cadena. 42
4.5.5.8. Vaciar una cadena. 42
4.5.5.9. Listar una cadena. 43
4.5.5.10. Restablecimiento (poniendo a cero) de los contadores. 44
4.5.5.11. Estableciendo la Política. 45
4.5.6. Operaciones en Enmascaramiento. 45
4.5.7. Verificar un paquete. 46
4.5.8. Multiples reglas de una vez y ver lo que pasa. 47
4.6. Ejemplos útiles 48
4.6.1. Usando.ipchains-save 50
4.6.2. Usando.ipchains-restore 51
5. Miscelánea. 53
5.1. Cómo organizar sus reglas del cortafuegos. 53
5.2. Qué no se filtra. 53
5.2.1. Paquetes de ICMP. 54
5.2.2. Conexiones TCP a DNS.(servidor de nombres) 54
5.2.3. Pesadillas de FTP. 55
5.3. Filtrando el Ping de la Muerte. 55
5.4. Filtrando Teardrop y Bonk. 56

5.5. Filtrando Bombas de Fragmento.....56
5.6. Cambiando las reglas del cortafuegos.....56
5.7. ¿ Cómo activo la protección de IP spoof?.....57
5.8. Proyectos avanzados. 59
5.8.1. SPF: Stateful Packet Filtering 59
5.8.2. ftp-data hack de Michael Hasenstein's. 60
5.9. Perfeccionamientos futuros.....60
6. Problemas comúnes. 62
6.1. ipchains -L se atasca! 62
6.2. La inversión no funciona.....62
6.3. Masquerading/forwarding no funciona! 62
6.4. ¡-j REDIR no funciona! 62
6.5. No funcionan los comodines de Interfaces ! 63
6.6. TOS no funciona!.....63
6.7. ipautofw e ipportfw y no funcionan! 63
6.8. xosview está roto! 64
6.9. Segmentation fault con -j REDIRECT 64
6.10. No puedo poner timeouts en enmascaramiento! 64
6.11. Quiero un cortafuegos IPX ! 64
7. Un ejemplo serio.....66
7.1. La situación. 66
7.2. Objetivos 67
7.3. Antes de filtrar paquetes.....69
7.4. Packet Filtering for Through Packets.....70
7.4.1. Configurar los saltos (JUMPS) en la cadena forward. 70
7.4.2. Definir la cadena icmp-acc.....71
7.4.3. Good (Interna) a DMZ (Servidores) 71
7.4.4. Bad (red externa) a DMZ (servidores). 72
7.4.5. Good (internal) to Bad (external). 73
7.4.6. DMZ a Good (interna). 74
7.4.7. DMZ a bad (externa). 75
7.4.8. Bad (externa) a Good (interna). 76
7.4.9. Filtrado de paquetes para la propia máquina de filtrado 77

7.4.9.1. Interfaz Bad (externa). 77
7.4.9.2. Interfaz DMZ. 78
7.4.9.3. Good (internal) interface. 78
7.5. Finalmente 79
8. Apéndice: Diferencias entre ipchains e ipfwadm 81
8.1. Tabla de referencia rápida 83
8.2. Ejemplos de comandos ipfwadm trasladados 85
9. Apéndice: Usando el script ipfwadm-wrapper 86
10. Apéndice: Gracias. 87
10.1. Traducciones 87
10.2. Sobre traducción 88

List of Figures

4-1. diagrama0 24
4-2. diagrama1 36
4-3. diagrama2 37
7-1. diagrama3 66

Chapter 1. Introducción

Éste es el Linux IPCHAINS-COMO; ver Section 1.4 Deberías leer también el Linux NET-3-HOWTO. El IP-Masquerading-HOWTO, el PPP-HOWTO y el Ethernet-HOWTO podrían ser lecturas interesantes. (Después, podría probar con la FAQ de alt.fan.bigfoot).

Si el filtrado de paquetes le es anticuado, lea la sección Section 1.2, la sección Section 2.3, y explore a través de los títulos de la sección Chains.

Si vienes de `ipfwadm`, lee la sección Chapter 1, la sección Section 2.3 y los Apéndices en la sección Chapter 8 y la sección .

1.1. ¿Qué?

Linux `ipchains` es una modificación de la codificación de Linux IPv4 firewalling, (que se obtuvo en su mayor parte de BSD), y una modificación de `ipfwadm`, que creo que fue a su vez una modificación de `ipfw` de BSD. Es necesaria para la administración de filtros de paquetes IP en las versiones 2.1.102 de Linux o posteriores.

1.2. ¿Por qué?

El código más viejo de Linux firewalling no opera con fragmentos, tiene contadores de 32-bit, (en Intel por lo menos), no permite especificación de protocolos que no sean TCP, UDP o ICMP, no puede hacer grandes cambios atómicamente, no puede especificar reglas inversas, (negación de reglas), tiene algunas incoherencias, y puede ser duro de manejar (haciéndolo propenso al error del usuario).

1.3. ¿Cómo?

Actualmente el código está en el núcleo 2.1.102. Para las series 2.0 del núcleo, necesitará descargar un parche del núcleo desde la página web. Si su núcleo 2.0 es más reciente que el parche proporcionado, el parche más viejo debe ser el adecuado; esta porción de los núcleos 2.0 es bastante estable (eg. el parche del núcleo 2.0.34 trabaja sobre el núcleo 2.0.35). Dado que el parche 2.0 es incompatible con los parches para ipportfw e ipautofw no recomiendo su instalación a menos que se necesite alguna utilidad que no proporcione ipchains.

1.4. ¿Dónde?

La pagina oficial está en tres lugares: Gracias al Penguin Computing (http://netfilter.filewatcher.org/ipchains) Gracias al equipo SAMBA (http://www.samba.org/netfilter/ipchains) Gracias a Harald Welte (http://netfilter.gnumonks.org/ipchains)

Hay una lista de informe de errores, discusión, desarrollo y uso. Subscribase a la lista enviando un mensaje que contenga la palabra "subscribe ipchain-list" en east.balius.com.

Chapter 2. Fundamentos sobre el Filtrado de Paquetes

2.1. ¿Qué?

Todo el tráfico a través de una red se envía en la forma de *paquetes*. Por ejemplo, descargar este paquete, (digamos de 50k de tamaño), podría hacer que recibiera como 36 paquetes de 1460 bytes cada uno (por poner números al azar).

La cabecera de cada paquete dice a dónde va, de dónde viene, el tipo del paquete, y otros detalles administrativos. Esta parte del paquete se llama *encabezado*. El resto del paquete, contiene los datos reales que se están transmitiendo, lo que normalmente se llama *cuerpo*.

Algunos protocolos, tal como *TCP*, que se usa para tráfico de la web, correo y logins remotos, usan el concepto de ’conexión’, antes de que cualquier paquete de datos reales se envien, se intercambian varios paquetes de configuración diciendo ’Quiero conectarme’, ’OK ’ y ’Gracias’. Luego, se intercambian paquetes normales.

Un filtro de paquetes es una porción de software que analiza el encabezado de los paquetes cuando lo atraviesan, y decide el destino del paquete entero. Podría decidir *denegar (deny)* el paquete, (ie. desecha el paquete como si nunca lo hubiese recibido), *aceptar (accept)* el paquete, (ie. permite que el paquete lo atraviese), o *rechazar(reject)* el paquete, (parecido a denegar, pero le dice al origen del paquete que ha sido rechazado).

Bajo Linux, el filtrado de paquetes está implementado en el núcleo, y hay cosas más complicadas que podemos hacer con los paquetes, pero el principio general es el de mirar los encabezados y decidir el destino del paquete.

2.2. ¿Por qué?

Control. Seguridad. Vigilancia.

Control:

Cuando usas una máquina Linux para conectar tu red interna a otra red, (digamos Internet), tienes la oportunidad de permitir ciertos tipos de tráfico, y desechar otros. Por ejemplo, el encabezado de un paquete contiene la dirección de destino del paquete, así que puedes evitar paquetes que van a una cierta parte de la red externa. Otro ejemplo, uso Netscape para acceder a los archivos de Dilbert. Hay anuncios de doubleclick.net en la página, y Netscape alegremente pierde mi tiempo descargándolos. Decirle al filtro de paquetes que no permita ningún paquete hacia o desde las direcciones propiedad de doubleclick.net resuelve ese problema, (aunque hay mejores maneras de hacer esto).

Seguridad:

Cuando tu máquina Linux es la única cosa entre el caos de Internet y tu agradable y ordenada red, es bueno conocer que puedes restringir lo que llama a tu puerta. Por ejemplo, podrías permitir que algo salga de tu red, pero podrías preocuparte por el muy conocido ’Ping de la Muerte’ que llega desde los intrusos malvados. Otro ejemplo, podrías no querer que los intrusos hagan telnet a tu máquina Linux, a pesar de que todas tus cuentas tienen contraseñas; quizás deseas, (como la mayoría de las personas), ser un observador en Internet, y no un servidor, en otras palabras: simplemente no permitir a nadie conectarse, teniendo el filtro de paquetes rechazando los paquetes entrantes que inician las conexiones.

Vigilancia:

A veces una máquina mal configurada en la red local decidirá mandar paquetes al mundo exterior. Es bueno decirle al filtro de paquetes que te permita saber si ocurre algo anormal, quizás puedas hacer algo, o simplemente, eres curioso por naturaleza.

2.3. ¿Cómo?

2.3.1. Un núcleo con filtrado de paquetes.

Necesitas un núcleo que tenga el nuevo IP firewall chains. Puedes saber si el núcleo que estás ejecutando actualmente lo tiene instalado simplemente buscando el archivo '/proc/net/ip_fwchains'.

Si no es así, necesitas un núcleo que tenga ip firewall chains. Primero, descarga las fuentes del núcleo que deseas. Si tienes el núcleo 2.1.102 o superior, no necesitarás parchearlo. En caso contrario, aplica el parche de la página web listada anteriormente y configúralo como se indica abajo. Si no sabes hacerlo, no tengas miedo: lee el Kernel-HOWTO.

Las opciones de configuración que necesitarás para las series 2.0 del núcleo son:

```
CONFIG_EXPERIMENTAL=y
CONFIG_FIREWALL=y
CONFIG_IP_FIREWALL=y
CONFIG_IP_FIREWALL_CHAINS=y
```

Para las series 2.1 o 2.2 del núcleo:

```
CONFIG_FIREWALL=y
CONFIG_IP_FIREWALL=y
```

La herramienta `ipchains` le dice al núcleo qué paquetes se filtran. A menos que seas programador, o muy curioso, está es la forma habitual de controlar el filtrado de paquetes.

2.3.2. ipchains

La herramienta `ipchain` inserta y borra reglas en la sección de filtrado de paquetes del núcleo. Esto significa que da igual cómo se configure, ya que se perderá cuando reinicies tu ordenador. Ver Section 2.3.3 para asegurarte de que se restauran la próxima vez que reinicias Linux.

`Ipchains` reemplaza a `ipfwadm` usado en el antíguo código de Ip firewall. Existe un conjunto de guiones muy útiles en:

http://netfilter.filewatcher.org/ipchains/ipchains-scripts-1.1.2.tar.gz

El paquete también contiene un guión shell llamado `ipfwadm-wrapper` que le permitirá hacer filtrado tal como se hacia antes. No debes usar este guión a menos que desees una manera rápida de actualizar un sistema que use `ipfwadm`, (es más lento, y no verifica argumentos, etc). En ese caso, no te servirá de mucho este HOWTO. Ver el apéndice Chapter 8 y el apéndice Chapter 9 para más detalles sobre problemas de `ipfwadm`.

2.3.3. Hacer que las reglas sean permanentes

Tu configuración de cortafuegos se almacena en el núcleo, por tanto se perderá cuando lo reinicies. Recomiendo que uses los guiones ‘ipchains-save’ e ‘ipchains-restore’ para mantener tus reglas de forma permanente. Para ello, configura tus reglas y después, como root:

```
#ipchains-save > /etc/ipchains.rules
#
```

Crea un script como el siguiente:

```
#! /bin/sh
# Script to control packet filtering.
# If no rules, do nothing.
[ -f /etc/ipchains.rules ] || exit 0
case "$1" in
    start)
```

```
echo -n "Turning on packet filtering:"
        /sbin/ipchains-restore < /etc/ipchains.rules || exit 1
echo 1 > /proc/sys/net/ipv4/ip_forward
echo "."
;;
    stop)
echo -n "Turning off packet filtering:"
echo 0 > /proc/sys/net/ipv4/ip_forward
/sbin/ipchains -F
/sbin/ipchains -X
/sbin/ipchains -P input ACCEPT
/sbin/ipchains -P output ACCEPT
/sbin/ipchains -P forward ACCEPT
echo "."
;;
    *)
echo "Usage: /etc/init.d/packetfilter {start|stop}"
exit 1
;;
esac
exit 0
```

Asegúrate de que se ejecuta en los primeros momentos del arranque. En mi caso (Debian 2.1), hago un enlace simbólico llamado ‘S39packetfilter’ en /etc/rcS.d, (esto se ejecuta antes que S40network)

Chapter 3. ¡Estoy confundido! Enrutamiento, enmascaramiento, port forwarding, ipautofw...

Este COMO trata sobre el filtrado de paquetes. Lo cual significa, decidir si a un paquete se le permite pasar o no. Sin embargo, Linux es el sitio de recreación de los hackers, y probablemente querras hacer más que eso.

Un inconveniente es que la misma herramienta, (ipchains), se usa para controlar enmascaramiento y proxy transparente, a pesar de que son dos conceptos de filtrado de paquetes totalmente distintos, (la implementación actual de Linux los confunde, dando la impresión de que se relacionan estrechamente).

Enmascaramiento y Proxies se tratan en COMOs separados, y las características auto forwarding, (auto redirección), y el port forwarding, (redirección a puerto), se manejan con herramientas separadas, a pesar de eso mucha gente me lo sigue preguntando. Incluiré una serie de situaciones comunes e indicaré cuándo debe ser aplicada cada una. La seguridad que requiere cada configuración no se discutirá aqui.

Chapter 4. La Guia de "tres lineas de Rusty" de Masquerading

Se asume que tu intefaz *externa* es ’ppp0’. Usa ifconfig para comprobarlo, y haz el cambio apropiado en caso contrario.

```
# ipchains -P forward DENY
# ipchains -A forward -i ppp0 -j MASQ
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

4.1. Publicidad Gratuita: Reglas WatchGuard.

Puedes comprar un cortafuegos. Uno, que es excelente, es el WatchGuard’s FireBox. Es excelente porque me gusta, es seguro, está basado en Linux, y porque ellos están consolidando el mantenimiento de ipchains asi como el nuevo código de firewalling (apuntado para 2.3). En pocas palabras, WatchGuard esta pagando para que yo coma mientras trabajo para ti. Por favor tenlo en cuenta.

http://www.watchguard.com

4.2. Configuraciones comunes de cortafuegos.

Diriges littlecorp.com. Tienes una red interna, y una simple conexión telefónica a Internet, (firewall.littlecorp.com que es 1.2.3.4). Usas ethernet en tu red local, y tu máquina personal se llama "myhost".

Esta sección ilustra las soluciones más comunes. Léela atentamente, porque se diferencian sútilmente.

4.2.1. Red Privada: Proxies Tradicionales.

En esta situación, los paquetes que provienen de la red privada nunca alcanzan Internet, y viceversa. La dirección IP de la red privada debe ser asignada siguiendo las especificaciones de dirección para redes privadas, (Private Network Allocations), del RFC1597 (ej. 10.*.*.*, 172.16.*.* o 192.168.*.*).

La única forma para que te conectes a Internet es a través del cortafuegos ,(firewall), que es la única máquina en ambas redes. Ejecutas un programa, (en el cortafuegos), llamado proxy para hacer esto, (hay proxies para FTP, acceso a la Web, telnet, RealAudio, Noticias de Usenet y otros servicios). Ver el Firewall-HOWTO.

Cualquier servicio al que desees acceder en Internet deberá estar en el cortafuegos. (Ver Section 4.2.5 más abajo).

Ejemplo: Permitir acceso al servicio web en internet desde la red privada.

1. La red privada tiene asignadas direcciones 192.168.1.*, siendo ’myhost’ 192.168.1.100, y la interfaz Ethernet del cortafuegos 192.168.1.1.
2. Se instala un proxy web, (ej. "squid"), en el cortafuegos, escuchando en el puerto 8080.
3. Se configura Netscape en la red privada para usar el puerto 8080 del cortafuegos como proxy.
4. No es necesario configurar DNS en la red privada.
5. Es necesario configurar el DNS en el cortafuegos.
6. No es necesario configurar ruta por defecto, (pasarela), en la red privada.

Escribes "http://slashdot.org" en el Netscape de ’myhost’.

1. Netscape conecta con el cortafuegos en el puerto 8080, usando el puerto 1050 en ’myhost’. Este pregunta por la pagina web de "http://slashdot.org".
2. El proxy consulta el nombre "slashdot.org", y obtiene 207.218.152.131. Luego, abre una conexión con esa dirección IP (usando el puerto 1025 en la interfaz

externa del cortafuegos), y pregunta al servidor web, (puerto 80), por la página.

3. Cuando recibe la página web que proviene de su conexión al servidor web, copia los datos a la conexión que proviene de Netscape.
4. Netscape te muestra la página.

ie.. Desde la perspectiva de slashdot.org, la conexión se hace desde 1.2.3.4, (la interfaz PPP del cortafuegos), puerto 1025, hacia 207.218.152.131, (slashdot.org), y puerto 80. Desde la perspectiva de ’myhost’, la conexión se hace desde 192.168.1.100 (myhost) puerto 1050, hacia 192.168.1.1 (la interfaz Ethernet del cortafuegos) puerto 8080.

4.2.2. Red Privada: Proxies Transparentes

En esta situación, los paquetes que provienen de la red privada nunca alcanzan Internet, y viceversa. La dirección IP de la red privada debe ser asignada con las Especificaciones de direccion para redes privadas, (Private Network Allocations), del RFC1597 (ej. 10.*.*.*, 172.16.*.* o 192.168.*.*).

La única forma de que las cosas conecten a Internet es pasando por el cortafuegos, que es la única máquina en ambas redes. Ejecutas un programa, (en el cortafuegos), llamado proxy transparente para hacer ésto; el núcleo envía los paquetes salientes al proxy transparente en lugar de enviarlos directamente.

Proxy transparente significa que los clientes no necesitan conocer que hay un proxy.

Cualquier servicio al que desees acceder en Internet debe ser por medio del cortafuegos. (Ver Section 4.2.5 más abajo).

Ejemplo: Permitir acceso al servio web en internet desde la red privada.

1. .La red privada tiene asignadas direcciones 192.168.1.*, con ’myhost’ siendo 192.168.1.100, el la interfaz Ethernet del cortafuegos siendo 192.168.1.1.
2. Un proxy de web transparente, (creo que hay unos parches para squid que le permiten operar de esa manera, o prueba "transproxy") se instala y configuran en el cortafuegos, escuchando en el puerto 8080.

3. Se le dice al núcleo que redirija las conexiones al puerto 80 del proxy, usando ipchains.
4. Se configura Netscape en la red privada para conexión directa.
5. Es necesario configurar DNS en la red privada, (ej. necesitas ejecutar un servidor DNS asi como un proxy en el cortafuegos).
6. Es necesario configurar la ruta por defecto, (pasarela), en la red privada, para enviar paquetes al cortafuegos.

Escribes http://slashdot.org en el Netscape de ’myhost’.

1. Netscape consulta el nombre "slashdot.org", y obtiene 207.218.152.131. Luego abre una conexión a esa dirección IP, usando el puerto local 1050, y pregunta al servidor web, (puerto 80), por la página.
2. Como los paquetes provenientes de ’myhost’ (puerto 1050) hacia slashdot.org, (puerto 80), pasan a través del cortafuegos, se redirigen al proxy transparente en el puerto 8080. El proxy transaparente abre una conexión, (usando el puerto local 1025), hacia 207.218.152.131 puerto 80, (donde los paquetes fueron enviados originalmente).
3. Cuando el proxy recibe la página web desde su conexión al servidor web, copia los datos a la conexión desde Netscape.
4. Netscape muestra la página.

ie. Desde la perspectiva de slashdot.org, la conexión se hace desde 1.2.3.4, (la interfaz PPP del cortafuegos), puerto 1025 a 207.218.152.131, (slashdot.org), puerto 80. Desde la perspectiva de ’myhost’, la conexión se hace desde 192.168.1.100 (myhost) puerto 1050, hacia 207.218.152.131, (slashdot.org) puerto 80, pero realmente esta hablando con el proxy transparente.

4.2.3. Red Privada: Enmascaramiento (Masquerading).

En esta situación, los paquetes que provienen de la red privada nunca alcanzan Internet, y viceversa. La dirección IP de la red privada debe ser asignada con las Especificaciones de direccion para redes privadas, (Private Network Allocations), del RFC1597 (ej. 10.*.*.*, 172.16.*.* o 192.168.*.*).

En lugar de usar un proxy, usaremos una capacidad especial del núcleo llamada "enmascaramiento" (masquerading). Masquerading reescribe los paquetes que pasan a través del cortafuegos, de manera que siempre se verán como si vinieran del propio cortafuegos. Luego reescribe la respuesta de tal forma que se ven como si fueran al remitente original.

Masquerading tiene módulos separados para manejar protocolos "complicados", tal como FTP, RealAudio, Quake, etc. Para protocolos de muy difícil manejo, la capacidad "auto forwarding" puede manejar algunos de ellos configurando automáticamente el "port forwarding" para conjuntos de puertos relacionados: busca "ipportfw" (núcleos 2.0) o "ipmasqadm" (núcleos 2.1).

Cualquier servicio al que desees acceder en Internet debe estar en el cortafuegos. (Ver Section 4.2.5 más abajo).

Ejemplo: Permitir acceso al servicio web en Internet desde la red privada.

1. La red privada tiene asignadas direcciones 192.168.1.*, con ’myhost’ siendo 192.168.1.100, el la interfaz Ethernet del cortafuegos siendo 192.168.1.1.
2. Se configura el cortafuegos enmascarar algunos paquetes que provienen de la red privada y que salen hacia el puerto 80 en un host en Internet.
3. Se configura Netscape para conexión directa.
4. El DNS debe estar correctamente configurado en la red privada.
5. El cortafuegos debe ser la ruta por defecto, (pasarela), para la red privada.

Escribes http://slashdot.org en el Netscape de ’myhost’.

1. Netscape consulta el nombre ’slashdot.org’, y obtiene 207.218.152.131. Luego abre una conexión a la dirección IP, usando el puerto local 1050, y solicita la página al servidor web (puerto 80).

2. Como los paquetes que provienen de ’myhost’ (puerto 1050) hacia slashdot.org (puerto 80) pasan a través del cortafuegos, son reescritos para llegar desde la interfaz PPP del cortafuegos, puerto 65000. El cortafuegos tiene una dirección válida de Internet, (1.2.3.4), de modo que los paquetes de respuesta que provienen de www.linuxhq.com obtienen la ruta de vuelta.
3. Cuando los paquetes que provienen de slashdot.org (puerto 80) hacia firewall.littlecorp.com (puerto 65000) entran, se reescriben para ir a ’myhost’, puerto 1050. Esto es lo realmente mágico del enmascaramiento: recuerda cuando reescribió los paquetes salientes y puede reescribirlos de nuevo como respuestasentrantes.
4. Netscape muestra la página.

ie.. Desde el punto de vista de slashdot.org, la conexión se hace desde 1.2.3.4 ,(la interfaz PPP del cortafuegos), puerto 65000 hacia 207.218.152.131, (slashdot.org), puerto 80. Desde la perspectiva de ’myhost’, la conexión se hace desde 192.168.1.100, (myhost), puerto 1050, hacia 207.218.152.131, (slashdot.org), puerto 80.

4.2.4. Red Pública.

En esta situación, tu red personal forma parte de Internet; los paquetes pueden fluir a través de ambas redes sin cambios. Las direcciones IP de la red interna deben ser asignadas aplicando un bloque de direcciones IP, de forma que, el resto de la red sabrá cómo llegar hasta ti. Esto implica una conexión permanente.

En este rol, el filtrado de paquetes se usa para restringir aquellos paquetes que pueden ser redirigidos entre la red y el resto de Internet, ej. para restringir al resto de Internet de forma que acceda solamente a sus servidores web internos.

Ejemplo: Permitir acceso al servicio web en internet desde la red privada.

1. Tu red internet es asignada conforme a un bloque de direcciones IP que tienes asignadas, (digamos 1.2.3.*).
2. Configuramos el cortafuegos para permitir todo el tráfico.

3. Configuramos Netscape para conectarse directamente.
4. El DNS debe estar correctamente configurado en tu red.
5. El cortafuegos deberia ser la ruta por defecto, (pasarela), de la red privada.

Escribes http://slashdot.org en el Netscape de ’myhost’.

1. Netscape consulta el nombre "slashdot.org", y obtiene 207.218.152.131. Luego abre una conexión a esa dirección IP, usando el puerto local 1050, y solicita la página al servidor web (puerto 80).
2. Los paquetes pasan a través del cortafuegos, de la misma forma como pasan a través de varios routers entre tú y slashdot.org.
3. Netscape muestra la página.

ie. Sólo existe una conexión: desde 1.2.3.100 (myhost) puerto 1050, hacia 207.218.152.131 (slashdot.org) puerto 80.

4.2.5. Servicios internos limitados

Hay unos pocos trucos para que Internet acceda a sus servicios internos, sin que sea necesario ejecutarlos en el cortafuegos. Esto funcionará para conexiones externas basadas en aproximaciones tanto de proxy como de enmascaramiento.

La aproximación mas simple es ejecutar un "redirector", que es un proxy de "pobres", que espera una conexión en un puerto dado y luego abre una conexión a un determinado host interno y un puerto, copiando los datos entre ambas conexiones. Un ejemplo de esto es el programa "redir". Desde el punto de vista de Internet, la conexión se hace a su cortafuegos. Desde el punto de vista de su servidor interno, la conexión se hace desde la interfaz interna del cortafuegos hacia el servidor.

Otra aproximación, (que requiere un parche del núcleo 2.0 para ipportfw, o un núcleo 2.1 o superior), es usar port forwarding en el núcleo. Esto hace lo mismo que "redir" pero de forma diferente: el núcleo reescribe los paquetes a medida que lo atraviesan, cambiando su dirección de destino y puerto para que apunte a un host y puerto internos.

Desde la perspectiva de Internet, la conexión se hace a tu cortafuegos. Desde el punto de vista del servidor interno, se hace una conexión directa desde el host situado en Internet hacia el servidor.

4.3. Más información de Masquerading

David Ranch ha escrito un nuevo y excelente HOWTO de Masquerading, que tiene mucho en común con este HOWTO. Podrás encontrarlo en:

http://www.linuxdoc.org/HOWTO/IP-Masquerade-HOWTO.html

La página oficial de masquerading está en:

http://ipmasq.cjb.net

4.4. Cadenas de filtrado (IP firewalling chains).

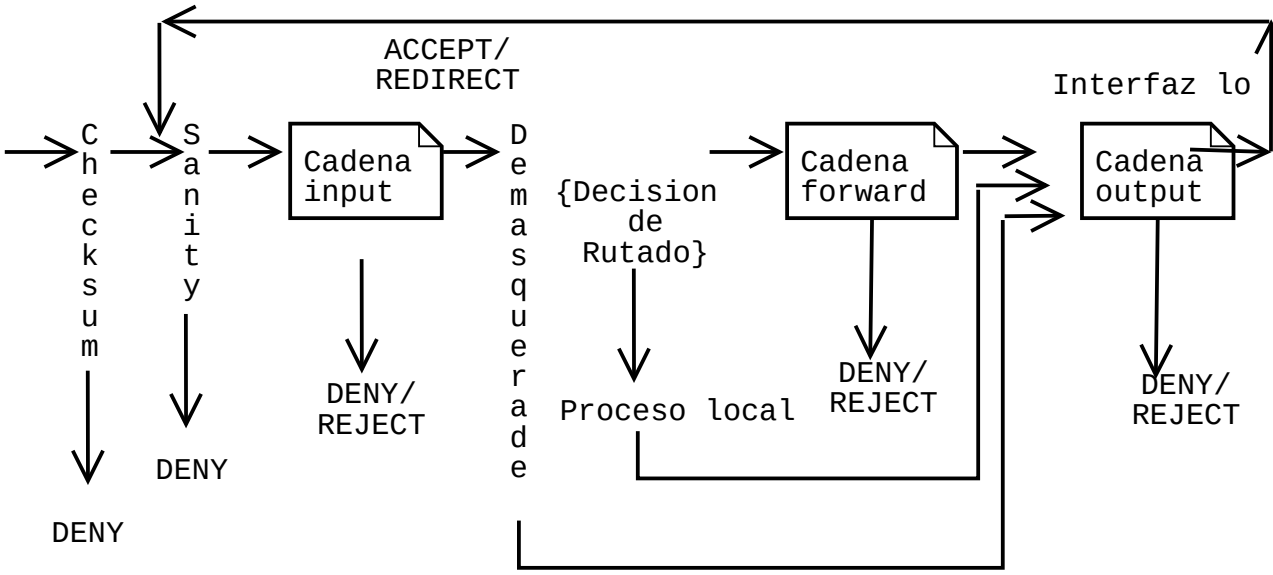
Esta sección describe lo que realmente necesitas saber para construir un filtro de paquetes que satisfaga tus necesidades.

4.5. Cómo atraviesan los filtros los paquetes.

El Kernel comienza con tres listas de reglas, estas listas se llaman *cadenas (chains) de cortafuegos* o simplemente *cadenas*. Las tres cadenas se llaman *input (entrada)*, *output (salida)* y *forward (reenvío)*. Cuando un paquete entra (digamos, a través de la tarjeta ethernet) el núcleo usa la cadena `input` para decidir su destino. Si sobrevive a este paso, entonces el núcleo decide a dónde enviar el paquete, (esto se llama enrutamiento --routing--). Si el destino es otra máquina, consulta la cadena `forward`, finalmente, justo antes de que el paquete saalga, el núcleo consulta la cadena `output`.

Una cadena es una lista de chequeo (checklist) de *reglas*. Cada regla dice ’si el encabezado del paquete tiene este aspecto, entonces esto es lo que deseo hacer con el paquete’. Si la regla no encaja con el paquete, entonces se consulta la próxima regla en la cadena. Finalmente, si no hay ninguna regla más por consultar, entonces el núcleo mira la política de la cadena para decidir qué hacer. En un sistema de seguridad-consciente, esta política normalmente le dice al núcleo que rechace o deniegue el paquete.

Figure 4-1. diagrama0



Esto es una descripción simplificada de cada fase:

Checksum:

> En esta fase se comprueba que el paquete no se ha adulterado de alguna manera. Si así pasa, se deniega.

Sanidad:

Efectivamente hay verificadores de sanidad antes de cada cadena del cortafuegos, pero la cadena de entrada (input) es más importante. Algunos paquetes malformados podrían confundir el código de verificación de la regla, y éstos son denegados aquí, (un mensaje queda en el syslog si esto pasa).

Cadena de entrada (Input chain):

Ésta es la primera cadena que deberá superar el paquete. Si el resultado de la cadena es no denegar o no rechazar, el paquete sigue su curso.

Desenmascaramiento (Demasquerade)

:

Si el paquete es una respuesta a un paquete previamente enmascarado, es desenmascarado, y pasa directamente a la cadena de salida, (output chain). Si no usa Ip Masquerading, puede borrar este diagrama de su mente.

Decisión de Enrutamiento (routing):

El campo de destino es examinado por el código de enrutamiento para decidir si este paquete debe ser enviado a un proceso local, (veáse proceso Local más abajo), o remitido a una máquina remota, (veáse Cadena de envio -- forward -- más abajo).

Proceso local:

Un proceso que corre en la máquina puede recibir paquetes después de la fase de decisión de enrutamiento, y puede enviar paquetes, (que pasan por la cadena de salida --output chain-- después a la cadena de entrada --input chain-- por medio de la interfaz ’lo’ si están destinados para un procceso local, caso contrario solo atraviesan la cadena de salida). Este cruce entre cadenas por inter-proceso no se muestra totalmente en el diagrama, es demasiado gráfico..

Local:

Si el paquete no fue creado por un proceso local, entonces se verifica la cadena de reenvio (forward chain), si no el paquete va directamente a la cadena de salida.

Cadena de reenvío (forward chain):

Todo paquete que intente pasar a través de esta máquina a otra, atravesará esta cadena.

Cadena de salida (output chain):

Justo antes de ser enviados atravesarán esta cadena.

4.5.1. Usando ipchains

Primero, verifique que tiene la versión de ipchains a la que este documento se refiere:

```
$ ipchains --version
ipchains 1.3.9, 17-Mar-1999
```

Nota: recomiendo 1.3.4 (que no tiene opciones largas como --sport), o 1.3.8 o superior; éstas son muy estables.

El ipchains tiene una página de manual bastante detallada (`man ipchains`), y si necesitas más detalles particulares, puedes comprobar la interfaz de programación (`man 4 ipfw`), o el archivo `net/ipv4/ip_fw.c` en las fuentes del núcleo 2.1.x que tienen, (obviamente), la última palabra.

Hay también una excelente guía de referencia de Scott Bronson en las fuentes, en A4 y US letter PostScript(TM).

Hay varias cosas que se pueden hacer con `ipchains`. Primero las operaciones para manejar cadenas enteras. Empieza con la construcción de tres cadenas `input`, `output` y `forward`, las cuales no puedes borrar.

1. Crear una nueva cadena (-N).
2. Borrar una cadena vacía (-X).
3. Cambiar la política para una cadena construida. (-P).
4. Listar las reglas de una cadena (-L).

5. Vaciar las reglas de una cadena (-F).
6. Poner a cero el contador de paquete y de byte en todas las reglas en una cadena (-Z).

Hay varias formas de manipular reglas dentro de una cadena:

1. Añadir una nueva regla a una cadena (-UN).
2. Insertar una nueva regla en alguna posición en una cadena (-yo).
3. Reemplazar una regla en alguna posición en una cadena (-R).
4. Anular una regla en alguna posición en una cadena (-D).
5. Anular la primera regla que se cumple en una cadena (-D).

Hay unas pocas operaciones para enmascarar, que se encuentran en `ipchains` por la necesidad de un buen sitio para ponerlas:

1. Listar las conexiones enmascaradas actualmente (-M -L).
2. Asignar valores de timeout de enmascaramiento (-M -S). ¡(Veáse Section 6.10).

La función final, (y quizás la más útil), es la que permite verificar lo que le pasaría a un paquete dado si éste atraviesa por una cadena dada.

4.5.2. Qué verás cuando enciendas el ordenador.

Antes de que se ejecute ninguna orden ipchains, (se cuidadoso: algunas distribuciones ejecutan ipchains en sus scripts de inicio), no habrá ninguna regla en ninguna de las cadenas existentes, (input, output, forward), y cada una de las reglas tendrá la política de ACCEPT. Esto es lo que obtendrás.

4.5.3. Operaciones en una sola regla.

Esto es la esencia de ipchains; manipulación de reglas. En la mayoría de casos, probablemente usarás los comandos de añadir (-A) y borrar (-D). Los otros (-I para insertar y -R para remplazar) son simples extensiones de estos conceptos.

Cada regla especifica un juego de condiciones que el paquete debe cumplir, y qué hacer si las cumple (objetivo -- target --). Por ejemplo, podrías querer denegar todos los paquetes ICMP que provienen de la dirección IP 127.0.0.1. Así que, nuestras condiciones son: el protocolo debe ser ICMP y la dirección orígen debe ser 127.0.0.1. Nuestro objetivo es denegar ’DENY’.

127.0.0.1 es la interfaz de bucle de retorno --loopback-- que existirá aún cuando no tengas ninguna conexión real de red. Puedes usar el programa ’ping’ para generar taless paquetes, (envía un ICMP de tipo 8 ,(echo request), al que todos los host coperativos deben obligatoriamente responder con un paquete ICMP de tipo 0 (echo reply)). Esto es útil para probar:

```
# ping -c 1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.2 ms
--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.2/0.2/0.2 ms
# ipchains -A input -s 127.0.0.1 -p icmp -j DENY
# ping -c 1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
--- 127.0.0.1 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
#
```

Puedes ver aquí que el primer ping fue exitoso, (el ‘-c 1’ indica a ping que envie sólo un paquete).

Luego añadimos, (-A), a una cadena ’input’, una regla que especifica que: para los paquetes que provienen de 127.0.0.1, (-s 127.0.0.1), con protocolo ICMP, (-p ICMP), debemos pasar a DENEGAR (’-j DENY’).

Luego probamos nuestra regla, usando el segundo ping. Habrá una pausa antes de que el programa deje de esperar una respuesta que nunca llegará.

Podemos borrar la regla de cualquiera de estas dos formas. Primero, puesto que sabemos que es la única regla en la cadena input, podemos usar un borrado numerado, así:

```
# ipchains -D input 1
#
```

Para borrar la regla número 1 de la cadena input.

La segunda forma es como con el parámetro -A, pero reemplazando -A por -D. Esto es útil cuando tiene una cadena compleja de reglas y no quiere tener que contarlos para concluir que es la regla 37 de la que desea librarse. En este caso, usamos:

```
# ipchains -D input -s 127.0.0.1 -p icmp -j DENY
#
```

La sintaxis de -D debe tener exactamente las mismas opciones que el comando -A (o -I o -R). Si hay multiples reglas idénticas en la misma cadena, sólo se borrará la primera de ellas.

4.5.4. Especificaciones de filtrado.

Hemos visto el uso de ’-p’ para especificar protocolo, y ’-s’ para especificar la dirección del orígen, pero hay otras opciones que podemos usar para especificar las características del paquete. Lo que sigue es un compendio exhaustivo.

4.5.4.1. Especificando direcciones IP orígen y destino.

Direcciones IP de orígen (-s) y destino (-d) pueden especificarse de cuatro maneras. La manera más común es usar el nombre completo, como ’localhost’ o ’www.linuxhq.com’. La segunda manera es especificar la dirección IP tal como ’127.0.0.1 ’.

La tercera y cuarta formas permiten especificar un grupo de direcciones IP, como ‘199.95.207.0/24 ’ o ‘199.95.207.0/255.255.255.0 ’. Estas dos especifican cualquier IP del rango desde 192.95.207.0 hasta 192.95.207.255; los dígitos después del ’/’ dicen qué partes de la dirección de IP son significativas. ’/32’ o ’/255.255.255.255’ es el valor por defecto, (todas las direcciones de IP). Para especificar cualquier IP se puede usar ’/0’, así:

```
# ipchains -A input -s 0/0 -j DENY
#
```

Esto rara vez se usa, puesto que el efecto es el mismo que no especificar la opción ’-s’.

4.5.4.2. Especificando Inversión (negaciones).

Muchas marcas, incluso las flags ‘ -s ’ y ‘ -d ’ pueden tener sus argumentos precedidos por ‘! ’ (‘pronunciado NOT’) para referir a direcciones que no son iguales a las dadas. Por ejemplo ‘ -s ! localhost’ se refiere a cualquier paquete que no proviene del localhost.

No olvide los espacios antes y después del ‘!’: s realmente necesario.

4.5.4.3. Especificando Protocolo.

El protocolo puede especificarse con la marca ‘ -p ’. El protocolo puede ser un número (si usted conoce los valores numéricos de protocolos para IP) o un nombre para los casos especiales de ‘TCP ’, ‘UDP ’ o ‘ICMP ’. No importa mínusculas o mayúsculas, ‘tcp ’ funciona igual que ‘TCP ’.

El nombre de protocolo puede estar precedido por ’!’, para invertirlo, tal como ’ -p ! TCP’ .

4.5.4.3.1. Especificando puertos UDP y TCP.

Para el caso especial donde se especifica un protocolo de TCP o UDP, puede haber un argumento extra que indica el puerto TCP o UDP, o un rango de puertos (Veáse Section 4.5.4.6 mas abajo). Un rango se representa usando el caracter ":", como "6000:6010" que cubre 11 números de puerto, desde 6000 hasta 6010. Si se omite el limite inferior, se asigna por defecto 0. Si se omite el límite superior, se asigna por defecto 65535. Así pues, para especificar conexiones de TCP en los puertos por debajo del 1024, la sintáxis sería ’ -p TCP -s 0.0.0.0/0 :1024 ’ . Los números de puerto se pueden especificar usando nombre, ej. ’www’.

Notar que el puerto especificado puede estar precedido por ’!’ para indicar "lo contrario" (inversion). Así pues, para especificar todos los paquetes TCP excepto los WWW, lo indicará así: -p TCP -d 0.0.0.0/0 ! www Es importante comprender que la especificación -p TCP -d ! 192.168.1.1 www es muy diferente de -p TCP -d 192.168.1.1 ! www

La primera especifica cualquier paquete de TCP al puerto de WWW en cualquier máquina excepto 192.168.1.1. El segundo especifica alguna conexión de TCP a cualquier puerto en 192.168.1.1 excepto el puerto de WWW.

Finalmente, este caso significa que no a los de puerto WWW y no a los de dirección 192.168.1.1: -p TCP -d ! 192.168.1.1 ! www

4.5.4.3.2. Especificando ICMP tipo & código.

ICMP también permite un argumento opcional, puesto que ICMP no tiene puertos, (ICMP tiene un *tipo* y un *código*) que tienen significados diferentes.

Puedes especificarlos como nombres ICMP (usar `ipchains -h icmp` para listar los nombres) después de la opción ‘ -s ’, o como un tipo y código númerico ICMP, donde el tipo sigue a la opción ’-s’ y el código sigue a la opción ’-d’.

Los nombres de ICMP son bastante largos: sólo necesitas usar las letras suficientes para hacer que un nombre sea distinto de otro.

Aquí está una pequeña tabla de algunos de los paquetes ICMP más comunes:

```
Número  Nombre                  Requerido por
0       echo-reply              ping
```

```
    3       destination-unreachable  Any TCP/UDP traffic.
    5       redirect                 routing if not run-
ning routing daemon
    8       echo-request             ping
    11      time-exceeded            traceroute
```

Notar que por el momento los nombres de ICMP no pueden ser precedidos por ‘! ’.

NO NO NO bloquee todos los mensajes ICMP tipo 3 (Vea Section 5.2.1 más abajo).

4.5.4.4. Especificando una interfaz.

La opción ‘ -i ’ especifica el nombre de una interfaz para comparar. Una interfaz es el dispositivo físico por el cual un paquete llega o sale. Puedes usar el comando ifconfig para listar las interfaces activas. (ie. que estén funcionando en el momento).

La interfaz para los paquetes entrantes (ie. paquetes que atraviesan la cadena input) se consideara la interfaz por donde entran. Lógicamente, la intercaz para los paquetes salientes (paquetes que atraviesan la cadena output) es la interfaz por donde salen. La interfaz para que los paquetes atraviesen la cadena forward es también la interfaz por salen; una decisión bastante arbitraria a mi modo de ver.

Es absolutamente legal especificar una interfaz que actualmente no existe; la regla no coincidirá con nada hasta que la interfaz sea activada. Esto es sumamente útil para enlaces telefónicos ppp (normalmente interfaz `ppp0`) y similares.

Como caso especial, un nombre de interfaz que termine con un ‘+ ’ coincidirá con todas las interfaces (tanto si existen actualmente como si no) qué empieza con esa cadena. Por ejemplo, para especificar una regla que se refiere a todas las interfaces PPP entonces se usaría `-i ppp+`.

El nombre de la interfaz puede ir precedido por ‘! ’ para indicar un paquete que no coincide con la interface(s) especificada.

4.5.4.5. Especificando sólo paquetes SYN TCP.

A veces es útil permitir conexiones de TCP en un sentido, pero no en el otro. Por ejemplo, podrías permitir conexiones hacia un servidor de WWW externo, pero no las que provienen desde él.

Una solucíon ingenua sería bloquear paquetes de TCP que vienen del servidor. Desafortunadamente, para que trabajen las conexiones de TCP requieren paquetes que van en ambos sentidos.

La solución es bloquear sólo los paquetes de petición de conexión. A estos paquetes se les llama paquetes *SYN* (ok, técnicamente son aquellos paquetes con el flag SYN activado (SYN flag) y los flags FIN y ACK limpios, pero nosotros los llamaremos paquetes SYN). Impidiendo sólo estos paquetes, podemos detener los intentos de conexión .

El flag ‘ -y ’ se usa para esto: es sólo válida para reglas que especifican TCP como su protocolo. Por ejemplo, especificar intentos de conexión TCP desde 192.168.1.1: -p TCP -s 192.168.1.1 -y

Una vez más, este flag se puede invertir precediéndolo de un ‘! ’ que significa todos los paquetes restantes a los que inician conexión.

4.5.4.6. Manejando Fragmentos.

A veces un paquete es demasiado grande como para "caber" en el cable de transmisión de una sola vez. Cuando esto pasa, el paquete se dividide en fragmentos , y se envía como múltiples paquetes. En el otro extremo se reensamblan los fragmentos para reconstruir el paquete entero.

El problema con los fragmentos es que algunas de las especificaciones que se mencionaron anteriormente (en particular, puerto orígen, puerto destino, tipo ICMP, código ICMP, o marca TCP SYN) requieren que el núcleo mire al inicio del paquete, que sólo esta en el primer fragmento.

Si tu máquina es la única conexión a una red externa, entonces puedes indicar al núcleo que reensamble los fragmentos que lo atraviesan, compilando el núcleo con la opción

`IP: always defragment` en ‘Y’ . Esto evita el problema limpiamente..

Por otra parte, es importante entender cómo se tratan los fragmentos por las reglas de filtración. Cualquier regla que nos pregunte por información que no se tiene,*no* se cumplirá. Esto significa que el primer fragmento se trata como cualquier otro paquete. El segundo y siguientes fragmentos no lo serán. Así pues, una regla `-p TCP -s 192.168.1.1 www` (especificando un puerto de orígen www) nunca coincidirá con un fragmento (salvo con el primer fragmento). La regla opuesta tampoco `-p TCP 192.168.1.1 ! www`

Sin embargo, puedes especificar una regla específicamente para el segundo y demás, usando el flag ‘ -f ’. Obviamente, es ilegal especificar un puerto TCP o UDP, tipo ICMP, código ICMP o marca TCP SYN en tal regla del fragmento.

También es legal especificar que una regla no se aplica para el segundo y demás fragmentos, precediendo ‘ -f ’ con ‘! ’.

Normalmente se considera seguro dejar pasar al segundo y demás fragmentos , puesto que el filtrado afectará al primer fragmento, de ese modo se evita reensamblar en el host destino, sin embargo, se han detectado bugs que hacen que la máquina falle simplemente por enviar fragmentos.

Nota: para los encabezados de red paquetes malformados (paquetes TCP, UDP e ICMP demasiado cortos para que el codigo del cortafuegos lea los puertos o los tipos o códigos ICMP) también se tratan como fragmentos. Sólo los fragmentos TCP que empiezan en la posición 8 son desechados explicitamente por el código de cortafuegos (un mensaje debe aparecer en el syslog si esto ocurre).

Como un ejemplo, la siguiente regla desecha cualquier fragmento que va a 192.168.1.1:

```
# ipchains -A output -f -D 192.168.1.1 -j DENY
#
```

4.5.5. Efectos secundarios del filtrado.

OK, hasta ahora sabemos todas las formas de coincidir con un paquete usando una regla. Si un paquete encaja con una regla, pasa lo siguiente.

1. .El contador de byte para esa regla es aumentado por el tamaño del paquete (encabezado y todo lo demás).
2. El contador de paquetes para esa regla se incrementa.
3. Si la regla lo requiere, el paquete es anotado.
4. Si la regla lo requiere, el campo Type Of Service es cambiado.
5. Si la regla lo requiere, el paquete es marcado (no en las series 2.0 del núcleo).
6. El objetivo de la regla es examinado para determinar que se hace después con el paquete..

Para variar, los he puesto en orden de importancia.

4.5.5.1. Especificando un objetivo.

Un objetivo *target* indica al núcleo que hacer con un paquete que encaja con una regla. El ipchains usa ’-j’ (saltar a) para especificar el objetivo. El nombre del objetivo (target) dee ser menor de 8 caracteres y es sensible a mayusculas/minusculas: "RETURN" y "return" son completamente diferentes.

El caso más simple es cuando no se ha especificado el objetivo. Este tipo de regla (a menudo llamada regla de "accounting") es útil simplemente para contar un cierto tipo de paquetes. Empareje o no con la regla, el núcleo examina la próxima regla en la cadena. Por ejemplo, para contar el número de paquetes provenientes de 192.168.1.1, podemos hacer esto:

```
# ipchains -A input -s 192.168.1.1
#
```

(Usando ‘ipchains -L -v ’ podemos ver el contador de bytes y de paquetes asociados con cada regla).

Hay seis objetivos especiales. Los tres primeros, son bastante simples. `ACEPTAR (ACCEPT) RECHAZAR (REJECT)` y `DENEGAR (DENY)`. `ACCEPT` permite que el paquete lo atraviese. `DENY` desecha el paquete como si nunca hubiera sido recibido. `REJECT` desecha el paquete, pero (si no es un paquete ICMP) genera una respuesta de ICMP al orígen que dice que el destino fue inalcanzable.

El siguiente, `MASQ` le dice al núcleo que enmascare el paquete. Para que esto funcione, su núcleo debe ser compilado con Ip Masquerading habilitado. Para mas detalles, ver el IP-Masquerading-HOWTO y el Apéndice Chapter 8 Este objetivo solo es válido para los paquetes que atraviesan la cadena forward.

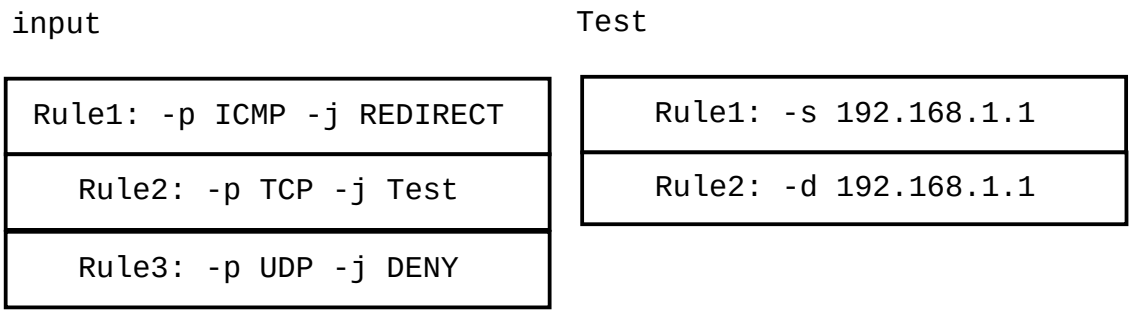
El otro objetivo especial es `REDIRECT` que le dice al núcleo que envie el paquete a un puerto local en lugar de un sitio cualquiera donde fuera dirigido. Puede ser especificado solo para reglas que contienen protocolos TCP y UDP. Opcionalmente, un puerto (nombre o número) puede especificarse después de '-j REDIRECT' lo cual hace que el paquete sea redirigido a un puerto particular incluso si fue direccionado a otro puerto. Este objetivo es válido solo para paquetes que atraviesan la cadena `input`.

El último objetivo especial es `RETURN` que es similar a descender de la cadena inmediatamente. (Ver Section 4.5.5.11 más abajo).

Cualquier otro objetivo indica una cadena definida por el usuario (como se describe en Section 4.5.5.5 más abajo). El paquete empezará atravesando las reglas en esa cadena. Si esa cadena no decide el destino del paquete, entonces una vez que la travesia por esa cadena ha terminado, la travesia se reasume en la próxima regla en la cadena actual.

Considere dos cadenas: input (la cadena construida --built-in--) y Test (una cadena definida por el usuario --user-defined--).

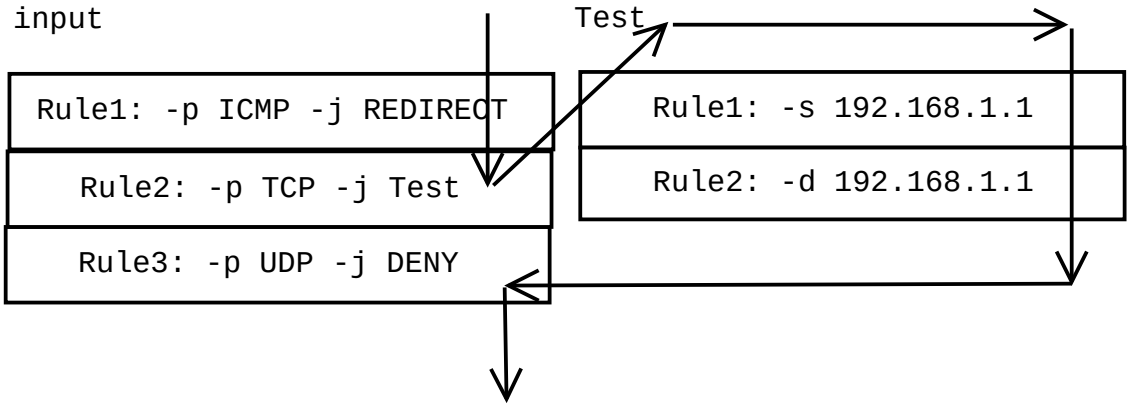
Figure 4-2. diagrama1



Considere un paquete de TCP que viene desde 192.168.1.1, que va hacia 1.2.3.4. Entra en la cadena `input`, y se prueba contra la Regla 1: no concuerda. Regla 2: concuerda, y el objetivo es `Test`, tal que la siguiente regla es la primera de `Test`. La regla 1 en test concuerda, pero no especifica un objetivo, tal que la próxima regla es examinada, Regla 2. Esta no concuerda, tal que hemos alcanzado el final de la cadena. Regresamos a la cadena `input`, ahora a la Regla 3, la cual no coincide.

Así que la ruta del paquete es:

Figure 4-3. diagrama2



Vea la sección Section 5.1 para saber la forma de cómo usar eficazmente las cadenas definidas por el usuario.

4.5.5.2. "logeando" paquetes.

Este es un efecto secundario que puede tener el coincidir con una regla; puede tener el log de paquete coincidente con usando la marca ’-l’. Usualmente no querrá esto para los paquetes rutinarios, pero es una característica muy útil si desea observar eventos fuera de lo común.

El núcleo anota esta información de esta forma:

```
    Packet log: in-
put DENY eth0 PROTO=17 192.168.2.1:53 192.168.1.1:1025
      L=34 S=0x00 I=18 F=0x0000 T=254
```

Este mensaje de log esta diseñado para ser conciso, y contener información técnica útil para los gurus de las redes, pero puede ser útil para el resto de nosotros. Se interpreta así:

1. ‘input’ es la cadena que contiene la regla la cual se ha emparejado con el paquete, causando el mensaje de anotación.
2. ‘DENY’ es lo que la regla le dice al paquete. Si es ’-’ entonces la regla no afecta absolutamente al paquete (en reglas de accounting).
3. ‘eth0’ es el nombre de la interface. Puesto que fue la cadena input, significa que el paquete llegó por ’eth0’.
4. ‘PROTO=17’ significa que el paquete fue del protocolo 17. Una lista de los números de protocolos se encuentra en ’/etc/protocols’. Los más comúnes son 1 (ICMP), 6 (TCP) y 17 (UDP).
5. ‘192.168.2.1’ significa que la dirección IP de orígen del paquete fue 192.168.2.1.
6. ‘:53’ significa que el puerto de orígen fue el 53. Mirando en ’/etc/services’ observa que este es el puerto ’domain’ (ej. Es probablemente el puerto de respuesta del

DNS). Para UDP y TCP, este número es el puerto de orígen. Para ICMP, es el tipo ICMP. Para otros, será 65535.

7. .‘192.168.1.1’ es la dirección IP de destino
8. ‘:1025’ significa que el puerto destino fue el 1025. Para UDP y TCP, este número es el puerto destino. Par ICMP, es el código ICMP. Para otros, será 65535.
9. ‘L=34’ significa que el paquete tenia una longitud total de 34 bytes.
10. .‘S=0x00’ significa que el campo Type Of Service (dividir por 4 para obtener el Tipo de Servicio tal como lo usa ipchains).
11. .‘I=18’ es el ID de IP.
12. .‘F=0x0000’ es el desplazamiento del fragmento de 16 bits más banderas. Un valor que comience con ’0x4’ o ’0x5’ significa que el bit de fragmento no está activo. ’0x2’ o ’0x3’ significa que el bit ’More Fragments’ (Más fragmentos) está activo; se esperan más fragmentos después de este. El resto de números es el desplazamiento de este fragmento, dividido por 8.
13. .‘T=254’ es el tiempo de vida (Time To Live) del paquete. Es deducido desde este valor para todo salto, y comunmente empieza en 15 o 255.

En sistemas Linux estándard, esta salida del núcleo es capturada por klogd (el demonio de logs del núcleo) el cual lo entrega a syslogd (el demonio de logs del sistema). El archivo ’/etc/syslog.conf’ controla el comportamiento de syslogd, porque especifica un destino para cada ’establecimiento -- facility’ (en nuestro caso, el establecimiento es "núcleo") y ’nivel -- level ’ (para ipchains, el nivel usado es "info").

Por ejemplo, mi (Debian) /etc/syslog.conf contiene dos lineas que concuerdan con "kern.info":

```
kern.*                          -/var/log/kern.log
*.=info;*.=notice;*.=warn;\
        auth,authpriv.none;\
        cron,daemon.none;\
        mail,news.none          -/var/log/messages
```

Esto significa que los mensajes son duplicados en ’/var/log/kern.log’ y ’/var/log/messages’. Para más detalles, ver ’man syslog.conf’.

4.5.5.3. Manipulando el Tipo de servicio (Type Of Service)

Hay cuatro bits en el encabezado IP, llamados bits del *Tipo de Servicio* (TOS). Afectan a la forma de tratar los paquetes. Los cuatro bits son "Retraso mínimo", "Máximo Throughput", "Fiabilidad Máxima" y "Costo Mínimo". Solo se permite configurar uno de estos bits. Rob van Nieuwkerk, autor del código TOS-mangling, dice:

“Especialmente el "Retraso Mínimo" es importante para mí. Lo cambio a "interactivo" en mi router (Linux). Estoy tras un enlace de modem de 33k6. Linux prioriza paquetes en 3 colas. De esta forma obtengo un rendimiento interactivo aceptable mientras mientras se hace descargas pesadas al mismo tiempo. (Incluso podría ser mejor si no hubiese semejante cola en el driver serial, excepto que la latencia se mantiene por debajo de 1.5 segundos). ”

Nota: obviamente, no tienes ningún control sobre los paquetes entrantes; sólo puedes controlar la prioridad de los paquetes que abandonan tu máquina. ara negociar prioridades con el otro extremo , se debe usar otro protocolo, como el RSVP (del que no sé nada, así que no me pregunteis). .

El uso más común es poner las conexiones telnet a "Retraso Mínimo" y los datos FTP a "Máximo Throughput." Esto se haría como sigue:

```
ipchains -A output -p tcp -d 0.0.0.0/0 telnet -t 0x01 0x10
ipchains -A output -p tcp -d 0.0.0.0/0 ftp -t 0x01 0x10
ipchains -A output -p tcp -s 0.0.0.0/0 ftp-data -t 0x01 0x08
```

La marca ‘ -t ’ toma dos parámetros extras, ambos en hexadecimal. Éstos permiten un juego completo de los bits de TOS: la primera máscara es con AND con el TOS actual de los paquetes, y luego la segunda máscara es XOR. Si esto lo confunde, simplemente use la siguiente tabla:

```
          Nombre del TOS          Valor         Usos típicos
```

```
Minimum Delay           0x01 0x10     ftp, telnet
Maximum Throughput      0x01 0x08     ftp-data
Maximum Reliability     0x01 0x04     snmp
Minimum Cost            0x01 0x02     nntp
```

Andi Kleen goes on to point out the following (mildly edited for posterity): “Maybe it would be useful to add an reference to the txqueuelen parameter of ifconfig to the discussion of TOS bits. The default device queue length is tuned for ethernet cards, on modems it is too long and makes the 3 band scheduler (which queues based on TOS) work suboptimally. It is a good idea to set it to a value between 4-10 on modem or single b channel ISDN links: on bundled devices a longer queue is needed. This is a 2.0 and 2.1 problem, but in 2.1 it is a ifconfig flag (with recent nettools), while in 2.0 it requires source patches in the device drivers to change.”

So, to see maximal benifits of TOS manipulation for modem PPP links, do ‘ifconfig $1 txqueuelen’ in your /etc/ppp/ip-up script. The number to use depends on the modem speed and the amount of buffering in the modem; here’s Andi setting me straight again:

“The best value for a given configuration needs experiment. If the queues are too short on a router then packets will get dropped. Also of course one gets benefits even without TOS rewriting, just that TOS rewriting helps to give the benefits to non cooperating programs (but all standard linux programs are cooperating). ”

4.5.5.4. Marcando un paquete. (no es mi culpa que suene así)

Permite poderosas y complejas interacciones con la nueva implementación "Quality Of Service" de Alexey Kuznetsov, así como el "redireccionamiento" basado en flags de los núcleos posteriores al 2.1. Más noticias cuando las tengamos. Esta opción se ignora totalmente en las series 2.0 de núcleo.

4.5.5.5. Operaciones en una cadena entera.

Un rasgo muy útil de ipchains es la habilidad de agrupar reglas relacionadas en las cadenas. Puede llamar a las cadenas como usted quiera, mientras no se llamen como las

cadenas contruidas (`input output forward`) o los objetivos (`MASQ REDIRECT ACCEPT DENY REJECT RETURN`). Sugiero evitar las mayúsculas completamente, puesto que se pueden usar para futuras ampliaciones. El nombre de la cadena puede estar por encima de los 8 caracteres.

4.5.5.6. Crear una nueva cadena.

Creemos una nueva cadena. Como soy un muy imaginativo, la llamaré test

```
# ipchains -N test
#
```

Es así de simple. Ahora usted puede poner reglas como se detalló anteriormente.

4.5.5.7. Borrar una cadena.

Borrar una cadena también es simple.

```
# ipchains -X test
#
```

¿Por qué ‘ -X ’? Bien, todas las letras buenas ya se usaron.

Hay un par de restricciones al borrar cadenas: deben estar vacías (vea Section 4.5.5.8 más abajo) y no deben ser el objetivo de cualquier otra regla. No puede borrar ninguna de las tres cadenas construidas (input output forward).

4.5.5.8. Vaciar una cadena.

Hay una manera simple de vaciar todas las reglas de una cadena, usando la orden ‘ -F ’.

```
# ipchains -F forward
#
```

Si usted no especifica una cadena, entonces *todas* las cadenas se vaciarán.

4.5.5.9. Listar una cadena.

Puede listar todas las reglas en una cadena usando la orden ‘ -L ’.

```
# ipchains -L input
Chain input (refcnt = 1): (policy ACCEPT)
target prot opt source destination ports
ACCEPT icmp ----- anywhere anywhere any
# ipchains -L test
Chain test (refcnt = 0):
target prot opt source destination ports
DENY icmp ----- localnet/24 anywhere any
```

El ‘refcnt ’ listado en `test`, es el número de reglas que tiene `test` como sus objetivos. Este debe ser cero (y la cadena estar vacía) antes de que la cadena pueda borrarse.

Si el nombre de la cadena se omite, todas las cadenas se listan, incluso las vacías.

Hay tres opciones que pueden acompañar a ‘-L ’. La opción ‘ -n ’ (numérico) es muy útil para evitar que ipchains intente mirar las direcciones IP, lo cual (si usa DNS como la mayoría de personas) causará grandes retrasos si su DNS no está configurado apropiadamente, o ha filtrado las peticiones al DNS. Esto también hace que los puertos sean mostrados como números en lugar de nombres.

La opción ‘ -v ’ muestra todos los detalles de las reglas, tal como los contadores de bytes y de paquetes, las mascaras de TOS, la interface, y la marca del paquete. Caso contrario, estos valores se omiten. Por ejemplo:

```
# ipchains -v -L input
Chain input (refcnt = 1): (policy ACCEPT)
pkts bytes target prot opt tosa tosx ifname mark source des-
tination ports
 10 840 ACCEPT icmp ----- 0xFF 0x00 lo anywhere anywhere any
```

Note que el contador de bytes y de paquetes se muestran usando los sufijos ‘K ’, ‘M ’ o ‘G ’ para 1000, 1,000,000 y 1,000,000,000 respectivamente. Usando la marca ‘ -x ’ (expandir números) se imprime los números completos, no importa lo grandes que ellos sean.

4.5.5.10. Restablecimiento (poniendo a cero) de los contadores.

Es útil poder restablecer los contadores. Esto puede hacerse con la opción ‘ -Z ’ (zero counters). Por ejemplo:

```
    # ipchains -v -L input
    Chain input (refcnt = 1): (policy ACCEPT)
    pkts bytes target prot opt tosa tosx ifname mark source des-
tination ports
     10 840 ACCEPT icmp ----- 0xFF 0x00 lo anywhere anywhere any
    # ipchains -Z input
    # ipchains -v -L input
    Chain input (refcnt = 1): (policy ACCEPT)
    pkts bytes target prot opt tosa tosx ifname mark source des-
tination ports
     0 0 ACCEPT icmp ----- 0xFF 0x00 lo anywhere anywhere any
```

El problema de esto es que a necesite conocer el valor del contador inmediatamente antes de que se restablezcan. En el ejemplo anterior, algunos paquetes podrían pasar a través de las ordenes ‘ -L ’ y ‘ -Z ’ . Por esta razón, puede usar ‘ -L ’ y ‘ -Z ’ *juntos*, para restablecer los contadores mientras los lee. Desgraciadamente, si hace esto, no puede operar en una sola cadena: tiene que listar y poner a cero todas las cadenas a la vez.

```
    # ipchains -L -v -Z
    Chain input (policy ACCEPT):
    pkts bytes target prot opt tosa tosx ifname mark source des-
tination ports
     10 840 ACCEPT icmp ----- 0xFF 0x00 lo anywhere anywhere any
    Chain forward (refcnt = 1): (policy ACCEPT)
```

```
    Chain output (refcnt = 1): (policy ACCEPT)
    Chain test (refcnt = 0):
     0 0 DENY icmp ----- 0xFF 0x00 ppp0 localnet/24 anywhere any
    # ipchains -L -v
    Chain input (policy ACCEPT):
    pkts bytes target prot opt tosa tosx ifname mark source des-
tination ports
     10 840 ACCEPT icmp ----- 0xFF 0x00 lo anywhere anywhere any
    Chain forward (refcnt = 1): (policy ACCEPT)
    Chain output (refcnt = 1): (policy ACCEPT)
    Chain test (refcnt = 0):
     0 0 DENY icmp -----
0xFF 0x00 ppp0 localnet/24 anywherep; any
```

4.5.5.11. Estableciendo la Política.

Anteriormente no se dijo lo que pasa cuando un paquete alcanza el final de una cadena construida, en Section 4.5.5.1. En este caso, la *política* de la cadena determina el destino del paquete. Sólo las cadenas construidas (`input` `output` `forward`) tienen políticas, porque si un paquete llega al final de una cadena definida por el usuario, pasaría a la cadena anterior.

La política puede ser cualquiera de los primeros cuatro objetivos especiales: `ACCEPT`, `DENY` `REJECT` `MASQ`. `MASQ` es sólo válido para la cadena ‘forward ’.

También es importante anotar que el objetivo `RETURN` en una regla de cadenas construidas es útil para especificar el objetivo de la política de la regla cuando un paquete encaja con una regla.

4.5.6. Operaciones en Enmascaramiento.

Hay varios parámetros que usted puede intentar para Ip Masquerading. Se han añadido a `ipchains` porque no vale la pena hacerlo en una herramienta por separado.

El comando IP Masquerading es ‘ -M ’, y puede combinarse con ‘ -L ’ para listar las conexiones actualmente enmascaradas, o ‘ -S ’ para configurar los parámetros de enmascaramiento.

El orden ‘ -L ’ puede ir con ‘ -n ’ (muestre números en lugar de los hostnames y nombres del puerto) o ‘ -v ’ (muestre la secuencia de números para conexión enmascarada, por si le importa).

La orden ‘ -S ’ debe ser seguido por tres valores de timeout, cada uno en segundos: para sesiones TCP, para sesiones TCP después del paquete FIN, y para los paquetes UDP. Si no desea cambiar los valores, simplemente de un valor de ’0’.

Los valores predefinidos se listan en ‘ /usr/include/net/ip_masq.h ’, actualmente 15 minutos, 2 minutos y 5 minutos respectivamente.

El valor que se cambia más comúnmente es el primero, para FTP (vea Section 5.2.3 más abajo).

Note los problemas al poner valores de timeout en Section 6.10.

4.5.7. Verificar un paquete.

A veces quiere ver lo que pasa cuando un cierto paquete entra en su máquina, como para hacer debug de las cadenas del cortafuegos. `Ipchains` tiene la orden ‘ -C ’ para hacerlo, usando exactamente las mismas rutinas que el núcleo usa para diagnosticar paquetes reales..

Usted especifica qué cadena probará el paquete siguiendo el argumento ‘ -C ’ con su nombre. Considerando que el núcleo siempre empieza atravesando las cadenas `input`, `output` y `forward`, le permite atravesar cualquier cadena con propósitos de comprobación.

Los detalles del ‘paquete ’ se especifican usando la misma sintaxis que se usa para especificar reglas del cortafuego. En particular, un protocolo (‘ -p ’), dirección del orígen (‘ -s ’), dirección del destino (‘ -d ’) e interfaz (‘ -i ’). Si el protocolo es TCP o UDP, entonces debe especificarse un solo orígen y un solo puerto del destino, y deben especificarse un tipo y un código de ICMP para el protocolo de ICMP (a menos que la

marca ‘ -f ’ se especifique para indicar una regla del fragmento, caso en que estas opciones son ilegales).

Si el protocolo es TCP (y la marca ‘ -f ’ no se especifica), la marca ‘ -y ’ puede especificarse, para indicar que el paquete de la prueba debe tener activo el bit de SYN.

Aquí va un ejemplo de testeo de un paquete TCP SYN desde 192.168.1.1 puerto 60000 al puerto www de 192.168.1.2, entrando la cadena ’input’ (Esta es una clásica iniciación de conexión WWW):

```
    # ipchains -C input -p tcp -y -s 192.168.1.1 60000 -
d 192.168.1.2 www
    packet accepted
```

4.5.8. Multiples reglas de una vez y ver lo que pasa.

A veces una simple línea de comando puede ocasionar que múltiples reglas sean afectadas. Esto se hace de dos formas: Primero, si especifica un hostname que se resuelve (usando DNS) a múltiples direcciones IP, `ipchains` actuará como si hubiese tecleado múltiples comandos con cada combinación de direcciones.

Así pues, si el hostname ‘www.foo.com ’ se resuelve a tres direcciones IP, y el hostname ’www.bar.com’ se resuelve a dos direcciones IP, entonces el comando ‘ipchains -A input -j reject -s www.bar.com -d www.foo.com’ añadirá seis reglas a la cadena `input`.

La otra forma de que `ipchains` realice acciones múltiples es usar la marca bidireccional (‘ -b ’). Esta marca hace que `ipchains` se comporte como si usted hubiera tecleado dos veces, en la segunda los argumentos ‘ -s ’ y ‘ -d ’ se invierten. Así, para evitar redirecciones (forwarding) hasta o desde 192.168.1.1, podría hacer a lo siguiente:

```
    # ipchains -b -A forward -j reject -s 192.168.1.1
    #
```

Personalmente, no me gusta mucho la opción ‘ -b ’; si desea conveniencia, vea Section 4.6.1 más abajo.

La opción -b puede usarse con las ordenes de inserción (‘ -I ’), borrado (‘ -D ’) (excepto en las variantes que toman un número de la regla), adición (‘ -A ’) y Chequeo (‘ -C ’).

Otra marca útil es ‘ -v ’ (verbose) qué muestra exactamente lo que `ipchains` está haciendo con los comandos. Es útil cuando esta trabajando con órdenes que pueden afectar a múltiples reglas. Por ejemplo:.

```
    # ipchains -v -b -C input -p tcp -f -s 192.168.1.1 -
d 192.168.1.2 -i lo
     tcp opt ---f- tos 0xFF 0x00 via lo 192.168.1.1 -
\062 192.168.1.2 * -\062 *
    packet accepted
     tcp opt ---f- tos 0xFF 0x00 via lo 192.168.1.2 -
\062 192.168.1.1 * -\062 *
    packet accepted
```

4.6. Ejemplos útiles

Tengo una conexión vía telefónica (`-i ppp0`). Tomo noticias (`-p TCP -s news.virtual.net.au nntp`) y correo (`-p TCP -s mail.virtual.net.au pop-3`) cada vez que me conecto. Uso el método ftp de Debian para actualizar mi máquina regularmente (`-p TCP -y -s ftp.debian.org.au ftp-data`). Navego en la web a través del proxy de mi ISP mientras tanto (`-p TCP -d proxy.virtual.net.au 8080`) , pero odio los anuncios de doubleclick.net en el archivo Dilbert (`-p TCP -y -d 199.95.207.0/24 & -p TCP -y -d 199.95.208.0/24`).

No me molesta que las personas intenten hacer ftp a mi máquina mientras estoy online (`-p TCP -d $LOCALIP ftp`) pero no quiero que nadie pretenda obtener una dirección de mi red interna (`-s 192.168.1.0/24`). Esto se llama comunmente IP

Spoofing, y hay una mejor manera de protegerse a si mismo, desde el núcleo 2.1.x y superiores. Ver Section 5.7 .

Esta configuración es bastante simple, ya que no hay ninguna otra máquina en mi red interna.

No quiero que ningún proceso local (ie. Netscape, lince etc.) se conecte con doubleclick.net:

```
    # ipchains -A output -d 199.95.207.0/24 -j REJECT
    # ipchains -A output -d 199.95.208.0/24 -j REJECT
```

Ahora quiero poner prioridades en varios paquetes salientes (no se gana mucho haciendolo en los paquetes entrantes). Puesto que tengo un número justo de estas reglas, tiene sentido ponerlos todos en una sola cadena, llamada. `ppp-out`

```
    # ipchains -N ppp-out
    # ipchains -A output -i ppp0 -j ppp-out
```

Retraso mínimo para tráfico de web & telnet.

```
    # ipchains -A ppp-out -p TCP -d proxy.virtual.net.au 8080 -
t 0x00 0x10
    # ipchains -A ppp-out -p TCP -d 0.0.0.0 telnet -t 0x00 0x10
```

Prioridad baja para los datos del ftp, nntp, pop-3,:

```
    # ipchains -A ppp-out -p TCP -d 0.0.0.0/0 ftp-data -
t 0x00 0x02
    # ipchains -A ppp-out -p TCP -d 0.0.0.0/0 nntp -t 0x00 0x02
    # ipchains -A ppp-out -p TCP -d 0.0.0.0/0 pop-3 -t 0x00 0x02
```

Hay unas restricciones en paquetes que entran por la interfaz ppp0: creemos una cadena llamada ‘ppp-in ’:

```
    # ipchains -N ppp-in
    # ipchains -A input -i ppp0 -j ppp-in
```

Ahora, ningún paquete que entra por `ppp0` debe estar exigiendo una dirección de orígen 192.168.1. *, así que lo anotamos y los denegamos:

```
    # ipchains -A ppp-in -s 192.168.1.0/24 -l -j DENY
    #
```

Permito conexiones DNS (remito todas las demandas a 203.29.16.1, así que espero respuesta DNS TCP sólo desde ellos), ftp, y retorno ftp-data solamente (las cuales solo deben ir a un puerto superior a 1023, y no a los puertos X11 alrededor de 6000)

```
    # ipchains -A ppp-in -p TCP -s 203.29.16.1 -d $LOCALIP dns -
j ACCEPT
    # ipchains -A ppp-in -p TCP -s 0.0.0.0/0 ftp-data -
d $LOCALIP 1024:5999 -j ACCEPT
    # ipchains -A ppp-in -p TCP -s 0.0.0.0/0 ftp-data -
d $LOCALIP 6010: -j ACCEPT
    # ipchains -A ppp-in -p TCP -d $LOCALIP ftp -j ACCEPT
```

Permito paquetes de respuesta TCP en

```
# ipchains -A ppp-in -p TCP ! -y -j ACCEPT
#
```

Finalmente, los paquetes local-a-local están OK:

```
    # ipchains -A input -i lo -j ACCEPT
```

Ahora, mi política predefinida en la cadena input es `DENY`, de tal forma que todo lo demás sea desechado.

```
    # ipchains -P input DENY
```

NOTA: No configuraría mis cadenas en este orden, mientras que los paquetes cruzan en el momento que estoy configurando. Lo más seguro es poner la política de DENY

primero, luego insertar las reglas. Por supuesto, que si sus reglas exigen ver al DNS, podría tener problemas.

4.6.1. Usando.ipchains-save

Configurar las cadenas de un cortafuegos como usted lo desea, y luego intentar recordar las ordenes que usted utilizó para hacerlo la próxima vez es doloroso.

Así pues, `ipchains-save` es un script que lee su actual configuración de cadenas y la graba en un archivo. Por el momento, le dejo la inquietud de lo que podría hacer `ipchains-restore`

`ipchains-save` puede grabar una sola cadena, o todas las cadenas (si ningún nombre de la cadena se especifica). La única opción actualmente permitida es ‘ -v ’ qué imprime las reglas (a stderr) cuando se graban. La política de la cadena también se graba para las cadenas `input`, `output` y `forward`.

```
$ ipchains-save \062 my_firewall
Saving ‘input’.
Saving ‘output’.
Saving ‘forward’.
Saving ‘ppp-in’.
Saving ‘ppp-out’.
$
```

4.6.2. Usando.ipchains-restore

`Ipchains-restore` restaura cadenas grabadas con `ipchains-save` Puede tomar dos opciones: ‘ -v ’ qué muestra cada regla mientras se agrega, y ‘ -f ’ qué forza el vaciado de las cadenas definidas por el usuario, si existen.

Si una cadena definida por el usuario se encuentra en la entrada, ipchains-restore verifica si esa cadena ya existe. Si es así, entonces le preguntará si las cadenas deben vaciarse (vaciado de todas las cadenas) o si la restauración de esta cadena debe saltarse.

Si usted especifica ’-f’ en la linea de comandos, usted no será consultado; la cadena se vaciará.

Por ejemplo:

```
# ipchains-restore < my_firewall
Restoring ‘input’.
Restoring ‘output’.
Restoring ‘forward’.
Restoring ‘ppp-in’.
Chain ‘ppp-in’ already exists. Skip or flush? [S/f]? s
Skipping ‘ppp-in’.
Restoring ‘ppp-out’.
Chain ‘ppp-out’ already exists. Skip or flush? [S/f]? f
Flushing ‘ppp-out’.
```

Chapter 5. Miscelánea.

Esta sección contiene toda la información y FAQs que no pude meter anteriormente dentro de la estructura.

5.1. Cómo organizar sus reglas del cortafuegos.

Esta pregunta requiere alguna reflexión. Puede intentar organizarlos para optimizar la velocidad (minimice el número de reglas verificadas para los paquetes más comúnes) o para aumentar flexibilidad en la administración.

Si tiene una conexión intermitente, tal como la conexión PPP, debería hacer que la primera regla de la cadena input sea '-i ppp0 -j DENY' en el momento del arranque, y luego tener algo como esto en tu script `ip-up`:

```
    # Re-crear la cadena 'ppp-in'
    ipchains-restore -f < ppp-in.firewall
    # Remplazar la regla DENY con salto a la ca-
dena que maneja ppp
    ipchains -R input 1 -i ppp0 -j ppp-in
```

Su script `ip-down` podría decir algo como:

```
    ipchains -R input 1 -i ppp0 -j DENY
```

5.2. Qué no se filtra.

Algunas cosas de las que debe ser consciente antes de que empiece filtrando lo que no quiere.

5.2.1. Paquetes de ICMP.

Se usan paquetes de ICMP (entre otras cosas) para indicar fracaso para otros protocolos (como TCP y UDP). Paquetes ‘destino-inalcanzable ’ en particular. Bloquear estos paquetes significa que nunca obtendrá errores de ‘Host unreachable’ o ‘No route to host’. Cualquier conexión esperará por una respuesta que nunca vendrá. Esto es irritante, pero raramente fatal.

Un problema peor es el rol de los paquetes de ICMP en descubrimiento de MTU. Todas las buenas implementaciones de TCP (incluido Linux) usan el MTU, para intentar deducir que tan largos pueden ir los paquetes a un destino sin fragmentarse (la fragmentación ralentiza el rendimiento, sobre todo cuando ocasionalmente los fragmentos son perdidos). El MTU trabaja enviando paquetes con el bit "Don’t fragment" seteado, y luego enviando paquetes más pequeños si obtiene un paquete ICMP indicando "Fragmentation needed but DF set" (necesidad de fragmentación). Este es un tipo de paquete "destination-unreachable", y si nunca se recibe, el host local nunca reducirá la MTU,

Todas las aplicaciones de TCP buenas (Linux incluyó) use descubrimiento de MTU para intentar deducir eso que el paquete más grande que puede conseguir a un destino sin fragmentarse (la fragmentación retarda actuación, sobre todo cuando los fragmentos ocasionales están perdidos). El descubrimiento de MTU trabaja enviando paquetes con el "no Fragmenta" el pedazo puso, y entonces enviando paquetes más pequeños si consigue un paquete de ICMP que indica "la Fragmentación necesitó pero DF puso" (‘fragmentation--ed ’). Éste es un tipo de ‘destination-inalcanzable ’ el paquete, y si nunca se recibe, el organizador local no reducirá MTU, y el rendimiento será abismal o no existente.

Note that it is common to block all ICMP redirect messages (type 5); these can be used to manipulate routing (although good IP stacks have safeguards), and so are often seen as slightly risky.

5.2.2. Conexiones TCP a DNS.(servidor de nombres)

Si está tratando de bloquear las conexiones TCP salientes, recuerde que DNS no

siempre usa UDP; si la respuesta del servidor excede los 512 bytes, el cliente usrá una conexión TCP (continúa siendo al puerto 53) para conseguir los datos.

Esto podría ser una trampa porque DNS "casi mayoritariamente" funcionará si no permite tales conexiones TCP; Podría usted experimentar extraños y largos retrasos y otros problemas DNS si lo hiciera.

Si sus consultas DNS se dirigen siempre a la misma fuente externa (ya sea directamente o usando la linea `nameserver` en `/etc/resolv.conf` o usando un servidor de nombres en caché en modo forward), entonces sólo necesitas permitir conexiones TCP al puerto `domain` en ese servidor de nombres desde el puerto `domain` local (si usas servidor de nombres de caché) o desde un puerto alto (> 1023) si usas `/etc/resolv.conf`.

5.2.3. Pesadillas de FTP.

El otro problema clásico es FTP. FTP tiene dos `modos`; el tradicional se llama modo `activo` y el más reciente se llama modo `pasivo`. Los browsers de web normalmente tienen como valor predefinido el modo pasivo, pero los programas de ftp de linea de comandos normalmente tienen como valor predefinido el modo activo.

En modo activo, cuando el extremo remoto quiere enviar un archivo (o incluso los resultados de un comando `ls` o `dir`) intenta abrir una conexión de TCP a la máquina local. Esto significa que no puede filtrar estas conexiones de TCP sin romper un FTP activo.

Si usted tiene la opción de usar modo pasivo, entonces bien; el modo pasivo hace conexiones de los datos del cliente al servidor, incluso para los datos entrantes. Por otra parte, se recomienda que usted sólo permita conexiones TCP de puertos por encima de 1024 y no entre 6000 y 6010 (se usa 6000 para X-Windows).

5.3. Filtrando el Ping de la Muerte.

Las máquinas Linux son ahora inmunes al famoso `Ping de la Muerte` que se produce enviando un paquete de ICMP ilegalmente grande que causa desbordamiento en los buffers en la pila TCP en el receptor y causa estragos.

Si está protegiendo máquinas que podrían ser vulnerables, podría simplemente bloquear fragmentos de ICMP. Los paquetes de ICMP normales no son tan grandes como para requerir fragmentación, así que usted no interrupirá nada excepto los pings grandes. Yo he oído (sin confirmar) informes de que algunos sistemas solo requieren del último fragmento de un gran paquete ICMP para ser adulterados, así que el bloqueo del primer fragmento sólo no es recomendable.

Aun cuando los programas de exploit que he visto solo usan ICMP, no hay razón para no que los fragmentos TCP o UDP (o de un protocolo desconocido) no puedan usarse en este ataque, asi que bloquear fragmentos de ICMP es sólo una solución temporal.

5.4. Filtrando Teardrop y Bonk.

Teardrop y Bonk son dos ataques (principalmente contra las máquinas de Microsoft Windows NT) qué se basa en el solapamiento de fragmentos. Hacer que su router Linux haga defragmentacion, o desaprobar todos los fragmentos hacia sus máquinas vulnerables son otras opciones.

5.5. Filtrando Bombas de Fragmento.

Se dice que algunas pilas de TCP poco fiables tienen problemas al tratar con números grandes de fragmentos de paquetes cuando no reciben todos los fragmentos. Linux no tiene este problema. Usted puede filtrar los fragmentos (lo cual podría interrumpir usos legítimos) o compilar su núcleo con ‘IP: always defragment’ puesta a ‘Y ’ (sólo si su máquina Linux es la única ruta posible para estos paquetes).

5.6. Cambiando las reglas del cortafuegos.

Hay algunos problemas involucrados en la alteración de las reglas del cortafuegos. Si usted no tiene cuidado, puede permitir que los paquetes crucen, mientras se estan realizando los cambios. Un simple acercamiento es hacer lo siguiente:

```
# ipchains -I input 1 -j DENY
# ipchains -I output 1 -j DENY
# ipchains -I forward 1 -j DENY
... make changes ...
# ipchains -D input 1
# ipchains -D output 1
# ipchains -D forward 1
#
```

Esto desecha todos los paquetes mientras duran los cambios.

Si restringe los cambios a una sola cadena, podría querer crear una nueva cadena con las nuevas reglas, y entonces reemplazar (‘ -R ’) la regla que apuntó a la vieja cadena con una que apunte a la nueva. Este reemplazo ocurrirá atómicamente.

5.7. ¿ Cómo activo la protección de IP spoof?

El spoofing de IP es una técnica donde un host manda paquetes que dicen ser de otro host. Puesto que el filtrado de paquetes toma decisiones basado en la dirección orígen, el Ip Spoofing es usado para engañar a los filtros de paquete. También se usa para ocultar la identidad de los atacantes que usan ataques de SYN, Teardrop, Ping de la Muerte y similares (no se preocupe si no sabe lo que son).

La mejor manera de protegerse del spoofing de IP se llama "Verificación de la dirección del orígen" y es hecha por el código de enrutamiento, y no en el de firewalling. Busque un archivo llamado `/proc/sys/net/ipv4/conf/all/rp_filter` Si existe, entonces activar la Verificación de la dirección de orígen en el arranque es la mejor solución para usted. Para hacer esto, inserte las siguientes lineas en alguna parte de sus

scripts de inicio, antes de que cualquier interfaz de red se inicialice (ej. Los usuarios de Debian las pondrian en /etc/init.d/netbase si es que ellas ya no esta allí).

```
# Esta es la mejor forma: activar la Verificación de Direc-
ción Orígen
# Protección de spoof en todas las interfaces
if [ -e /proc/sys/net/ipv4/conf/all/rp_filter ]; then
   echo -n "Configurando protección de IP spoofing"
   for f in /proc/sys/net/ipv4/conf/*/rp_filter; do
      echo 1 > $f
   done
   echo "done."
   else
   echo PROBLEMAS PARA CONFIGURAR LA PROTECCION DE IP SPOOF-
ING.  PREOCUPESE.
   echo "Presione CONTROL-D para continuar..."
   echo
   # Iniciar un simple shell de usuario en la consola
  /sbin/sulogin $CONSOLE
fi
```

Si no puede hacer esto, puede insertar reglas manualmente para proteger cada interface. Esto requiere conocimiento de cada interface. Los núcleos 2.1 desechan automáticamente los paquetes que dicen venir de las direcciones 127.* (reservadas para el loopback local `lo`).

Por ejemplo, digamos que tenemos tres interfaces, `eth0`, `eth1` y `ppp0`. Podemos usar `ifconfig` para indicar la dirección y la máscara de subred de las interfaces. Digamos que `eth0` pertenece a la red 192.168.1.0 con máscara de subred 255.255.255.0, eth1 pertenece a la red 10.0.0.0 con máscara 255.0.0.0 y que `ppp0` se conectóa internet (donde cualquier dirección excepto las direcciones IP privadas reservadas se permiten), insertaríamos las siguientes reglas:

```
# ipchains -A input -i eth0 -s ! 192.168.1.0/255.255.255.0 -
j DENY
```

```
# ipchains -A input -i ! eth0 -s 192.168.1.0/255.255.255.0 -
j DENY
# ipchains -A input -i eth1 -s ! 10.0.0.0/255.0.0.0 -j DENY
# ipchains -A input -i ! eth1 -s 10.0.0.0/255.0.0.0 -j DENY
#
```

Esto no es tan bueno como la Verificación de la Dirección Orígen, porque si su red cambia, entonces tiene que cambiar sus reglas de firewalling.

Si usted está ejecutando un núcleo de la serie 2.0 , podría querer proteger el loopback también, usando una regla como:

```
# ipchains -A input -i ! lo -s 127.0.0.0/255.0.0.0 -j DENY
#
```

5.8. Proyectos avanzados.

Hay una librería que he escrito y que está incluida con la distribución de fuentes llamada ‘libfw ’. Usa la habilidad de IP chains 1.3 y superiores para copiar un paquete al userspace (usando la opción de configuración IP_FIREWALL_NETLINK).

El valor marca se puede usar para especificar los parametros de Calidad de Servicio para los paquetes, o para especificar como se deberían redirigir los paquetes (port-forwarded). Nunca los he usado, pero si quieres escribir sobre ellos, por favor contacta conmigo.

Cosas como "*stateful inspection*" (Yo (el autor)prefiero el termino cortafuegos dinamico) puede implementarse usando esta libreria. *Other nifty ideas include controlling packets on a per-user basis by doing a lookup in a userspace daemon*. Esto deberia ser bastante sencillo.

5.8.1. SPF: Stateful Packet Filtering

ftp://ftp.interlinx.bc.ca/pub/spf es el sitio del proyecto SPF de Brian Murrell , que hace rastreo de conexiones en el *userspace*. Añade una seguridad significativa para sitios de bajo ancho de banda.

Actualmente hay poca documentación, pero hay en la lista de correo hay algunos respuestas de Brian answered a algunas preguntas:

```
> I believe it does exactly what I want: Installing a temporary
> "backward"-rule to let packets in as a response to an
> outgoing request.
Yup, that is exactly what it does.  The more protocols it
understands, the more "backward" rules it gets right.  Right
now it has support for (from memory, please excuse any errors
or omissions) FTP (both active and passive, in and out), some
RealAudio, traceroute, ICMP and basic ICQ (inbound from the ICQ
servers, and direct TCP connections, but alas the secondary
direct TCP connections for things like file transfer, etc. are
not there yet)
> Is it a replacement for ipchains or a supplement?
It is a supplement.  Think of ipchains as the engine to allow
and prevent packets from travelling across a Linux box.  SPF is
the driver, constantly monitoring traffic and telling ipchains
how to change it’s policies to reflect the changes in traffic
patterns.
```

5.8.2. ftp-data hack de Michael Hasenstein’s.

Michael Hasenstein de SuSE ha escrito un parche para el núcleo que añade rastreo de conexiones ftp. Actualmente se puede encontrar en
http://www.suse.de/~mha/patch.ftp-data-2.gz

5.9. Perfeccionamientos futuros.

Se han rediseñado el Cortafuegos y NAT para el 2.4. Los planes y discusiones estan disponibles en la lista de netfilter (ver http://lists.samba.org). Estas mejoras deberían aclarar muchas cuestiones de utilidad importantes (firewalling y masquerading shouldn’t be *this hard*), and allow growth for far more flexible firewalling.

Chapter 6. Problemas comúnes.

6.1. ipchains -L se atasca!

Probablemente esta bloqueando las consultas al DNS; Eso eventualmente lo retardará. Pruebe el flag ’-n’ (numérico) para ipchains, la cual evita la consulta de nombres.

6.2. La inversión no funciona.

Debes poner la opción ‘!’, con espacios. Un error clasico (advertido en 1.3.10) es:

```
# ipchains -A input -i !eth0 -j DENY
#
```

Nunca existirá una interfaz ‘!eth0’, pero ipchains no lo sabe.

6.3. Masquerading/forwarding no funciona!

Asegúrese de que el envio de paquetes este habilitado (en núcleos recientes esto está deshabilitado por defecto, lo cual significa que los paquetes nunca atraviesan la cadena ’forward’). Puede corregir esto tecleando (como root)

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Si esto funciona, puede ponerlo en algun script de arranque para que se habilite en todos los casos; aunque querras configurar tu cortafuegos antes de ejecutar esta orden, sino habrá una posibilidad de que se te escapen algunos paquetes.

6.4. ¡-j REDIR no funciona!

Debes habilitar el reenvío(forwarding) de paquetes (ver arriba) para que funcione el redireccionamiento; sino el codigo de rutado se deshará del paquete. Así que si sólo estas usando redireccionamiento (redirect), y no usas el reenvío(forwarding) para nada, estate atento a ésto.

Notar que REDIR (estando en la cadena input) no afecta a conexiones desde procesos locales.

6.5. No funcionan los comodines de Interfaces !

Había un bug en versiones 2.1.102 y 2.1.103 del núcleo (y algunos parches viejos que construí) que hace que el comando de ipchains donde se especifica una interfaz wildcard (comodín) (como `-i ppp+`) falle.

Esto se ha corregido en recientes núcleos, y en el parche 2.0.34 en el sito web. Puede arreglarlo manualmente en la el código fuente del núcleo, cambiando la línea 63 en include/linux/ip_fw.h:

```
    #define IP_FW_F_MASK    0x002F  /* All possi-
ble flag bits mask   */
```

Aqui debe decir “ 0x003F ”. Arregle el núcleo y recompile.

6.6. TOS no funciona!

Esto fue un error mío: configurar el campo "Type of Service" (tipo de servicio) no configura realmente el "Type of Service" (Tipo de servicio) en versiones del núcleo desde la 2.1.102 hasta la 2.1.111. Esto se solucionó en el 2.1.112.

6.7. ipautofw e ipportfw y no funcionan!

Para 2.0.x, esto es correcto; no tengo tiempo para crear y mantener un parche enorme para el ipchains e ipautofw/ipportfw.

Para los 2.1.x, bajate el ipmasqadm de Juan Ciarlante's <url url="http://juanjox.linuxhq.com/" name="http://juanjox.linuxhq.com/"> y usalo exactamente como usarías `ipautofw` o `ipportfw`, salvo que para `ipportfw` teclearas `ipmasqadm portfw`, y para `ipautofw` teclearás `ipmasqadm autofw`.

6.8. xosview está roto!

Actualicelo a la versión 1.6.0 o superiores, las cuales no requieren reglas de cortafuegos para los núcleos 2.1.x. MIRAR ESTE PARRAFO EN EL ORIGINAL This seems to have broken again in the 1.6.1 release; please bug the author (it's not my fault!).

6.9. Segmentation fault con -j REDIRECT

Fue un bug en ipchains versión 1.3.3. Por favor actualicelo.

6.10. No puedo poner timeouts en enmascaramiento!

Cierto (para núcleos 2.1.x) hasta 2.1.123. En 2.1.124, el tratar de configurar timeouts de masquerading provoca un bloqueo del núcleo (cambia `return` a `ret =` en la linea 1328 de net/ipv4/ip_fw.c). En 2.1.125, funciona bien.

6.11. Quiero un cortafuegos IPX !

Tambien otros, según parece. Mi código sólo cubre IP, desafortunadamente. On the good side, all the hooks are there to firewall IPX! You just need to write the code; I will happily help where possible.

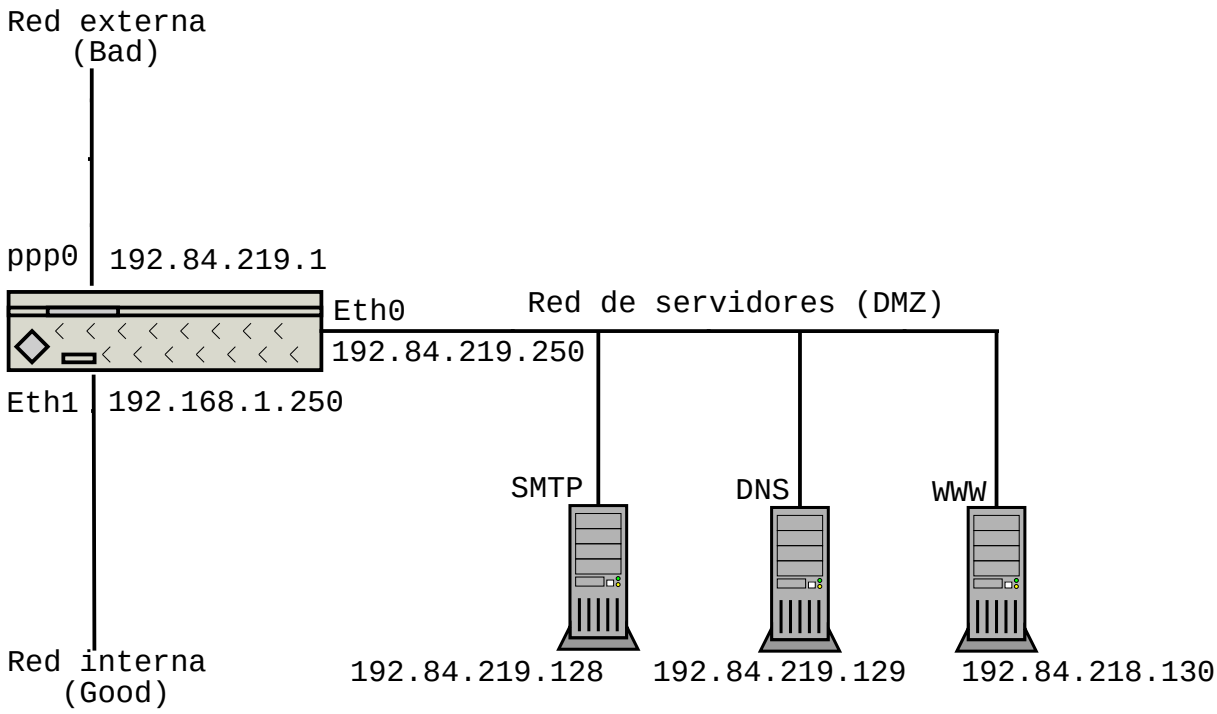
Chapter 7. Un ejemplo serio.

Este ejemplo se extrajo del Linuxworld tutorial de 1999 de Michael Neuling y mio; no es la única manera de resolver el problema dado, pero es probablemente la más simple. Espero que lo encuentre informativo.

7.1. La situación.

- Red interna enmascarada (con varios sistemas operativos) a la que llamamos GOOD.
- Servidores expuestos en una red separada (llamad "DMZ" de Demilitarized Zone, en español, zona desmilitarizada).
- Conexión PPP a Internet (llamada "BAD").

Figure 7-1. diagrama3



7.2. Objetivos

Maquina del filtrado de paquetes:

PING a cualquier red

Es realmente útil para saber si una máquina ha caído.

TRACEROUTE a cualquier red

Una vez más útil para diagnósticos.

Acceso a DNS

Para hacer ping y DNS más útiles.

Dentro de la DMZ:

Servidor de correo

- SMTP a la externa
- Aceptar SMTP de la interna y externa
- Aceptar POP-3 desde la interna

Servidor de nombres

- Envía DNS a la externa
- Acepta DNS de la interna, externa y de la máquina de filtrado

Servidor web

- Acepta HTTP de la interna y externa
- Acceso Rsync desde la interna

Interna:

Permite WWW, ftp, traceroute, ssh a la externa

Estas son más o menos lo standard a la hora de permitir: algunos sitios empiezan permitiendo casi todo desde la interna, pero estamos siendo restrictivos.

Permitir SMTP al servidor de correo

Obviamente, querremos que pueda se pueda enviar correo fuera.

Permitir POP-3 al servidor de correo

Así es cómo la gente leerá su mail.

Permitir DNS al servidor de nombres

Se necesita poder buscar nombres externos WWW, ftp, traceroute y ssh.

Permitir rsync al servidor Web

Así es cómo se sincronizará el servidor web externo con el interno.

Permitir WWW al sservidor Web

Obviamnete, deben ser capaces de conectarse al servidor web externo.

Permitir ping a la maquina de filtrado de paquetes

Es cortes permitirlo: esto significa que se puede comprobar si el firewall se ha caido (así no nos hechan la culpa si un servidor web se ha caido).

7.3. Antes de filtrar paquetes.

- Anti-spoofing

  Dado que no tenemos ningun enrutado asimétrico, podemos simplemente activar el anti-spoofing en todas las interfaces.

  ```
  # for f in /proc/sys/net/ipv4/conf/*/rp_filter; do echo 1 > $f; done
  #
  ```

- Configurar las reglas de filtrado para DENEGAR todo:

  Todavía permitimos el tráfico loopback local, pero denegamos el resto.

  ```
  # ipchains -A input -i ! lo -j DENY
  ```

```
# ipchains -A output -i ! lo -j DENY
# ipchains -A forward -j DENY
#
```

- Configurando las interfaces.

  Esto se hace generalmente en los scripts de inicio. Saegúrate de ejecutar los pasos anteriores antes de configurar las interfaces, para evitar la pérdida de paquetes antes de que se hayan configurado las reglas.

- Insertar los módulos per-protocol masquerading.

  Necesitamos insertar el módulo de masquerading para el FTP, de modo que funcione desde la red interna el FTP pasivo y activo.

  ```
  # insmod ip_masq_ftp
  #
  ```

7.4. Packet Filtering for Through Packets

Con masquerading, es mejor filtrarlo en la cadena forward.

Divide la cadena forward en varios cadenas de ususario chains dependiendo de las interfaces origen/destino; esto descompone el problema en trocitos más manejables.

```
ipchains -N good-dmz
ipchains -N bad-dmz
ipchains -N good-bad
ipchains -N dmz-good
ipchains -N dmz-bad
ipchains -N bad-good
```

Es común ACCEPTar errores ICMP standard, por tanto creamos una cadena para ello.

```
ipchains -N icmp-acc
```

7.4.1. Configurar los saltos (JUMPS) en la cadena forward.

Desafortunadamente, sólo sabemos (en la cadena forward) la interfaz de salida. Por tanto, para imaginarnos desde que interfaz está entrando, usamos la dirección de origen (el anti-spoofing evita el falseo de direcciones).

Notar que mandaremos a los logs cualquier cosa que no encaje con ésto (obviamente, ésto no debería ocurrir).

```
ipchains -A forward -s 192.168.1.0/24 -i eth0 -j good-dmz
ipchains -A forward -s 192.168.1.0/24 -i ppp0 -j good-bad
ipchains -A forward -s 192.84.219.0/24 -i ppp0 -j dmz-bad
ipchains -A forward -s 192.84.219.0/24 -i eth1 -j dmz-good
ipchains -A forward -i eth0 -j bad-dmz
ipchains -A forward -i eth1 -j bad-good
ipchains -A forward -j DENY -l
```

7.4.2. Definir la cadena icmp-acc

Los paquetes que sean mensaje de ICMP serán ACCEPTados, sino, el control pasará hacia abajo en la cadena.

```
ipchains -A icmp-acc -p icmp --icmp-type destination-
unreachable -j ACCEPT
ipchains -A icmp-acc -p icmp --icmp-type source-quench -j ACCEPT
ipchains -A icmp-acc -p icmp --icmp-type time-exceeded -j ACCEPT
ipchains -A icmp-acc -p icmp --icmp-type parameter-problem -
j ACCEPT
```

7.4.3. Good (Interna) a DMZ (Servidores)

Restricciones internas:

- Permitir WWW, ftp, traceroute, ssh to external
- *Permitir SMTP to Mail server*
- *Permitir POP-3 to Mail server*
- *Permitir DNS to Name server*
- *Permitir rsync to Web server*
- *Permitir WWW to Web server*
- Permitir ping to packet filter box

Podríamos hacer masquerading desde la red interna hacia la DMZ, peo aquí no lo hacemos. Dado que ninguno en la red interna debería intentar hacer "el mal", mantendremos un log de los paquetes denegados.

Notar que en versiones viejas de Debian llamaban ‘pop3’ ‘pop-3’ en /etc/services, lo cual incumple el RFC1700.

```
ipchains -A good-dmz -p tcp -d 192.84.219.128 smtp -j ACCEPT
ipchains -A good-dmz -p tcp -d 192.84.219.128 pop3 -j ACCEPT
ipchains -A good-dmz -p udp -d 192.84.219.129 domain -j ACCEPT
ipchains -A good-dmz -p tcp -d 192.84.219.129 domain -j ACCEPT
ipchains -A good-dmz -p tcp -d 192.84.218.130 www -j ACCEPT
ipchains -A good-dmz -p tcp -d 192.84.218.130 rsync -j ACCEPT
ipchains -A good-dmz -p icmp -j icmp-acc
ipchains -A good-dmz -j DENY -l
```

7.4.4. Bad (red externa) a DMZ (servidores).

- Restricciones DMZ:
  - Servidor de correo
    - *SMTP a la externa*
    - *Aceptar SMTP desde* la interna y la *externa*
    - Aceptar POP-3 desde la interna
  - Servidor de nombres
    - *Enviar DNS a la externa*
    - *Aceptar DNS desde la* interna, *externa* y la máquina de filtrado de paquetes
  - Web server
    - *Accept HTTP from* internal and *external*
    - Rsync access from internal
- Things we allow from external network to DMZ.
  - Don’t log violations, as they may happen.

```
ipchains -A bad-dmz -p tcp -d 192.84.219.128 smtp -j ACCEPT
ipchains -A bad-dmz -p udp -d 192.84.219.129 domain -j ACCEPT
ipchains -A bad-dmz -p tcp -d 192.84.219.129 domain -j ACCEPT
ipchains -A bad-dmz -p tcp -d 192.84.218.130 www -j ACCEPT
ipchains -A bad-dmz -p icmp -j icmp-acc
ipchains -A bad-dmz -j DENY
```

7.4.5. Good (internal) to Bad (external).

- Restricciones internas:
  - *Permitir WWW, ftp, traceroute, ssh to external*
  - Permitir SMTP to Mail server
  - Permitir POP-3 to Mail server
  - Permitir DNS to Name server
  - Permitir rsync to Web server
  - Permitir WWW to Web server
  - Permitir ping to packet filter box
- Muchas personas permiten todo desde la red interna a la externa, y luego añade restricciones. Nosotros estamos siendo fascistas.
  - Violaciones de Logs.
  - FTP pasivo gestionado por el módulo masq.
  - Traceroute usará puertos de destino UDP desde el 33434 en adelante.

```
ipchains -A good-bad -p tcp --dport www -j MASQ
ipchains -A good-bad -p tcp --dport ssh -j MASQ
ipchains -A good-bad -p udp --dport 33434:33500 -j MASQ
ipchains -A good-bad -p tcp --dport ftp -j MASQ
ipchains -A good-bad -p icmp --icmp-type ping -j MASQ
ipchains -A good-bad -j REJECT -l
```

7.4.6. DMZ a Good (interna).

- Restriciones internas:
  - Permitir WWW, ftp, traceroute, ssh a la externa
  - *Permitir SMTP al servidor de correo*
  - *Permitir POP-3 al servidor de correo*
  - *Permitir DNS to servidor de nombres*
  - *Permitir rsync al servidor web*
  - *Permitir WWW al servidor web*
  - Permitir ping a la maquina de filtrado de paquetes
- Si estuvieramos haciendo enmascaramiento desde la red interna a la DMZ, simplemente denegaremos cualquier paquete que venga desde el otro lado. Como así es, sólo permitiremos paquetes que pudieran ser parte de un conexión establecida.

```
ipchains -A dmz-good -p tcp ! -y -s 192.84.219.128 smtp -
j ACCEPT
ipchains -A dmz-good -p udp -s 192.84.219.129 domain -j ACCEPT
ipchains -A dmz-good -p tcp ! -y -s 192.84.219.129 domain -
j ACCEPT
ipchains -A dmz-good -p tcp ! -y -s 192.84.218.130 www -
j ACCEPT
ipchains -A dmz-good -p tcp ! -y -s 192.84.218.130 rsync -
j ACCEPT
ipchains -A dmz-good -p icmp -j icmp-acc
ipchains -A dmz-good -j DENY -l
```

7.4.7. DMZ a bad (externa).

- Restricciones en la DMZ:
  - Servidor de correo
    - *SMTP a la externa*
    - *Aceptar SMTP desde la* interna y *externa*
    - Aceptar POP-3 desde la interna
  - Servidor de nombres
    - *Enviar DNS a la externa*
    - *Aceptar DNS desde la* interna, *externa* y máquina de filtrado de paquetes
  - Servidor Web
    - *Aceptar HTTP desde la* interna y *externa*
    - Acceso Rsync desde la interna
-

```
ipchains -A dmz-bad -p tcp -s 192.84.219.128 smtp -j ACCEPT
ipchains -A dmz-bad -p udp -s 192.84.219.129 domain -j ACCEPT
ipchains -A dmz-bad -p tcp -s 192.84.219.129 domain -j ACCEPT
ipchains -A dmz-bad -p tcp ! -y -s 192.84.218.130 www -
j ACCEPT
ipchains -A dmz-bad -p icmp -j icmp-acc
ipchains -A dmz-bad -j DENY -l
```

7.4.8. Bad (externa) a Good (interna).

- No permitimos nada (no-enmascarado) desde la red externa a la red interna

```
ipchains -A bad-good -j REJECT
```

7.4.9. Filtrado de paquetes para la propia máquina de filtrado

- Si queremos usar filtrado de paquetes en los paquetes entrantes en la propia máquina de filtrado de paquetes, necesitamos filtrar en la cadena input. reamos una cadena para cada interfaz de destino:

```
ipchains -N bad-if
ipchains -N dmz-if
ipchains -N good-if
```

- Crear saltos a ellas:

```
ipchains -A input -d 192.84.219.1 -j bad-if
ipchains -A input -d 192.84.219.250 -j dmz-if
ipchains -A input -d 192.168.1.250 -j good-if
```

7.4.9.1. Interfaz Bad (externa).

- Máquina de filatrado de paquetes:
  - *PING a cualquier red*

  - *TRACEROUTE a cualquier red*
  - Acceso a DNS

- La interfaz externa tambien recibe respuestas to masqueraded packets (masquerading uses source ports 61000 to 65095) and ICMP errors for them and PING replies.

```
ipchains -A bad-if -i ! ppp0 -j DENY -l
ipchains -A bad-if -p TCP --dport 61000:65095 -j ACCEPT
ipchains -A bad-if -p UDP --dport 61000:65095 -j ACCEPT
ipchains -A bad-if -p ICMP --icmp-type pong -j ACCEPT
ipchains -A bad-if -j icmp-acc
ipchains -A bad-if -j DENY
```

7.4.9.2. Interfaz DMZ.

- Packet Filter box restrictions:

  - *PING any network*
  - *TRACEROUTE any network*
  - *Access DNS*

- DMZ interfaz receives DNS replies, ping replies and ICMP errors.

```
ipchains -A dmz-if -i ! eth0 -j DENY
ipchains -A dmz-if -p TCP ! -y -s 192.84.219.129 53 -j ACCEPT
ipchains -A dmz-if -p UDP -s 192.84.219.129 53 -j ACCEPT
ipchains -A dmz-if -p ICMP --icmp-type pong -j ACCEPT
ipchains -A dmz-if -j icmp-acc
ipchains -A dmz-if -j DENY -l
```

7.4.9.3. Good (internal) interface.

- Packet Filter box restrictions:
  - *PING any network*
  - *TRACEROUTE any network*
  - *Access DNS*
- Internal restrictions:
  - Permitir WWW, ftp, traceroute, ssh a la interna .
  - Permitir SMTP al servidor de correo.
  - Permitir POP-3 al servidor de correo.
  - Permitir DNS al servidor de nombres.
  - Permitir rsync al servidor Web.
  - Permitir WWW al servidor Web.
  - *Permitir a la máquina de filtrado de paquetes*
- La interfaz interna recibe pings, ping responde mensajes de error ICMP.

```
ipchains -A good-if -i ! eth1 -j DENY
ipchains -A good-if -p ICMP --icmp-type ping -j ACCEPT
ipchains -A good-if -p ICMP --icmp-type pong -j ACCEPT
ipchains -A good-if -j icmp-acc
ipchains -A good-if -j DENY -l
```

7.5. Finalmente.

- Borrar bloques de reglas:

```
ipchains -D input 1
ipchains -D forward 1
ipchains -D output 1
```

Chapter 8. Apéndice: Diferencias entre ipchains e ipfwadm

Algunos de estos cambios son un resultado de modificaciones en el núcleo, y algunos resultado de la diferencia entre `ipchains` e `ipfwadm`

1. Se han remapeado muchos argumentos: las mayúsculas ahora indican un comando, y las minúsculas una opción.
2. Se soportan cadenas arbitrarias, incluso cadenas de construcción que tienen nombres completos en lugar de flags (ej. ‘input ’ en lugar de ‘ -I ’).
3. La opción ‘ -k ’ ha desaparecido: use ‘! -y ’.
4. La opción ‘ -b ’ realmente inserta/añade/borra dos reglas, en lugar de una sola regla ’bidireccional’.
5. La opción ‘ -b ’ puede pasarse a ‘ -C ’ para hacer dos chequeos (uno en cada dirección).
6. La opción ‘ -x ’ para ‘ -l ’ se ha reemplazado por ‘ -v ’.
7. Ya no se soporta múltiples puertos de orígen y destino. Pudiendo negar un rango de puertos que puede hacer más que éso.
8. Las interfaces sólo pueden ser especificadas por nombre (no por dirección). De cualquier forma la vieja semántica se cambió en el núcleo 2.1.
9. Los fragmentos son examinados, no se permiten que crucen automáticamente.
10. Las cadenas de accounting explícitas se han hecho afuera.
11. Los protocolos arbitrarios por encima de IP pueden probarse.
12. El viejo comportamiento de los emparejamientos de SYN y ACK (previamente ignorados para paquetes NO-TCP) ha cambiado. la opción de SYN no es válida para las reglas que no especificas de TCP.
13. Los contadores son ahora de 64 bits en máquinas de 32 bits, no de 32 bits.

14. Se soportan opciones inversas ahora.
15. Códigos ICMP son ahora soportados.
16. Interfaces Wildcard son soportadas ahora.
17. Las manipulaciones de TOS están ahora chequeadas de sanidad: el viejo código de núcleo lo detendría silenciosamente (ilegalmente) manipulando el bit TOS "Must be Zero"; ipchains ahora retorna un error si lo intenta, tal como para otros casos ilegales.

COMPARAR PARRAFO ANTERIOR CON EL SIGUIENTE

1. Many arguments have been remapped: capitals now indicates a command, and lower case now indicates an option.
2. Arbitrary chains are supported, so even built-in chains have full names instead of flags (eg. ‘input’ instead of ‘-I’).
3. The ‘-k’ option has vanished: use ‘! -y’.
4. The ‘-b’ option actually inserts/appends/deletes two rules, rather than a single ‘bidirectional’ rule.
5. The ‘-b’ option can be passed to ‘-C’ to do two checks (one in each direction).
6. The ‘-x’ option to ‘-l’ has been replaced by ‘-v’.
7. Multiple source and destination ports are not supported anymore. Hopefully being able to negate the port range will somewhat make up for that.
8. Interfaces can only be specified by name (not address). The old semantics got silently changed in the 2.1 núcleo series anyway.
9. Fragments are examined, not automatically allowed through.
10. Explicit accounting chains have been done away with.
11. Arbitrary protocols over IP can be tested for.
12. The old behavior of SYN and ACK matching (which was previously ignored for non-TCP packets) has changed; the SYN option is not valid for non-TCP-specific

rules.

13. Counters are now 64-bit on 32-bit machines, not 32-bit.
14. Inverse options are now supported.
15. ICMP codes are now supported.
16. Wildcard interfaces are now supported.
17. TOS manipulations are now sanity-checked: the old núcleo code would silently stop you from (illegally) manipulating the ‘Must Be Zero’ TOS bit; ipchains now returns an error if you try, as well as for other illegal cases.

8.1. Tabla de referencia rápida

[Principalmente, los argumentos de comandos son en MAYUSCULAS, y los argumentos de opciones son en minúsculas]

Una cosa para notar, el enmascaramiento se especifica por ‘ -j MASQ’; es completamente diferente de ‘ -j ACCEPT’, y no se trata de un mero efecto lateral, al contrario que `ipfwadm`

Diferencias entre ipchains e ipfwadm

ipfwadm	ipchains	Notas
-A [both]	-N acct -I 1 input -j acct -I 1 output -j acct acct	Crea una cadena ’acct’ y tiene paquetes salientes y entrantes cruzandola
-A in	input	Una regla sin objetivo
-A out	output	Una regla sin objetivo
-F	forward	Use esto como [Cadena]

ipfwadm	ipchains	Notas
-I	Input	Use esto como [cadena]
-O	output	Use esto como [cadena].
-M -l	-M -L	
-M -s	-M -S	
-a policy	-A [chain] -j POLICY	ver -r y -m
-d policy	-D [chain] -j POLICY	ver -r -m
-i policy	-I 1 [chain] -j POLICY	ver -r y -m
-l	-L	
-z	-Z	
-f	-F	
-p	-P	
-c	-C	
-P	-p	
-S	-s	sólo toma un puerto o un rango, no múltiples
-D	-d	sólo toma un puerto o un rango, no múltiples
-V		Use -i [nombre]
-W	-i	
-b	-b	Ahora hace dos reglas
-e	-v	
-k	! -y	No funciona a menos que se especifique -p tcp
-m	-j MASQ	
-n	-n	
-o	-l	
-r [redirpt]	-j REDIRECT [redirpt]	

ipfwadm	ipchains	Notas
-t	-t	
-v	-v	
-x	-x	
-y	-y	No funciona a menos que se especifique -p tcp

8.2. Ejemplos de comandos ipfwadm trasladados

Antiguo comando: ipfwadm -F -p deny Nuevo comando: ipchains -P forward DENY

Antiguo comando: ipfwadm -F -a m -S 192.168.0.0/24 -D 0.0.0.0/0 Nuevo comando: ipchains -A forward -j MASQ -s 192.168.0.0/24 -d 0.0.0.0/0

Antiguo comando: ipfwadm -I -a accept -V 10.1.2.1 -S 10.0.0.0/8 -D 0.0.0.0/0 Nuevo comando: ipchains -A input -j ACCEPT -i eth0 -s 10.0.0.0/8 -d 0.0.0.0/0

(Note que no hay ningún equivalente para especificar interfaces por dirección: use el nombre de la interface. En esta máquina, 10.1.2.1 corresponde a eth0).

Chapter 9. Apéndice: Usando el script ipfwadm-wrapper

El script de shell `ipfwadm-wrapper` debe ser un medio para reemplazar `ipfwadm` para compatibilidad hacia atrás con ipfwadm 2.3a.

La única característica que realmente no puede manejar es la opción ‘ -V ’. Cuando se usa, se da una advertencia. Si se usa la opción ‘ -W ’ la opción ‘ -V ’ se ignora. Por otra parte, el script intenta encontrar el nombre de interfaz asociado con esa dirección, usando `ifconfig`. Si eso falla (tal como sucede si la interfaz no esta funcionando) entonces terminará con un mensaje de error.

Esta advertencia puede ser eliminada cambiando ‘ -V ’ por ‘ -W ’, o redirigiendo la salida estándar del script a /dev/null.

Si encuentra cualquier error en este script, o cualquier cambio entre el ipfwadm real y este script, *por favor* informeme del bug: envie un EMail a rusty@linuxcare.com con "BUG-REPORT" en el subject. Por favor liste su vieja versión de `ipfwadm` (`ipfwadm -h`), su versión de `ipchains` (`ipchains --version`), la version del script `ipfwadm wrapper` (`ipfwadm-wrapper --version`). También envíe la salidaa `ipchains-save`. Gracias de antemano.

Mezcle `ipchains` con el script `ipfwadm-wrapper` bajo su riesgo??????.

Chapter 10. Apéndice: Gracias.

Muchas gracias a Michael Neuling, que escribió la primera porción del código IP chains mientras trabajaba para mi. Disculpas públicas por Public apologies for nixing his result-caching idea, which Alan Cox later proposed and I have finally begun implementing, having seen the error of my ways.

Gracias a Alan Cox por su soporte técnico por mail H-24, y por su aliento.

Gracias a todos los autores del código de ipfw e ipfwadm, especialmente a Jos Vos. Standing on the shoulders of giants and all that... Esto sirve para Linus Torvalds y a todos los hackers del núcleo.

Gracias a los "beta testers" y busca-fallos, especialmente a Jordan Mendelson, Shaw Carruthers, Kevin Moule, Dr. Liviu Daia, Helmut Adams, Franck Sicard, Kevin Littlejohn, Matt Kemner, John D. Hardin, Alexey Kuznetsov, Leos Bitto, Jim Kunzman, Gerard Gerritsen, Serge Sivkov, Andrew Burgess, Steve Schmidtke, Richard Offer, Bernhard Weisshuhn, Larry Auton, Ambrose Li, Pavel Krauz, Steve Chadsey, Francesco Potorti‘, Alain Knaff, Casper Boden-Cummins and Henry Hollenberg.

10.1. Traducciones

La gente que haga traducciones deben ponerse a ellos mismos al principio de la página de agradecimientos, tal como sigue: ‘Special thanks to XXX, for translating everything exactly from my English.’. Después hazme saber de tu traducción para que pueda colocarla aquí.

Arnaud Launay, asl@launay.org:
http://www.freenix.fr/unix/linux/HOWTO/IPCHAINS-HOWTO.html

Giovanni Bortolozzo, borto@pluto.linux.it:
http://www.pluto.linux.it/ildp/HOWTO/IPCHAINS-HOWTO.html

Herman Rodríguez, herman@maristas.dhis.org:
http://netfilter.gnumonks.org/ipchains/spanish/HOWTO.html

10.2. Sobre traducción

Este documento es el resultado de una revisión de otro documento en html traducido al español por *Herman Rodríguez, herman@maristas.dhis.org.*. Sólo pude encontrar su traducción en formato html y era de la versión *V0.1, Apr 5 1999*. Así que me puse manos a la obra y lo pasé a sgml mientras le añadía partes de la versión *v1.0.9, de Mayo del 2001*, todo ello para www.cortafuegos.org (http://www.cortafuegos.org).