

iptables en español

poné el paquete sobre la mesa

Bellone, Matías
matias@enespanol.com.ar

15 de noviembre de 2006

Índice general

| | |
|--|-----------|
| 1. Semántica | 1 |
| 1.1. Historia | 1 |
| 1.2. Funcionamiento interno | 2 |
| 1.3. Tablas | 2 |
| 1.4. Cadenas | 3 |
| 1.5. Reglas | 4 |
| 1.5.1. Destino | 4 |
| 1.6. Módulos extras | 5 |
| 1.6.1. Módulos de tablas y cadenas | 6 |
| 2. Sintáxis | 8 |
| 2.1. Manejo de Cadenas | 8 |
| 2.1.1. Crear una nueva cadenas | 9 |
| 2.1.2. Ver cadenas existentes | 9 |
| 2.1.3. Eliminar cadenas | 10 |
| 2.1.4. Definir la política de una cadena | 10 |
| 2.2. Opciones manejadas por la interfaz | 10 |
| 2.3. Construcción de filtros | 11 |
| 2.3.1. Destino del filtro | 11 |
| 2.3.2. Filtrado por protocolo | 12 |
| 2.3.3. Filtrado por dirección | 12 |
| 2.3.4. Filtrado por interfaz | 12 |
| 2.3.5. Manejo de fragmentos | 13 |
| 2.4. Extensiones | 14 |
| 2.4.1. Incluidas en el sistema | 14 |
| 2.4.2. No incluidas en el sistema | 18 |
| 3. Práctica | 20 |
| 3.1. Problemas de iptables | 20 |
| 3.1.1. Sintáxis complicada | 20 |
| 3.1.2. Guardando la configuración | 21 |
| 3.2. Respaldos | 22 |
| 3.2.1. Actualizaciones | 22 |
| 3.3. Frontends | 23 |

| | | |
|-----------|------------------|-----------|
| 3.3.1. | Firestarter | 23 |
| 3.3.2. | KNetFilter | 23 |
| 3.3.3. | Firewall Builder | 23 |
| 3.4. | Wrappers | 24 |
| 3.4.1. | Shorewall | 24 |
| 3.4.2. | Ferm | 24 |
| 3.4.3. | gShield | 24 |
| 3.5. | Otros | 25 |
| 3.5.1. | Auto FireWall | 25 |
| 3.5.2. | Floppy FireWall | 25 |
| 4. | Enlaces | 26 |

Resumen

El objetivo de este laboratorio fue investigar acerca de **iptables**, comprender su funcionamiento y sintaxis. Más allá de una simple recopilación de información, es nuestra intención servir de introducción tanto teórica como práctica a esta potente herramienta.

Capítulo 1

Semántica

1.1. Historia

En el año 1998 el equipo de `netfilter`¹ decidió tomar el código de `ipfwadm` e `ipchains` unificándolo y mejorándolo. Esto no fue sólo para consolidar dos aplicaciones que cumplían la misma función sino también para poder sacar lo mejor de cada una eliminando la necesidad de comprender el funcionamiento (y su sintáxis) de dos aplicaciones para poder cubrir todas las necesidades.

Por otro lado, era necesario dicha transición debido a los extensos cambios en el código del kernel pertinente. Dichas aplicaciones tenían serios problemas en sus dependencias con la implementación de `sockets` y las funciones de `Glibc` que se fueron modificando a lo largo de las diferentes versiones de las mismas.

`ipfwadm` se desarrolló como una forma sencilla de administración de las funciones de *firewall* ofrecidas por el kernel. Dependía de la aplicación `ipfw` que se encargaba de modificar las opciones de los sockets correspondientes. Tenía serios problemas para hacer NAT (*Network Address Translation*) debido a que dependía de los puertos utilizados para hacerlo. Además, el manejo de UDP era complicado debido a la falta de un socket permanente utilizando funciones especiales para manejarlo y el manejo de TCP no tenía en cuenta los estados de las conexiones. Por otro lado, su comportamiento - a pesar de estar dividido en módulos - era demasiado monolítico lo que complicaba el extenderlo.

Con la evolución del kernel hacia su versión 2.2 ciertas funciones de manejo de sockets y las librerías GNU de C estándar (la tan conocida `Glibc`) fueron modificadas. Dichas modificaciones provocaban que `ipfwadm` dejara de funcionar; afortunadamente `ipchains` ya había visto la luz. Su modularización era mucho mejor, mejoraba el manejo de conexiones, puertos y la independencia de otros cambios en el kernel y/o librerías utilizadas. Sin embargo, el manejo de NAT era realizado por una aplicación separada `ipnatct1`. Un nivel más de modularización es lo que hizo (y hace) a `iptables` tan poderoso y versátil.

El trabajo del equipo de `netfilter` fue lo suficientemente bueno como para

¹<http://www.iptables.org/>

que el código que necesitaban modificar en el kernel para permitir el manejo de paquetes fuese incluido dentro del kernel principal por defecto. Anteriormente, se debía tener el código del kernel, agregar el parche² a mano y compilar el kernel.

1.2. Funcionamiento interno

Como recién mencionamos, para que `iptables` funcione, debemos de tener un kernel con las modificaciones hechas por el equipo de `netfilter`. Éstas, sin embargo, fueron incluidas en el código oficial toma su nombre de un **funcionamiento según tablas**. Cada tabla contiene **cadenas**, nombre heredado de `ipchains` para determinar a un conjunto de reglas de filtrado.

Para poder funcionar, sin embargo, necesita del módulo de kernel `ip_tables` y del programa `iptables` que sirve como interfaz de usuario. Éste es el módulo principal y se encuentra disponible para todas las versiones 2.4 y 2.6 de kernels existentes y puede utilizarse bien compilado monolíticamente o bien como si fuese un módulo que se cargue cuando sea necesario. Se pueden, además, agregar otros módulos que expanden la funcionalidad del mismo.

Cabe destacar, sin embargo, que las tablas se cargan al iniciar el módulo pero están vacías. Se deben de utilizar otros métodos para hacerlas permanentes.

1.3. Tablas

Por defecto `iptables` no posee tabla alguna. Dejando al usuario la creación de las mismas. Sin embargo, debido a que las tareas de filtrado son relativamente estándares, existen módulos de kernel que trabajan sobre el módulo `ip_tables` agregando las tablas más utilizadas. La tabla más utilizada es la que proporciona el módulo: `iptables_filter`, casualmente la tabla se llama **filter**

Obviamente, como el sistema fue modularizado pensado también en que sea extensible. Sin embargo, el agregado de tablas con significado directo sólo puede realizarse mediante módulos de kernel. Esto se debe a que deben interactuar directamente con el módulo principal `ip_tables` aprovechando los *hooks* que las modificaciones de `netfilter` introdujeron al kernel.

- `iptables_nat`
- `iptables_mangle`

Cada una de éstas agrega una tabla a `iptables` con el nombre que indica.

²*parche* es una modificación de código guardada en formato especial para distribuir sólo las diferencias y reconstruir la versión deseada

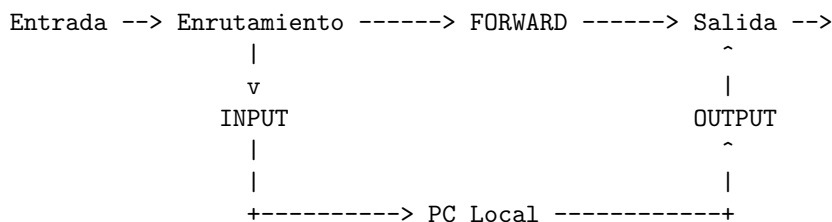
1.4. Cadenas

Las cadenas son, como ya mencionamos, simples conjuntos de reglas. Sin embargo, esto es una simplificación de un concepto un poco más complicado. Originalmente (en `ipchains`) el filtrado de paquetes se hacía en cadena debido a la forma en la que éste atravesaba el sistema. Se hacían chequeos, controles, ruteo, filtrado, etc. en forma secuencial, como una cadena de producción - de ahí el nombre -.

Sin embargo, este comportamiento fue modificado en `iptables` para optimizar el filtrado. Cada cadena está asociada a un punto de la ruta que puede tomar un paquete determinado que ya no es lineal. La tabla más utilizada (*filter*) posee tres cadenas distintas:

- INPUT
- OUTPUT
- FORWARD

Como se mencionó anteriormente las tablas agregadas por módulos contienen ciertas cadenas por defecto. La razón por la que dichas tablas deben ser cargadas como módulos es que éstas cadenas se corresponden con la ruta de un paquete. Esto quiere decir que se utilizan *ganchos (hooks)* en el kernel³ para forzar el filtrado por las cadenas correspondientes según el siguiente gráfico⁴:



Esto significa que todo paquete atraviesa sólo una de las cadenas. El *enrutamiento* no es más que decidir si el paquete está dirigido hacia la propia PC o no. En caso de estarlo, pasaría por la cadena de **INPUT**, de lo contrario, por la de **FORWARD**. Por otro lado, cualquier paquete generado localmente pasará por la cadena **OUTPUT**.

Evidentemente, la modularización es mucho más marcada facilitando la comprensión del sistema. Como si fuese poco, los nombres representativos de cada cadena evitan confusiones o problemas en los filtros por superposición o contradicción de reglas. Lo que es más, el funcionamiento de `iptables` permite el agregado de nuevas cadenas mediante la interfaz de usuario. Evidentemente, éstas no estarán relacionadas con la ruta seguida por el paquete ya que eso

³Éstos son provistos por el soporte de `netfilter` en el kernel

⁴Perdón por el ascii-art pero me falta práctica para imágenes en `LATEX`... y tiempo para hacer las imágenes

se logra interactuando directamente con el módulo principal (`ip_tables`). Sin embargo, permite un manejo más sencillo y modular de los filtros por parte del usuario.

1.5. Reglas

Cada una de las cadenas se compone de filtros, aquí llamados *reglas*. Éstas se componen de **una extensión** que indica qué sucederá con los paquetes recibidos por dicha regla y una serie de parámetros (**un patrón**) que indican las condiciones que debe de cumplir un paquete para obedecer dicha regla.

Las cadenas son construidas secuencialmente. Esto quiere decir que las reglas poseen un orden dado por el orden en el que se agregan. Este será también el orden en el que se evaluarán las reglas, determinando una debilidad del sistema en sí: **no se tiene en cuenta qué regla concuerda más específicamente con el paquete sino sólo la primera⁵ que coincide.**

La razón por la que ésto es importante es que el orden de las reglas es realmente importante. Los errores más comunes en filtrado de paquetes se deben a un orden inapropiado de las reglas para lo que se desea hacer. Por otro lado, el agregar una regla en una posición específica significaría eliminar las reglas siguientes, agregar la regla deseada y volver a agregar la regla original lo que implica una tarea tediosa y engorrosa, especialmente cuando la cantidad de reglas aumenta o en un sistema que necesite de un control complejo.

1.5.1. Destino

Llamamos *destino* a la acción a realizar con un paquete según una regla dada. Por defecto, los destinos son:

- ACCEPT
- QUEUE
- RETURN
- DROP

Como seguramente se adivina **ACCEPT** permite el paso del paquete por la ruta que transita. **QUEUE** provoca que el paquete sea transferido a una cola de espera en el espacio del usuario para que sea revisado; podrá revisarlo cualquier aplicación que haga uso de las librerías `libipq`. **RETURN**, por su parte, provoca un salto hacia el final de la cadena; en las cadenas definidas por el usuario esto implicaría volver a la cadena anterior.

Finalmente, **DROP** desecha completamente el paquete. Cabe destacar que el paquete es *desechado*, no *rechazado*. Esto implica que el paquete no llegará a

⁵Existen excepciones en las que la evaluación de reglas continúa después de una coincidencia, pero serán vistos más adelante

las capas superiores, como si nunca hubiese existido. En tales casos, la PC destino jamás recibirá el paquete y agotará su tiempo de espera para un acuse de recibo (*un ACK*) en caso de estar esperando uno. Este comportamiento puede llegar ser deseable, impidiendo establecer conexiones no deseadas o dar indicios de presencia ante intentos de verificación externos⁶ por parte de sistemas no regulados (por medio de una regla análoga que sí lo permita).

Sin embargo, pueden existir casos en los que se prefiera enviar el paquete de error, o inclusive otras acciones distintas a cualquiera de esas dos. Para ello existen módulos que agregan destinos o condiciones de filtro.

Por otro lado, también se puede especificar como destino de un paquete otra cadena. Normalmente esto sería utilizado con cadenas definidas por el usuario permitiendo, de esta forma, un mejor manejo de las reglas. Lo que es más, una correcta modularización permitiría reducir al mínimo los inconvenientes que significan agregar una regla y que ésta quede en el orden deseado.

Políticas

Hay algo sin embargo que no se mencionó: **¿qué sucede cuando se utiliza un *RETURN* en una cadena que no es del usuario? ¿qué sucede cuando un paquete no coincide con ninguna regla dada?** Sería esperable pensar que, dado que el sistema *interfiere* con el ruteo del paquete, éste siga su curso normalmente.

De hecho, si bien es el caso, esto no siempre es el comportamiento deseado. Por ello, se puede determinar la acción por defecto de una cadena. Esta acción por defecto se denomina **política** de la cadena y puede ser cualquiera de las extensiones disponibles (que no sean una cadena definida por el usuario).

1.6. Módulos extras

Como ya se mencionó, `iptables` fue diseñado para ser extensible y adaptable a muchas y diversas necesidades.

En general, se necesita que el kernel haya sido compilado con soporte para `netfilter` lo que agrega instrucciones en el kernel que permite tomar paquetes en puntos específicos de la ruta que siguen. Además, es necesario el módulo `ip_tables` que utiliza dichos puntos en el kernel para proveer de una interfaz más sencilla de manejar y la estructura interna del sistema como ya se explicó. Y es sobre éste módulo que se trabaja finalmente.

Sin embargo, para poder agregar tablas, cadenas y extensiones, se necesitan de otros módulos. Por otro lado, existen otros módulos de los que `iptables` puede aprovecharlos módulos también permiten agregar otras funcionalidades. Dividiremos, entonces, los módulos en tres grandes grupos:

- Módulos de `netfilter`

⁶Es muy común por parte de atacantes tener un sistema que envía paquetes para determinar la existencia de posibles objetivos si éstos devuelven algún tipo de respuesta.

- Módulos de tablas y cadenas
- Módulos de destinos

Si bien no se encontró información que así lo asegure, parecería existir algún tipo de convención o estándar en el nombre que reciben los módulos. Como se verá más adelante, se cumplirían las siguientes reglas:

Módulos de netfilter Están prefijados por "ip_" y un nombre descriptivo de lo que realizan. El módulo `ip_tables` estaría en este grupo.

Módulos de tablas Contienen el prefijo "iptables_" seguido del nombre de la tabla que agregan.

Módulos de destinos Contienen el prefijo "ipt_" seguido del nombre del destino mayúsculas. Puede suceder que el destino no sea exactamente el nombre del módulo sino que sea una forma corta (para facilitar el uso). Serán cubiertos más adelante ya que no atañen al funcionamiento interno

Cabe destacar que, si bien aquí se los menciona por separado, la mayoría (sino todos) de los módulos que se describirán – y muchos otros – están incluidos en una instalación por defecto. No sólo eso, sino que incluso algunos nisiquiera son un módulo extra que debe ser instalado y cargado pero se lo reconoce como una extensión; aún así, no dejan de ser módulos prácticamente independientes.

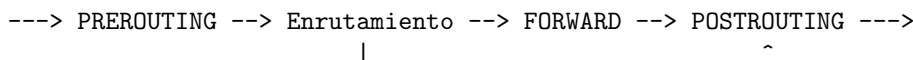
1.6.1. Módulos de tablas y cadenas

Estos módulos se caracterizan por agregar nuevas tablas y cadenas por defecto en éstas tablas. Dichas cadenas se corresponden con algún punto específico de la ruta seguida por un paquete. Los más utilizados⁷ ya fueron mencionados: `iptables_filter`, `iptables_nat` e `iptables_mangle`. El primero ya fue analizado en detalle, veamos ahora qué funcionalidades proveen los otros dos.

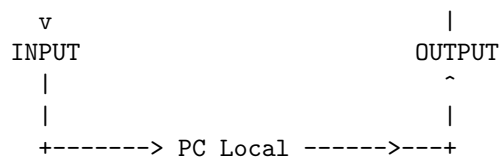
`iptables_nat`

Este módulo, como bien indica su nombre sirve para poder hacer NAT (*Network Address Translation*) mediante `iptables`. Para poder lograrlo agrega al sistema una tabla, casualmente llamada `nat`, y tres cadenas por defecto: **OUTPUT**, **PREROUTING** y **POSTROUTING**.

La cadena **OUTPUT** es idéntica a aquella presente en la tabla `filter` en cuanto al papel que representa en el esquema de ruteo de un paquete. La principal diferencia radica en las otras dos cadenas ubicadas en dos puntos distintos del ruteo del paquete a todas las demás cadenas vistas hasta ahora. El gráfico a continuación así lo evidencia:



⁷ Además, éstos son los únicos de los que conocemos existencia



Para quienes sepan inglés habrán notado los significados de las nuevas cadenas implican un *pre-ruteo* y un *post-ruteo* para todos los paquetes que atraviesan el alcance del sistema. Obviamente, esto se debe a los momentos en los que se pueden tomar decisiones sobre NAT evitando perder tiempo. Si la cadena **PREROUTING** estuviese después del ruteo, no se podría hacer un simple *enmascaramiento* de la red ya que todos los paquetes hacia la red interna atravesarían la cadena **INPUT** primero, implicando la necesidad de generar un nuevo paquete de forma local para poder enviarlo hacia la red interna.

Obviamente, se agregan una serie de extensiones que nos permitirán el manejo de paquetes y transformaciones de IPs haciendo **DNAT** (*Destination NAT* ó **SNAT** (*Source NAT*) sin problema alguno.

`iptables_mangle`

No conforme con las capacidades actuales del sistema existe aún un módulo más de amplia utilización. `iptables_mangle` agrega, como ya se mencionó, una nueva tabla llamada **mangle** (*alterar hasta hacerlo irreconocible* en inglés) que posee todas las cadenas hasta ahora vistas: **INPUT**, **OUTPUT**, **FORWARD**, **PREROUTING** y **POSTROUTING**.

Las capacidades de esta tabla son las de modificar prácticamente cualquier aspecto conocido de un paquete dado. De allí el nombre.

Capítulo 2

Sintáxis

Con el conocimiento del funcionamiento interno de `iptables` el poder utilizar el sistema se limita a aprender la sintáxis necesaria para poder majar los mismos. Evidentemente, para ello se dispone de una aplicación que interactúa con las tablas y reglas de una forma sencilla: `iptables`.

El uso que se le pueda dar está limitado por nuestra creatividad y los límites innatos del sistema en sí. A continuación se exponen las opciones para el manejo de dicha aplicación en conjunto con sencillos ejemplos. Como se comprobó en el código fuente, para el manejo de opciones `iptables` hace uso de la abstracción brindada por `getopts` facilitando la utilización de versiones cortas y largas – pero más expresivas – de los parámetros. Ésto permite a usuarios novatos comprender lo que hace cada regla si utilizan las versiones largas sin perder las ventajas de evitar tipear nombres largos.

2.1. Manejo de Cadenas

Si bien `iptables` incluye cadenas por defecto, se puede modularizar el sistema de filtrado haciéndolo más sencillo de comprender y mantener. Ésto se logra agregando cadenas propias, bajo la única condición que sus nombres no se superpongan con cadenas existentes.

Para trabajar con una tabla en especial se la puede especificar agregando `-t nombre_tabla`. Esto no es necesario, sin embargo, debido a que el sistema interpreta que, por defecto, uno siempre se refiere a la tabla `filter`, si ésto no es lo deseado entonces se debe especificar la tabla sobre la que se está trabajando.

También es de destacar el hecho que la utilización de módulos modifica la cantidad de opciones aceptadas por la interfaz de usuario. De ahí que exista la opción `-m nombre_módulo` que fuerza la carga de un módulo determinado para poder aprovechar las opciones de las que dispone. La extensa utilización de ciertos módulos, sin embargo, permite la omisión de dicha opción para el uso de ciertas opciones.

Para mayor simplicidad, de ahora en adelante se supondrá que, a menos que

no sea posible, se trabajará sobre la tabla **filter** y no se forzará la carga de módulos siempre que sea posible. Por su parte, los parámetros serán indicados según la notación más concisa y aceptada, de gran similitud con expresiones regulares.

Corchetes Indicarán parámetros opcionales. Esto es, que pueden aparecer o no

Llaves Cada elemento separado por comas dentro de ellas es una de los posibles valores aceptados.

Texto en español Siempre que se observe un texto en español con un nombre bastante descriptivo se debe reemplazar el mismo por lo que esté pidiendo.

Existen además, una serie de opciones generales que pueden ser especificadas en cualquier instrucción. A saber:

- v** El siempre conocido *verbose* que provoca mensajes más descriptivos y detallados de la actividad circundante.
- n** La salida de direcciones IPs y puertos se mantiene en forma numérica (por ejemplo: 127.0.0.1 en lugar de localhost)
- line-numbers** Cada regla tendrá asignado un número dentro de la cadena. Esta opción provoca que dichos números sean mostrados antes de cada regla.

2.1.1. Crear una nueva cadenas

Para hacerlo sólo se debe ejecutar el siguiente comando:

```
-N nombre_cadena
```

2.1.2. Ver cadenas existentes

Evidentemente se debe de poder ver qué reglas contiene una cadena dada. Para hacerlo, basta con ejecutar:

```
-L [nombre_cadena]
```

Cabe la posibilidad de no proveer nombre de cadena alguno. Caso en el que se mostrarán por la salida estándar todas las reglas asociadas a cadena.

2.1.3. Eliminar cadenas

Si podemos agregar cadenas, entonces deberíamos de poder eliminarlas. Para ello contamos con la siguiente regla:

```
-X [nombre_cadena]
```

De forma consistente con las demás, esta regla elimina todas las cadenas creadas por el usuario si ninguna es especificada.

2.1.4. Definir la política de una cadena

Como bien se mencionó anteriormente, cada cadena posee un destino que acompañará a todo paquete que no coincida con ninguna regla. Dicho destino determina la **política** de la cadena y se puede especificar mediante la siguiente instrucción:

```
-P nombre_cadena destino_por_defecto
```

2.2. Opciones manejadas por la interfaz

Cada cadena tiene sus propias reglas. Para poder agregar una regla a una cadena especificada debemos decidir si queremos hacerlo al principio o al final de la cadena en cuestión.

Si lo queremos hacer al final, basta con ejecutar:

```
-A regla_deseada
```

Si, en cambio, deseamos agregarla en un punto en particular (por defecto la agrega al comienzo) :

```
-I regla_deseada [numero_de_orden]
```

Por otro lado, también puede suceder que deseemos eliminar una regla específica. Para ello podemos hacerlo bien por medio de un filtro exacto o por el número de orden de la orden en una cadena:

```
-D regla_deseada  
-D número_de_regla
```

Evidentemente, la primera es de más fácil lectura pero presenta un inconveniente: es necesario recordar exactamente el enunciado de la misma para que sea eliminada. El utilizar los números de reglas tendría un beneficio evidente.

Si se desea, sin embargo, eliminar todas las reglas de una cadena el trabajo es lento y tedioso. Para ello, disponemos de otra opción:

`-F cadena`

Esto provoca que todas las reglas de la cadena especificado sean desechadas y se vuelva al estado inicial de la configuración dicha tabla en el que no tiene regla alguna. Queda, sin embargo, un valor que no es reseteado; éste valor se trata de una extensión del sistema que permite llevar cuenta de ciertos paquetes.

Para poder resetear dichos contadores, disponemos de la opción:

`-Z cadena`

Por último, suponiendo el caso en que uno estuviera probando una nueva regla pero no está del todo seguro de su funcionamiento. A tal efecto, uno suele probar hasta que se logra el efecto deseado y el agregar y quitar reglas permanentemente es demasiado trabajo; especialmente cuando se dispone de la posibilidad de reemplazar una regla en una cadena determinada:

`-R cadena numero_de_regla`

2.3. Construcción de filtros

La potencia de `iptables` radica en su excelente nivel de modularización. Por ello, en lugar de estudiar grandes y complicadas reglas para construir filtros, daremos las opciones más comunes para que cada quien las utilice según sus necesidades.

Todas las opciones que se especificarán a continuación pueden ser negadas. Para ello se debe preceder a los valores deseados (o la opción en uno de los casos) con un signo de exclamación ("!" sin comillas) de forma similar a la negación de una variable en el lenguaje de programación C. De esta forma se podrán hacer filtros de una forma más sencilla.

A continuación se describirán aquellas opciones incluidas por defecto dentro de la aplicación. Éstas pueden ser rápidamente revisadas gracias a la ayuda de la misma al ejecutar:

`iptables -h`

A diferencia de dicha ayuda, aquí se encontrarán ordenadas según utilidad o el orden que se creyó más útil o sencillo para comprender. En la ayuda incluída, como es práctica usual, se encuentran ordenadas por orden alfabético.

2.3.1. Destino del filtro

Lo primero que se debe saber para poder construir un filtro con `iptables` es cómo especificar el destino de los paquetes que coincidan con él. La opción

que controla éste aspecto del filtro se puede resumir en:

```
-j destino_válido [opciones]
```

Recordemos que los valores posibles de `destino_válido` dependen de los módulos cargados en un momento dado. Por defecto serían **ACCEPT** y **DROPL**. Las `opciones` son modificadores extras que dependen del destino.

2.3.2. Filtrado por protocolo

Una de las primeras distinciones a realizar en un paquete es el protocolo según el que transita. Como administradores de sistemas puede ser de nuestro interés el filtrar alguno de los protocolos más utilizados. Esto se logra mediante:

```
-p {tcp,udp,icmp}
```

Si no se especifica protocolo alguno, directamente no se lo tendrá en cuenta. Si es especificado, en cambio - dependiendo del protocolo especificado - se podrían generar una serie de parámetros extras específicos de dicho protocolo para facilitar el trabajo.

2.3.3. Filtrado por dirección

Pudiendo bloquear por protocolo, sería un despropósito no poder filtrar por IP. Para ello, podemos filtrar a aquellos paquetes que estén originados o destinados a cierta IP mediante las reglas:

```
-s {dirección_IP[/máscara_de_red],nombre_de_host}  
-d {dirección_IP[/máscara_de_red],nombre_de_host}
```

Indica la dirección de origen (o destino, respectivamente) del paquete. Ésta puede ser indicada por el nombre de host (por ejemplo: "google.com") o con la dirección IP en forma numérica. La máscara de red, a su vez, se puede indicar del modo tradicional de 4 números decimales separados por puntos o un entero entre 0 y 32 indicando la cantidad de bits que se fijan de la máscara.

2.3.4. Filtrado por interfaz

Por otro lado, también se puede filtrar por interfaz¹ (la aplicación `ifconfig` mostrará la lista de interfaces activas) gracias a las opciones:

```
-i nombre_interfaz  
  
-o nombre_interfaz
```

¹"Interfaz" es el término que define una abstracción sobre los dispositivos de red.

Estos pueden filtrar según la interfaz por la que entre o salga (respectivamente) un paquete. Parecería ser inútil el poder filtrar por interfaz si ya se dispone de la capacidad de filtrar por IP. Sin embargo, no es extraño el encontrar ataques que se basan en la creación de paquetes especiales. Por ejemplo, crear un paquete que parezca provenir de una IP local (técnica denominada "spoofing"). La forma más fácil de controlar estos ataques sería revisando las interfazs involucradas.

Evidentemente, sólo los paquetes que atraviesen la cadena **FORWARD** tendrán ambas interfaces. Aquellos que sean dirigidos a la propia PC (y que atraviesen la cadena **INPUT**) no tendrán interfaz de salida. Por lo tanto, cualquiera regla de dicha cadena que incluya esa opción no coincidirá con ningún paquete. Secede algo análogo con los paquetes que atraviesan la cadena **OUTPUT** que no tendrán interfaz de entrada, así que la regla -i no funcionará.

También es de destacar que es perfectamente válido especificar un interfaz que no existe. La regla en cuestión simplemente no coincidirá con ningún paquete. Como un caso especial, un nombre de interfaz con el sufijo "+" coincidirá con todas las interfaces que comiencen con dicha cadena.

2.3.5. Manejo de fragmentos

Aquellos familiares con la forma en la que se encapsula la información debido al modelo por capas con el que se trabaja se habrán percatado de que no todos los paquetes poseen toda la información necesaria para poder filtrarlos. Más puntualmente, el puerto de origen o destino sólo pueden ser encontrados en la cabecera del protocolo. Si el paquete se fragmenta en varias porciones: ¿cómo hago para filtrar el resto de los paquetes que no contienen esa información?

Teóricamente, no debería de ser necesario. Si la cabecera de un mensaje contiene toda la información entonces será filtrada en caso que así se lo desee. Por más que los demás fragmentos no posean dicha información y pasen los filtros desapercibidos las capas correspondientes no serán capaces de reconstruirlo sin la cabecera que sí fue filtrada.

En la práctica sin embargo, hay muchos efectos no deseados. El primero es que un paquete "roto" significa un pedido de retransmisión. Como el paquete fue roto voluntariamente todas las retransmisiones también lo estarán. Esto provoca un tráfico no sólo inútil sino también innecesario desperdiciando recursos (tanto económicos como de ancho de banda).

Por otro lado, todo el tráfico de los fragmentos hasta las capas superiores implica una cantidad de procesamiento. Si es nuestra intención impedir el acceso a esa información sería más eficiente el bloquear todos los fragmentos que sólo algunos de ellos e impedir la reconstrucción del original.

Para ello, `iptables` hace un seguimiento de los paquetes para que todos los fragmentos coincidan con las reglas del paquete que les corresponda. Aún así, provee la siguiente opción para diferenciar el primer paquete de los demás:

-f

En caso de aparecer en una regla, indica que sólo se aplicará a los fragmentos que no sean el primero.

2.4. Extensiones

Como bien fue mencionado varias veces ya, `iptables` fue diseñado teniendo en cuenta la modularización para hacerlo fácilmente extendible. Esto se puede lograr, bien mediante módulos - para tareas complicadas - o simplemente con librerías que interactúan con el usuario.

Las extensiones existentes responden a diferentes necesidades que se presentaron con el uso del sistema y fueron desarrollados aprovechando la extensibilidad del mismo. Podemos clasificarlas según dos criterios distintos pero para nada disjuntos: **inclusión en el sistema ó área que extienden**. Contrario a lo que es usual, será el primero el principal criterio de división utilizado - dejando el otro para una subclasificación - debido a que este documento está pensado para facilitar el uso del sistema.

Es de destacar que, en algunos casos será necesario explicitar el deseo de uso de alguna de dichas extensión por medio de la opción:

```
verb+-m extension+
```

En otros casos, sin embargo, esto puede ser omitido y la aplicación cargará la extensión automáticamente.

Para saber cuáles son las extensiones de las que disponemos, nada más sencillo que buscar entre los módulos de kernel con el comando `lsmod` o simplemente listar las librerías que figuran en `/lib/iptables`

2.4.1. Incluidas en el sistema

Muchas de las extensiones fueron consideradas tan útiles o populares que fueron incluidas directamente en el sistema base. Esto quiere decir que deberían de estar disponibles en una instalación limpia de `iptables`.

Si bien se listarán las opciones más utilizadas con una breve descripción, siempre se podrá consultar la ayuda incluida con el comando:

```
iptables -m modulo --help
```

TCP, UDP, ICMP

Como bien vimos, `iptables` puede filtrar nativamente por protocolo. Es de notar que cada uno de estos protocolos lleva asociado una extensión que permite la utilización de opciones adicionales.

La más sencilla de todas es **ICMP** que sólo agrega una opción: `-icmp-type`. Ésta permite, como seguramente se imaginan, filtrar los paquetes ICMP según el tipo (pings, errores de red, etc).

UDP por su parte, sólo agrega dos opciones capaces de ayudarnos a filtrar por puerto o rango de puertos y si es el puerto de origen ó destino con: *-sport* y *-dport*.

Finalmente **TCP** es la más completa ya que, además de las opciones brindadas por la extensión **UDP**, provee dos filtros adicionales: *-tcp-option* y *-tcp-flags* que permite distinguir el valor del campo de opción de la cabecera TCP y saber qué flags están activadas. Sobre éste último, necesita de dos valores:

una máscara una lista separada por comas de uno o más de **SYN**, **ACK**, **FIN**, **RST**, **URG** y **PSH** o bien **ALL** como una forma corta de agregar todas las opciones ó **NONE** para no indicar ninguna

un filtro otra lista separada por comas de aquellas opciones que aparecen en la máscara que deben de estar activadas para que coincida el paquete

mac

Algo también de gran interés para asegurar una red privada es el filtro para comparar paquetes según direcciones Ethernet (MAC). Es de destacar, sin embargo, que sólo puede filtrar por dirección entrante² por lo que sólo se lo puede utilizar en las cadenas **PREROUTING** e **INPUT**.

Necesita de ser utilizado explícitamente mediante *-m mac* y provee, de esa forma, sólo de la opción *-mac-source*. Sus resultados son obvios.

limit

Este módulo debe ser especificado explícitamente con *-m limit* y se utiliza para restringir la tasa de coincidencias indicando una máxima cantidad de paquetes por unidad de tiempo que seguirán esa regla. Agotada dicha cantidad de paquetes, las reglas que contengan este condicionador actuarán como si tuviesen un destino **DROP**.

Provee de dos opciones con las que trabajar: *-limit-burst* y *-limit*. La primera indica la cantidad máxima de paquetes a analizar antes de comenzar a descartarlos (5 paquetes por defecto) y la segunda indica la tasa de recuperación de coincidencias por unidad de tiempo (3 paquetes por hora por defecto).

Se puede, además, especificar la unidad de tiempo utilizando diferentes unidades como *second*, *minute*, *hour* ó *day*. Acepta inclusive abreviaciones: (5/*second* es lo mismo que 5/s). Nótese sin embargo, que no es posible crear una regla en la que se se especifique una tasa de recuperación multiplicada por el límite de paquetes deberá ser menor a 59 horas.

La combinación de estas opciones permite indicar una tasa máxima. En el caso por defecto (5 de máxima, recuperando 3 paquetes por hora) cada paquete que coincida disminuirá la máxima y ésta aumentará en 1 por cada 20 minutos (3

²Esto se debe a que estamos trabajando a nivel de capa de red, donde un paquete está dirigido a la dirección mac del dispositivo siguiente en la ruta. Lo que quiere decir que todos los paquetes que pasen por una PC estarán dirigidos a su dirección mac, más no necesariamente a su dirección IP

por hora). En cuanto a las limitaciones de contabilización, si se desea establecer una tasa de 1/día el máximo deberá ser menor a 3 (1/day con limit de 3 fuerza la contabilización de 72 horas continuas).

mark y MARK

Existe un módulo especial que provee tanto de una opción para filtrar paquetes como un destino. Está restringido a la tabla **mangle** ya que modifica los paquetes: los marca. Utilizándolo como destino provee de las opciones *-set-mark*, *-and-mark* y *-or-mark* que establecen el valor de la marca u operan con un "y lógico" u "ó lógico" según el parámetro pasado.

Sabiendo, además, que la evaluación de las reglas no se detendría es posible utilizar las marcas como una forma de agrupar paquetes y luego actuar sobre ellos mediante el filtro *-mark* (notar que en este caso es necesario cargar la extensión con *-m mark*). También, la aplicación **iproute2** es capaz de leer las marcas para, gracias a **iptables**, modificar el ruteo de los paquetes.

CLASSIFY

De la misma forma que la extensión *mark* puede ayudarnos a agrupar paquetes, existe esta otra capaz de clasificarlos. La diferencia, sin embargo, es las aplicaciones para las que dicha clasificación es útil.

El destino **CLASSIFY** provee la opción *-set-class* que acepta dos números separados por dos puntos (:). Esta clasificación luego puede ser leída por la aplicación **tc** para poder hacer *traffic shaping*. Correctamente implementado evitaría el tener que indicar también a **tc** rangos de IP, mac, etc. para poder dividir redes o filtrar servicios de una forma más específica evitando la reprogramación de accesos y permisos

NOTRACK

En algunos casos, la performance del sistema es crítica y, por lo tanto, los recursos disponibles deben de ser aprovechados al máximo. A tal efecto, disponemos del destino **NOTRACK**, que simplemente evita el seguimiento de una conexión. Esto libera recursos para los paquetes de conexiones que no se deseen controlar. Su uso sencillo ya que no toma opciones y su utilidad es evidente en grandes redes con varias sub-redes que se encargan de seguir sus propias conexiones

REDIRECT

El poder hacer NAT muchas veces no es suficiente. A veces sólo necesitamos cambiar el puerto de una conexión entrante ya que el presupuesto no nos permite tener dos equipose separados con los puertos por defecto funcionando. O tal vez queremos utilizar un caché de forma transparente. A tal efecto, el la extensión de destino **REDIRECT** añade la opción *-to-ports* que permite hacer el cambio pertinente.

REJECT

Como se había mencionado, el sistema de `iptables` por defecto sólo puede descartar paquetes mediante **DROP**. Este paquete agrega un nuevo destino para los paquetes con un comportamiento similar. No sólo descarta el paquete sino que, además, genera un paquete de error ICMP y lo envía de vuelta al origen del paquete filtrado.

Permite, además, configurar el tipo de paquete de error ICMP que se envía. Por defecto es `port unreachable` pero puede cambiarse con la opción `-reject-with` seguida del tipo deseado.

LOG

Cualquier administrador de sistemas sabe que algo tan importante como la información que contienen sus sistemas son los registros *logs* generados por el funcionamiento de los mismos. Éstos contienen información sobre toda la actividad que se desarrolla en los mismos.

La actividad de red, actualmente, es una de las más importantes. No sólo por la cantidad de sistemas que dependen de la red para funcionar, sino también por el simple hecho de que es la principal puerta para ataques. Un sistema indefenso se caracteriza por permitir un acceso sencillo al sistema por medio de la red. Si no se lo desea, se debe utilizar algún tipo de filtrado para evitarlo y, lo que es más importante, nunca está de más registrar cuando dichos intentos suceden.

`iptables` contiene una extensión para generar logs de una forma sencilla. Simplemente se agrega una regla cuyo destino sea **LOG**. Este destino es especial, a diferencia de las demás reglas, aquellas con éste destino no concluirán la evaluación de reglas. De esta forma, si se quiere rechazar o descartar un paquete y loguearlo se debe de poner la regla que lo loguee primero; de lo contrario, el paquete se desechará y jamás llegará hasta la regla que lo loguee.

El logueo se realiza en conjunto con los demás logs del sistema operativo y es posible especificar el nivel de logueo con `-log-level` e inclusive una cadena de hasta 29 caracteres que servirán de prefijo con `-log-prefix`. Esto posibilita el diferenciar la cadena utilizada para cada paquete logueado. Existen otras opciones que permiten loguear opciones, números de secuencia o el ID de usuario del socket utilizado.

Es de notar que, para una PC conectada a internet, los registros pueden volverse muy extensos; especialmente si se intenta reducir el tráfico al mínimo. Por ello, es necesario un buen manejo de los logs del sistema operativo para evitar agotar el espacio del disco duro. La extensión *limit* descrita anteriormente puede ser de gran ayuda.

MASQUERADE

Con el transcurso del tiempo, la difusión de redes locales fue cada vez mayor... al igual que el deseo de interconectarlas. Obviamente, el conectar cada dispositivo con todos los demás sería no sólo engorroso sino también costoso.

Por ello, toda red local dispone de uno o más dispositivos que sirve como **puerta de enlace** hacia las demás redes.

Sin embargo esto plantea un problema sencillo en cuanto a comunicación de dos equipos en la sección interna de cada red. Si sólo dichas puertas de enlace pueden comunicarse directamente, se necesita de una forma de simular una conexión entre los dispositivos de las redes locales. Para ello existe el **enmascaramiento** (*masquerading*).

Como las reglas para poder manejar estos casos son en exceso complicadas debido a los cambios que se deben de realizar en los paquetes, se creó un destino específico a tal efecto. Cualquier paquete que atravesase algún punto de la tabla **nat** puede tener **MASQ** como destino, implicando tal acción. Esto implica no tener que utilizar toda una serie complicada de reglas para controlarlo

2.4.2. No incluídas en el sistema

La cantidad de extensiones no incluídas en el sistema es inmensa. Esto se debe, no sólo a la facilidad del desarrollo de una, sino también al hecho que fueron pensadas para solucionar problemas muy específicos o facilitar alguno ya solucionado.

Se pueden encontrar en dos formas distintas: **módulos extras** ó **parches para el código**. Estos últimos implican la recompilación de la aplicación o inclusive de algún módulo por no ser oficiales (a pesar de estar extensamente probados algunos); es por ello que nos centraremos en los primeros.

`ip_conntrack: state`

Este es un módulo de kernel que se acopla al sistema de netfilter. Sin ir más lejos, fue desarrollado por netfilter para proveer de una forma de mantener un registro de las conexiones existentes. En particular, este módulo es incluido dentro de la instalación base por una cuestión de dependencias del módulo `ipt_nat` demostrando ser un pilar importante del funcionamiento del sistema. No lo mencionamos anteriormente porque es en sí un módulo separado de `iptables`.

Es éste módulo que permite la existencia del módulo `state`, capaz de filtrar paquetes según el estado de su conexión (leído directamente de la información que mantiene `conntrack`). Debe de ser explícito en su uso mediante `-m state` y ofrece sólo una opción: `-state` que toma como parámetro una lista separada por comas de los siguientes valores:

NEW Conexiones nuevas, por lo general, el primer paquete que entra o sale

ESTABLISHED Todos los demás paquetes de una conexión después del primero

INVALID Un paquete cuya conexión no pudo ser identificada, ya sea por cuestiones de memoria o paquetes ICMP que no correspondieron a ninguna conexión

RELATED Paquetes que no son parte de una conexión, pero derivan de ella (como paquetes ICMP)

ftp

A pesar de todos los módulos y extensiones de los que ya hemos hablado. Sigue habiendo un problema: las conexiones FTP. Sucede que el protocolo FTP utiliza una conexión de control pero otra conexión completamente distinta para la transmisión de datos. Esta última no es permanente y mantener el puerto abierto sería un despropósito para un firewall.

Es por eso que netfilter pone a nuestra disposición dos módulos más: *ip_conntrack_ftp* e *ipt_nat_ftp*. El primero se encarga de que el módulo de seguimiento de conexiones pueda relacionar las dos conexiones que acabamos de mencionar sobre el protocolo FTP. El segundo en cambio, se encarga de fisgonear en los paquetes que pasan por las cadenas de la tabla `nat` para poder hacer lo mismo.

Sin estos módulos, no hay forma alguna en la que se pueda utilizar dicho modo de FTP teniendo un firewall restrictivo al mismo tiempo. De la misma forma, existen módulos para muchos otros protocolos con inconvenientes similares como H.323, IRC, amanda, PPTP, TFPT, etc.

Capítulo 3

Práctica

La teoría de construcción de firewalls es muy sencilla, simplemente es aprender la sintáxis o métodos de configuración de la aplicación deseada y punto. Nada que no pueda encontrarse en las páginas de manual o repartido en varias páginas de internet. Lo que sí cuesta es encontrar las herramientas que nos ayudan a hacer el trabajo más sencillo o guías sobre las prácticas más usuales.

3.1. Problemas de iptables

En un intento de ser objetivos, no todo es de color de rosa con `iptables`, tiene defectos como cualquier otro sistema. Defectos para el que existen soluciones pero que pueden no ser del todo convenientes para el criterio de algunos. Los más comunes son dos:

3.1.1. Sintáxis complicada

Como bien se repasó a lo largo de todo este documento, la sintáxis de `iptables` no es para nada sencilla. Las opciones básicas son poderosas pero para cualquier uso que se le quiera dar se necesita de un poco más de conocimiento.

La cantidad de opciones y extensiones disponibles no facilitan el trabajo tampoco. Las opciones se multiplican, las combinaciones posibles, restricciones y requisitos para utilizar cada una de ellas no son siempre de lo más sencillas de comprender. Y, si queremos utilizar extensiones no estándar todo es aún más interesante ya que algunas implican tener que compilar porciones de código a mano.

Las soluciones a todo esto son variadas. Algunas las revisaremos más adelante: **frontends** y **wrappers**, pero todas tienen sus problemas. Las primeras no permiten un control completo y minucioso de todas las posibilidades y cualquier uso demasiado avanzado requerirá de conocimientos directos de `iptables`. Los

segundos, en cambio, no simplifican la sintáxis sino que sólo la cambian por otra que puede (o no) resultarnos más cómoda o familiar.

Constructores

De todas formas podemos tener un firewall sin tener la más remota idea de `iptables`. Para ello existen, pululando por internet sistemas completos en los que uno selecciona servicios, puertos y topología de la red y éste nos devuelve las reglas de `iptables` necesarias para implementarlo.

Uno de ellos, y a mi criterio el más completo, es el **On-line FireWall generator**¹. Con unos cuantos clicks nos brinda las reglas capaces de manejar un equipo aislado o un servidor que funcionará de puerta de enlace hacia internet para una red completa o brindará uno que otro servicio. Si bien está en inglés, provee de una extensa ayuda sobre cada opción configurable para un mejor entendimiento de lo que se está haciendo.

Como si esto fuese poco, el código generado (que no es más que un script de shell) está muy bien organizado y es fácilmente entendible. Lo que es más, los extensos comentarios sobre el mismo hace casi innecesario tener conocimiento alguno sobre `iptables` para saber qué es lo que hace cada línea.

3.1.2. Guardando la configuración

Si no lo han notado, toda la sintaxis explicada anteriormente trabaja directamente sobre los módulos de kernel agregando - en tiempo de ejecución - las reglas a las cadenas deseadas. Cada una de éstas es mantenida en memoria para el permanente chequeo de cada paquete con ventajas de ser posible modificarlo en tiempo real ... pero se pierde al reiniciar la PC.

La aplicación `iptables` posee, sin embargo, dos aplicaciones que nos podrían ayudar a tal efecto: `iptables-restore` e `iptables-save`. Como bien indican sus nombres, sirven para restaurar y guardar la configuración de `iptables` en un momento dado. Ofrecen la posibilidad de guardar cada tabla por separado, guardar los contadores de paquetes y bytes e inclusive hacer restauraciones incrementales del firewall.

Para utilizarlo basta con ejecutarlos y redireccionando la entrada o salida a un archivo correspondiente. La opción `-c` es la que controla el tema de los contadores, `-t` sirve para indicar la tabla a guardar y `-n` evita el vaciado de las tablas al restaurar el firewall. Sin embargo, es algo tedioso de hacer a mano cada vez; ni hablar cuando se trata de un servidor remoto y autónomo que puede llegar a sufrir de cortes de luz o inconvenientes inesperados similares. Definitivamente no queremos que quede sin firewall.

Carga al arranque

Una solución posible sería generar un script que sea ejecutado automáticamente durante el arranque de la PC. Dicho script no haría más que invocar la

¹Lo podrán encontrar en <http://easyfwgen.morizot.net/gen/>

aplicación `iptables-restore` leyendo desde un archivo en el que hayamos guardado previamente nuestras reglas de firewall con la aplicación `iptables-save`.

Obviamente esto puede complicarse un poco más, modularizando el firewall guardando cada tabla por separado. O más aún, utilizando scripts propios que ejecuten una por una las instrucciones para reconstruir nuestro firewall. La idea es siempre la misma y es aquella utilizada por defecto en la mayoría de los sistemas agregando dicho script a la lista de scripts ejecutados al iniciar la PC en el directorio `/etc/init.d/` con algún nombre apropiado.

Carga junto con la red

La solución anterior es la correcta y la necesaria para sistemas que necesitan de un firewall corriendo permanentemente ya que están siempre conectadas a internet o funcionan como server o puerta de enlace para una red. Para un equipo personal que no está conectado permanentemente a internet, eso puede ser no sólo un despropósito sino también un desperdicio de recursos.

Para ello se puede configurar el sistema para que ejecute el script al levantar una interfase de red. Esto sería extremadamente similar a lo anteriormente descrito, sólo que los scripts deberían de ubicarse o referenciarse de otra forma. En sistemas Debian o similares, esto podría ser en `/etc/network/if-up.d/`.

3.2. Respaldos

Es importante siempre tener copias de respaldo de las reglas del firewall. Esto es porque, con el tiempo, los servicios que uno va abriendo o cerrando, las reglas y módulos que utiliza. En casos críticos, la información que se respalda son siempre los datos del sistema de archivos, los usuarios y servicios brindados por la PC... rara vez incluye el firewall.

Esto puede ser extremadamente engorroso ya que nos obliga a recurrir a un firewall deficiente e improvisado. Lo que es más, causa incesantes dolores de cabeza sobre las cosas que se fueron modificando sobre la marcha y que deben de ser realizadas nuevamente.

3.2.1. Actualizaciones

De la misma manera, al momento de hacer una actualización al firewall, la versión corriente debe de estar correctamente respaldada y algún tipo de medida de contingencia. Esto nos permitirá actuar de manera rápida y eficiente en caso que alguna modificación no resultare como esperamos.

Es de vital importancia considerar dos casos completamente separados. El primero sería cuando uno está trabajando directamente sobre la PC en la que funciona el firewall, **cuando uno está frente al hardware** en cuestión. En dichos casos, la solución a este tipo de problemas puede ser tan sencillo como eliminar todas las reglas y poner las políticas a DROP hasta tanto el firewall esté nuevamente en funcionamiento.

El segundo caso es cuando uno está trabajando de forma remota. En estos casos uno no puede hacer lo mismo que antes se mencionó por la simple razón de que ello bloquearía permanentemente cualquier tipo de conexión desde el exterior... incluso la que estaba utilizando para ingresar al servidor.

3.3. Frontends

Otra solución a los problemas de `iptables` en cuanto a su complejidad son los frontends. Estos consisten en simplemente una interfaz gráfica para un entorno X que ayudará a la administración del firewall. Algunos de ellos son:

3.3.1. Firestarter

Sitio oficial: <http://www.fs-security.com/>

El firewall manager de GNOME. Lo he utilizado y es extremadamente sencillo, inclusive para manejar una PC que funcione como puerta de enlace. Se encarga de cargar todos los módulos necesarios y brinda excelentes opciones de personalización del firewall.

Utiliza las reglas de logeo para mostrar en tiempo real las conexiones existentes, los paquetes rechazados y mucha otra información útil. Permite prender y apagar el firewall o modificarlo *en caliente* quedando como residente para un mejor control en tiempo real. Muy recomendado para usuarios novatos, o aquellos que no tienen muchas ganas de ponerse a aprender sintáxis.

3.3.2. KNetFilter

Sitio oficial: <http://expansa.sns.it/knetfilter/>

Es la versión de KDE de Firestarter. No la he utilizado y no he podido encontrar referencia alguna que indique su funcionamiento en kernels posteriores a 2.4. Se veía bien en los screenshots sin embargo.

3.3.3. Firewall Builder

Sitio oficial: <http://www.fwbuilder.org/>

Uno de los GUIs más poderosos que he visto. De los más completos... y complicados. Necesita de sendos conocimientos sobre redes y lo que se desea hacer con el firewall.

Está muy bien documentado y tiene extensas ayudas. Todo en inglés sin embargo y no es recomendable para alguien que quiere hacer algo sencillo y rápido. Para un usuario avanzado con un gusto por interfases gráficas, es lo mejor que he visto.

3.4. Wrappers

Así como existen interfaces gráficas para implementar `iptables`, existen también sistemas completos que se encargan de proveer de una nueva forma de configurar el firewall (y alguna que otra cosa también). Estos sistemas se encargan de parsear estas configuraciones y traducirlas a las reglas de `iptables` necesarias.

Muchas veces proveen de más utilidades al integrarse también con otras aplicaciones. Proveen, además, de un estándar en cuanto al almacenamiento de configuraciones del firewall y no lo dejan librado al gusto del administrador. Sin embargo, sus sintáxis de configuración pueden ser tanto o más complicadas que las de `iptables`; pero eso queda a criterio de cada uno.

3.4.1. Shorewall

Sitio oficial: <http://www.shorewall.net/>

Uno de los más populares y mejor documentados. Con una sintáxis de configuración sencilla pero con parámetros *sospechosamente similares* a aquellos que podemos utilizar en `iptables`.

Se basa en la configuración de **zonas** y sus políticas de seguridad. Configuramos qué se permite y que no para cada zona y las conexiones que corresponden a cada una. Permite integración con `tc` para hacer traffic shaping desde el mismo archivo de configuración.

Tiene la gran ventaja de que una extensa porción de su documentación también puede encontrarse en español.

3.4.2. Ferm

Sitio oficial: <http://ferm.foo-projects.org/>

Un proyecto cuyo nombre proviene del acrónimo de *Para la Simple Creación de Reglas* (del inglés "For Easy Rule Making"). Al igual que Shorewall, sus parámetros de configuración son *sospechosamente similares* a los que se pueden encontrar en `iptables`; pero su configuración se realiza mediante una sintaxis muy parecida a código C.

Permite utilizar nombres de protocolos en lugar de los números de puerto por éstos utilizados. Además de poder agrupar reglas, utilizar variables, funciones, bloques, etc para un mejor control.

3.4.3. gShield

Sitio oficial: <http://muse.linuxmafia.org/gshield/>

Es un potente sistema que toma scripts de shell y los transforma en reglas de `iptables`. Excelentemente documentado y muy potente. Como si fuese poco, cuenta con una completa interfase gráfica: **gShieldConf**². Aprovechando la potencia del lenguaje bash permite construir configuraciones complicadas con

²Sitio oficial: <http://members.shaw.ca/vhodges/gshieldconf.html>

sencillos scripts para quienes tengan el conocimiento necesario o la utilización de la interfase gráfica para quienes quieran algo más sencillo.

3.5. Otros

3.5.1. Auto FireWall

Sitio oficial: <http://baruch.ev-en.org/proj/autofw/autofw.html>

Este es una aplicación completa que genera reglas de ruteo y un firewall muy decente para el usuario promedio. Basta con correr la aplicación para que ésta auto-detecte el estado del sistema y se configure a sí misma.

No es la mejor solución, pero una de las mejores que existen que funcionan bien así como vienen. Especialmente recomendado para usuarios nóveles que quieren conectarse a internet y olvidarse del asunto. Eso sí, deben recordar ejecutarla manualmente al conectarse (o antes).

3.5.2. Floppy FireWall

Sitio Oficial: <http://www.zelow.no/floppyfw/>

Es una distribución completa de Linux diseñada especialmente para correr sólo desde un disquette y administrar un servidor completo. La configuración es muy simple e intuitiva modificando archivos de texto y puede ser realizada tanto desde Windows como de Linux.

Yo la estoy utilizando con excelentes resultados en una PC sin disco rígido, poco procesador y poca memoria (Pentium 586 con 64 Mb) para hacer de puerta de enlace a 6 PCs. No ha fallado casi nunca; pero depende en gran medida de la vida útil del disquette.

Al estar orientada a servidores provee de archivos de configuración para determinar cómo se realizará la conexión a internet (PPP o PPPoE), si las IPs serán estáticas o se utilizará DHCP dentro de la red, si se permitirá algún tipo de conexión hacia dentro de la red - como para permitir la conexión remota a alguna de ellas - y muchas otras cosas comentadas en exceso dentro de los mismos archivos. En constante desarrollo y muy bien documentado.

Capítulo 4

Enlaces

ADMLogger: una herramienta que parsea logs de iptables <http://aaron.marasco.com/linux.html>

Auto FireWall <http://baruch.ev-en.org/proj/autofw/autofw.html>

Comparación entre distintos wrappers de iptables <http://www.securityfocus.com/infocus/1410>

Ferm <http://ferm.foo-projects.org/>

FireStarter <http://www.fs-security.com/>

Firewall Builder <http://www.fwbuilder.org/>

Floppy FireWall <http://www.zelow.no/floppyfw/>

gShield <http://muse.linuxmafia.org/gshield/>

gShieldConf <http://members.shaw.ca/vhodges/gshieldconf.html>

kNetfilter <http://expansa.sns.it/knetfilter/>

Módulos raros y no tanto de iptables http://www.tummy.com/journals/entries/jafo_20050717_16453

On-line Firewall Generator <http://easyfwgen.morizot.net/gen/>

Shorewall <http://www.shorewall.net/>