Linux netfilter Hacking COMO

Rusty Russell, lista de correo `netfilter@lists.samba.org`
Traducido por Gabriel Rodríguez Alberich *chewie@asef.us.es*
v1.0.1 sábado 1 de julio 1 18:24:41 EST 2000, traducción del 7 de noviembre de 2000

Este documento describe la arquitectura de netfilter en Linux, cómo *hackearla*, y algunos de los sistemas más importantes que se asientan en ella, como el filtrado de paquetes, seguimiento de conexiones (connection tracking) y Traducción de Direcciones de Red (Network Address Translation).

Contents

1 Introducción

Hola.

Este documento es un viaje; algunas partes se recorren con comodidad, y en otras zonas se encontrará casi en la soledad. El mejor consejo que puedo darle es que coja una gran taza de café o chocolate caliente, se siente en un cómodo asiento, y absorba los contenidos antes de aventurarse en el a veces peligroso mundo del *hacking* de redes.

Para un mayor entendimiento del uso de la infraestructura existente sobre el sistema netfilter, recomiendo la lectura del Packet Filtering HOWTO y el NAT HOWTO (disponibles en castellano). Para más información sobre la programación del kernel, sugiero el *Rusty's Unreliable Guide to Kernel Hacking* y el *Rusty's Unreliable Guide to Kernel Locking*.

1.1 ¿Qué es «netfilter»?

netfilter es un sistema para manipular paquetes que se encuentra fuera del interfaz normal de sockets de Berkeley. Consta de cuatro partes. Primero, cada protocolo define "ganchos" (IPv4 define 5), que son puntos bien definidos en el recorrido de un paquete a través de la pila de ese protocolo. En cada uno de estos puntos, el protocolo llamará al sistema netfilter con el paquete y el número del gancho.

En segundo lugar, hay partes del kernel que pueden registrarse para escuchar los diferentes ganchos de cada protocolo. Entonces, cuando se le pasa un paquete al sistema netfilter, éste comprueba si alguien se ha registrado para ese protocolo y ese gancho; si es así, cada uno de los que se ha registrado tiene la posibilidad de examinar (y quizá alterar) el paquete en cuestión, y luego rechazarlo, permitir que pase, o pedirle a netfilter que ponga el paquete en una cola para el espacio de usuario.

En tercer lugar, los paquetes que han sido colocados en la cola son recogidos (por el controlador ip_queue_driver) para enviarlos al espacio de usuario; estos paquetes se manejan asincrónicamente.

La parte final está constituida por comentarios útiles en el código y por la documentación. Esto juega un papel decisivo en cualquier proyecto experimental. El lema de netfilter es (robado descaradamente a Cort Dougan):

```
‘‘Entonces... ¿en qué es mejor esto que KDE?’’
```

(Este lema casi dice ‘Azótame, pégame, hazme usar ipchains’).

Además de este sistema crudo, se han escrito varios módulos que proporcionan una funcionalidad similar a la que tenían los kernels anteriores (pre-netfilter). En particular, un sistema NAT extensible, y un sistema de filtrado de paquetes extensible (iptables).

1.2 ¿Qué hay de malo en lo que teníamos con el 2.0 y el 2.2?

1. No hay establecida una infraestructura para pasar paquetes al espacio de usuario:
   - La programación del kernel se hace difícil
   - La programación del kernel tiene que hacerse en C/C++
   - Las políticas de filtrado dinámicas no pertenecen al kernel
   - 2.2 introdujo una manera de pasar paquetes al espacio de usuario mediante netlink, pero la reinyección de paquetes es lenta, y sujeta a comprobaciones de ‘sanidad’. Por ejemplo, reinyectar un paquete que afirma venir de una interfaz existente no es posible.
2. Montar un proxy transparente es una chapuza:
   - Tenemos que observar **todos** los paquetes para ver si hay un socket ligado a esa dirección
   - root puede ligar (bind :-) a direcciones externas
   - No se pueden redirigir paquetes generados localmente
   - REDIRECT no maneja respuestas UDP: redirigir paquetes UDP al puerto 1153 no funciona porque a algunos clientes no les gustan las respuestas que vienen de otro puerto que no sea el 53.
   - REDIRECT no se coordina con la asignación de puertos tcp/udp: un usuario podría conseguir un puerto (shadowed) por una regla REDIRECT. (a user may get a port shadowed by a REDIRECT rule)
   - Ha sido interrumpido al menos dos veces desde la serie 2.1. Has been broken at least twice during 2.1 series.

- El código es extremadamente molesto. Considere el número de apariciones de #ifdef CONFIG_IP_TRANSPARENT_PROXY en el 2.2.1: 34 apariciones en 11 ficheros. Compare esto con CONFIG_IP_FIREWALL, que aparece 10 veces en 5 ficheros.

3. No es posible crear reglas de filtrado de paquetes independientes de las direcciones de interfaz:
    - Se deben conocer las direcciones de interfaz locales para distinguir los paquetes generados localmente o los que terminan localmente de los paquetes redirigidos.
    - Incluso esto es insuficiente en casos de redirección o enmascaramiento.
    - La cadena forward (redirección) sólo tiene información de la interfaz de salida, lo que significa que usted tiene que figurarse de dónde proviene el paquete utilizando sus conocimientos sobre la topología de la red.
4. El enmascaramiento está encima del filtrado de paquetes:Las interacciones entre el filtrado de paquetes y el enmascaramiento hacen que sea complejo manejar un cortafuegos:
    - En el filtrado de entrada (input), los paquetes de respuesta parecen ir destinados a la propia máquina
    - En el filtrado de redirección (forward), los paquetes desenmascarados no se pueden ver
    - En el filtrado de salida (output), los paquetes parecen venir de la máquina local
5. La manipulación del TOS, la redirección, el ICMP unreachable y el marcado (mark) (que pueden proporcionan redirección de puertos, enrutado y QoS) también están encima del código de filtrado de paquetes.
6. El código de ipchains no es ni modular ni extensible (p.ej. filtrado de direcciones MAC, opciones de filtrado, etc).
7. La falta de una infraestructura suficiente ha llevado a la profusión de distintas técnicas:
    - Enmascaramiento, además de módulos por cada protocolo
    - NAT estático veloz mediante código de enrutamiento (no tiene manejo por protocolo)
    - Redirección de puertos (port forwarding), redirección, auto redirección (auto forwarding)
    - Los Proyectos NAT y Servidor Virtual para Linux.
8. Incompatibilidad entre CONFIG_NET_FASTROUTE y el filtrado de paquetes:
    - Los paquetes redirigidos tienen que atravesar tres cadenas de todos modos
    - No hay manera de saber si estas cadenas pueden evitarse
9. No es posible la inspección de los paquetes rechazados a causa de una protección de enrutado (p.ej. Source Address Verification).
10. No hay manera de leer automáticamente los contadores de las reglas de filtrado de paquetes.
11. CONFIG_IP_ALWAYS_DEFRAG es una opción en tiempo de compilación, cosa que le complica la vida a las distribuciones que quieren hacer un kernel de propósitos generales.

1.3 ¿Quién eres?

Soy el único lo suficientemente tonto para hacer esto. Como coautor de ipchains y como actual mantenedor del Cortafuegos IP del Kernel de Linux, puedo ver muchos de los problemas que la gente tiene con el sistema actual, además de saber lo que tratan de hacer.

1.4 ¿Por qué se cuelga?

¡Bueno! ¡Debería haberlo visto la semana **pasada**!

Porque no soy un gran programador como todos desearíamos, y ciertamente no he comprobado todas las situaciones, por falta de tiempo, equipo y/o inspiración. Sí que tengo una batería de pruebas, a la que le animo que contribuya.

2 ¿Dónde puedo conseguir el último?

Hay un servidor CVS en samba.org que contiene los últimos HOWTOs, las herramientas de usuario y la batería de pruebas. Para el que quiera una navegación rápida, puede usar la *Interfaz Web* `http://www.samba.org/cgi-bin/cvsweb/netfilter/`.

Para conseguir los últimos fuentes, puede hacer lo siguiente:

1. Entre en el servidor CVS de SAMBA anónimamente:

   ```
   cvs -d :pserver:cvs@cvs.samba.org:/cvsroot login
   ```

2. Cuando pida la clave, escriba 'cvs'.

3. Mire el código usando:

   ```
   cvs -d :pserver:cvs@cvs.samba.org:/cvsroot co netfilter
   ```

4. Para actualizarse a la última versión escriba

   ```
   cvs update -d -P
   ```

3 La arquitectura de Netfilter

Netfilter es meramente una serie de ganchos en varios puntos de la pila de un protocolo (a estas alturas IPv4, IPv6 y DECnet). El diagrama de recorrido (idealizado) de IPv4 se parece a lo siguiente:

```
Un paquete atravesando el sistema Netfilter:

   --->[1]--->[ROUTE]--->[3]--->[4]--->
                 |            ^
                 |            |
                 |         [ROUTE]
                 v            |
                [2]          [5]
                 |            ^
                 |            |
                 v            |
```

Los paquetes entran desde la izquierda: tras haber pasado las sencillas comprobaciones de sanidad (es decir, no está truncado, la suma de control IP es correcta y no es una recepción promiscua), son pasados al gancho NF_IP_PRE_ROUTING [1] del sistema netfilter.

Luego entran en el código de enrutamiento, que decide si el paquete está destinado a otra interfaz o a un proceso local. El código de enrutamiento puede rechazar paquetes que no se pueden enrutar.

Si está destinado a la propia máquina, se llama de nuevo al sistema netfilter para el gancho NF_IP_LOCAL_IN [2], antes de ser enviado al proceso (si hay alguno).

Si, en cambio, está destinado hacia otra interfaz, se llama al sistema netfilter para el gancho NF_IP_FORWARD [3].

Luego el paquete pasa por un gancho final, el gancho NF_IP_POST_ROUTING [4], antes de ser enviado de nuevo al cable.

Para los paquetes creados localmente, se llama al gancho NF_IP_LOCAL_OUT [5]. Aquí puede ver que el enrutamiento ocurre después haber llamado a este gancho: de hecho, se llama primero al código de enrutamiento (para averiguar la dirección IP y algunas opciones IP), y luego se le llama otra vez si el paquete ha sido alterado.

3.1 La base de Netfilter

Ahora que tenemos un ejemplo de netfilter en IPv4, ya puede ver cuándo se activa cada gancho. Ésta es la esencia de netfilter.

Uno o varios módulos del kernel pueden registrarse para escuchar en alguno de estos ganchos. Luego, cuando se llama al gancho de netfilter desde el código de red, los módulos registrados en ese punto tienen libertad para manipular el paquete. Un módulo puede decirle a netfilter que haga una de estas cinco cosas:

1. NF_ACCEPT: continúa el recorrido normalmente.
2. NF_DROP: rechaza el paquete; no continúes el recorrido.
3. NF_STOLEN: me hago cargo del paquete; no continúes el recorrido.
4. NF_QUEUE: pon el paquete en una cola (normalmente para tratar con el espacio de usuario).
5. NF_REPEAT: llama de nuevo a este gancho.

Las otras partes de netfilter (manejo de paquetes en la cola, comentarios útiles) se cubrirán luego en la sección del kernel.

Sobre esta base, podemos construir manipulaciones de paquetes bastante complejas, como se muestra en las dos próximas secciones.

3.2 Selección de paquetes: IP Tables

Se ha construido una sistema de selección de paquetes llamado IP Tables sobre el sistema netfilter. Es un descendiente directo de ipchains (que vino de ipfwadm, que vino del ipfw IIRC de BSD), con extensibilidad. Los módulos del kernel pueden registrar una tabla nueva, e indicarle a un paquete que atraviese una tabla dada. Este método de selección de paquetes se utiliza para el filtrado de paquetes (la tabla 'filter'), para la Traducción de Direcciones de Red (la tabla 'nat') y para la manipulación general de paquetes antes del enrutamiento (la tabla 'mangle').

3.2.1 Filtrado de Paquetes

Esta tabla, 'filter', nunca altera los paquetes: sólo los filtra.

Una de las ventajas de iptables sobre ipchains es que es pequeño y rápido, y se engancha a netfilter en los puntos NF_IP_LOCAL_IN, NF_IP_FORWARD y NF_IP_LOCAL_OUT. Esto significa que para cualquier paquete dado, existe un (y sólo un) posible lugar donde pueda ser filtrado. Esto hace las cosas mucho más sencillas. Además, el hecho de que el sistema netfilter proporcione las dos interfaces de entrada (input) y salida (output) para el gancho NF_IP_FORWARD significa que hay bastantes tipos de filtrado que se simplifican mucho.

Nota: He portado las porciones del kernel de ipchains e ipfwadm en forma de módulos sobre netfilter, permitiendo el uso de las viejas herramientas de usuario ipfwadm y ipchains sin que se requiera una actualización.

3.2.2 NAT

Esto es el reino de la tabla 'nat', que se alimenta de paquetes mediante tres ganchos de netfilter: para los paquetes no locales, los ganchos NF_IP_PRE_ROUTING y NF_IP_POST_ROUTING son perfectos para las alteraciones en el destino y el origen, respectivamente. Para alterar el destino de los paquetes locales, se utiliza el gancho NF_IP_LOCAL_OUT.

Esta tabla es ligeramente distinta a la tabla 'filter' en el sentido de que sólo el primer paquete de una conexión nueva atravesará la tabla: el resultado de este recorrido se aplica luego a todos los paquetes futuros de la misma conexión.

Enmascaramiento, redireccionamiento de puertos y proxys transparentes Divido NAT en NAT de Origen (en el que se altera el origen del primer paquete), y NAT de Destino (en el que se altera el destino del primer paquete).

El enmascaramiento es una forma especial de NAT de Origen; el redireccionamiento de puertos y los proxys transparentes son formas especiales de NAT de Destino. Ahora todas se hacen utilizando el sistema NAT, en vez de ser entidades independientes.

3.3 Seguimiento de conexiones

El seguimiento de conexiones es fundamental para NAT, pero está implementado en un módulo aparte; esto permite una extensión del filtrado de paquetes para utilizar de manera limpia y sencilla el seguimiento de conexiones (el módulo 'state').

3.4 Otros añadidos

La nueva flexibilidad nos da la oportunidad de hacer cosas realmente chulas, y permitir a la gente que escriba mejoras o complete recambios, que pueden mezclarse y combinarse.

4 Información para programadores

Le voy a contar un secreto: mi hámster hizo todo el código. Yo sólo era una vía, una 'fachada' si quiere, en el gran plan de mi mascota. Por tanto, no me culpe a mí si existen fallos. Culpe al lindo peludo.

4.1 Comprendiendo ip_tables

iptables proporciona simplemente un vector de reglas en memoria (de ahí el nombre 'iptables'), e información tal como por dónde deberían comenzar el recorrido los paquetes de cada gancho. Después de que una tabla es registrada, el espacio de usuario puede leer y reemplazar sus contenidos utilizando getsockopt() y setsockopt().

iptables no se registra en ningún gancho de netfilter: cuenta con que otros módulos lo hagan y le administren los paquetes apropiados.

4.1.1 Estructuras de datos de ip_tables

Por conveniencia, se utiliza la misma estructura de datos para representar un regla en el espacio de usuario y dentro del kernel, aunque algunos campos sólo se utilizan dentro del kernel.

Cada regla consiste en las partes siguientes:

1. Una estructura 'struct ipt_entry'.
2. Cero o más estructuras 'struct ipt_entry_match', cada una con una cantidad variable de datos (0 o más bytes) dentro de ella.
3. Una estructura 'struct ipt_entry_target', con una cantidad variable de datos (0 o más bytes) dentro de ella.

La naturaleza variable de las reglas proporciona una enorme flexibilidad a las extensiones, como veremos, especialmente porque cada concordancia (match) u objetivo (target) puede llevar una cantidad de datos arbitraria. Eso, sin embargo, acarrea unas cuantas trampas: tenemos que tener cuidado con la alineación. Esto lo hacemos asegurándonos de que las estructuras 'ipt_entry', 'ipt_entry_match' e 'ipt_entry_target' tienen el tamaño conveniente, y de que todos los datos son redondeados a la máxima alineación de la máquina, utilizando la macro IPT_ALIGN().

La estructura 'struct ipt_entry' tiene los siguientes campos:

1. Una parte 'struct ipt_ip', que contiene las especificaciones para la cabecera IP que tiene que concordar.
2. Un campo de bits 'nf_cache' que muestra qué partes del paquete ha examinado esta regla.
3. Un campo 'target_offset' que indica el *offset* del principio de esta regla donde comienza la estructura ipt_entry_target. Esto siempre debe alinearse correctamente (con la macro IPT_ALIGN).
4. Un campo 'next_offset' que indica el tamaño total de esta regla, incluyendo las concordancias y el objetivo. Esto siempre debe alinearse correctamente con la macro IPT_ALIGN.
5. Un campo 'comefrom', utilizado por el kernel para seguir el recorrido del paquete.
6. Un campo 'struct ipt_counters' que contiene los contadores de paquetes y de bytes que han concordado con esta regla.

Las estructuras 'struct ipt_entry_match' y 'struct ipt_entry_target' son muy similares, en el sentido de que contienen un campo de longitud total, alineado con IPT_ALIGN, ('match_size' y 'target_size' respectivamente) y una unión que contiene el nombre de la concordancia u objetivo (para el espacio de usuario), y un puntero (para el kernel).

Debido a la naturaleza engañosa de la estructura de datos de las reglas, se proporcionan algunas rutinas de ayuda:

ipt_get_target()
: Esta función (inline) devuelve un puntero al objetivo de una regla.

IPT_MATCH_ITERATE()
: Esta macro llama a la función especificada cada vez que se produce una concordancia en la regla en cuestión. El primer argumento de la función es la estructura 'struct ipt_match_entry', y el resto de argumentos (si los hay) son los proporcionados por la macro IPT_MATCH_ITERATE().

IPT_ENTRY_ITERATE()

Esta función recibe un puntero a una entrada, el tamaño total de la tabla de entradas, y una función a la que llamar. El primer argumento de la función es la estructura 'struct ipt_entry', y el resto de argumentos (si los hay) son los proporcionados por la macro IPT_ENTRY_ITERATE().

4.1.2 ip_tables desde el espacio de usuario

El espacio de usuario dispone de cuatro operaciones: puede leer la tabla actual, leer la información (posiciones de los ganchos y tamaño de la tabla), reemplazar la tabla (y obtener los contadores antiguos), y añadir nuevos contadores.

Esto permite simular cualquier operación atómica desde el espacio de usuario: se hace mediante la biblioteca libiptc, que proporciona una cómoda semántica "añadir/borrar/reemplazar" para los programas.

Ya que estas tablas son trasladadas al espacio del kernel, la alineación se convierte en un asunto importante en máquinas que tienen reglas de tipo distintas para el espacio de usuario y el espacio del kernel (p.ej. Sparc64 con un (userland) de 32 bits). Estos casos se resuelven cancelando la definición de IPT_ALIGN para estas plataformas en 'libiptc.h'.

4.1.3 Recorrido y uso de ip_tables

El kernel comienza el recorrido en la posición indicada por el gancho en particular. Esa regla se examina, y si los elementos de 'struct ipt_ip' concuerdan, se comprueba cada 'struct ipt_entry_match' en orden (se llama a la función de concordancia asociada con esa concordancia). Si la función de concordancia devuelve 0, la iteración se deteiene en esa regla. Si establece el valor de 'hotdrop' a 1, el paquete será rechazado inmediatamente (esto se usa para algunos paquetes sospechosos, como en la función de concordancia tcp).

Si la iteración continúa hasta el final, los contadores se incrementan y se examina la estructura 'struct ipt_entry_target': si es un objetivo estándar, se lee el campo 'veredict' (negativo significa el veredicto de un paquete, positivo significa un offset al que saltar). Si la respuesta es positiva y el offset no es el de la regla siguiente, se establece la variable 'back', y el valor anterior de 'back' se coloca en el campo 'comefrom' de esa regla.

Para objetivos no estándar, se llama a la función de objetivo: ésta devuelve un veredicto (los objetivos no estándar no pueden saltar, ya que esto interrumpiría el código estático de loop-detection). El veredicto puede ser IPT_CONTINUE, para continuar en la siguiente regla.

4.2 Extendiendo iptables

Como soy vago, `iptables` es muy extensible. Esto es básicamente una excusa para encajarle el trabajo a otras personas, que es de lo que se trata el open source (cf. el Software Libre, como diría RMS, va sobre la libertad, y yo me he basado en sus palabras para escribir esto).

Extender `iptables` implica potencialmente dos partes: extender el kernel escribiendo un nuevo módulo, y posiblemente extender el programa de espacio de usuario `iptables`, escribiendo una nueva biblioteca compartida.

4.2.1 El kernel

Escribir un módulo para el kernel es bastante sencillo, como puede ver a partir de los ejemplos. Una cosa de la que hay que estar avisado es que su código debe ser reentrante: puede haber un paquete entrando desde el espacio de usuario, mientras que otro llega a través de una interrupción. De hecho, con SMP puede haber un paquete por interrupción y por CPU en las versiones 2.3.4 y superiores.

Las funciones que necesita conocer son las siguientes:

init_module()
: Ésta es el punto de entrada del módulo. Devuelve un número de error negativo, o 0 si se registra con éxito en netfilter.

cleanup_module()
: Ésta es el punto de salida del módulo; lo desregistra de netfilter.

ipt_register_match()
: Ésta se utiliza para registrar un nuevo tipo de concordancia. Se le pasa una estructura 'struct ipt_match', que normalmente se declara como una variable estática (file-scope).

ipt_register_target()
: Ésta se utiliza para registrar un nuevo tipo de objetivo. Se le pasa una estructura 'struct ipt_target', que normalmente se declara como una variable estática (file-scope).

ipt_unregister_target()
: Utilizada para desregistrar su objetivo.

ipt_unregister_match()
: Utilizada para desregistrar su concordancia.

Una advertencia sobre hacer cosas delicadas (como proporcionar contadores) en el espacio extra de su nueva concordancia u objetivo. En las máquinas SMP, toda la tabla se duplica utilizando memcpy en cada CPU: si realmente quiere preservar información central, debería echarle un vistazo al método utilizado en la concordancia 'limit'.

Nuevas funciones de concordancia Las nuevas funciones de concordancia se escriben normalmente como un módulo independiente. Es posible tener extensibilidad en estos módulos, aunque normalmente no es necesario. Una manera sería utilizar la función 'nf_register_sockopt' del sistema netfilter para permitir a los usuarios hablar directamente al módulo. Otra manera sería exportar los símbolos para que otros módulos se registren, de la misma manera que lo hacen netfilter e iptables.

El corazón de su nueva función de concordancia es la estructura 'struct ipr_match' que le pasa a 'ipt_register_match()'. Esta estructura tiene los siguientes campos:

list
: En este campo se puede poner lo que sea, digamos '{ NULL, NULL }'.

name
: Este campo es el nombre de la función de concordancia, referido por el espacio de usuario. El nombre deber ser igual al nombre del módulo (es decir, si el nombre es "mac", el módulo debe ser "ipt_mac.o") para que funcione la auto-carga.

match
: Este campo es un puntero a una función de concordancia, que recibe el skb, los punteros a los dispositivos *in* y *out* (uno de los cuales podría ser NULL, dependiendo del gancho), un puntero a los datos de concordancia de la regla que concordó, el tamaño de esa regla, el offset IP (si es distinto de cero

se refiere a un fragmento no inicial), un puntero a la cabecera del protocolo (es decir, justo después de la cabecera IP), la longitud de los datos (es decir, la longitud del paquete menos la longitud de la cabecera IP), y finalmente un puntero a una variable 'hotdrop'. Devuelve algo distinto de cero si el paquete concuerda, y puede poner 'hotdrop' a 1 si devuelve 0, para indicar que el paquete debe rechazarse inmediantemente.

checkentry

Este campo es un puntero a una función que comprueba las especificaciones de una regla; si devuelve 0, no se aceptará la regla del usuario. Por ejemplo, el tipo de concordancia "tcp" sólo aceptará paquetes tcp, y por tanto, si la estructura 'struct ipt_ip' de la regla no especifica que el protocolo debe ser tcp, se devuelve cero. El argumento tablename permite a su concordancia controlar en qué tablas puede utilzarse, y 'hook_mask' es una máscara de bits de ganchos desde los que se puede llamar a esta regla: si su concordancia no tiene sentido desde algunos ganchos netfilter, puede evitarlo aquí.

destroy

Este campo es un puntero a una función que es llamada cuando se borra una entrada que utiliza esta concordancia. Esto le permite reservar recursos dinámicamente en el checkentry y limpiarlos aquí.

me

A este campo se le asigna '&__this_module', que da un puntero a su módulo. Hace que el contador de uso suba y baje al crearse y destruirse reglas de ese tipo. Esto impide que un usuario pueda eliminar el módulo (y por tanto llamar a cleanup_module()) si una regla se refiere a él.

Nuevos objetivos Los objetivos nuevos se escriben normalmente como un módulo independiente. Las discusiones de la sección de arriba sobre las 'Nuevas funciones de concordancia' se aplican igualmente aquí.

El núcleo de su nuevo objetivo es la estructura 'struct ipt_target' que le pasa a 'ipt_register_target()'. Esta estructura consta de los siguientes campos:

list

En este campo se puede poner lo que sea, digamos '{ NULL, NULL }'.

name

Este campo es el nombre de la función de objetivo, referido por el espacio de usuario. El nombre debe concordar con el nombre del módulo (es decir, si el nombre es "REJECT", el módulo debe ser "ipt_REJECT.o") para que funcione la auto-carga.

target

Esto es un puntero a la función de objetivo, que recibe el skbuff, los punteros a los dispositivos *in* y *out* (de los que cualquiera puede ser NULL), un puntero a los datos del objetivo, el tamaño de los datos del objetivo, y la posición de la regla en la tabla. La función de objetivo devuelve una posición absoluta no negativa hacia la que saltar, o un veredicto negativo (que es el veredicto negado menos uno).

checkentry

Este campo es un puntero a una función que comprueba las especificaciones de una regla; si devuelve 0, entonces no se aceptará la regla del usuario.

destroy

Este campo es un puntero a una función que es llamada cuando se borra una entrada que utiliza este objetivo. Esto le permite reservar recursos dinámicamente en el checkentry y limpiarlos aquí.

me

A este campo se le asigna '&__this_module', que da un puntero a su módulo. Hace que el contador de uso suba y baje al crearse reglas con este objetivo. Esto impide que un usuario pueda eliminar el módulo (y por tanto llamar a cleanup_module()) si una regla se refiere a él.

Nuevas tablas Puede crear una tabla nueva para sus propósitos específicos si lo desea. Para hacerlo, llame a 'ipt_register_table()' con una estructura 'struct ipt_table', que tiene los siguientes campos:

list

En este campo se puede poner lo que sea, digamos '{ NULL, NULL }'.

name

Este campo es el nombre de la función de tabla, referido por el espacio de usuario. El nombre debe concordar con el nombre del módulo (es decir, si el nombre es "nat", el módulo debe ser "iptable_nat.o") para que funcione la auto-carga.

table

Esto es una estructura 'struct ipt_replace' completamente rellenada, utilizada por el espacio de usuario para reemplazar una tabla. Al puntero 'counters' debe asignársele NULL. Esta estructura de datos puede declararse como '__initdata' para que sea descartada después del arranque.

valid_hooks

Esto es una máscara de bits de los ganchos IPv4 de netfilter que introducirá en la tabla: se utiliza para comprobar que esos puntos de entrada son válidos, y para calcular los posibles ganchos para las funciones 'checkentry()' de ipt_match e ipt_target.

lock

Esto es el read-write spinlock para toda la tabla; inicialícela a RW_LOCK_UNLOCKED. This is the read-write spinlock for the entire table; initialize it to RW_LOCK_UNLOCKED.

private

Este campo es utilizado internamente por el código de ip_tables.

4.2.2 Herramienta del espacio de usuario

Ahora que ya ha escrito un bonito y reluciente módulo del kernel, quizá quiera controlar sus opciones desde el espacio de usuario. En vez de tener una versión independiente de `iptables` para cada extensión, he utilizado la última tecnología de los años 90: los furbies. Perdón, quería decir bibliotecas compartidas.

Generalmente, las nuevas tablas no requieren ninguna extensión de `iptables`: el usuario sólo tiene que utilizar la opción '-t' para hacer que use la nueva tabla.

La biblioteca compartida debe tener una función '_init()', a la que se llamará automáticamente durante la carga: el equivalente moral de la función del módulo del kernel 'init_module()'. Ésta llamará a 'register_match()' o a 'register_target()', dependiendo de si su biblioteca compartida proporciona una nueva concordancia o un nuevo objetivo.

Sólo necesita proporcionar una biblioteca compartida si quiere inicializar parte de la estructura o proporcionar opciones adicionales. Por ejemplo, el objetivo 'REJECT' no requiere nada de esto, por lo que no hay ninguna biblioteca compartida.

Hay funciones útiles definidas en la cabecera 'iptables.h', especialmente:

check_inverse()

comprueba si un argumento es realmente un '!', y si es así, activa el flag 'invert' si no estaba ya activado. Si devuelve verdadero, hay que incrementar optind, como en los ejemplos.

string_to_number()

convierte una cadena en un número dentro del rango dado, devolviendo -1 si está malformada o fuera de rango.

exit_error()

debe llamarse si se encuentra un error. Normalmente, el primer argumento es 'PARAMETER_PROBLEM', que significa que el usuario no usó correctamente la línea de comandos.

Nuevas funciones de concordancia Su función de biblioteca compartida _init() le pasa a 'register_match()' un puntero a una estructura estática 'struct iptables_match' que tiene los siguientes campos:

next

Este puntero se utiliza para construir una lista enlazada de concordancias (como la utiliza para listar las reglas). Inicialmente debe asignársele el valor NULL.

name

El nombre de la función de concordancia. Debe concordar con el nombre de la librería (por ejemplo "tcp" para 'libipt_tcp.so').

version

Normalmente se le asigna la macro NETFILTER_VERSION: esto se hace para asegurar que el binario `iptables` no utiliza por error una biblioteca compartida equivocada.

size

El tamaño de los datos de concordancia para esta concordancia; debe utilizar la macro IPT_ALIGN() para asegurarse de que está alineado correctamente.

userspacesize

En algunas concordancias, el kernel cambia internamente algunos campos (el objetivo 'limit' es un caso). Esto significa que un simple 'memcmp()' es insuficiente para comparar dos reglas (algo requerido para la funcionalidad de borrar reglas concordantes). Si éste es el caso, coloque todos los campos que no cambian al principio de la estructura, y ponga aquí el tamaño de estos campos. De todas formas, esto será casi siempre idéntico al campo 'size'.

help

Una función que imprime la sintaxis de la opción.

init

Esta función se puede utilizar para inicializar el espacio extra (si lo hay) de la estructura ip_entry_match, y establecer algún bit de nfcache; si está examinando algo no expresable mediante los contenidos de 'linux/include/netfilter_ipv4.h', entonces haga simplemente un OR con el bit NFC_UNKNOWN. Se llamará a la función después de 'parse()'.

parse

Esta función es llamada cuando se observa una opción desconocida en la línea de comandos: devuelve distinto de cero si la opción era realmente para su bilbioteca. 'invert' es verdadero si ya se ha observado un '!'. El puntero 'flags' es de uso exclusivo para su biblioteca de concordancia, y normalmente se utiliza para guardar una máscara de bits de opciones que se han especificado. Asegúrese de ajustar el campo nfcache. Si es necesario, puede extender el tamaño de la estructura 'ipt_entry_match' haciendo una nueva reserva de memoria, pero entonces debe asegurarse de que el tamaño se pasa a través de la macro IPT_ALIGN()

final_check

Esta función es llamada después de que se haya analizado sintácticamente la línea de comandos, y se le pasa el entero 'flags' reservado para su biblioteca. Esto le permite comprobar si no se ha especificado alguna de las opciones obligatorias, por ejemplo: llamar a 'exit_error()' si éste es el caso.

print

Esta función es utilizada por el código de listado de cadenas, para imprimir (a la salida estándar) la información de concordancia extra (si la hay) de una regla. El flag numérico se activa si el usuario especificó la opción '-n'.

save

Esta función es el inverso de parse: es utilizada por 'iptables-save' para reproducir las opciones que crearon la regla.

extra_opts

Esto es una lista (terminada en NULL) de las opciones extra que ofrece su biblioteca. Se unen a las opciones actuales y son pasadas a getopt_long; para más detalles, lea la página man. El código devuelto por getopt_long se convierte en el primer argumento ('c') de su función 'parse()'.

Existen elementos adicionales al final de esta estructura para el uso interno de `iptables`: no necesita asignarles nada.

Nuevos objetivos La función _init() de su biblioteca compartida le pasa a 'register_target()' un puntero a una estructura estática 'struct iptables_target', que tiene campos similares a la estructura iptables_match detallada más arriba.

A veces, un objetivo no necesita de una biblioteca de espacio de usuario; de todas formas, debe crear una trivial: existían demasiados problemas con bibliotecas mal colocadas.

4.2.3 Utilizando 'libiptc'

`libiptc` es la biblioteca de control de iptables, diseñada para listar y manipular las reglas del módulo del kernel iptables. Aunque su aplicación actual es para el programa iptables, hace muy sencillo escribir otras herramientas. Necesita ser root para utilizar estas funciones.

Las propias tablas del kernel son simplemente una tabla de reglas, y una serie de números que representan los puntos de entrada. Mediante la biblioteca, se proporcionan los nombres de las cadenas ("INPUT", etc.) como una abstracción. Las cadenas definidas por el usuario se etiquetan insertando un nodo de error antes de la cabecera de la cadena, que contiene el nombre de la cadena de la sección de datos extra del objetivo (las posiciones de la cadena montada están definidas por los tres puntos de entrada de la tabla).

Cuando se llama a 'iptc_init()', se lee la tabla, incluyendo los contadores. La tabla se manipula mediante las funciones 'iptc_insert_entry()', 'iptc_replace_entry()', 'iptc_append_entry()', 'iptc_delete_entry()', 'iptc_delete_num_entry()', 'iptc_flush_entries()', 'iptc_zero_entries()', 'iptc_create_chain()' 'iptc_delete_chain()', y 'iptc_set_policy()'.

Los cambios en la tabla no se efectúan hasta que se llama a la función 'iptc_commit()'. Esto significa que es posible que dos usuarios de la biblioteca operando en la misma cadena compitan; para prevenir esto habría que hacer un bloqueo, y actualmente no se hace.

Sin embargo, no existe carrera entre los contadores; los contadores se añaden al kernel de tal manera que los incrementos de contador que hay entre la lectura y escritura de la tabla todavía siguen presentándose en la nueva tabla.

Hay varias funciones de ayuda:

iptc_first_chain()
: Esta función devuelve el primer nombre de cadena de la tabla.

iptc_next_chain()
: Esta función devuelve el siguiente nombre de cadena de la tabla: NULL significa que no hay más cadenas.

iptc_builtin()
: Devuelve verdadero si el nombre de cadena dado es el nombre de una cadena montada.

iptc_first_rule()
: Esta función devuelve un puntero a la primera regla del nombre de cadena dado: NULL si es una cadena vacía.

iptc_next_rule()
: Esta función devuelve un puntero a la siguiente regla de la cadena: NULL significa el final de la cadena.

iptc_get_target()
: Esta función obtiene el objetivo de una regla dada. Si es un objetivo extendido, se devuelve el nombre del objetivo. Si es un salto a otra cadena, se devuelve el nombre de esa cadena. Si es un veredicto (p.ej. DROP), se devuelve su nombre. Si no tiene objetivo (una regla tipo accounting), entonces se devuelve la cadena vacía.

  Tenga en cuenta que debe utilizarse esta función en vez de utilizar directamente el valor del campo 'veredicto' de la estructura ipt_entry, ya que ofrece todas las interpretaciones del veredicto estándar especificadas arriba.

iptc_get_policy()
: Esta función obtiene la política de una cadena montada, y rellena el argumento 'counters' con las estadísticas de esa política.

iptc_strerror()
: Esta función devuelve una explicación más detallada de un fallo de código en la biblioteca iptc. Si una función falla, siempre establece la variable errno: este valor puede pasarse a iptc_strerror() para producir un mensaje de error.

4.3 Comprendiendo NAT

Bienvenido a la Traducción de Direcciones de Red del kernel. Tenga en cuenta que la infraestructura ofrecida está diseñada más para ser completa que para ser eficiente, y puede que algunos ajustes futuros aumenten notablemente la eficiencia. Por el momento estoy contento de que al menos funcione.

El NAT está separado en el seguimiento de conexiones (que no manipula paquetes) y el propio código NAT. El seguimiento de conexiones también está diseñado para que pueda utilizarlo un módulo de iptables, por lo que hace distinciones sutiles en los estados, que a NAT no le interesan en absoluto.

4.3.1 Seguimiento de conexiones

El seguimiento de conexiones se acopla en los ganchos de alta prioridad NF_IP_LOCAL_OUT y NF_IP_PRE_ROUTING para poder interceptar los paquetes antes de que entren en el sistema.

El campo nfct del skb es un puntero al interior de la estructura ip_conntrack, a un elemento del vector infos[]. Así podemos saber el estado del skb mediante el elemento de este vector al que está apuntando: este puntero codifica la estructura de estado y la relación de este skb con ese estado.

La mejor manera de extraer el campo 'nfct' es llamando a 'ip_conntrack_get()', que devuelve NULL si no está inicializado, o el puntero de conexión, y rellena ctinfo, que describe la relación del paquete con esa conexión. Este tipo enumerado tiene varios valores:

IP_CT_ESTABLISHED
: El paquete es parte de una conexión establecida, y va en la dirección original.

IP_CT_RELATED
: El paquete está relacionado con la conexión, y está pasando en la dirección original.

IP_CT_NEW
: El paquete intenta crear una nueva conexión (obviamente, va en la dirección original).

IP_CT_ESTABLISHED + IP_CT_IS_REPLY
: El paquete es parte de una conexión extablecida, en la dirección de respuesta.

IP_CT_RELATED + IP_CT_IS_REPLY
: El paquete está relacionado con la conexión, y está pasando en la dirección de respuesta.

Por tanto, se puede identificar un paquete de respuesta comprobando si es >= IP_CT_IS_REPLY.

4.4 Extendiendo el seguimiento de conexiones/NAT

Estos sistemas están diseñados para alojar cualquier número de protocolos y diferentes tipos de correspondencia (mapping). Algunos de estos tipos de correspondencia pueden ser bastante específicos, como el tipo de correspondencia load-balancing/fail-over.

Internamente, el seguimiento de conexiones convierte un paquete en una "n-upla", que representa las partes interesantes del paquete, antes de buscar ligaduras o reglas que concuerden con él. Esta n-upla tiene una parte manipulable y una parte no manipulable, llamadas "src" y "dst", ya que éste es el aspecto del primer paquete en el mundo del Source NAT [SNAT, NAT de origen] (sería un paquete de respuesta en el mundo del Destination NAT [DNAT, NAT de destino]). En todos los paquetes del mismo flujo y en la misma dirección, esta n-upla es igual.

Por ejemplo, la parte manipulable de la n-upla de un paquete TCP son la IP de origen y el puerto de origen, y la parte no manipulable son la IP de destino y el puerto de destino. Sin embargo, las partes manipulable y no manipulable no necesitan ser del mismo tipo; por ejemplo, la parte manipulable de la n-upla de un paquete ICMP es la IP de origen y el id ICMP, y la parte no manipulable es la IP de destino y el tipo y código ICMP.

Toda n-upla tiene una inversa, que es la n-upla de los paquetes de respuesta del flujo. Por ejemplo, la inversa de un paquete ICMP ping con id 12345, desde 192.168.1.1 y hacia 1.2.3.4, es un paquete ping-reply con id 12345, desde 1.2.3.4 hacia 192.168.1.1.

Estas n-uplas, representadas por la estructura 'struct ip_conntrack_tuple', se utilizan ampliamente. De hecho, junto con el gancho desde el que vino el paquete (que tiene influye en el tipo de manipulación esperada) y el dispositivo implicado, suponen toda la información del paquete.

La mayoría de las n-uplas están contenidas dentro de una estructura 'struct ip_conntrack_tuple_hash', que añade una entrada que es una lista doblemente enlazada, y un puntero a la conexión a la que pertenece la n-upla.

Una conexión está representada por la estructura 'struct ip_conntrack'; tiene dos campos 'struct ip_conntrack_tuple_hash': uno referido a la dirección del paquete original (tuplehash[IP_CT_DIR_ORIGINAL]), y otro referido a los paquetes de la dirección de respuesta (tuplehash[IP_CT_DIR_REPLY]).

De todas maneras, la primera cosa que hace el código NAT es ver si el código de seguimiento de conexiones consiguió extraer una n-upla y encontrar una conexión existente, mirando el campo nfct del skbuff; esto nos dice si es un intento de conexión nueva, y si no lo es, qué dirección tiene; en el último caso, se realizan con anterioridad las manipulaciones determinadas para esa conexión.

Si era el comienzo de una conexión nueva, buscamos una regla para esa n-upla, utilizando el mecanismo de recorrido estándar de iptables. Si una regla concuerda, se utiliza para inicializar las manipulaciones para esa dirección y para la respuesta; se le dice al código de seguimiento de conexiones que la respuesta que espera ha cambiado. Luego, es manipulado como se explica arriba.

Si no hay regla, se crea una ligadura (binding) 'null': normalmente, esto no hace corresponder al paquete, pero existe para asegurarnos de que no hacemos corresponder otro flujo sobre uno ya existente. A veces no puede crearse la ligadura null, porque ya hemos hecho corresponder un flujo existente sobre ella, en cuyo caso la manipulación por-protocolo (per-protocol) puede intentar rehacer la correspondencia (remap), aunque sea nominalmente una ligadura 'null'.

4.4.1 Objetivos NAT estándar

Los objetivos NAT son como cualquier otro objetivo de iptables, excepto en que insisten en ser utilizados sólo en la tabla 'nat'. Los objetivos SNAT y DNAT reciben una estructura 'struct ip_nat_multi_range' como datos extra; esto se utiliza para especificar el rango de direcciones a los que se puede enlazar una correspondencia. Un elemento de rango, la estructura 'struct ip_nat_range', consiste en una dirección IP inclusiva mínima y máxima, y un valor específico de protocolo (p.ej. puertos TCP) inclusivo máximo y mínimo. También hay sitio para flags, que dicen si la dirección IP puede corresponderse (a veces sólo queremos corresponder la parte específica de protocolo de una n-upla, no la IP), y otra para decir que la parte específica de protocolo del rango es válida.

Un multi-rango es un vector de estos elementos 'struct ip_nat_range'; esto significa que un rango podría ser "1.1.1.1-1.1.1.2 ports 50-55 AND 1.1.1.3 port 80". Cada elemento se añade al rango (una unión, para los que les guste la teoría).

4.4.2 Nuevos protocolos

Dentro del kernel Implementar un protocolo nuevo significa primero decidir cuáles deben ser las partes manipulables y no manipulables de la n-upla. Todo en la n-upla tiene la propiedad de que identifica al flujo unívocamente. La parte manipulable de la n-upla es la parte con la que usted puede hacer NAT: para el TCP esto es el puerto de origen, para el ICMP es el id; algo que se utiliza para que sea un "identificador de flujo". La parte no manipulable es el resto del paquete que identifica unívocamente al flujo, pero con lo que no podemos trastear (p.ej. el puerto TCP de destino, o el tipo ICMP).

Una vez que ha decidido esto, puede escribir una extensión al código de seguimiento de conexiones en el directorio, y meterse a rellenar la estructura 'ip_conntrack_protocol' que necesita pasarle a 'ip_conntrack_register_protocol()'.

Los campos de 'struct ip_conntrack_protocol' son:

list

Asígnele '{ NULL, NULL }'; utilizado para coserle a la lista.

proto

Su número de protocolo; vea '/etc/protocols'.

name

El nombre de su protocolo. Éste es el nombre que verá el usuario; normalmente, es mejor si es el nombre canónico que aparece en '/etc/protocols'.

pkt_to_tuple

La función que rellena las partes específicas de protocolo de la n-upla, dado un paquete. El puntero 'datah' apunta al principio de su cabecera (justo después de la cabecera IP), y datalen es la longitud del paquete. Si el paquete no es lo suficientemente largo para contener la información de la cabecera, devuelve 0; sin embargo, datalen siempre tendrá al menos 8 bytes (forzado por el sistema).

invert_tuple

Esta función se usa simplemente para transformar la parte específica de protocolo de la n-upla en el aspecto que tendría una respuesta a ese paquete.

print_tuple

Esta función se utiliza para imprimir la parte específica de protocolo de una n-upla; normalmente se almacena mediante sprintf() en el búfer especificado. Se devuelve el número de caracteres utilizados del búfer. Esto se utiliza para imprimir los estados para la entrada en /proc.

print_conntrack

Esta función se utiliza para imprimir la parte privada de la estructura conntrack, si hay alguna. También se utiliza para imprimir los estados en /proc.

packet

Se llama a esta función cuando se observa un paquete que es parte de una conexión establecida. Se obtiene un puntero a la estructura conntrack, la cabecera IP, la longitud y el ctinfo. Hay que devolver un veredicto para el paquete (normalmente NF_ACCEPT), o -1 si el paquete no es una parte válida de la conexión. Puede borrar la conexión de esta función si lo desea, pero debe utilizar el siguiente idioma para evitar carreras (vea ip_conntrack_proto_icmp.c):

```
if (del_timer(&ct->timeout))
        ct->timeout.function((unsigned long)ct);
```

new

Se llama a esta función cuando un paquete crea una conexión por primera vez; no hay argumento ctinfo, ya que el primer paquete tiene ctinfo IP_CT_NEW por definición. Devuelve 0 para no aprobar la creación de la conexión, o el timeout de la conexión en *jiffies*.

Una vez que ha escrito su nuevo protocolo y comprobado que puede hacer seguimiento con él, es hora de enseñarle a NAT cómo traducirlo. Esto significa escribir un nuevo módulo, una extensión al código NAT, y meterse a rellenar la estructura 'ip_nat_protocol' que necesita pasarle a 'ip_nat_protocol_register()'.

list

Asígnele '{ NULL, NULL }'; utilizado para coserle a la lista.

name

El nombre de su protocolo. Éste es el nombre que verá el usuario; es mejor si es el nombre canónico que aparece en '/etc/protocols' para que funcione la auto-carga, como veremos después.

protonum

Su número de protocolo; vea '/etc/protocols'.

manip_pkt

Ésta es la otra mitad de la función de seguimiento de conexiones pkt_to_tuple: puede pensar en ella como en "tuple_to_pkt". Sin embargo, hay algunas diferencias: se obtiene un puntero al comienzo de la cabecera IP y la longitud total del paquete. Esto es así porque algunos protocolos (UDP, TCP) necesitan conocer la cabecera IP. Se obtiene el campo ip_nat_tuple_manip de la n-upla (es decir, el campo "src"), en vez de toda la n-upla, y el tipo de manipulación que se va a realizar.

in_range

Esta función se usa para saber si la parte manipulable de una n-upla dada está dentro del rango dado. Esta función tiene un poco de trampa: obtenemos el tipo de manipulación que se ha aplicado a la n-upla, que nos dice cómo interpretar el rango (¿es un rango de origen o un rango de destino lo que tratamos de obtener?).

Esta función se utiliza para comprobar si una correspondencia (mapping) existente nos coloca dentro del rango adecuado, y también comprueba si no se necesita ninguna manipulación.

unique_tuple

Esta función es el corazón de NAT: dada una n-upla y un rango, vamos a alterar la parte del protocolo de la n-upla para colocarla dentro del rango, y hacerla única. Si se puede encontrar una n-upla sin utilizar dentro del rango, devuelve 0. También obtenemos un puntero a la estructura conntrack, que se requiere para ip_nat_used_tuple().

El método usual es simplemente iterar la parte del protocolo de la n-upla a través del rango, aplicando 'ip_nat_used_tuple()' sobre ella, hasta que una devuelva falso.

Tenga en cuenta que ya se ha comprobado el caso de correspondencia nula (null-mapping): o está fuera del rango dado, o ya está cogido.

Si IP_NAT_RANGE_PROTO_SPECIFIED no está activado, significa que el usuario está haciendo NAT, no NAPT: hace algo razonable con el rango. Si no es deseable ninguna correspondencia (por ejemplo, en TCP, una correspondencia de destino no debe cambiar el puerto TCP a menos que se le ordene), devuelve 0.

print

Dado un búfer de caracteres, una n-upla de concordancia y una máscara, escribe la parte específica de protocolo y devuelve la longitud del búfer utilizado.

print_range

Dado un búfer de caracteres y un rango, escribe la parte de protocolo del rango y devuelve la longitud del búfer utilizado. Si IP_NAT_RANGE_PROTO_SPECIFIED no está activado para este rango, no se llamará a esta función.

4.4.3 Nuevos objetivos NAT

Ésta es la parte realmente interesante. Se pueden escribir nuevos objetivos NAT que proporcionen un nuevo tipo de correspondencia; el paquete por defecto trae dos nuevos objetivos adicionales: MASQUERADE y REDIRECT. Son bastante sencillos e ilustran el potencial que tiene escribir un objetivo NAT nuevo.

Están escritos igual que cualquier otro objetivo de iptables, pero internamente extraen la conexión y llaman a 'ip_nat_setup_info()'.

4.4.4 Ayudantes de protocolo para UDP y TCP

Esto todavía está en desarrollo.

4.5 Comprendiendo netfilter

Netfilter es muy sencillo, y está descrito con bastante profundidad en las secciones anteriores. Sin embargo, a veces es necesaro ir más allá de lo que ofrecen las infraestructuras de NAT o ip_tables, o usted puede querer reemplazarlas completamente.

Una cuestión importante de netfilter (bueno, en el futuro) es el cacheado. Todo skb tiene un campo 'nfcache': una máscara de bits que indica qué campos de la cabecera se examinaron, y si el paquete fue alterado o no. La idea es que cada gancho desactivado de netfilter haga OR en su bit relevante, *The idea is that each hook off netfilter OR's in the bit relevant to it,* para que luego podamos escribir un sistema de caché que sea lo suficientemente listo para darse cuenta de cuándo no es necesario que los paquetes pasen a través de netfilter.

Los bits más importantes son NFC_ALTERED, que significa que el paquete fue alterado (esto ya se utiliza en el gancho IPv4 NF_IP_LOCAL_OUT para re-enrutar los paquetes alterados), y NFC_UNKNOWN, que significa que no debe hacerse cacheado porque se ha examinado una propiedad que no puede ser expresada. En caso de duda, simplemente active el flag NFC_UNKNOWN del campo nfcache del skb de su gancho.

4.6 Escribiendo nuevos módulos netfilter

4.6.1 Conectándose a los ganchos netfilter

Para recibir/filtrar paquetes dentro del kernel, simplemente hay que escribir un módulo que registre un "gancho netfilter". Esto es básicamente una expresión de interés en algún punto dado; los puntos actuales son específicos para un protocolo, y están definidos en las cabeceras de netfilter específicas para un protocolo, como "netfilter_ipv4.h".

Para registrar y desregistrar ganchos netfilter, se utilizan las funciones 'nf_register_hook' y 'nf_unregister_hook'. Ambas reciben un puntero a una estructura 'struct nf_hook_ops', que se rellenan de la manera siguiente:

list

Utilizado para coserle a la lista enlazada: asígnele '{ NULL, NULL }'

hook

La función a la que se llama cuando un paquete llega a este punto de gancho. Su función debe devolver NF_ACCEPT, NF_DROP o NF_QUEUE. Si es NF_ACCEPT, se llamará al próximo gancho enlazado a ese punto. Si es NF_DROP, el paquete es rechazado. Si es NF_QUEUE, se coloca el paquete en la cola. Se recibe un puntero a un puntero skb, por lo que puede reemplazar completamente el skb si lo desea.

flush

Actualmente no se usa: está diseñada para transmitir la cuenta de paquetes cuando se limpia la caché. Puede que nunca se implemente: asígnele NULL.

pf

La familia de protocolos, por ejemplo, 'PF_INET' para IPv4.

hooknum

El número del gancho en el que está interesado, por ejemplo, 'NF_IP_LOCAL_OUT'.

4.6.2 Procesando paquetes en la cola

Actualmente, esta interfaz la utiliza ip_queue; puede registrarse para manejar los paquetes de la cola de un protocolo dado. Esto tiene una semántica parecida a registrarse para un gancho, excepto en que puede bloquearse procesando un paquete, y sólo puede ver los paquetes a los que un gancho haya respondido 'NF_QUEUE'.

Las dos funciones utilizadas para registrar interés en los paquetes de la cola son 'nf_register_queue_handler()' y 'nf_unregister_queue_handler()'. La función que usted registra será llamada con el puntero 'void *' que le pasó a 'nf_register_queue_handler()'.

Si nadie está registrado para manejar el protocolo, entonces devolver NF_QUEUE es lo mismo que devolver NF_DROP.

Una vez que ha registrado interés en los paquetes de cola, empiezan a entrar en la cola. Puede hacer lo que quiera con ellos, pero debe llamar a 'nf_reinject()' cuando haya acabado (no sirve hacer simplemente un kfree_skb()). Se le pasa el dkb, la estuctura 'struct nf_info' que recibió el manejador de la cola, y un veredicto: NF_DROP hace que sean rechazados, NF_ACCEPT hace que continúen iterando a través de los ganchos, NF_QUEUE hace que entren de nuevo en la cola, y NF_REPEAT hace que se consulte de nuevo el gancho que puso al paquete en la cola (cuidado con los bucles infinitos).

Puede mirar dentro de la estructura 'struct nf_info' si quiere información auxiliar sobre el paquete, como las interfaces y el gancho en el que estaba.

4.6.3 Recibiendo comandos desde el espacio de usuario

Es corriente que los componentes de netfilter quieran interactuar con el espacio de usuario. El método para hacer esto es utilizar el mecanismo setsockopt. Tenga en cuenta que cada protocolo tiene que modificarse para llamar a nf_setsockopt() para los números setsockopt que no entiende (y nf_getsockopt() para los números getsockopt), y hasta ahora sólo se han modificado IPv4, IPv6 y DECnet.

Utilizando una técnica ya familiar, registramos una estructura 'struct nf_sockopt_ops' utilizando la llamada nf_register_sockopt(). Los campos de esta estructura son como sigue:

list

Utilizado para coserlo a la lista enlazada: asígnele '{ NULL, NULL }'.

pf

La familia de protocolos que está manejando, p.ej. PF_INET.

set_optmin

y

set_optmax

Éstos especifican el rango (exclusivo) de números setsockopt manejados. Por tanto, poner 0 y 0 significa que no tiene números setsockopt.

set

Ésta es la función a la que se llama cuando el usuario llama a uno de sus setsockopts. Debe comprobar que tienen capacidad NET_ADMIN dentro de esta función.

get_optmin

y

get_optmax

Éstos especifican el rango (exclusivo) de números getsockopt manejados. Por tanto, poner 0 y 0 significa que no tiene números getsockopt.

get

Ésta es la función que se llama cuando el usuario llama a uno de sus números getsockopt. Debe comprobar que tienen capacidad NET_ADMIN dentro de esta función.

Los dos campos finales son de uso interno.

4.7 Manejo de paquetes en el espacio de usuario

Utilizando la bilbioteca libipq y el módulo 'ip_queue', ahora casi todo lo que se puede hacer dentro del kernel se puede hacer desde el espacio de usuario. Esto significa que, con una pequeña pérdida de velocidad, puede desarrollar completamente su código en el espacio de usuario. A menos que trate de filtrar anchos de banda muy grandes, este método es superior a la manipulación de paquetes desde el kernel.

En los primeros días de netfilter, probé esto portando al espacio de usuario una versión embrionaria de iptables. Netfilter abre las puertas para que la gente escriba sus propios y eficientes módulos de manipulación de paquetes en el lenguaje que quieran.

5 Portando los módulos de filtrado de paquetes desde 2.0 y 2.2

Mire el fichero ip_fw_compat.c para una sencilla capa que hace bastante fácil la traducción.

6 La batería de pruebas

Dentro del repositorio CVS reside una batería de pruebas: cuando más cubra la batería, más confianza se puede tener en que los cambios en el código no han roto algo silenciosamente. Las pruebas triviales son como poco tan importantes como las pruebas concienzudas: son las pruebas triviales las que simplifican las pruebas complejas (ya sabe que las bases tienen que funcionar bien antes de que se realicen las pruebas complejas).

Las pruebas son sencillas: tan sólo son *shell scripts* dentro del subdirectorio testsuite/. Se espera de ellos que se ejecuten sin dar error. Los scripts se ejecutan en orden alfabético, por lo que '01test' se ejecuta antes que '02test2'. Actualmente existen 5 directorios:

00netfilter/

Pruebas generales del sistema netfilter.

01iptables/

Pruebas de iptables.

02conntrack/

Pruebas del seguimiento de conexiones.

03NAT/

Pruebas del NAT.

04ipchains-compat/

Pruebas de compatibilidad ipchains/ipfwadm.

Dentro del directorio testsuite/ hay un script llamado 'test.sh'. Configura dos interfaces falsas (tap0 y tap1), activa el redireccionamiento, y elimina todos los módulos de netfilter. Luego entra en todos los directorios de arriba y ejecuta uno a uno los scripts test.sh hasta que uno falla. Este script recibe dos argumentos opcionales: '-v', que imprime los tests al ejecutarse, y el nombre de un script (si se especifica uno, se omitirán todos los scripts hasta que se encuentre éste).

6.1 Escribiendo una prueba

Cree un fichero nuevo en el directorio apropiado: intente numerar su script de manera que se ejecute en el momento adecuado. Por ejemplo, para probar el seguimiento de las respuestas ICMP (02conntrack/02reply.sh), primero necesitamos comprobar que los paquetes ICMP de salida tienen un seguimiento correcto (02conntrack/01simple.sh).

Normalmente es mejor crear muchos ficheros pequeños, cada uno cubriendo un área, porque así el que ejecute la batería de pruebas puede aislar el problema inmediatamente.

Si algo va mal en la prueba, haga simplemente un 'exit 1', que causa error; si es algo que usted espera que vaya a fallar, debería imprimir un mensaje. Si todo va bien, la prueba debe acabar con un 'exit 0'. Debe comprobar que **todos** los comandos se ejecutan con éxito, bien poniendo 'set -e' al principio del script, o añadiendo '|| exit 1' al final de cada comando.

Se pueden utilizar las funciones de ayuda 'load_module' y 'remove_module' para cargar módulos: nunca debe fiarse de la auto carga en la batería de pruebas, a menos que sea eso lo que está probando.

6.2 Variables y entorno

Dispone de dos interfaces con las que jugar: tap0 y tap1. Sus direcciones de interfaz están en las variables `$TAP0` y `$TAP1` respectivamente. Ambas tienen máscaras de red 255.255.255.0; sus redes están en $TAP0NET y $TAP1NET respectivamente.

Existe un archivo temporal vacío en $TMPFILE. Es borrado al final de la prueba.

Su script se ejecutará desde el directorio testsuite/, esté donde esté. Por tanto debe acceder a las herramientas (como iptables) utilizando una ruta que empiece por '../userspace'.

Su script puede imprimir más información si $VERBOSE está activado (lo que significa que el usuario especificó '-v' en la línea de comandos).

6.3 Herramientas útiles

Hay varias herramientas útiles para la batería de pruebas en el subdirectorio "tools": todas acaban con un status de salida distinto de cero si hubo algún problema.

6.3.1 gen_ip

Puede generar paquetes IP utilizando 'gen_ip', que imprime un paquete IP en la salida estándar. Puede alimentar a las interfaces tap0 y tap1 redireccionando la salida estándar a /dev/tap0 y /dev/tap1 (si no existen, se crean la primera vez que se ejecuta la batería de pruebas si).

gen_ip es un programa simplista que actualmente es muy quisquilloso con el orden de los argumentos. Primero están los argumentos opcionales:

FRAG=offset,longitud

Genera el paquete y luego lo convierte en un fragmento con el offset y la longutud especificados.

MF

Activa el bit 'More Fragments' del paquete.

MAC=xx:xx:xx:xx:xx:xx

Especifica la dirección MAC de origen del paquete.

TOS=tos

Especifica el campo TOS del paquete (de 0 a 255).

Ahora vienen los argumentos obligatorios:

source ip

La dirección IP de origen del paquete.

dest ip

La dirección IP de destino del paquete.

length

Longitud total del paquete, incluyendo las cabeceras.

protocol

Número de protocolo del paquete, p.ej. 17 = UDP.

Además, los argumentos dependen del protocolo: para UDP (17), están los puertos de origen y destino. Para ICMP (1), están el tipo y código del mensaje ICMP: si el tipo es 0 u 8 (ping-reply o ping), se requieren dos argumentos adicionales (los campos ID y secuencia). Para TCP, se requieren los puertos de origen y destino, y los flags ("SYN", "SYN/ACK", "ACK", "RST" o "FIN"). Hay tres argumentos opcionales: "OPT=" seguido de una lista de opciones separadas por comas, "SYN=" seguido de un número de secuencia, y "ACK=" seguido de un número de secuencia. Finalmente, el argumento opcional "DATA" indica que el cuerpo del paquete TCP tiene que llenarse con los contenidos de la entrada estándar.

6.3.2 rcv_ip

Puede ver los paquetes IP utilizando 'rcv_ip', que imprime la línea de comandos lo más parecidamente posible al valor original que se le pasó a gen_ip (los fragmentos son una excepción).

Esto es extremadamente útil para analizar paquetes. Recibe dos argumentos obligatorios:

wait time

El tiempo máximo en segundos que se esperará a un paquete por la entrada estándar.

iterations

El número de paquetes a recibir.

Hay un argumento especial, "DATA", que hace que el cuerpo de un paquete TCP se imprima en la salida estándar después de la cabecera.

La manera estándar de usar 'rcv_ip' en un script es como sigue:

```
# Activa el control de trabajos para poder utilizar & en los scripts.
set -m

# Espera un paquete desde tap0 durante dos segundos
../tools/rcv_ip 2 1 < /dev/tap0 > $TMPFILE &

# Se asegura de que rcv_ip ha comenzado a ejecutarse.
sleep 1

# Envía un paquete ping
../tools/gen_ip $TAP1NET.2 $TAP0NET.2 100 1 8 0 55 57 > /dev/tap1 || exit
1

# Espera a rcv_ip,
if wait %../tools/rcv_ip; then :
else
    echo rcv_ip failed:
    cat $TMPFILE
    exit 1
fi
```

6.3.3 gen_err

Este programa toma un paquete (generado con gen_ip, por ejemplo) de la entrada estándar, y lo convierte en un error ICMP.

Recibe tres argumentos: una dirección IP de origen, un tipo y un código. La dirección IP de destino será la IP de origen del paquete recibido desde la entrada estándar.

6.3.4 local_ip

Éste toma un paquete de la entrada estándar y lo inyecta dentro del sistema mediante un socket *raw*. Esto da la apariencia de un paquete generado localmente (a diferencia de alimentar un paquete mediante uno de los dispositivos ethertap, que aparentan ser paquetes generados remotamente).

6.4 Random Advice

Todas las herramientas asumen que pueden hacerlo todo en una sola lectura o escritura: esto es cierto para los dispositivos ethertap, pero podría no serlo si está haciendo cosas delicadas con tuberías.

Puede utilizar dd para cortar paquetes: dd tiene una opción obs (output block size, o tamaño del bloque de salida) que puede usarse para que produzca el paquete en una sola lectura.

Compruebe primero el funcionamiento correcto: p.ej. al probar que los paquetes se bloquean con éxito. Primero pruebe que los paquetes pasan normalmente, y **luego** pruebe que algunos paquetes quedan bloqueados. De otra manera, cualquier otro fallo podría estar parando los paquetes...

Trate de escribir pruebas precisas, no pruebas de 'enviar cosas al azar y ver lo que pasa'. Si una prueba precisa falla, es útil saberlo. Si una prueba aleatoria falla una vez, no ayuda demasiado.

Si una prueba falla sin dejar un mensaje, puede añadir '-x' en la primera línea del script (es decir, '#! /bin/sh -x') para ver qué comandos está ejecutando.

Si una prueba falla aleatoriamente, compruebe si hay tráfico de red aleatorio interfiriendo (pruebe desactivando todas sus interfaces externas). Un ejemplo: como comparto la misma red con Andrew Tridgell, suelo recibir plagas de broadcasts de Windows.

7 Motivación

Mientras desarrollaba ipchains, me di cuenta (en uno de esos momentos de destello-cegador-mientras-esperas-los-entrantes en un restaurante chino de Sydney) de que el filtrado de paquetes estaba haciéndose de la manera equivocada. No puedo encontrarlo, pero recuerdo haberle enviado un correo a Alan Cox, que respondió algo como 'aunque problemente tengas razón, por qué no acabas primero lo que estás haciendo'. En pocas palabras, el pragmatismo ganaba sobre El Modo Correcto.

Cuando acabé ipchains, que inicialmente iba a ser una pequeña modificación de la parte del kernel de ipfwadm, y luego se convirtió en una reescritura mucho mayor, y escribí el HOWTO. Me di cuenta de cuánta confusión existe en la mayoría de la comunidad Linux acerca de cuestiones como el filtrado de paquetes, enmascaramiento, redireccionamiento de puertos y cosas así.

Ésta es la satisfacción de hacer tu propio soporte: tienes una mejor percepción de lo que tratan de hacer los usuarios, y con qué cosas se están peleando. El software libre es más gratificante cuando está en manos de la mayoría de los usuarios (de eso se trata, ¿no?), y eso significa hacerlo más fácil. La arquitectura, no la documentación, era el defecto clave.

Por tanto, tenía la experiencia, con el código de ipchains, y una buena idea de lo que la gente de fuera estaba haciendo. Sólo había dos problemas.

Primero, no quería volver al tema de la seguridad. Ser un experto en seguridad es un juego moral de la cuerda entre tu conciencia y tu cartera. A un nivel fundamental, estás vendiendo la sensación de seguridad, que está reñida con la verdadera seguridad. Quizá trabajando en un cuartel militar, donde entienden la seguridad, sería distinto.

El segundo problema es que los usuarios novatos no son los únicos interesados; hay un número en aumento de grandes empresas y PSIs que están utilizando esto. Necesitaba The second problem is that newbie users aren't the only concern; an increasing number of large companies and ISPs are using this stuff. I needed reliable input from that class of users if it was to scale to tomorrow's home users.

Estos problemas se resolvieron cuando me topé con David Bonn, de WatchGuard, en el Usenix de julio de 1998. Estaban buscando un programador del kernel de Linux; al final acordamos que iría a sus oficinas de Seattle durante un mes, y veríamos si podíamos sacar un acuerdo por el cual ellos patrocinarían mi código nuevo y mis esfuerzos por realizar este soporte. El precio que acordamos era más de lo que yo pedía, These problems were resolved, when I ran into David Bonn, of WatchGuard fame, at Usenix in July 1998. They were looking for a Linux kernel coder; in the end we agreed that I'd head across to their Seattle offices for a month and we'd see if we could hammer out an agreement whereby they'd sponsor my new code, and my current support efforts. The rate we agreed on was more than I asked, so I didn't take a pay cut. This means I don't have to even think about external conslutting for a while.

El acceso a WatchGuard me dio acceso a los grandes clientes que necesitaba, y ser independiente de ellos me permitió dar soporte a todos los usuarios (por ejemplo, a la compentencia de WatchGuard) por igual.

Podría simplemente haber escrito netfilter y portado ipchains, y habría acabado con eso. Desafortunadamente, eso habría dejado todo el código de enmascaramiento dentro del kernel: hacer el enmascaramiento independiente del filtrado es uno de los principales puntos a favor de mover los puntos de filtrado de paquetes, pero para hacer eso, el enmascaramiento también necesitaba moverse al sistema netfilter. So I could have simply written netfilter, ported ipchains over the top, and been done with it. Unfortunately, that would leave all the masquerading code in the kernel: making masquerading independent from filtering is the one of the major wins point of moving the packet filtering points, but to do that masquerading also needed to be moved over to the netfilter framework as well.

Además, mi experiencia con la característica 'interface-address' de ipfwadm (la que eliminé en ipchains) me había enseñado que no era factible quitar el código de enmascaramiento y esperar que alguien que lo necesitase hiciese por mí el trabajo de portarlo a netfilter.

Por tanto, necesitaba tener al menos tantas características como en el código de entonces; preferiblemente unas cuantas más, para animar a los usuarios de nicho (??) a hacerse los primeros en adoptarlo. Esto significa reemplazar el proxy transparente (¡felizmente!), el enmascaramiento y el redireccionamiento de puertos. En otras palabras, una capa NAT completa. So I needed to have at least as many features as the current code; preferably a few more, to encourage niche users to become early adopters. This means replacing transparent proxying (gladly!), masquerading and port forwarding. In other words, a complete NAT layer.

Aunque hubiese decidido portar la capa de enmascaramiento existente, en vez de escribir un sistema NAT genérico, el código de enmascaramiento mostraba ya una edad y una falta de mantenimiento. No había nadie manteniendo el enmascaramiento, y se notaba. Parece que los usuarios serios generalmente no usan enmascaramiento, y no hay muchos usuarios domésticos que se dediquen a la tarea de llevar el mantenimiento. Gente animosa como Juan Ciarlante hacía correcciones, pero se había llegado a un punto (que se alargaba más y más) en el que era necesario una reescritura.

Por favor, tenga en cuenta que yo no era la persona para hacer una reescritura del NAT: no utilicé más el enmascaramiento, y no había estudiado el código existente entonces. Probablemente por eso me llevó más tiempo del que hubiese debido. Pero el resultado es bastante bueno, en mi opinión, y está claro que he aprendido mucho. Sin duda la segunda versión será incluso mejor, una vez que veamos cómo lo utiliza la gente.

8 Agradecimientos

Gracias a todos los que han ayudado.