NFTABLES

Autor: Manuel Luna Pérez Fecha: 17/06/2016

Índice

1. Documentación

1.1 Introducción

1.1.1 ¿QUÉ ES?

Nftables es una nueva infraestructura de clasificación de paquetes cuya intención es reemplazar la infraestructura actual {ip,ip6,arp,eb}_tables.

- Está disponible en los kernels de linux >= 3.13.
- Tiene una nueva línea de comandos (nft) cuya sintaxis es diferente de la de iptables.
- Viene con una capa de compatibilidad que permite ejecutar comandos de iptables sobre la nueva infraestructura de nftables
- Proporciona una infraestructura que permite construir mapas y concatenaciones. Se puede utilizar esta nueva característica para organizar un ruleset en árbol multidimensional, que hace que se reduzca drásticamente el número de reglas que necesitan consultarse hasta que se encuentre qué acción realizar sobre el paquete.

1.1.2 ¿Por qué utilizar nftables?

Iptables nos ha servido (y probablemente seguirá sirviendo aún durante un tiempo en muchas implementaciones) para filtrar el tráfico tanto por paquete como por flujo, registrar actividades sospechosas en el tráfico, realizar NAT y otras muchas cosas. Viene con más de un centenar de extensiones que han sido presentadas a lo largo de los últimos 15 años. Sin embargo, la infraestructura de iptables sufre de limitaciones que no pueden trabajar fácilmente en torno a:

- Evitar la duplicación de código y las incoherencias: Muchas de las extensiones de iptables son de protocolo específico, por lo que no es una forma consolidada para que coincida con los campos de paquetes, en su lugar tenemos una extensión para que cada protocolo sea soportado. Esto incha el código base con un código muy similar para realizar una tarea similar. Coincidencia de carga útil.
- Clasificación de paquetes más rápida a través de un conjunto genérico mejorado e infraestructura de mapeo de datos.
- Simplificada la administración dual de IPv4/IPv6, a través de la nueva familia inet que permite registrar cadenas base que vean el tráfico tanto de IPv4 como IPv6.
- Soporte de actualizaciones dinámicas a un ruleset.
- Proporcionar una API Netlink para aplicaciones de terceros, al igual que otras redes linux y el subsistema netfilter hace.
- Inconsistencias en la sintaxis de direcciones y proporcionar la sintaxis más agradable y más compacta.

1.1.3 Principales diferencias con iptables

Las principales diferencias entre nftables e iptables desde el punto de vista del usuario son:

- La sintaxis. La herramienta de línea de comandos iptables utiliza un analizador basado en getopt_long(), donde las claves son siempres precedidas por doble guión, (ej. --key) o un solo guión (ej. -p tcp). En ese sentido, nftables es más agradable, más intuitivo y más compacto respecto a la sintaxis que se inspira en la de tcpdump.
- Las tablas y las cadenas son totalmente configurables. En nftables, las tablas son contenedores de cadenas sin una semántica determinada. Iptables viene con tablas y con un número predefinido de cadenas, que son registradas en el sistema independientemente de si van a ser usadas o no. Se han generado informes en los que se dice que las cadenas no realizadas perjudican el

rendimiento, incluso si no añadimos ninguna regla. Con este nuevo enfoque, se pueden registrar las cadenas necesarias que necesitamos dependiendo de su configuración. Por otra parte, también se puede modelar la canalización mediante las prioridades de cadenas en la forma que necesitemos y seleccionar cualquier nombre para las tablas y cadenas.

- No hay más distinciones entre "matches" y "targets". En nftables, las expresiones son el componente básico de una regla, por lo tanto, una regla es básicamente una composición de expresiones que se evalúan de forma lineal de izquierda a derecha: si la primera expresión coincide, entonces la siguiente expresión será evaluada y así sucesivamente hasta llegar a la última expresión al final de la regla.
- Podemos especificar varias acciones en una sola regla. En iptables sólo puede especificarse un solo target. Esta ha sido una limitación durante muchos años, que los usuarios resuelvan saltando a las cadenas personalizadas, a costa de hacer un poco más compleja la estructura del conjunto de reglas.
- Sin contador integrado por cadena y reglas. En nftables, esto es opcional para que pueda permitir a los contadores en demanda.
- Mejor soporte para actualizaciones dinámicas del ruleset.
- Simplificada la administración de IPv4/IPv6, a través de la nueva familia inet, que le permite registrar cadenas que vean el tráfico IPv4 e IPv6. Por lo tanto no es necesarios depender de scripts para duplicar un conjunto de reglas.
- Infraestructura genérica de conjuntos y mapeos de datos. Esta nueva infraestructura se integra perfectamente en el núcleo nftables y permite configuraciones avanzadas tales como diccionarios, mapas e intervalos para lograr la clasificación de paquetes orientado al rendimiento. Lo más importante es que se pueden usar selectores soportados para clasificar el tráfico.
- Soporte para concatenaciones. Desde el kernel linux 4.1, se puede concatenar varias claves y combinarlas con los diccionarios y mapas. La idea es construir

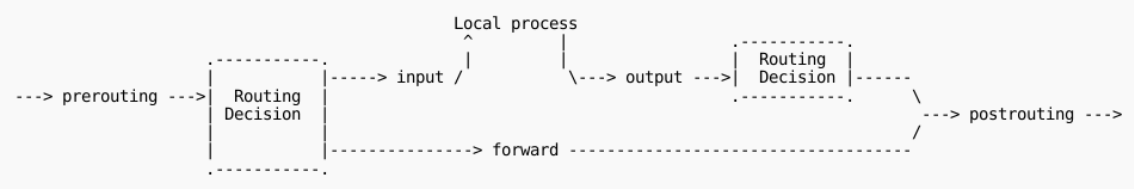
una tupla cuyos valores se encripten para obtener la acción que debe realizarse cerca de 0(1).

- Nuevos protocolos soportados sin actualizaciones de kernel. Las actualizaciones del kernel pueden ser una tarea desalentadora y consumir mucho tiempo. En concreto, si tenemos que mantener más de un servidor de seguridad en la red. Los distribuidores incluyen generalmente versiones un poco más viejas de kernel linux por razones de estabilidad. Con el nuevo enfoque de nftables, lo más probable es que no tengamos que actualizar para que un protocolo sea soportado. Una actualización de software nft debería ser más que suficiente para soportar nuevos protocolos.

1.1.4 Hooks netfilter

La mayor parte de la infraestructura sigue siendo la misma, nftables vuelve a utilizar la infraestructura existente de hooks, sistema de seguimiento de conexiones, motor NAT, registro de infraestructura, gestión de colas y así sucesivamente. Por lo tanto, sólo se ha reemplazado la infraestructura de clasificación de paquetes.

Para aquellos que no estén familiarizados con Netfilter, aquí tenemos una representación de estos hooks.



Básicamente, el flujo de tráfico entrante hacia la máquina local verá el hook de entrada y prerouting . Entonces, el tráfico que se genera por los procesos locales sigue la ruta de salida y postrouting.

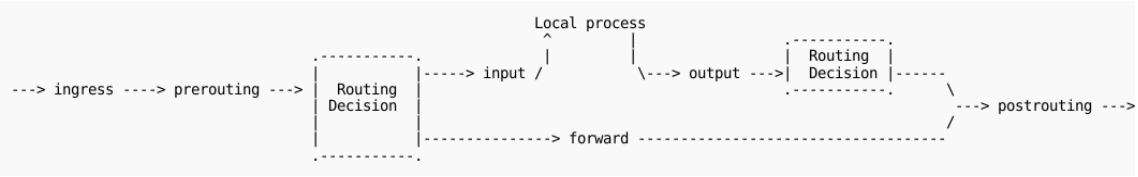
Si configuramos nuestra máquina como router, no podremos olvidar activar forwarding haciendo:

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

Y luego, los paquetes que no tienen como destino nuestra máquina, serán vistos por el hook de forward. En resumen, los paquetes que no están dirigidos hacia un proceso local seguirán esta ruta: prerouting, forward y postrouting.

Hook de ingreso

Desde el kernel linux 4.2, Netfilter también viene con un hook de entrada que se puede utilizar desde nftables. Así que el proceso ahora será el siguiente:



Podemos utilizar este nuevo hook de ingreso para filtrar el tráfico de capa 2, este nuevo hook que se encuentra antes de prerouting.

1.2 Empezando

1.2.1 Construcción e instalación de nftables desde código fuente.

Nftables requiere varias librerías de espacio de usuario, la utilidad "nft"en línea de comandos y los módulos del núcleo.

Si estamos utilizando una distribución de linux importante, deberíamos considerar el uso de nftables desde las distribuciones.

Instalación librerías de espacio de usuario.

Necesitamos instalar las siguientes librerías:

- libmnl ⇒ Esta biblioteca proporciona las interfaces necesarias para que puedan comunicarse el kernel y el espacio de usuario mediante Netlink. Es muy probable que algunas distribuciones ya ofrezcan esta librería. Si decidimos utilizar el paquete del distribuidor,

tenemos que asegurarnos de instalar el paquete de desarrollo también.

- libnft (anteriormente conocido como libnftables) ⇒ Esta biblioteca proporciona la API de nivel bajo para transformar los mensajes de red a los objetos.

También son necesarias las librerías "libgmp" y "libreadline", la mayoría de las distribuciones ya las ofrecen, así que debemos de asegurarnos de instalar las extensiones de desarrollo de estos paquetes para compilar correctamente nftables.
Si vamos a realizar pruebas con nftables, se recomienda usar instantáneas mediante git para "libnftnl" y "nft".

Instalación librerías desde git.

Para instalar "libnftnl" tenemos que clonar el repositorio que compilar la librería.

```
$ git clone git://git.netfilter.org/libnftnl
$ cd libnftnl
$ sh autogen.sh
$ ./configure
$ make
$ sudo make install
```

Con esto sería suficiente.

Instalación librerías desde snapshot.

Podemos recuperar los snapshot diarios de esta librería desde FTP Netfilter.

El proceso de instalación sería el siguiente:

```
$ wget ftp://ftp.netfilter.org/pub/libnftnl/snapshot/libnftnl-20140217.tar.bz2
$ tar xvjf libnftnl-20140217.tar.bz2
$ ./configure
$ make
$ sudo make install
```

Instalación de utilidad "nft"

Este comando proporciona una interfaz de usuario para configurar nftables.

Instalando desde git

Basta con ejecutar los siguientes comandos:

```
% git clone git://git.netfilter.org/nftables
% cd nftables
% sh autogen.sh
% ./configure
% make
% make install
```

Podemos comprobar que nftables ha sido instalado en el sistema de la siguiente forma:

```
% nft
nft: no command specified
```

Esto significa que nftables se ha instalado correctamente.

Instalación de kernel linux con soporte nftables.

Requisitos: nftables está disponible en el kernel de linux desde la versión 3.13, pero es el software en fase de desarrollo, por lo que se recomienda utilizar la última versión estable del kernel.

Comprobando la instalación.

Podemos comprobar que la instalación se ha realizado correctamente si podemos arrancar el módulo "nf_tables" :

```
% modprobe nf_tables
```

Comprobamos si el módulo se está ejecutando con *"lsmod"*:

```
# lsmod | grep nf_tables
nf_tables              42349  0
```

"dmesg" debería mostrar el siguiente mensaje:

```
% dmesg
...
[13939.468020] nf_tables: (c) 2007-2009 Patrick McHardy
<kaber@trash.net>
```

Tenemos que asegurarnos que también se ha cargado el apoyo de la familia "nf_tables", por ejemplo:

```
% modprobe nf_tables_ipv4
```

El comando “lsmod” debería mostrar algo similar a:

```
# lsmod | grep nf_tables
nf_tables_ipv4        12869  0
nf_tables             42349  1 nf_tables_ipv4
```

Otros módulos de la familia nf_tables son “nf_tables_ipv6”, “nf_tables_bridge”, “nf_tables_arp” y desde versiones de kernel superiores a la 3.14 “nf_tables_inet”.

Estos módulos proporcionan las correspondiente tabla y el soporte de filtro de cadenas para la familia dada.

Podemos comprobar qué módulos son compatibles con el kernel actual.

Tendremos que comprobar el fichero /boot/config-XXX-YYY , donde “XXX” es la versión del kernel e “YYY” la arquitectura, ej: “/boot/config-4.2.0-1-amd64”

1.2.2 Nftables desde las distribuciones

La mayoría de las distribuciones de Linux tienen soporte para nftables:

- Incluyen el kernel con soporte para nf_tables
- Incluyen el soporte de espacio de usuario.

Normalmente, podemos obtener nftables con tan solo instalar el software utilizando el gestor de paquetes correspondiente.

Tenemos que tener en cuenta que las distribuciones normalmente no utilizan la última versión de nftables o del kernel. Si necesitamos la última versión de esta infraestructura, necesitaremos instalar desde las fuentes.

Como referencia, aquí tenemos algunos enlaces donde podemos obtener más información sobre nftables en cada distribución:

Debian:

Debian también incluye las últimas versiones de nftables en los repositorios “stable-backports”, por lo que no es necesario utilizar “testing” para obtener nftables.

- wiki: https://wiki.debian.org/nftables
- kernel: https://tracker.debian.org/pkg/linux
- libnftnl: https://tracker.debian.org/pkg/libnftnl

- nft utility: https://tracker.debian.org/pkg/nftables

arch linux

- wiki: https://wiki.archlinux.org/index.php/Nftables
- kernel: https://www.archlinux.org/packages/core/x86_64/linux/
- nft utility: https://www.archlinux.org/packages/extra/x86_64/nftables/
- libnftnl: https://www.archlinux.org/packages/extra/x86_64/libnftnl/

ubuntu

- kernel: https://launchpad.net/ubuntu/+source/linux
- nft utility: https://launchpad.net/ubuntu/+source/nftables
- libnftnl: https://launchpad.net/ubuntu/+source/libnftnl

fedora

- kernel: https://admin.fedoraproject.org/pkgdb/package/kernel/
- nft utlity: https://admin.fedoraproject.org/pkgdb/package/nftables/
- libnftnl: https://admin.fedoraproject.org/pkgdb/package/libnftnl/

1.3 Operaciones básicas

1.3.1 Configuración de tablas

En nftables, las tablas son contenedores de cadenas sin una semántica específica. Así que lo primero que tenemos que hacer para empezar a trabajar con nftables es añadir al menos una tabla. Entonces, podremos añadir cadenas y a esas cadenas le agregaremos las reglas.

Pero antes vamos a centrarnos en este artículo sobre los objetos de la tablas:

Para aquellos que estén familiarizados con iptables, la diferencia principal es que no hay ninguna tabla predefinida. Por lo que cualquier tablas que creamos vendrá inicialmente vacía.

Por el momento tenemos seis tipos diferentes de tablas en función de la familia que sean:

- ip

- arp
- ip6
- bridge
- inet, que está disponible desde el kernel de Linux 3.14. Esta tabla es una tabla híbrida de IPV4 + IPV6 que debería ayudar a simplificar la carga en la administración de servidores de seguridad dual. Por lo tanto, las cadenas que registremos en esta tabla servirán para ver el tráfico de IPv4 y de IPv6.
- netdev, disponible desde el kernel linux 4.2. Esta familia viene con un hook de entrada que se puede utilizar para registrar una cadena que filtre en una etapa muy temprana, es decir, antes de prerouting, como alternativa a la infraestructura existente.

Añadiendo tablas

La sintaxis para añadir una tabla es la siguiente:

```
% nft add table [<familia>] <nombre>
```

Para añadir una “foo” para IPv4, se haría de la siguiente manera:

```
% nft add table ip foo
```

Tenemos que tener en cuenta que en este caso, el argumento de la familia es opcional, ya que si no se especifica ninguna, se asumirá que es IPv4.

Si queremos añadir una tabla para IPv6, tendremos que escribir el siguiente comando:

```
% nft add table ip6 bar
```

Listado de tablas

Para ver el listado de tablas existentes:

```
% nft list tables
table ip foo
table ip6 bar
```

También podemos ver las tablas pertenecientes a una familia:

```
% nft list tables ip
table ip foo
```

Para ver el contenido de una tabla correspondiente lo haremos de la siguiente manera:

```
% nft list table ip filter
```

```
table ip filter {
}
```

Como hemos visto anteriormente, en principio, cualquier tabla está vacía, ya que las tablas no vienen con cadenas preconfiguradas como en iptables.

Eliminación de tablas

Para eliminar las tablas se hará de la siguiente forma:

```
% nft delete table ip foo
```

Solución de problemas: Desde el kernel linux 3.18, podemos eliminar las tablas y su contenido con este comando. Sin embargo, antes de esta versión, es necesario eliminar su contenido en primer lugar, de lo contrario nos devolverá un error similar a este:

```
% nft delete table filter
<cmdline>:1:1-19: Error: Could not delete table: Device or resource busy
delete table filter
^^^^^^^^^^^^^^^^^^^
```

Limpiado de tablas

Podemos eliminar todas las reglas pertenecientes a una tabla con el siguiente comando:

```
% nft flush table ip filter
```

Esto eliminará las reglas de todas las cadenas que se registren en esta tabla.

1.3.2 Configuración de cadenas

Al igual que en iptables, debemos añadir reglas a las cadenas. Sin embargo, la infraestructura de nftables viene sin cadenas predefinidas, por lo que necesitamos añadir cadenas base en primer lugar antes de añadir cualquier regla. Esto permite configuraciones muy flexibles.

Añadiendo cadenas base

La sintaxis para añadir cadenas base es la siguiente:

```
% nft add chain [<familia>] <nombre-tabla> <nombre-cadena> { type <tipo> hook <hook> priority <valor> \; }
```

Las cadenas base con las que están registradas en los hooks de Netfilter, es decir, estas cadenas verán los paquete que fluyen a través de la pila Linux TCP/IP. El siguiente ejemplo muestra cómo se puede añadir una cadena base a la tabla “foo” a través del siguiente comando:

```
% nft add chain ip foo input { type filter hook input priority 0 \; }
```

Importante: Tenemos que evitar el punto y coma si estamos ejecutando este comando desde bash.

Este comando registra la cadena de entrada, que está en el hook de entrada por lo que verá los paquetes destinados a los procesos de nuestra máquina.

La prioridad es importante, ya que determina el orden de las cadenas, por lo tanto, si tenemos varias cadenas en el hook de entrada, podemos decidir qué tipo de paquete serán vistos antes que otros.

Si vamos a utilizar nftables para filtrar el tráfico de un pc de escritorio, es decir, un ordenador que no transmita tráfico, también se puede registrar la cadena output.

```
% nft add chain ip foo output { type filter hook output priority 0 \; }
```

Ahora ya podemos filtrar el tráfico entrante (dirigido a los procesos locales) y saliente (generado por los procesos locales).

Nota importante: Si no incluimos la configuración de la cadena que se especifica entre las llaves, se está creando una cadena “no-base” que no verá ningún paquete.

Desde nftables 0.5, también podemos especificar una política por defecto para las cadenas base como en iptables.

```
% nft add chain ip foo output { type filter hook output priority 0 \; policy accept\; }
```

Al igual que en iptables, las dos políticas predeterminadas son aceptar y tirar.

Al añadir una cadena en el hook de entrada, es obligatorio especificar el dispositivo por el que se captarán los paquetes:

```
% nft add chain netdev foo dev0filter { type filter hook ingress device eth0 priority 0 \; }
```

Tipos de cadena base

Los posibles tipos de cadena son:

- ***filter*** → obviamente se utiliza para filtrar paquetes. Esto es apoyado por las familias de las tablas arp, bridge, ip, ip6 e inet.
- ***route*** → se utiliza para reencaminar los paquetes si se modifica cualquier campo de la cabecera IP correspondiente o la mark del paquete. Si estamos familiarizados con iptables, este tipo de cadena proporciona la semántica equivalente a la tabla mangle. Esto es apoyado por las familias de la tabla ip e ip6.
- ***nat*** → se utiliza para llevar a cabo la traducción de direcciones de red (NAT). El primer paquete que pertenece a un flujo siempre dará con esta cadena, el seguimiento de paquetes no. Por lo tanto, nunca utlizaremos esta cadena para filtrar. Esto es apoyado por las familias de la tabla ip e ip6.

Los posibles hooks que se pueden utilizar cuando se configura la cadena son:

- ***prerouting*** → La decisión de enrutamiento de los paquetes no sucedió todavía, así que no se sabe si se dirigen al sistema local o remoto.
- ***input*** → Ocurre después de la decisión de encaminamiento, se verán los paquetes que se dirigen a nuestro sistema.

- ***forward*** → Sucede después de la decisión de encaminamiento, se pueden ver los paquetes que no están dirigidos a la máquina local.
- ***output*** → Captará los paquetes que se originan en la máquina local con destino otra máquina.
- ***postrouting*** → Después de la decisión de enrutamiento de los paquetes que salen del sistema local.
- ***ingress*** (sólo disponible en la familia netdev) → Desde el kernel Linux 4.2, podemos filtrar el tráfico antes de prerouting, después del que paquete se mande desde el controlador NIC. Por lo que tenemos una alternativa a tc.

La prioridad se puede utilizar para ordenar las cadenas o ponerlas antes o después de algunas operaciones internas de Netfilter. Por ejemplo, se colocará una cadena en el hook PREROUTING con la prioridad -300 antes de las operaciones de seguimiento de conexiones.

Como referencia, aquí está la lista de prioridad diferente utilizada en iptables:

- NF_IP_PRI_CONNTRACK_DEFRAG (-400): prioridad de desfragmentación.
- NF_IP_PRI_RAW (-300): prioridad tradicional de la tabla colocada en bruto antes de la operación de seguimiento de conexiones.
- NF_IP_PRI_SELINUX_FIRST (-225): operaciones de SELinux.
- NF_IP_PRI_CONNTRACK (-200): operaciones de seguimiento de conexión.
- NF_IP_PRI_MANGLE (-150): operación de mangle.
- NF_IP_PRI_NAT_DST (-100): NAT destino.
- NF_IP_PRI_FILTER (0): operación de filtrado, la tabla filter.
- NF_IP_PRI_SECURITY (50): Lugar de la tabla de seguridad donde secmark se puede establecer, por ejemplo.

- NF_IP_PRI_NAT_SRC (100): source NAT
- NF_IP_PRI_SELINUX_LAST (225): SELinux en la salida de paquetes
- NF_IP_PRI_CONNTRACK_HELPER (300): el seguimiento de conexiones en la salida

Añadiendo cadenas "no-base"

También podemos crear cadenas "no-base" como en iptables via:

```
% nft add chain ip foo test
```

Debemos de tener en cuenta que esta cadena no ve ningún tráfico, ya que no está unida a ningún hook, pero puede ser muy útil para organizar un conjunto de reglas en un árbol de cadenas utilizando el salto a la acción de la cadena.

Borrado de cadenas

Podemos eliminar las cadenas que no necesitamos, por ejemplo:

```
% nft delete chain ip foo input
```

La única condición es que la cadena que queremos borrar debe estar vacía, de lo contrario el kernel nos dirá que dicha cadena está en uso.

```
% nft delete chain ip foo input
<cmdline>:1:1-28: Error: Could not delete chain: Device or resource busy
delete chain ip foo input
^^^^^^^^^^^^^^^^^^^^^^^^^
```

Tendremos que limpiar el rulseset en esa cadena antes de poder eliminar la cadena.

Vaciado de cadenas

También podemos vaciar el contenido de una cadena. Si queremos eliminar todas las reglas en la cadena de entrada de la tabla foo, tendríamos que escribir:

```
nft flush chain foo input
```

Ejemplo de configuración: Filtrado de tráfico para nuestro equipo.

Podemos crear una tabla con dos cadenas base para definir reglas para filtrar el tráfico que viene y sale de nuestra máquina, asumiendo que la conectividad es ipv4.

```
% nft add table ip filter
% nft add chain ip filter input { type filter hook input priority 0 \; }
% nft add chain ip filter output { type filter hook output priority 0 \; }
```

Ahora, podemos comenzar a fijar normas para estas dos cadenas base. Debemos de tener en cuenta que no es necesaria la cadena forward en este caso ya que este ejemplo supone que está configurado nftables para filtrar el tráfico para un equipo cliente.

1.3.3 Gestión sencilla de reglas

Para añadir nuevas reglas, tenemos que especificar la tabla correspondiente y la cadena que queremos utilizar, por ejemplo:

```
% nft add rule filter output ip daddr 8.8.8.8 counter
```

Donde "filter" es la tabla y "output" es la cadena. El ejemplo anterior agrega una regla para que coincida con todos los paquetes vistor por la cadena de salida cuyo destino es 8.8.8.8, en caso de que coincida actualiza los contadores de reglas. Tengamos en cuenta que los contadores son opcionales en nftables.

Esto es equivalente al comando "iptables -A" en iptables.

Listado de reglas

Se pueden listar las reglas que están en una tabla con el siguiente comando:

```
% nft list table filter
table ip filter {
      chain input {
                  type filter hook input priority 0;
      }

      chain output {
                  type filter hook output priority 0;
                  ip  daddr  google-public-dns-a.google.com  counter
packets 0 bytes 0
      }
}
```

Suponemos que añadimos una regla que utiliza un puerto:

```
% nft add rule filter output tcp dport ssh counter
```

Podemos desactivar la resolución de nombre de host mediante el uso de la opción "-n":

```
% nft list -n table filter
```

```
table ip filter {
      chain input {
                type filter hook input priority 0;
      }

      chain output {
                type filter hook output priority 0;
                ip daddr 8.8.8.8 counter packets 0 bytes 0
                tcp dport ssh counter packets 0 bytes 0
      }
}
```

También se puede desactivar la resolución de nombres de un servicio con "-nn":

```
% nft list -nn table filter
table ip filter {
        chain input {
                type filter hook input priority 0;
        }

        chain output {
                type filter hook output priority 0;
                ip daddr 8.8.8.8 counter packets 0 bytes 0
                tcp dport 22 counter packets 0 bytes 0
        }
}
```

Comprobando las reglas

Vamos a probar esta regla con un simple ping a 192.168.1.1:

```
% ping -c 1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_req=1 ttl=64 time=1.31 ms
```

Entonces, si se indican el conjunto de reglas, obtenemos:

```
% nft -nn list table filter
table ip filter {
        chain input {
                type filter hook input priority 0;
        }
```

```
        chain                          output                          {
                type    filter    hook    output    priority    0;
                ip  daddr  8.8.8.8  counter  packets  1  bytes  84
                tcp   dport   22   counter   packets   0   bytes   0
        }
}
```

Comprobamos que los contadores se han actualizado.

Añadir una regla en una posición dada

Si queremos agregar una regla en una posición dada, tendremos que utilizar como referencia:

```
% nft list table filter -n -a
table filter {
      chain output {
                type filter hook output priority 0;
                ip protocol tcp counter packets 82 bytes 9680 #
handle 8
                ip  saddr  127.0.0.1  ip  daddr  127.0.0.6  drop  #
handle 7
      }
}
```

Si queremos añadir una regla después de la regla en la octava posición, tendremos que escribir:

```
% nft add rule filter output position 8 ip daddr 127.0.0.8 drop
```

Ahora, comprobamos la posición de la regla enumerando el conjunto de reglas:

```
%       nft       list       table       filter       -n       -a
table                            filter                            {
        chain                          output                          {
                type    filter    hook    output    priority    0;
                ip protocol tcp counter packets 190 bytes 21908 #
handle                                                            8
                ip    daddr    127.0.0.8    drop    #    handle    10
                ip  saddr  127.0.0.1  ip  daddr  127.0.0.6  drop  #
handle                                                            7
        }
}
```

Si queremos añadir una regla antes de la regla número 8, tenemos que hacerlo de la siguiente forma:

```
% nft insert rule filter output position 8 ip daddr 127.0.0.8 drop
```

Borrando reglas

Tendremos que obtener el identificador para eliminar una regla con la opción -a. El número de regla se asigna automáticamente por el núcleo que hace que se identifique una regla de manera única.

```
% nft list table filter -a
table ip filter {
      chain input {
                  type filter hook input priority 0;
      }

      chain output {
                  type filter hook output priority 0;
                  ip daddr 192.168.1.1 counter packets 1 bytes 84 #
handle 5
      }
}
```

Podemos eliminar la regla número 5 con el siguiente comando:

```
% nft delete rule filter output handle 5
```

Nota: Existen planes de apoyo al borrado de reglas, mediante:

```
% nft delete rule filter output ip saddr 192.168.1.1 counter
```

Pero esto aún no se ha implementado. Por lo que tenemos que utilizar el número de regla para eliminarlas hasta que se implemente esta característica.

Eliminación de todas las reglas de una cadena

Podemos eliminar todas las reglas de una cadena con el siguiente comando:

```
% nft delete rule filter output
```

También se pueden eliminar todas las reglas de una tabla:

```
% nft flush table filter
```

Anteponiendo nuevas reglas

Para anteponer nuevas reglas a través de la inserción de comandos:

```
% nft insert rule filter output ip daddr 192.168.1.1 counter
```

Esto antepone una regla que actualiza los paquetes por regla y los contadores para el tráfico dirigido a 192.168.1.1.

Lo equivalente en iptables es:

```
% iptables -I OUTPUT -t filter -d 192.168.1.1
```

Iptables siempre proporciona contadores por regla.

Sustitución de reglas

Podemos reemplazar cualquier regla a través del comando "replace", indicando el número de regla. Por lo tanto, primero tenemos que enumerar las reglas con la opción "-a" vista anteriormente:

```
# nft list ruleset -a
table ip filter {
      chain input {
                  type filter hook input priority 0; policy accept;
                  ip protocol tcp counter packets 0 bytes 0 # handle
2
      }
}
```

Asumiendo que queremos sustituir la regla número 2, tenemos que especificar este número y la nueva regla con la que queremos sustituir a esta:

```
nft replace rule filter input handle 2 counter
```

Entonces, si comprobamos de nuevo el número de reglas:

```
# nft list ruleset -a
table ip filter {
      chain input {
                  counter packets 0 bytes 0
      }
}
```

Podemos comprobar que la regla ha sido sustituida por una simple regla para contar los paquetes, en lugar de contar solo los paquetes TCP, que es lo que hacía la regla anterior.

1.3.4 Reemplazo masivo de reglas

Podemos actualizar las reglas de forma masiva con la opción *"-f"*

```
% nft -f file
```

Donde “file” contiene el conjunto de reglas:

Para guardar el conjunto de reglas en un fichero de la siguiente forma:

```
% nft list table filter > filter-table
```

Para restaurar el conjunto de reglas utilizaremos:

```
% nft -f filter-table
```

Si anteponemos “flush table filter” en el comienzo del archivo “filter-table”, será equivalente a los que realiza “iptables-restore”. Tengamos en cuenta que el kernel maneja los comandos de regla en el archivo en una sola transición, por lo que, el limpiado y la carga de nuevas reglas se hará de una sola vez.

Algunas personas prefieren tener un script con el conjunto de reglas. Aunque tenemos que tener cuidado con esto, ya que no podemos actualizar las reglas de forma masiva con un shell script. Por lo tanto es mejor hacerlo con la forma nativa de scripting nftables y para restaurar el conjunto de reglas con“*nft -f*”.

1.3.5 Informe de errores desde línea de comandos

La utilidad de línea de comandos “nft” trata de ayudar cuando se utiliza un tipo de datos incorrecto:

Los siguientes ejemplos muestran la salida de un error si se pasa una dirección ipv4 como un puerto TCP.

```
% nft add rule filter input tcp dport 1.1.1.1 counter drop
<cmdline>:1:33-39: Error: Could not resolve service: Servname not
supported for ai_socktype
add rule filter input tcp dport 1.1.1.1 counter drop
                               ^^^^^^^
```

Si el comando está incompleto el fallo será del siguiente tipo:

```
% nft add rule filter input tcp dport
<cmdline>:1:32-32: Error: syntax error, unexpected end of file
add rule filter input tcp dport
                             ^
```

1.3.6 Construcción de reglas con expresiones

nftables proporciona las siguientes operaciones integradas:

- ne → que significa diferente, como alternativa se puede usar " != "
- lt → significa menos que, alternativamente se puede usar " < "
- gt → mayor que, también se puede utilizar " > "
- le → menos que o igual, como alternativa tenemos " <= "
- ge →mayor que o igual, alternativamente se puede utilizar " >= "

Cuidado: Si utilizamos los símbolos " < " y " > " desde la shell, lo interpretara como entrada estándar y redirección de salida, respectivamente. Tendremos que utilizarlo de la siguiente forma " \< "

El siguiente puerto muestra cómo hacer coincidir todo el tráfico entrante que no venga con destino puerto TCP 22.

```
nft add rule filter input tcp dport != 22
```

Del mismo modo, también puede coincidir con el tráfico que viene a altos puertos con el siguiente comando:

```
nft add rule filter input tcp dport >= 1024
```

1.3.7 Operaciones a nivel de ruleset

Utilizando la sintaxis nativa de nft

El kernel linux 3.18 incluye algunas mejoras con respecto a las operaciones disponibles para administrar el ruleset en su conjunto.

Listado

Listar el ruleset completo:

```
% nft list ruleset
```

Listar el ruleset por familia:

```
% nft list ruleset arp
% nft list ruleset ip
% nft list ruleset ip6
% nft list ruleset bridge
% nft list ruleset inet
```

Estos comandos mostrarán todas las tablas/cadenas/conjuntos/reglas de la familia dada.

Limpiado

Además, también se podrá limpiar el ruleset al completo.

```
% nft flush ruleset
```

También por familia:

```
% nft flush ruleset arp
% nft flush ruleset ip
% nft flush ruleset ip6
% nft flush ruleset bridge
% nft flush ruleset inet
```

Copia de seguridad / restauración

Se pueden combinar estos dos comandos de arriba para hacer una copia de seguridad del ruleset:

```
% echo "nft flush ruleset" > backup.nft
% nft list ruleset >> backup.nft
```

Y cargarlo de forma automática:

```
% nft -f backup.nft
```

En formato JSON o XML

También se puede exportar el ruleset en formato JSON o XML.

En este caso, tendremos que utilizar el comando “export”:

```
% nft export xml > ruleset.xml
% nft export json > ruleset.json
```

Tengamos en cuenta que el comando “export” volcará todas las tablas de todas las familias (ip, ip6, inet, arp, bridge).

Actualmente se está trabajando en la operación “import” para XML y JSON.

1.3.8 Monitorización de actualizaciones del ruleset

Nft puede mostrar notificaciones de actualizaciones del ruleset a través de :

% nft monitor

Esto suscribe nft a cualquier tipo de actualización del ruleset.

Podemos filtrar los eventos por tipo de:

- objeto → tablas, cadenas, reglas, conjuntos y elementos.
- evento → nuevo y destruir.

El formato de salida puede ser:

- texto plano (formato nativo nft)
- XML
- JSON

El siguiente ejemplo muestra cómo seguir las actualizaciones de las reglas:

```
% nft monitor rules
```

En el caso de que sólo quisiéramos recibir las nuevas reglas:

```
% nft monitor new rules
```

Un ejemplo más desarrollado

Suponemos que escribimos lo siguiente en una terminal:

```
term1% nft monitor
```

Desde un terminal diferente, si escribimos esto:

```
term2% nft add table inet filter
term2% nft add chain inet filter forward
term2% nft add rule inet filter forward counter accept
term2% nft flush table inet filter forward
term2% nft flush ruleset
```

Entonces en la primer terminal veremos esto:

```
term1% nft monitor
add table inet filter
add chain inet filter forward
add rule inet filter forward counter packets 0 bytes 0 accept
delete rule inet filter forward handle 4
delete chain inet filter forward
```

```
delete table inet filter
```

1.3.9 Scripting

Muchas personas se decantan por mantener las reglas en un shell script, esto les permite añadir comentarios y organizar las reglas de una forma más sencilla. Esto es problemático, ya que los shell scripts rompen la atomicidad cuando se aplica al ruleset, por lo tanto, la política de filtrado se aplica de una manera inconsciente durante el tiempo de carga de las reglas.

Afortunadamente, nftables proporciona un entorno de programación nativo para evitar este tipo de preocupaciones, que básicamente permite incluir otros archivos ruleset, definir variables y añadir comentarios. Tenemos que restaurar el contenido de este script a través del comando *"nft -f my-ruleset.file"*

Para crear un script nftables, tenemos que añadir al encabezado del script esto:

```
#!/usr/sbin/nft
```

Adición de comentarios

Al igual que en muchos lenguajes de programación, podremos añadir comentarios a nuestro fichero usando el caracter " # ". Cualquier cosa después de "#" será ignorada. Ej:

```
#!/usr/sbin/nft
#                          table                          declaration
#
add                           table                            filter
#
#                          chain                          declaration
add chain filter input { type filter hook input priority 0; policy
drop;                                                              }
#                           rule                          declaration
add rule filter input ct state established,related counter accept
```

Incluyendo ficheros

El siguiente ejemplo muestra cómo incluir otros ficheros de reglas:

```
#!/usr/sbin/nft
include                                          "ipv4-nat.ruleset"
include "ipv6-nat.ruleset"
```

Definiendo variables

Se pueden definir variables, el siguiente ejemplo muestra un ruleset muy simple para tener en cuenta el trafico que viene de 8.8.8.8 (DNS Google):

```
#!/usr/sbin/nft
define google_dns = 8.8.8.8
add table filter
add chain filter input { type filter hook input priority 0; }
add rule filter input ip saddr $google_dns counter
```

También se pueden definir variables para un conjunto de direcciones:

```
define ntp_servers = { 84.77.40.132, 176.31.53.99, 81.19.96.148,
138.100.62.8 }
add table filter
add chain filter input { type filter hook input priority 0; }
add rule filter input ip saddr $ntp_servers counter
```

No olvidemos que los corchetes tienen una semántica especial, ya que indican que esta variable representa un conjunto. Por lo tanto hay que evitar cosas como:

```
define google_dns = { 8.8.8.8 }
```

Esto es un fallo común para definir un conjunto que sólo almacena un solo elemento, en lugar de utilizar la definición simple:

```
define google_dns = 8.8.8.8
```

1.3.10 Depuración/rastreo de ruleset

Desde nftables 0.6 y kernel linux 4.6, es compatible el rastreo del rulset.

Esto es equivalente al antiguo método de iptables “-J TRACE”, pero con algunas grandes mejoras.

Los pasos para habilitar la depuración/rastreo es la siguiente:

- Dar soporte en el ruleset para ello (conjunto nftrace en cualquiera de las reglas)
- Monitorizar los eventos de seguimiento de la herramienta nft.

Activando nftrace

Para habilitar nftrace en un paquete, utilizaremos esta regla con esta declaración:

```
meta nftrace set 1
```

Después de todo, nftrace es parte de la metainformación del paquete.

Por supuesto, es posible que sólo permitirá nftrace cuando exista la coincidencia dada. En el siguiente ejemplo, solo se permite nftrace para los paquetes TCP que utilizan la interfaz loopback:

```
iif lo ip protocol tcp meta nftrace set 1
```

Ajustar nftrace sólo a un subconjunto de los paquetes deseados es la clave para depurar adecuadamente el ruleset, de lo contrario podemos obtener una gran cantidad de información que puede ser abrumadora.

Monitorización de los eventos de rastreo

En nftables, conseguir la depuración/rastreo de eventos es un poco diferente del mundo iptables. Ahora, tenemos un monitor basado en eventos para el kernel para notificar a la herramienta nft.

La sintaxis básica es:

```
% nft monitor trace
```

A cada evento de rastreo se le asigna un id para que sea más fácil seguir los diferentes paquetes en la misma sesión de rastreo.

Ejemplo

Aquí un par de ejemplos completos de este mecanismo de depuración/rastreo:

Test simple de rastreo:

```
% nft add rule inet filter input iif lo counter nftrace set 1 accept
% nft monitor trace
trace id 530fa6dd inet filter input packet: iif lo
trace id 530fa6dd inet filter input rule iif lo accept (verdict accept)
trace id 87a375ea inet filter input packet: iif lo
trace id 87a375ea inet filter input rule iif lo accept (verdict accept)
```

Rastreo de dos tipos diferentes de paquetes en la misma sesión:

```
% nft filter input tcp dport 10000 nftrace set 1
```

```
% nft filter input icmp type echo-request nftrace set 1
% nft -nn monitor trace
trace id e1f5055f ip filter input packet: iif eth0 ether saddr 63:f6:4b:00:54:52 ether daddr c9:4b:a9:00:54:52 ip saddr 192.168.122.1 ip daddr 192.168.122.83 ip tos 0 ip ttl 64 ip id 32315 ip length 84 icmp type echo-request icmp code 0 icmp id 10087 icmp sequence 1
trace id e1f5055f ip filter input rule icmp type echo-request nftrace set 1 (verdict continue)
trace id e1f5055f ip filter input verdict continue
trace id e1f5055f ip filter input
trace id 74e47ad2 ip filter input packet: iif vlan0 ether saddr 63:f6:4b:00:54:52 ether daddr c9:4b:a9:00:54:52 vlan pcp 0 vlan cfi 1 vlan id 1000 ip saddr 10.0.0.1 ip daddr 10.0.0.2 ip tos 0 ip ttl 64 ip id 49030 ip length 84 icmp type echo-request icmp code 0 icmp id 10095 icmp sequence 1
trace id 74e47ad2 ip filter input rule icmp type echo-request nftrace set 1 (verdict continue)
trace id 74e47ad2 ip filter input verdict continue
trace id 74e47ad2 ip filter input
trace id 3030de23 ip filter input packet: iif vlan0 ether saddr 63:f6:4b:00:54:52 ether daddr c9:4b:a9:00:54:52 vlan pcp 0 vlan cfi 1 vlan id 1000 ip saddr 10.0.0.1 ip daddr 10.0.0.2 ip tos 16 ip ttl 64 ip id 59062 ip length 60 tcp sport 55438 tcp dport 10000 tcp flags == syn tcp window 29200
trace id 3030de23 ip filter input rule tcp dport 10000 nftrace set 1 (verdict continue)
trace id 3030de23 ip filter input verdict continue
trace id 3030de23 ip filter input
```

1.4. Selectores de coincidencia de paquetes soportados

1.4.1 Coincidencia de campos según cabecera de paquetes.

La utilidad “nft” es compatible con la capa 4 de protocolos: AH, ESP, UDP, UDPlite, TCP, DCCP, SCTP y IPComp.

Coincidencia de protocolo de transporte

La siguiente regla muestra cómo hacer coincidir cualquier tipo de tráfico TCP:

```
% nft add rule filter output ip protocol tcp
```

Coincidencia de campo cabecera ethernet

Si queremos hacer coincidir el tráfico Ethernet cuya dirección sea destino "ff:ff:ff:ff:ff:ff", debemos hacerlo de la siguiente forma:

```
% nft add rule filter input ether daddr ff:ff:ff:ff:ff:ff counter
```

No olvidemos que la información de cabecera de capa 2 sólo está disponible en la ruta input.

Coincidencia de campo cabecera IPV4

También puede coincidir con tráfico en función del origen y destino, el siguiente ejemplo muestra como tener en cuenta todo el tráfico con origen "192.168.1.100" y que está dirigido a "192.168.1.1":

```
% nft add rule filter input ip saddr 192.168.1.100 ip daddr 192.168.1.1 counter
```

Tengamos en cuenta que, dado que la regla se une a la cadena de input, nuestra máquina tendrá que tener la dirección 192.168.1.1, de lo contrario no habrá ninguna coincidencia.

Para filtrar un protocolo de capa 4, como TCP, podemos utilizar la palabra clave "protocol":

```
% nft add rule filter input protocol tcp counter
```

Coincidencia de campo cabecera IPV6

Si queremos tener en cuenta el tráfico IPv6 que se dirige a "abdc::100", utilizaremos la siguiente regla:

```
% nft add rule filter output ip6 daddr abcd::100 counter
```

Para filtrar un protocolo de capa 4, como TCP, utilizaremos la palabra "nexthdr":

```
% nft add rule filter input ip6 nexthdr tcp counter
```

No debemos olvidar crear una tabla ip6 y registrar las correspondientes cadenas.

Coincidencia de tráfico TCP/UDP/UDPlite

Los siguientes ejemplos muestran cómo eliminar todo el tráfico TCP de puertos bajos:

```
% nft add rule filter input tcp dport 1-1024 counter drop
```

Tengamos en cuenta que esta regla está utilizando un intervalo (de 1 a 1024) .

Para hacer coincidir los indicadores TCP, es necesario utilizar una operación binaria.

Por ejemplo, para contar los paquetes que no son “SYN”:

```
% nft add rule filter input tcp flags != syn counter
```

Más filtros complejos pueden ser usados, por ejemplo, para contar y registrar paquetes TCP con indicadores SYN y ACK:

```
% nft -i
nft> add rule filter output tcp flags & (syn | ack) == (syn | ack)
counter log
```

Coincidencia de tráfico ICMP

Podemos eliminar toda solicitud de “echo” ICMP a través de:

```
% nft add rule filter input icmp type echo-request counter drop
```

Esta es la lista de los distintos tipos disponibles icmp:

`echo-reply, destination-unreachable, source-quench, redirect, echo-request, time-exceeded, parameter-problem, timestamp-request, timestamp-reply, info-request, info-reply, address-mask-request, address-mask-reply.`

1.4.2 Coincidencia de paquetes según metainformacion

Nftables viene con selectores de paquetes según su metainformacion, se pueden utilizar para que coincida la metainformacion que se almacenan en el paquete.

Los meta selectores

La metainformacion actual que se puede encontrar son las siguientes:

- Nombre de la interfaz e índice del dispositivo: iifname, oifname, iif y oif.
- Tipo de interfaz: iiftyte y oiftype.
- Prioridad: priority
- Usuario y grupo identificador del socket: skuid y skgid
- Longitud del paquete: lenght

Coincidencia de paquetes por nombre de interfaz

Se pueden utilizar los siguientes selectores para que coincida con el nombre de la interfaz:

- iifname → Para que coincida con el nombre de la interfaz de entrada.
- oifname → Para que coincida con el nombre de la interfaz de salida.
- iif → Para que coincida con el índice de la interfaz de red. Esto es más rápido que "iifname" ya que solo tiene que comparar 32 bits en lugar de una cadena. El índice de la interfaz se asigna dinámicamente, así que no utilizar esto para las interfaces que son creadas y destruidas dinámicamente, ej : ppp0.
- oif → como iif pero coincide con el índice de interfaz de salida.

Un ejemplo de uso de nombre de la interfaz:

```
% nft add rule filter input meta oifname lo accept
```

Esta regla acepta todo el tráfico para el dispositivo lo.

Coincidencia por mark de paquete

Se pueden hacer coincidir paquetes cuya mark es 123 con la siguiente regla:

```
nft add rule filter output meta mark 123 counter
```

Coincidencia de paquetes por socket UID

Podemos utilizar el nombre de un usuario para hacer coincidir el tráfico, por ejemplo:

```
% nft add rule filter output meta skuid manuel counter
```

Y en el caso de que no exista una entrada para el usuario en el fichero /etc/passwrd, podemos utilizar el UID.

```
% nft add rule filter output meta skuid 1000 counter
```

Por ejemplo, si generamos algo de tráfico, podemos comprobar como los contadores se actualizan si miramos la regla.

```
% nft list table filter
table ip filter {
```

```
        chain output {
                type filter hook output priority 0;
                skuid manuel counter packets 7 bytes 510
        }

        chain input {
                type filter hook input priority 0;
        }
}
```

1.4.3 Coincidencia según estado de conexión

Como en iptables, podemos hacer coincidir los paquetes con la información de seguimiento de estado (a veces referido como "conntrack" o "cr information") que Netfilter recoge a través del sistema de conexiones para implementar cortafuegos de estado.

Coincidencia de información conntrack

Nftables proporciona el selector "ct" que se puede utilizar para que concida con:

- Información de estado: new, established, related e invalid. En este sentido, no hay cambios con iptables.
- La mark conntrack

Coincidencia de información de estado

El siguiente ejemplo muestra como implementar un cortafuegos de estado sencillo con nftables:

```
nft add rule filter input ct state established,related counter accept
nft add rule filter input counter drop
```

La primera regla permite los paquetes que son parte de una comunicación ya establecida con la red. De este modo, se eliminará cualquier tipo de intento de conexión de un ordenador que quiera comunicarse con el nuestro. Sin embargo, el tráfico que forma parte de un flujo que nosotros hemos iniciado será aceptado. Tenemos que tener en cuenta que el ejemplo anterior utiliza una lista de los estados a utilizar separados por comas .

Coincidencia con conntrack mark

El siguiente ejemplo muestra cómo hacer coincidir los paquetes en base a la conntrack mark :

```
nft add rule filter input ct mark 123 counter
```

Para saber más acerca de las conntrack mark y las marks de paquetes, ver configuración de metainformacion de paquetes.

1.4.4 Coincidencia según límites

Podemos filtrar el tráfico marcando límites. El siguiente ejemplo muestra cómo aceptar un máximo de 10 peticiones de echo ICMP por segundo:

```
% nft add rule filter input icmp type echo-request limit rate 10/second accept
```

Desde el kernel linux 4.3, también podemos establecer límites por bytes:

```
% nft add rule filter input limit rate 10 mbytes/second accept
```

La regla anterior acepta el tráfico por debajo de la tasa de 10 mbytes por segundo.

También se puede utilizar el parámetro “burst” para indicar el número de paquetes/bytes que puede exceder el límite.

```
% nft add rule filter input limit rate 10 mbytes/second burst 9000 kbytes accept
```

Esto indica que se puede superar el límite en 9000 kbytes.

También se pueden establecer límites para los paquetes:

```
% nft add rule filter input icmp type echo-request limit rate 10/second burst 2 packets counter accept
```

Por lo que pueden exceder el límite 2 paquetes.

La expresión “limit” se puede utilizar para vigilar el tráfico de entrada también, como alternativa a “tc” de la nueva familia netdev.

1.5 Acciones posibles sobre paquetes

1.5.1 Aceptando y descartando paquetes

Descartando paquetes

Se puede utilizar la opción “drop” para descartar paquetes. Tenemos que tener en cuenta que esta es una acción de terminación, es decir, una vez que se realiza la acción “drop” , el paquete es desechado y no se puede realizar ninguna otra acción sobre este.

```
nft add rule filter output drop
```

Esta acción es probable que nos deje sin conectividad a internet ya que se descartan todos los paquetes salientes.

Aceptando paquetes

Una regla sencilla para aceptar cualquier tipo de tráfico es:

```
nft add rule filter output accept
```

Sería más lógico añadir un contador a la regla:

```
nft add rule filter output counter accept
```

Con esto podemos contabilizar el tráfico aceptado:

```
nft list table filter
table ip filter {
      chain output {
                  type filter hook output priority 0;
                  counter packets 1 bytes 84 accept
      }
}
```

1.5.2 Saltando a cadena

Como en iptables, se puede estructurar el conjunto de reglas en modo árbol. Para ello, primero tenemos que crear la cadena personalizada a través de:

```
% nft add chain ip filter tcp-chain
```

El ejemplo anterior crea la cadena tcp-chain que se utilizará para añadir reglas para filtrar el tráfico TCP, por ejemplo:

```
% nft add rule ip filter input ip protocol tcp jump tcp-chain
```

Ahora podemos añadir una regla simple para que la cadena “tcp-chain” cuente los paquetes y los bytes:

```
% nft add rule ip filter tcp-chain counter
```

La lista debería mostrar algo similar a:

```
% nft list table filter
table ip filter {
      chain input {
                  type filter hook input priority 0;
                  ip protocol tcp jump tcp-chain
      }

      chain tcp-chain {
                  counter packets 8 bytes 2020
      }
}
```

Los contadores deben actualizarse a medida que se genere tráfico TCP.

Nota: solo se puede saltar a las cadenas “no-base”.

Jump vs goto

Debemos tener en cuenta la diferencia entre “jump” y “goto”.

Si utilizamos “jump” para obtener el paquete procesado en otra cadena, el paquete volverá a la cadena de la regla llamada después del final.

Sin embargo, si se utiliza “goto”, los paquetes se procesan en otra cadena, pero no volverán a la cadena de la regla llamada. En este caso, la política por defecto aplicada al paquete será la política por defecto de la cadena base que comenzó a procesar el paquete.

Ejemplo:

El paquete es: SRC=1.1.1.1 DST=2.2.2.2 TCP SPORT 111 DPORT 222

```
table ip filter {
      chain input {
                  type filter hook input priority 0; policy accept;
                  ip saddr 1.1.1.1 ip daddr 2.2.2.2 tcp dport 111
tcp dport 222 jump other-chain
```

```
                ip saddr 1.1.1.1 ip daddr 2.2.2.2 tcp dport 111
tcp dport 222 accept
      }
      chain other-chain {
                counter packets 8 bytes 2020
      }
}
```

Ejemplo de “goto”:

```
table ip filter {
      chain input {
                type filter hook input priority 0; policy accept;
                ip saddr 1.1.1.1 ip daddr 2.2.2.2 tcp dport 111
tcp dport 222 goto other-chain
                ip saddr 1.1.1.1 ip daddr 2.2.2.2 tcp dport 111
tcp dport 222 accept
      }
      chain other-chain {
                counter packets 8 bytes 2020
      }
}
```

1.5.3 Rechazando tráfico

La siguiente regla muestra cómo rechazar cualquier tráfico de la red:

```
% nft add rule filter input reject
```

Si no se especifica ninguna razón, el paquete ICMP/ICMPv6 “port unreachable” se envía al origen.

Podemos filtrar esto a través del selector “ct”, así que esto sólo rechaza el tráfico que viene a la máquina local que no se originó por nosotros.

```
% nft add rule filter input ct state new reject
```

También se puede especificar el motivo de rechazo, por ejemplo:

```
% nft add rule filter input reject with icmp type host-unreachable
```

Para los paquetes ICMP, se pueden utilizar las siguientes razones de rechazo:

- net-unreachable: Red de destino inaccesible

- host-unreachable: Host inaccesible
- prot-unreachable: Protocolo de destino inaccesible
- port-unreachable: Puerto destino inaccesible
- net-prohibited: Red prohibida administrativamente
- host-prohibited: Host prohibido administrativamente
- admin-prohibited: Comunicación prohibida administrativamente

También podemos rechazar el tráfico IPV6 indicando el motivo de rechazo, por ejemplo:

```
% nft add rule ip6 filter input reject with icmpv6 type no-route
```

Para ICMPv6, podemos utilizar las siguientes razones:

- no-route: No hay ruta hasta el destino.
- admin-prohibited: Comunicación prohibida administrativamente
- addr-unreachable: Dirección inaccesible
- port-unreachable: Puerto inaccesible

Para la familia inet, podemos utilizar una abstracción, el llamado icmpx, para rechazar el tráfico IPv4 e IPv6 usando una sola regla, por ejemplo:

```
% nft add rule inet filter input reject with icmpx type no-route
```

Esta regla rechaza el tráfico IPv4 con el motivo "net unreachable" y el tráfico IPv6 con el motivo "no route". El mapeo se muestra en la siguiente tabla:

ICMPX REASON	ICMPv6	ICMPv4
admin-prohibited	admin-prohibited	admin-prohibited
port-unreachable	port-unreachable	port-unreachable
no-route	no-route	net-unreachable
host-unreachable	addr-unreachable	host-unreachable

1.5.4 Registrando el tráfico

Nota: El soporte de registro completo (log) está disponible a partir del kernel linux 3.17. Si estamos utilizando un kernel más antiguo, tendremos que cargar el módulo "ipt_LOG" para habilitar el registro.

Podemos registrar los paquetes utilizando la acción “log”. La regla más simple para registrar todo el tráfico de entrada es:

```
% nft add rule filter input log
```

Una regla típica, es registrar y aceptar el tráfico entrante por ssh:

```
% nft add rule filter input tcp dport 22 ct state new log prefix \"New SSH connection: \" accept
```

“Prefix” indica la cadena inicial que se utiliza como prefijo para el mensaje del log.

Tengamos en cuenta que nftables permite llevar a cabo dos acciones en una sola regla, contrariamente a iptables que requería dos reglas para esto.

También tenemos que tener en cuenta que la regla se consulta de izquierda a derecha, así que si tenemos la siguiente regla:

```
nft add rule filter input iif lo log tcp dport 22 accept
```

Registrará todos los paquetes que vienen en la interfaz de Loopback y no solo los que vienen con destino puerto 22.

1.5.5 Traducción de direcciones de red (NAT)

La cadena nat permite realizar NAT. Este tipo de cadena viene con una semántica especial:

- El primer paquete de un flujo se utiliza para buscar una regla que coincida, que establecerá la unión NAT para este flujo. En consecuencia, esto también manipula este primer paquete.
- Ninguna búsqueda de regla pasa para el seguimiento de paquetes en el flujo: el motor NAT utiliza la información de enlace NAT ya establecido por el primer paquete para llevar a cabo la manipulación de paquetes.

Source NAT

Si queremos hacer Source NAT con el tráfico que sale de nuestra red local a internet, podemos crear una tabla nat con las cadenas de “prerouting” y “postrouting”:

```
% nft add table nat
% nft add chain nat prerouting { type nat hook prerouting priority 0 \; }
```

```
% nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
```

Entonces añadimos la siguiente regla:

```
% nft add rule nat postrouting ip saddr 192.168.1.0/24 oif eth0 snat 1.2.3.4
```

Esto hará coincidir todo el tráfico de la red 192.168.1.0/24 que salga por la interfaz eth0. La dirección IPv4 1.2.3.4 se utiliza como origen para los paquetes que coincidan con esta regla.

Tenemos que registrar la cadena "PREROUTING", incluso si no tenemos reglas en esa cadena, pero esta cadena invoca al motor NAT para los paquetes que vienen en el path de INPUT.

Destination NAT

Es necesario agregar la siguiente configuración de tabla y cadenas:

```
% nft add table nat
% nft add chain nat prerouting { type nat hook prerouting priority 0 \; }
% nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
```

A continuación podemos añadir la siguiente regla:

```
% nft add rule nat prerouting iif eth0 tcp dport { 80, 443 } dnat 192.168.1.120
```

Esto hace que se redirija el tráfico de entrada por los puertos 80 y 443 a 192.168.1.120. No debemos olvidar registrar la cadena POSTROUTING ya que esto invoca al motor NAT para el seguimiento de paquetes que van en la dirección de respuesta.

Enmascaramiento

Nota: esta función está disponible a partir del kernel linux 3.18.

El enmascaramiento es un caso especial de SNAT, donde la dirección de origen se establece automáticamente a la dirección de interfaz de salida. Por ejemplo:

```
% nft add rule nat postrouting masquerade
```

Redireccionamiento

Nota: esta función está disponible a partir del Kernel linux 3.19

Mediante el uso de redireccionamiento, los paquetes serán enviados al equipo local. Es un caso especial de DNAT, donde el destino es el equipo actual.

```
% nft add rule nat prerouting redirect
```

En este ejemplo se redirige el tráfico tcp por el puerto 22 a el 2222:

```
% nft add rule nat prerouting tcp dport 22 redirect to 2222
```

Tenemos que tener en cuenta que:

- "redirect" sólo tiene sentido en una cadena PREROUTING de tipo NAT.
- Tenemos que registrar una cadena POSTROUTING NAT, por lo que el tráfico se traduce en la dirección de respuestas

Flags NAT

Desde el kernel linux 3.18, se pueden combinar las siguientes flags con los estados de NAT:

- random → aleatorizar la asignación de puertos de origen.
- fully-random → aleatorización completa de puertos.
- persistent → da a un cliente la misma dirección origen/destino para cada conexión.

Por ejemplo:

```
% nft add rule nat postrouting masquerade random,persistent
% nft add rule nat postrouting ip saddr 192.168.1.0/24 oif eth0
snat 1.2.3.4 fully-random
```

Incompatibilidades

No se puede usar iptables y nft para realizar NAT al mismo tiempo. Así que debemos asegurarnos que el módulo "iptables_nat" no esté cargado:

```
% rmmod iptable_nat
```

1.5.6 Configuración de metainformación de paquetes

Se puede establecer alguna metainformacion en un paquete: mark, priority o nftrace. Para utilizar estas funciones se requiere tener kernel linux superior a 3.14.

Mark

El siguiente ejemplo muestra cómo establecer la mark de paquetes:

```
% nft add rule route output mark set 123
```

Mark y conntrack mark

Podemos guardar y restaurar la conntrack mark como en iptables.

En este ejemplo, el motor "nf_tables" estableció mark a 1. En la última regla, esa mark se almacena en la entrada conntrack asociada con el flujo.

```
 % nft add rule filter forward meta mark 1
% nft add rule filter forward ct mark set mark
```

En este ejemplo, la conntrack mark se almacena en el paquete.

```
% nft add rule filter forward meta mark set ct mark
```

Prioridad

Se puede establecer la prioridad de un paquete. Este ejemplo muestra una operación similar a lo que haría iptables con "-j CLASSIFY":

```
% nft add table mangle
% nft add chain postrouting {type route hook output priority -
150\; }
% nft add rule mangle postrouting tcp sport 80 meta priority set 1
```

Nftrace

Configurar nftrace en un paquete guardara la ruta a través de la pila "nf_tables":

```
% nft add rule filter forward udp dport 53 meta nftrace set 1
```

Combinación de opciones

Dado el diseño flexible de nftables, tenemos que recordar que se pueden realizar varias acciones a un paquete en una regla:

```
% nft add rule filter forward ip saddr 192.168.1.1 meta nftrace
set 1 meta priority set 2 meta mark set 123
```

1.5.7 Duplicación de paquetes

Desde el kernel linux 4.3, podemos duplicar los paquetes a otra destino de las familias ip e ip6. Se puede utilizar esta función para hacer frente a este tráfico a otro interlocutor remoto para continuar la inspección.

La siguiente regla duplica todo el tráfico a 172.20.0.2:

```
% nft add rule mangle prerouting dup to 172.20.0.2
```

También se puede forzar el estado de la duplicación para utilizar un determinado dispositivo para enrutar el tráfico a partir de:

```
% nft add rule mangle prerouting dup to 172.20.0.2 device eth1
```

Por lo tanto, la regla anterior indica que los paquetes duplicados deben ir via th1. No debemos olvidar tener una ruta para llegar a 172.20.0.2 a través de ethi1, de lo contrario esto no funcionará.

También es posible combinar el estado de la duplicación con los mapas, por ejemplo:

```
% nft add rule mangle prerouting dup to ip saddr map { 192.168.0.1
: 172.20.0.2, 192.168.0.1 : 172.20.0.3 }
```

En esta regla anterior, el destino que se utiliza para duplicar paquetes depende de la dirección IPv4 de origen.

1.5.8 Contadores

Los contadores son opcionales en nftables, por lo tanto, es necesario especificar explícitamente en la regla si lo queremos.

El siguiente ejemplo permite contar todo el tráfico TCP que recibe nuestra máquina:

```
% nft add rule filter input ip protocol tcp counter
```

Una característica interesante de la acción del contador es que su posición en la sintaxis de la regla importa. Esta regla no es equivalente a la regla anterior:

```
% nft add rule filter input counter ip protocol tcp
```

La regla se evalúa de izquierda a derecha, por lo que cualquier tipo de paquete actualiza los contadores, no solo los paquetes TCP.

1.6 Estructura de datos avanzadas para el rendimiento de clasificación de paquetes

1.6.1 Conjuntos

Nftables viene con una infraestructura de conjunto genérico incorporado que le permite utilizar cualquier selector soportado para construir conjuntos. Esta infraestructura hace posible la representación de los diccionarios y mapas.

Los elementos de conjunto se representan internamente usando estructuras de datos, tales como tablas hash y árboles rojo-negro.

Conjuntos anónimos

Conjuntos anónimos son aquellos que son:

- Unido a una regla, si se elimina la regla, ese conjunto se libera también.
- No tienen ningún nombre específico, el kernel asigna un identificador interno.
- No se pueden actualizar. Así no se pueden añadir y eliminar elementos de él una vez que se una a una regla.

El siguiente ejemplo muestra cómo crear un conjunto sencillo:

```
% nft add rule filter output tcp dport { 22, 23 } counter
```

Esta regla anterior atrapa todo el tráfico que va a los puertos TCP 22 y 23, en caso de coincidir los contadores se actualizan.

Conjuntos nombrados

Podemos crear conjuntos con nombre con el siguiente comando:

```
% nft add set filter blackhole { type ipv4_addr\;}
```

Tenemos que tener en cuenta que “blackhole” es el nombre del conjunto en este caso. La opción “type” indica el tipo de datos que se almacenará ese conjunto, lo cual es una dirección IPv4. El máximo de caracteres permitidos son 16.

```
% nft add element filter blackhole { 192.168.3.4 }
% nft add element filter blackhole { 192.168.1.4, 192.168.1.5 }
```

A continuación, se puede utilizar para establecer la regla:

```
% nft add rule ip input ip saddr @blackhole drop
```

Los tipos de datos soportados son:

- ipv4_addr:Dirección IPv4.
- ipv6_addr: Dirección IPv6 .
- ether_addr: Dirección Ethernet.
- inet_proto: Tipo de protocolo.
- inet_service: Servicios de internet (leer puerto TCP por ejemplo)
- mark: Tipo de marca.

Los conjuntos con nombre se pueden actualizar en cualquier momento, por lo que se pueden añadir y eliminar elementos de ellos.

Aquí tenemos un ejemplo que compara iptables con nftables:

```
ip6tables -A INPUT -p tcp -m multiport --dports 22,80,443 -j ACCEPT
ip6tables -A INPUT -p icmpv6 --icmpv6-type neighbor-solicitation -j ACCEPT
ip6tables -A INPUT -p icmpv6 --icmpv6-type echo-request -j ACCEPT
ip6tables -A INPUT -p icmpv6 --icmpv6-type router-advertisement -j ACCEPT
ip6tables -A INPUT -p icmpv6 --icmpv6-type neighbor-advertisement -j ACCEPT
```

Lo que en nftables sería:

```
% nft add rule ip6 filter input tcp dport {telnet, http, https} accept
% nft add rule ip6 filter input icmpv6 type { nd-neighbor-solicit, echo-request, nd-router-advert, nd-neighbor-advert } accept
```

Listando conjuntos nombrados

Se puede listar el contenido de un conjunto nombrado via:

```
% nft list set filter myset
```

1.6.2 Diccionarios

Los diccionarios, también conocidos como mapas de veredicto, son una de las característica más interesantes disponibles en nftables. Básicamente, permiten adjuntar una acción para un elemento. Utilizan internamente la infraestructura de conjunto genérico.

Diccionarios literales

El siguiente ejemplo muestra cómo crear un árbol de cadenas cuyo recorrido depende la capa 4 de tipo protocolo:

```
% nft add rule ip filter input ip protocol vmap { tcp : jump tcp-chain, udp : jump udp-chain , icmp : jump icmp-chain }
% nft add rule ip filter input counter drop
```

Este ejemplo anterior supone que ya hemos creado las cadenas “tcp-chain”, “udp-chain” e “icmp-chain”. Entonces, uniendo reglas simples para contar el tráfico, es decir:

```
% nft add rule filter icmp-chain counter
% nft add rule filter tcp-chain counter
% nft add rule filter udp-chain counter
```

Se puede comprobar que la clasificación está realizándose:

```
% nft list table filter
table ip filter {
        chain input {
                type filter hook input priority 0;
                ip protocol vmap { udp : jump udp-chain, tcp :
jump tcp-chain, icmp : jump icmp-chain}
        }
        chain tcp-chain {
                counter packets 4 bytes 520
        }
        chain udp-chain {
                counter packets 4 bytes 678
        }
        chain icmp-chain {
                counter packets 4 bytes 336
        }
}
```

Una de las cosas buenas de nftables es que se pueden utilizar diccionarios para construir conjunto de reglas. Este conjunto de reglas permite reducir la cantidad de inspecciones lineales para clasificar los paquetes.

Declaraciones

También se pueden declarar diccionarios y poblar su contenido de forma dinámica, por ejemplo:

```
% nft add map filter mydict { type ipv4_addr : verdict\; }
```

Entonces podremos agregar elementos:

```
% nft add element filter mydict { 192.168.0.10 : drop,
192.168.0.11 : accept }
```

Podemos enlazar este conjunto de reglas usando el siguiente comando:

```
% nft add rule filter input ip saddr vmap @mydict
```

En caso de querer construir nuestro propio diccionario, pero no sabemos el nombre del tipo de datos, podemos consultar nft para saber el nombre:

```
% nft describe tcp dport
payload expression, datatype inet_service (internet network service) (basetype integer), 16 bits
```

Ver la cadena “inet_service” después del tipo de datos.

1.6.3 Intervalos

Los intervalos se expresan como valor-valor.

El siguiente ejemplo muestra cómo desechar el tráfico que viene del ranfo 192.168.0.1 a 192.168.0.250:

```
% nft add rule filter input ip daddr 192.168.0.1-192.168.0.250 drop
```

También se puede utilizar con los puertos TCP:

```
% nft add rule filter input tcp ports 1-1024 drop
```

Y cualquier tipo de valor constante.

También pueden utilizarse intervalos de conjuntos, el siguiente ejemplo muestra cómo sería la lista negra de dos intervalos de direcciones IP.

```
% nft add rule ip filter input ip saddr { 192.168.1.1-192.168.1.200, 192.168.2.1-192.168.2.200 } drop
```

Y también se puede utilizar en los diccionarios:

```
% nft add rule ip filter forward ip daddr vmap { 192.168.1.1-192.168.1.200 : jump chain-dmz, 192.168.2.1-192.168.20.250 : jump chain-desktop }
```

1.6.4 Mapas

Los mapas son otra característica interesante que ha estado en nftables desde el principio. Podemos usar un mapa para buscar los datos basados en alguna tecla específica que se utiliza como entrada. Estos mapas usan internamente la infraestructura de conjunto genérico.

Mapas literales

El siguiente ejemplo muestra cómo el puerto destino TCP selecciona la dirección ip destino del paquete DNAT.

```
% nft add rule ip nat prerouting dnat tcp dport map { 80 : 192.168.1.100, 8888 : 192.168.1.101 }
```

Se trata de cómo se puede expresar una redirección de puertos clásica cuando el servidor real se encuentra detrás del firewall. Esto se puede leer como:

- Si el puerto TCP es 80, entonces el paquete hará DNAT a 192.168.1.100.
- Si el puerto TCP destino es 8888, entonces el paquete hará DNAT hacia 192.168.1.101.

Declarando mapas

También podemos declarar mapas, que se pueden rellenar de forma dinámica, por ejemplo:

```
% nft add map nat porttoip  { type inet_service: ipv4_addr\; }
% nft add element nat porttoip { 80 : 192.168.1.100, 8888 : 192.168.1.101 }
```

Este ejemplo muestra que el mapa "porttoip" mantiene la correspondencia entre servicios de internet (que es el tipo de datos de un puerto TCP destino) y direcciones IP.

A continuación, se puede utilizar a partir de una regla como:

```
% nft add rule ip nat postrouting snat tcp dport map @porttoip
```

Esta regla dice que la dirección IP de origen que se utiliza para SNAT depende del puerto TCP destino. El contenido del mapa puede actualizarse de forma dinámica mediante la adición de nuevos elementos.

1.6.5 Concatenaciones

Desde el kernel Linux 4.1 nftables soporta concatenaciones.

Esta nueva característica permite poner dos o más selectores juntos para realizar búsquedas rápidamente combinándolos con conjuntos, diccionarios y mapas.

Conjuntos literales

```
% nft add rule ip filter input ip saddr . ip daddr . ip protocol { 1.1.1.1 . 2.2.2.2 . tcp, 1.1.1.1 . 3.3.3.3 . udp} counter accept
```

Así que si el paquete tiene dirección IP origen e IP destino y el puerto TCP destino coincide:

- 1.1.1.1 y 2.2.2.2 y TCP.

  o

- 1.1.1.1 y 3.3.3.3 y UDP.

Nftables actualiza el contador para esta regla y luego acepta el paquete.

Declaración de diccionarios

En el siguiente ejemplo se crea el diccionario "whitelist" usando una concatenación de dos selectores:

```
% nft add map filter whitelist { type ipv4_addr . inet_service : verdict \; }
```

Una vez creado el diccionario, se puede utilizar para una norma que crea la siguiente concatenación:

```
% nft add rule filter input ip saddr . tcp dport vmap @whitelist
```

Por lo tanto, la regla anterior utiliza veredicto basado en la dirección IP origen y el puerto TCP destino.

Dado que el diccionario está inicialmente vacío, se puede rellenar dinámicamente con este diccionario de elementos a través de:

```
% nft add element filter whitelist { 1.2.3.4 . 22 : accept}
```

Mapas literales

La siguiente regla determina la dirección IP destino que se utiliza para realizar DNAT, pero con el paquete basándose en la dirección IP origen y el puerto TCP destino:

```
% nft add rule ip nat prerouting dnat ip saddr . tcp dport map { 1.1.1.1 . 80 : 192.168.1.100, 2.2.2.2 . 8888 : 192.168.1.101 }
```

1.7 Ejemplos

1.7.1 Ruleset simple para un workstation

fw.basic

```
table ip filter {
```

```
    chain input {
        type filter hook input priority 0;
        # accept traffic originated from us
        ct state established,related accept

        # accept any localhost traffic
        iif lo accept
        # count and drop any other traffic
        counter drop
    }
}
```

fw6.basic

```
table ip6 filter {
        chain input {
                type filter hook input priority 0;
                # accept any localhost traffic
                iif lo accept
                # accept traffic originated from us
                ct state established,related accept
                #    accept    neighbour    discovery    otherwise
connectivity breaks
                icmpv6  type  {  nd-neighbor-solicit,  echo-request,
nd-router-advert, nd-neighbor-advert } accept
                # count and drop any other traffic
                counter drop
        }
}
```

fw.inet.basic

La tabla inet está disponible a partir del kernel linux 3.14 y permite hacer una tabla de IPv4 e IPv6. Esto es un simple cambio comparado con el conjunto de reglas anterior que es la palabra clave inet.

```
table inet filter {
        chain input {
```

```
                type filter hook input priority 0;
                 # accept any localhost traffic
                iif lo accept
                 # accept traffic originated from us
                ct state established,related accept
                #    accept    neighbour    discovery    otherwise
connectivity breaks. daddr filter is a workaround to set l3
protocol.
                ip6 nexthdr icmpv6 icmpv6 type { nd-neighbor-
solicit,  echo-request,  nd-router-advert,  nd-neighbor-advert  }
accept
                 # count and drop any other traffic
                counter drop
        }
}
```

1.7.2 Filtrado de bridge

Limitaciones

Actualmente no existe una conexión de seguimiento disponible para el filtrado de puente.

Ejemplos:

Filtro en el puerto TCP destino:

```
nft add rule bridge filter forward ether type ip tcp dport 22
accept
```

Aceptar el paquete ARP:

```
nft add rule bridge filter forward ether type arp accept
```

1.7.3 Múltiples NAT's utilizando mapas nftables

Gracias a los mapas de nftables, si tenemos en iptables un ruleset como este de DNAT:

```
% iptables -t nat -A PREROUTING -p tcp --dport 1000 -j DNAT --to-
destination 1.1.1.1:1234
% iptables -t nat -A PREROUTING -p udp --dport 2000 -j DNAT --to-
destination 2.2.2.2:2345
```

```
% iptables -t nat -A PREROUTING -p tcp --dport 3000 -j DNAT --to-destination 3.3.3.3:3456
```

Se puede traducir fácilmente a nftables en una sola línea:

```
% nft add rule nat prerouting dnat \
      tcp dport map { 1000 : 1.1.1.1, 2000 : 2.2.2.2, 3000 : 3.3.3.3} \
      : tcp dport map { 1000 : 1234, 2000 : 2345, 3000 : 3456 }
```

Del mismo modo, en iptables DNAT:

```
% iptables -t nat -A POSTROUTING -s 192.168.1.1 -j SNAT --to-source 1.1.1.1
% iptables -t nat -A POSTROUTING -s 192.168.2.2 -j SNAT --to-source 2.2.2.2
% iptables -t nat -A POSTROUTING -s 192.168.3.3 -j SNAT --to-source 3.3.3.3
```

Traducido a nftables sería:

```
% nft add rule nat postrouting snat \
      ip saddr map { 192.168.1.1 : 1.1.1.1, 192.168.2.2 : 2.2.2.2, 192.168.3.3 : 3.3.3.3
```

2. Instalación de cortafuegos nftables con control de versiones mediante git.

Al estar utilizando debían jessie, tenemos que añadir los repositorios de backports para poder instalar nftables:

```
#jessie-backports
deb http://ftp.cica.es/debian/ jessie-backports main
```

Procedemos a instalar nftables:

```
# apt-get -t jessie-backports install nftables
```

Para evitar problemas vamos a desinstalar iptables:

```
# apt-get purge iptables
```

Para llevar a cabo un control de versiones el cortafuegos instalaremos el paquete "git":

```
# apt-get install git
```

Creamos el directorio "/srv/git/nft-firewall.git", donde crearemos el servior git con "git init --bare":

```
# mkdir -p /srv/git/nft-firewall.git
# git init --bare
Initialized empty Git repository in /srv/git/nft-firewall.git/
```

Dentro del directorio hooks creado automáticamente vamos a añadir 2 scripts, "post-recive" y "post-update"

Contenido de "post-recive":

```
#!/bin/bash
NAME="hooks/post-receive"
NFT_ROOT="/etc/nftables.d"
RULESET="${NFT_ROOT}/ruleset.nft"
export GIT_WORK_TREE="$NFT_ROOT"
info()
{
      echo "$NAME $1 ..."
}
info "checkout latest data to $GIT_WORK_TREE"
sudo git checkout -f
info "cleaning untracked files and dirs at $GIT_WORK_TREE"
sudo git clean -f -d
info "deploying new ruleset"
set -e
cd $NFT_ROOT && sudo nft -f $RULESET
info "new ruleset deployment was OK"
```

Contenido de "update":

```
#!/bin/bash
NAME="hooks/update"
info()
{
      echo "$NAME $1 ..."
}
```

```
warning()
{
      echo "W: ${NAME}: $1 ..." >&2
      rm -rf $DEST 2>/dev/null
      exit 0
}
error()
{
      echo "E: ${NAME}: $1 ..." >&2
      rm -rf $DEST 2>/dev/null
      exit 1
}
DEST=$(mktemp -d)
if [ ! -d "$DEST" ] ; then
      warning "unable to create temp dir"
fi
REV=$3
if [ -z "$REV" ] ; then
      error "invalid input argument for revision"
fi
NFT="/usr/sbin/nft"
if [ ! -x "$NFT" ] ; then
      warning "no nft binary found"
fi


RULESET="${DEST}/ruleset.nft"
NETNS="nft-test-ruleset"
info "exporting new revision"
git archive --format=tar $REV | ( cd $DEST && tar xf -)
if [ "$?" != "0" ] ; then
      warning "unable to export revision"
fi
if [ ! -r $RULESET ] ; then
      warning "$RULESET doesn't exists"
fi
info "testing new ruleset"
# Create netns, ignore if alredy exists
sudo ip netns add $NETNS 2>/dev/null
# Check if exists
NETNS_LIST=$(sudo ip netns list)
```

```
grep $NETNS <<< $NETNS_LIST >/dev/null 2>/dev/null
if [ "$?" != "0" ] ; then
      warning "unable to create netns $NETNS"
fi
# Load ruleset
cd $DEST && sudo ip netns exec $NETNS $NFT -f $RULESET
if [ "$?" != "0" ] ; then
      error "failed to load $RULESET"
fi
# Clear ruleset
sudo ip netns exec $NETNS $NFT flush ruleset
if [ "$?" != "0" ] ; then
      warning "failed to flush ruleset after testing"
fi
# Delete netns
sudo ip netns delete $NETNS
if [ "$?" != "0" ] ; then
      warning "failed to clean netns $NETNS"
fi
info "ruleset test was OK"
rm -rf $DEST
exit 0
```

Creamos el directorio /etc/nftables.d/ donde se añadirá automáticamente el ruleset y los ficheros con la configuración de las cadenas apropiadas con sus respectivas reglas.

```
# mkdir /etc/nftables.d/
```

Antes de realizar cualquier otro paso, debemos asegurarnos que se está cumpliendo la política de seguridad. Con el fin de permitir todas estas operaciones a un usuario sin privilegios, necesitamos una polÃ-tica sudo.

Se requiere la opción NOPASSWD en sudo para que no pida contrasña al usuario mientras se ejecutan los hooks de git.

Debemos tener cuidado también con los usuarios/grupos estándar de Unix. Debemos dar los permisos apropiados en los directorios/scripts para que todo funcione

correctamente. Por ejemplo, poner los operadores en el grupo de git, cambiar el grupo del directorio del servidor git y ponerle g+w.

Para hacer esto instalamos sudo:

```
# apt-get install sudo
```

Dentro del direcctorio /etc/sudoers.d/ creamos un fichero: "nft-git" con el siguiente contenido:

```
User_Alias  OPERATORS = usuario
Defaults    env_keep += "GIT_WORK_TREE"
OPERATORS   ALL=(ALL) NOPASSWD:/bin/ip netns add nft-test-ruleset
OPERATORS   ALL=(ALL) NOPASSWD:/bin/ip netns list
OPERATORS   ALL=(ALL)   NOPASSWD:/bin/ip   netns   exec   nft-test-ruleset
/usr/sbin/nft -f *
OPERATORS   ALL=(ALL)   NOPASSWD:/bin/ip   netns   exec   nft-test-ruleset
/usr/sbin/nft flush ruleset
OPERATORS   ALL=(ALL) NOPASSWD:/bin/ip netns delete nft-test-ruleset
OPERATORS   ALL=(ALL) NOPASSWD:/usr/bin/git checkout -f
OPERATORS   ALL=(ALL) NOPASSWD:/usr/bin/git clean -f -d
OPERATORS   ALL=(ALL) NOPASSWD:/usr/sbin/nft -f *
```

Ahora cambiamos los permisos de los directorios "/srv/git/nft-firewall.git" y el propietario. Crearemos un grupo "git" y añadiremos los usuarios encargados de la gestión de nftables.

```
chown -R root:git /srv/git/nft-firewall.git
chmod -R g+rw /srv/git/nft-firewall.git
chmod -R g+rwx /srv/git/nft-firewall.git/hooks
```

Ahora clonamos el repositorio creado:

```
$ git clone /srv/git/nft-firewall.git/
```

Añadimos ficheros de configuración de nftables, por ejemplo el ruleset tendrá el siguiente contenido:

```
# don't delete this
flush ruleset

include "./defines.nft"
```

```
# The main table
table inet inet-filter {
     include "./inet-filter-chain-global.nft"
     include "./inet-filter-chain-input.nft"
     include "./inet-filter-chain-forward.nft"
     include "./inet-filter-chain-output.nft"
}
```

Este fichero hará un limpiado de reglas.

Llamará al fichero "defines.nft" donde vamos a declarar las variables.

Creará la tabla inet "inet-filter" que a su vez llamará a los ficheros de configuración de las cadenas.

Ahora creamos los ficheros indicados con el siguiente contenido:

defines:

```
#Interfaces
define nic_internet = eth0
define nic_aulas= eth1
define nic_andared = eth2
define nic_papion = eth3
define nic_babuino = eth4

#REDES
define net_aulas = 172.22.0.1/16
define net_satelites = 172.22.222.1/24
define net_papion = 192.168.102.1/24
define net_babuino = 192.168.103.1/24

#Máquinas
define papion = 192.168.102.2
```

```
define babuino = 192.168.103.2
define jupiter = 172.22.222.1
define corot = 172.22.222.2
define io = 172.22.222.21
define europa = 172.22.222.22
define ganimedes = 172.22.222.23
define calisto = 172.22.222.24
define saturno = 172.22.222.11
define internet = 80.59.1.2
define bulerias = 172.22.200.61
```

---

inet-filter-chain-input.nft :

```
chain input {
     type filter hook input priority 0; policy drop;
     jump global
     counter
}
```

---

inet-filter-chain-output.nft

```
chain output {
     type filter hook output priority 0; policy drop;
     jump global

     # contar tráfico antes del policy drop
     counter

}
```

---

inet-filter-chain-forward.nft

```
chain forward {
     type filter hook forward priority 0;
     counter
     policy drop;
```

```
}
```

---

inet-filter-chain-global.nft

```
chain global {
}
```

Con esto tendremos creados las cadenas input,forward y output con una política drop.

Como habrás podido observar, en los ficheros de las cadenas "input" y "output" se ha añadido "jump global", esto hará que se consulte la cadena "global" en la que habrá reglas globales.

Añadimos los ficheros al servidor git y observamos en la salida como se ejecutan los hooks de git:

```
remote: hooks/update exporting new revision ...
remote: hooks/update testing new ruleset ...
remote: hooks/update ruleset test was OK ...
remote: hooks/post-receive checkout latest data to
/etc/nftables.d ...
remote: hooks/post-receive cleaning untracked files and dirs
at /etc/nftables.d ...
remote: hooks/post-receive deploying new ruleset ...
remote: hooks/post-receive new ruleset deployment was OK ...
To /srv/git/nft-firewall.git/
   b140206..892950c  master -> master
```

Si todo está bien configurado deberán haberse añadido los ficheros al directorio "/etc/nftables.d/" :

```
$ ls /etc/nftables.d/
defines.nft  inet-filter-chain-forward.nft inet-filter-chain-
global.nft  inet-filter-chain-input.nft    inet-filter-chain-
output.nft  ruleset.nft
```

Comprobamos las tablas creadas y su contenido:

```
table inet inet-filter {
	chain global {

	}
	chain input {
		type filter hook input priority 0; policy drop;
		jump global
		counter packets 0 bytes 0
	}
	chain forward {
		type filter hook forward priority 0; policy drop;
		counter packets 0 bytes 0
	}
	chain output {
		type filter hook output priority 0; policy drop;
		jump global
		counter packets 0 bytes 0
	}
}
```

Vamos a insertar una regla que permita las peticiones icmp a/desde todas las interfaces menos $nic_internet(eth0).

Añadimos al fichero inet-filter-chain-input.nft:

```
#permitir ping desde todos las interfaces menos eth0
iifname !=$nic_internet ip protocol icmp counter accept
```

Añadimos al fichero inet-filter-chain-output.nft:

```
#permitir ping a todas las interfaces menos eth0
oifname !=$nic_internet ip protocol icmp counter accept
```

La interfaz eth0 es la que tiene salida a internet por lo que si hacemos un ping a "8.8.8.8" no deberáa funcionar:

```
$ ping -c1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
^C
--- 8.8.8.8 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

Sin embargo si hacemos ping a una máquina que está detrás de otra interfaz:

```
$ ping -c1 192.168.102.2
PING 192.168.102.2 (192.168.102.2) 56(84) bytes of data.
64 bytes from 192.168.102.2: icmp_seq=1 ttl=64 time=0.704 ms
--- 192.168.102.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.704/0.704/0.704/0.000 ms
```

Ahora vamos a añadir una regla que permita cierto tráfico a la máquina babuino, por lo que añadiremos a los ficheros de input y output la siguiente regla respectivamente:

```
#Permitir tráfico que viene desde babuino
ip saddr $babuino tcp dport { 25, 4949, 80 } ct state new,established counter accept

#Permitir tráfico que va a babuino
ip daddr $babuino tcp sport { 25, 4949, 80 } ct state new,established counter accept
```

Para comprobar que funciona correctamente vamos a ejecutar netcat en el servidor del cortafuegos y nos conectaremos desde babuino a uno de los puertos permitidos:

```
usuario@babuino:~$ nc 192.168.103.1 80
Hola
root@macaco:/home/usuario# netcat -lk -p 80
Hola
```

La conexión funciona correctamente.

Gracias al control de versiones de git podemos ver la lista de cambios, así- como quién realiza dicho cambio. Para comprobarlo

```
$ git log --pretty=format:"%h - %an, %ar : %s"
ac1a21e - usuario, 5 minutes ago : Permitir tráfico babuino
```

También podemos ver el contenido del commit ejecutando "git show [numerodecommit]":

```
$ git show ac1a21e
commit ac1a21e239f37f37e26de9b65ef41c59798e6695
Author: usuario <usuario@macaco>
Date:   Fri Jun 17 10:18:57 2016 +0200
    Permitir tráfico babuino
diff   --git   a/inet-filter-chain-input.nft   b/inet-filter-chain-
input.nft
index 4d3118b..974652f 100644
--- a/inet-filter-chain-input.nft
+++ b/inet-filter-chain-input.nft
@@ -2,5 +2,8 @@ chain input {
        type filter hook input priority 0; policy drop;
        jump global
         iifname !=$nic_internet ip protocol icmp counter accept
+
+       #Permitir tráfico que viene desde babuino
+         ip saddr $babuino tcp dport { 25, 4949, 80 } ct state
new,established counter accept
        counter
 }
diff   --git   a/inet-filter-chain-output.nft   b/inet-filter-chain-
output.nft
index bd61611..2f85216 100644
--- a/inet-filter-chain-output.nft
+++ b/inet-filter-chain-output.nft
@@ -2,6 +2,10 @@ chain output {
```

```
        type filter hook output priority 0; policy drop;
        jump global
         oifname !=$nic_internet ip protocol icmp counter accept
+
+       #Permitir tráfico que va a babuino
+         ip daddr $babuino tcp sport { 25, 4949, 80 } ct state
new,established counter accept
+
        # contar y logear tráfico antes del policy drop
        counter
```

Ahora vamos a ver cómo permitir tráfico de forward:

Para tener mejor organizado el tráfico de forward podemos crear ficheros ".nft" con nuevas cadenas. Estos ficheros se incluirán en el fichero de configuración de forward y a su vez invocará a la cadena apropiada incluida en cada fichero.

Vamos a modificar el fichero "inet-filter-chain-forward.nft" y vamos a añadir lo siguiente:

```
include "./inet-filter-chain-babuino_all_in.nft"
include "./inet-filter-chain-babuino_all_out.nft"
include "./inet-filter-chain-babuino_aulas.nft"
include "./inet-filter-chain-aulas_babuino.nft"

chain forward-split {
#Tráfico de babuino
     #Canalizamos el tráfico viene desde cualquier red con destino
babuino
     ip daddr $babuino oifname $nic_babuino jump babuino_all_in

     #tráfico que viene desde babuino con destino cualquier red
     ip saddr $babuino iifname $nic_babuino jump babuino_all_out

     #tráfico que viene desde la red aulas con destino babuino
```

```
      ip saddr $net_aulas ip daddr $babuino iifname $nic_aulas
oifname $nic_babuino jump aulas_babuino

      #tráfico desde babuino con destino la red aulas
      ip saddr $babuino ip daddr $net_aulas iiname $nic_babuino
oifname $nic_aulas jump babuino_aulas

}

chain forward {
      type filter hook forward priority 0; policy drop;
      jump forward-split
}
```

La configuración de las cadenas creadas es la siguiente:

Fichero “inet-filter-chain-babuino_all_in.nft”:

```
chain babuino_all_in {
        tcp dport 25 ct state new,established counter accept
        tcp dport 80 ct state new,established counter accept
        tcp dport 443 ct state new,established counter accept
}
```

Fichero “inet-filter-chain-babuino_all_out.nft":

```
chain babuino_all_out {
        tcp sport 25 ct state established counter accept
        tcp sport 80 ct state established counter accept
        tcp sport 443 ct state established counter accept
}
```

Fichero “inet-filter-chain-babuino_aulas.nft":

```
chain babuino_aulas {
        tcp sport 22 ct state established counter accept
}
```

Fichero “inet-filter-chain-aulas_babuino.nft":

```
chain aulas_babuino {
        tcp dport 22 ct state new,established counter accept
}
```

Si comprobamos el contenido de la tabla veremos las cadenas creadas con sus respectivas reglas:

```
table inet inet-filter {
      chain babuino_all_in {
            tcp dport smtp ct state established,new counter packets 0
bytes 0 accept
            tcp dport http ct state established,new counter packets 0
bytes 0 accept
            tcp dport https ct state established,new counter packets 0
bytes 0 accept
      }
      chain babuino_all_out {
            tcp sport smtp ct state established counter packets 0 bytes 0
accept
            tcp sport http ct state established counter packets 0 bytes 0
accept
            tcp sport https ct state established counter packets 0 bytes
0 accept
      }
      chain aulas_babuino {
            tcp dport ssh ct state established,new counter packets 0
bytes 0 accept
      }
      chain babuino_aulas {
            tcp sport ssh ct state established counter packets 0 bytes 0
accept
      }
      chain forward-split {
            ip saddr 172.22.0.0/16 ip daddr 192.168.103.2 iifname "eth1"
oifname "eth4" jump aulas_babuino
            ip saddr 192.168.103.2 ip daddr 172.22.0.0/16 iifname "eth4"
oifname "eth1" jump babuino_aulas
            ip daddr 192.168.103.2 oifname "eth4" jump babuino_all_in
            ip saddr 192.168.103.2 iifname "eth4" jump babuino_all_out
```

```
	}
	chain forward {
		type filter hook forward priority 0; policy drop;
		jump forward-split
		counter packets 0 bytes 0
	}
}
```

Comprobamos que podemos conectarnos por ssh desde una máquina en la red aulas a babuino:

```
root@aulas:/home/usuario# ssh usuario@192.168.103.2
usuario@192.168.103.2's password:
The programs included with the Debian GNU/Linux system are free
software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Jun 17 19:38:46 2016 from 172.22.222.5
usuario@babuino:~$
```

Observamos cómo se actualizan los contadores:

```
	chain aulas_babuino {
		tcp dport ssh ct state established,new counter packets
46 bytes 8234 accept
	}
	chain babuino_aulas {
		tcp sport ssh ct state established counter packets 57
bytes 7866 accept
	}
```

Ahora vamos a permitir tráfico entre las redes de papion y babuino, añadimos lo siguiente al fichero de forward:

```
include "./inet-filter-chain-babuinonet_papionnet.nft"
include "./inet-filter-chain-papionnet_babuinonet.nft"
```

```
#TRÁ□FICO DE RED BABUINO-PAPION
      ip saddr $net_babuino ip daddr $net_papion iifname
$nic_babuino oifname $nic_papion jump babuinonet_papionnet

#TRÁ□FICO DE RED PAPION-BABUINO
      ip saddr $net_papion ip daddr $net_babuino iifname
$nic_papion oifname $nic_babuino jump papionnet_babuinonet
```

Creamos los ficheros indicados:

Fichero "inet-filter-chain-babuinonet_papionnet.nft":

```
chain babuinonet_papionnet{
      udp dport { 53, 123 } ct state new,established counter accept
      tcp dport { 80, 389, 3306 } ct state new,established counter
accept
}
```

Fichero "inet-filter-chain-papionnet_babuinonet.nft":

```
chain papionnet_babuinonet{
      udp sport { 53, 123 } ct state established counter accept
      tcp sport { 80, 389, 3306 } ct state established counter
accept
}
```

Observamos las cadenas creadas:

```
      chain babuinonet_papionnet {
            udp dport { ntp, domain} ct state established,new counter
packets 0 bytes 0 accept
            tcp dport { mysql, ldap, http} ct state established,new
counter packets 0 bytes 0 accept
      }
      chain papionnet_babuinonet {
            udp sport { ntp, domain} ct state established counter packets
0 bytes 0 accept
            tcp sport { mysql, http, ldap} ct state established counter
packets 0 bytes 0 accept
      }
```

```
	chain forward-split {
		ip saddr 172.22.0.0/16 ip daddr 192.168.103.2 iifname "eth1" oifname "eth4" jump aulas_babuino
		ip saddr 192.168.103.2 ip daddr 172.22.0.0/16 iifname "eth4" oifname "eth1" jump babuino_aulas
		ip daddr 192.168.103.2 oifname "eth4" jump babuino_all_in
		ip saddr 192.168.103.2 iifname "eth4" jump babuino_all_out
		ip saddr 192.168.103.0/24 ip daddr 192.168.102.0/24 iifname "eth4" oifname "eth3" jump babuinonet_papionnet
		ip saddr 192.168.102.0/24 ip daddr 192.168.103.0/24 iifname "eth3" oifname "eth4" jump papionnet_babuinonet
	}
```

Comprobamos la conexión desde babuino a papion:

```
root@babuino:/home/usuario# nc 192.168.102.2 80
hola
root@papion:/home/usuario# netcat -lk -p 80
hola
```

Comprobamos cómo se han actualizado los contadores:

```
	chain babuinonet_papionnet {
		udp dport { ntp, domain} ct state established,new counter packets 0 bytes 0 accept
		tcp dport { mysql, ldap, http} ct state established,new counter packets 6 bytes 333 accept
	}

	chain papionnet_babuinonet {
		udp sport { ntp, domain} ct state established counter packets 0 bytes 0 accept
		tcp sport { mysql, http, ldap} ct state established counter packets 4 bytes 204 accept
	}
```

3. Referencias

https://wiki.nftables.org/wiki-nftables/index.php/Main_Page

http://ral-arturo.blogspot.com.es/2015/02/nftables-ruleset-managed-with-git.html