Firewalls en Linux: NetFilter

Tabla de contenidos

1. Licencia

2. Topologías de red

Uno de los aspectos más críticos a la hora de crear y administrar una red es diseñar y planificar la topología de red.

Una **topología de red** (también llamada arquitectura) es simplemente un diseño en que se emplean determinados componentes con la finalidad de canalizar tráfico, este tráfico se permite o se deniega atendiendo a ciertos criterios fijados a priori.

Existen varias topologías de red, desde las más simples en las que solo existe un router, hasta las más complejas basadas en varios routers, proxies y redes perimetrales (o zonas neutras, o zonas desmilitarizadas).

En cuanto a los elementos básicos que conforman una red tenemos:

- **Routers**.
  - Un router es un dispositivo que interconecta dos redes llevando paquetes de una a otra.
  - Un router opera en el nivel 3 de OSI.
- **Firewalls**.
  - Un firewall (o cortafuegos) es un dispositivo o conjunto de dispositivos diseñados para permitir/denegar tráfico de red según un conjunto de reglas definidas.
- **Proxies**.
  - Un proxy es un servidor (máquina o servicio) que actúa como un intermediario en un petición cliente/servidor de un servicio de red.

- **Bridge**.
    - Un bridge (o puente) es un dispositivo de interconexión de redes de ordenadores que opera en el nivel 2 OSI.
    - Un bridge interconecta dos segmentos de red intercambiando datos de una red a otra.
- **Bastion Host**.
    - Un "bastion host" es un ordenador de propósito específico conectado a una red, diseñado y preparado para resistir ataques.
    - Un bastion host, por norma general, solo ejecuta un servicio de red, como puede ser un servicio proxy; el resto de servicios se limitan o eliminan para reducir posibles amenazas.
    - Se encuentran normalmente al otro lado de un firewall o DMZ donde puede ser accedido desde redes no confiables.
- **DMZ o red perimetral**.
    - Una DMZ (DeMilitarized Zone) es una red añadida entre dos redes con el objetivo de proporcionar mayor protección a una de ellas.
    - En esta red se suelen colocar los servidores públicos de la organización/empresa. Ante una posible intrusión, ésta quedaría aislada en la DMZ, impidiéndose el acceso a la red interna.

3. Cortafuegos y DMZ

En cualquier empresa, es habitual ofrecer distintos servicios accesibles desde Internet, ya sea para empleados o clientes, tales como una página web, correo electrónico o un simplemente, un servidor de ficheros. Estos servicios pueden:

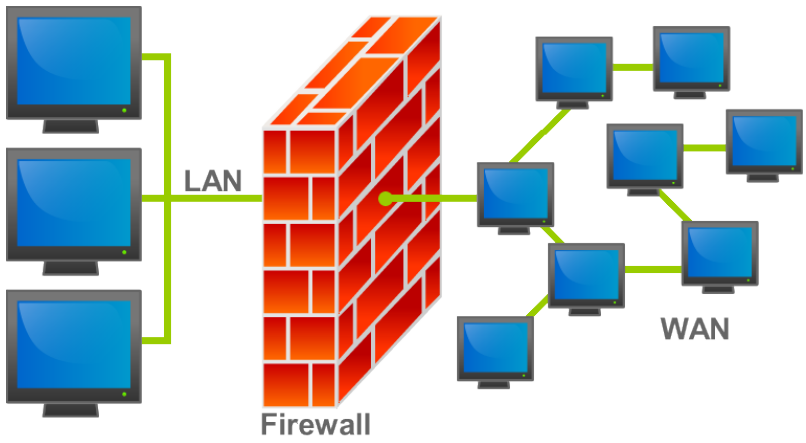
- Subcontratarse a una empresa especializada en cloud.
- Afrontarse de forma interna, desde la organización, mediante recursos propios.
    - (+) Se mantiene el control de la información.
    - (+) Se puede diseñar el servicio en función exacta de las necesidades.
    - (-) Se aumenta el riesgo de sufrir un incidente de seguridad.
- Para minimizar los riesgos derivados de un servidor con acceso desde Internet que pudiera comprometer la seguridad de la organización, se deben desplegar todas las medidas de seguridad necesarias, comenzando por la implantación de:
    - Cortafuegos o firewalls.
    - Una red local denominada zona desmilitarizada o DMZ.

3.1. ¿Qué es un cortafuegos?

Un **cortafuegos** (firewall) es un dispositivo o conjunto de dispositivos que controlan el tráfico entre redes.

Un cortafuegos puede ser:

- Un dispositivo físico (hardware).
- Un software sobre un sistema operativo.



Podemos ver un cortafuegos como una caja negra con dos o más interfaces de red en las que se establecen unas reglas de filtrado que deciden:

- Si una conexión determinada **se establece**.
- Si una conexión determinada **NO se establece**.

Podemos hablar en términos de conexiones iniciadas o en términos de permitir o no determinado tipo de tráfico. Un firewall, además, puede realizar ciertas modificaciones sobre el tráfico que filtra.

Siendo más concretos podemos definir un cortafuegos como:

Cortafuegos o firewall

> Un cortafuegos o firewall es un hardware específico con un sistema operativo o un IOS (Internetworking Operative System) que filtra el tráfico TCP/UDP/IP/ICMP/... y decide si un paquete:
>
> - Pasa.
> - Se modifica.
> - Se convierte.
> - Se descarta.

Sea del tipo que sea, un cortafuegos es un dispositivo basado en reglas, en el se definen un conjunto de reglas y en base a esas reglas toma decisiones.

Un ejemplo de reglas podría ser el siguiente:

- Política por defecto: DENEGAR.
- Todo lo que venga de la LAN interna al firewall: ACEPTAR.
- Todo lo que venga de la IP concreta del administrador de la red al puerto TCP 22: ACEPTAR.

- Todo lo que venga de la LAN interna hacia el exterior puerto TCP 80: ACEPTAR.
- Todo lo que venga de la LAN interna hacia el exterior puerto TCP 443: ACEPTAR.
- ...

Hay dos formas básicas de implementar un cortafuegos:

- Política por defecto **ACEPTAR**:
  - Todo el tráfico que entra entra y sale por el firewall se acepta y solo se denegará lo que se diga explícitamente.
- Política por defecto **DENEGAR**:
  - Todo el tráfico está denegado y sólo se permitirá pasar por el firewall aquello que se diga explícitamente.

Cada una de las anteriores políticas tiene sus ventajas y desventajas:

- Política ACEPTAR:
  - (+) Fácil de configurar y poner en marcha.
  - (+) Rápido de configurar y poner en marcha.
  - (+) Da menos problemas a los usuarios de la red.
  - (-) Poca seguridad.
  - (-) Poco control: no sabemos qué es lo que estamos permitiendo.
- Política DENEGAR:
  - (+) Seguridad.
  - (+) Más control de lo que hacemos.
  - (+) Más control de nuestra red.
  - (-) Difícil construir las reglas.
  - (-) Difícil depurar las reglas.
  - (-) Problemas con los usuarios, algunas aplicaciones pueden dejar de funcionar, las nuevas aplicaciones probablemente tampoco funcionen.

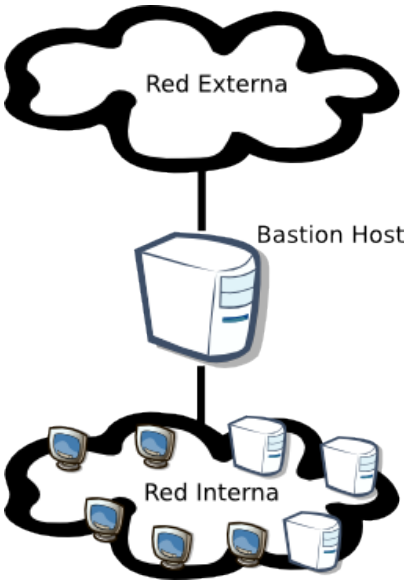
Desde el punto de vista de la seguridad, siempre optaremos por la política más segura: **DENEGAR**.

3.2. Topologías básicas

A la hora de configurar nuestra red podemos decantarnos por una de las siguientes topologías o incluso por una mezcla de varias:

- Dual homed gateway.
- Screened host.
- Screened subnet / DMZ.

3.2.1. Dual homed gateway

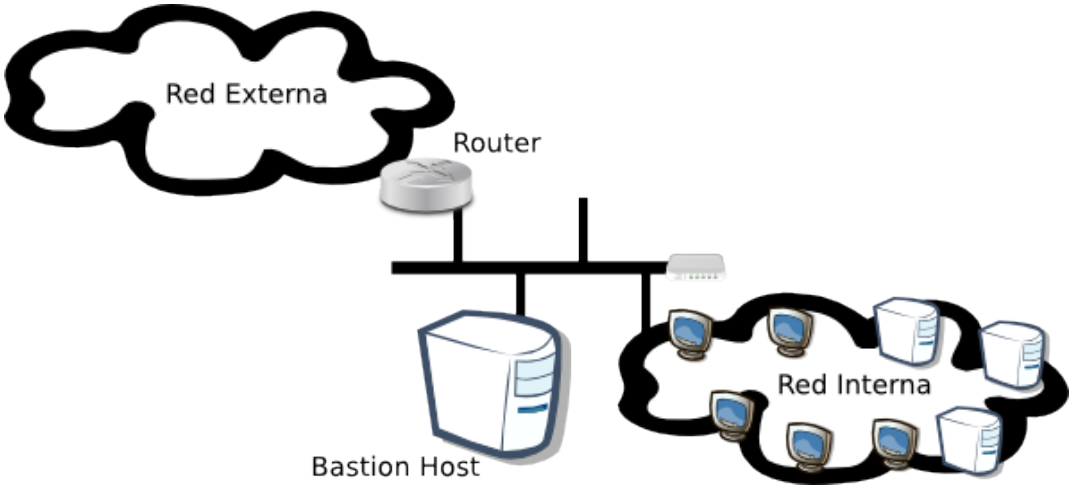


En esta topología de red tenemos:

- Un host con dos interfaces de red, cada una conectada a dos redes diferentes.
- El host tiene deshabilitado el enrutamiento.
- Solo se soportan servicios mediante proxy.
- Realmente, no se soporta el filtrado de paquetes, no hay cortafuegos.

Su principal ventaja es que es una topología bastante segura, barata y fácil de implementar, pero tiene la gran desventaja de que solo soporta servicios para los que exista o se puede utilizar un proxy. No podremos utilizar servicios que no se puedan "consumir" a través de un proxy.

3.2.2. Screened host



Topología similar a la anterior, pero con la principal diferencia que se incluye una red a la que se conectan un router y un proxy. Está formada por:

- Un router, bloquea todo el tráfico entre la red interna y externa, menos el tráfico que proviene del bastión.
- El bastión proporciona servicios a través de proxy.

- La topología soporta filtrado del paquetes, el router ofrece servicios de cortafuegos.
- La topología soporta servicios mediante proxy (bastión).

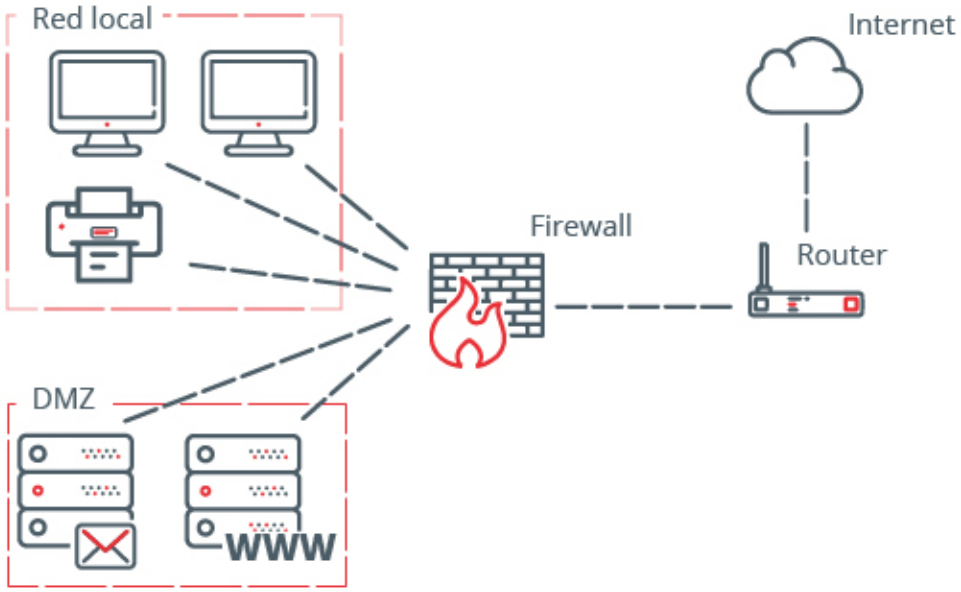
Es una topología fácil de implementar, pero tiene el inconveniente que no hay seguridad si el bastión se ve comprometido.

3.2.3. Screened subnet / DMZ

En esta topología se establece una red DMZ o red desmilitarizada (Demilitarized Zone).

Una zona desmilitarizada es una red aislada que se encuentra dentro de la red interna de la organización, en ella se encuentran ubicados exclusivamente todos los recursos de la empresa/organización que deban ser accesibles desde Internet (servidores web, servidores de correo, etc.).

Es una topología que tiene el inconveniente de ser difícil de implementar/mantener y más cara, como contrapartida es de las topologías más seguras.



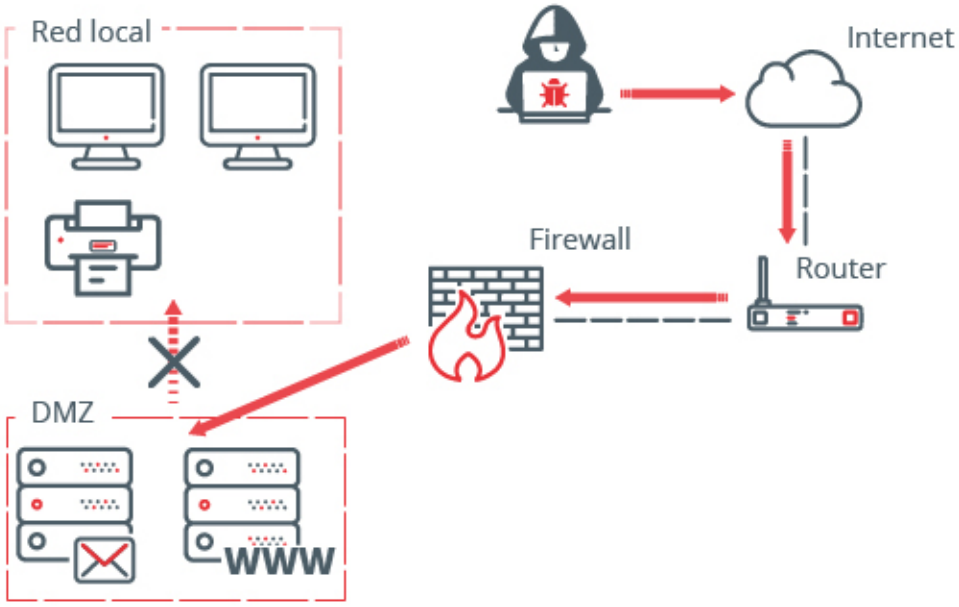
Por lo general una DMZ **permite**:

- Conexiones procedentes de Internet.
- Conexiones procedentes de la red local de la empresa donde están los equipos de los usuarios.

Mientras que **no permite**:

- Conexiones que van desde la DMZ a la red local.

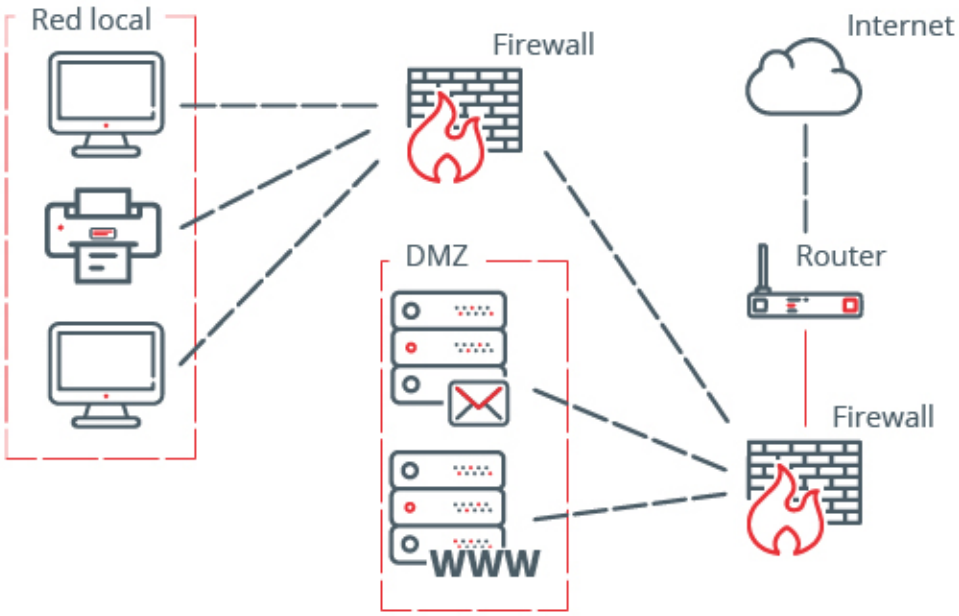
Los servidores accesibles desde Internet son más susceptibles a sufrir un ataque que pueda comprometer su seguridad, por eso se colocan en la DMZ. Si un ciberdelincuente comprometiera un servidor de la zona desmilitarizada, tendría muchos más complicado acceder a la red local de la organización, ya que las conexiones procedentes de la DMZ se encuentran bloqueadas.



Un resumen de las reglas de filtrado sería el siguiente:

Origen	Destino	Política
Internet	DMZ	Permitido
Internet	LAN	Denegado
DMZ	Internet	Permitido
DMZ	LAN	Denegado
LAN	DMZ	Permitido
LAN	Internet	Permitido

También se podría añadir un segundo firewall:



Al final lo normal es implementar una topología híbrida en la que se mezclan varios cortafuegos y routers, dependiendo del nivel de seguridad a implementar y del presupuesto

del que se disponga, hay multitud de combinaciones:

- Un solo cortafuegos.
- Dos o más cortafuegos.
- Solo DMZ.
- DMZ y MZ.
- Servidores públicos en la nube y servidores empresariales sin acceso desde el exterior o a través de VPN.
- Etc…

3.3. ¿Por qué necesitamos todo esto?

hay varios motivos para implementar una topología de red, pero los principales serían:

- **Control**.
  - Como administradores de red, queremos permitir o denegar ciertos tipos de tráfico atendiendo a diferentes criterios.
- **Seguridad**.
  - Debemos separar el tráfico interno del externo y no permitir intrusiones así como accesos no autorizados a nuestra red.
  - De la misma forma, tenemos que controlar el tráfico saliente para prevenir fugas de datos (DLP), shells inversas, C2, etc.
- **Vigilancia**.
  - Queremos ser informados cuando algo anómalo esté sucediendo.

Una vez configurada nuestra topología necesitaremos de máquinas de salto y probablemente de VPN para el acceso a determinadas partes de la red.

Una **máquina de salto** (jump server, jump host o jump box) es un sistema situado en una red que se utilizar para acceder y gestionar dispositivos en una zona de seguridad separada; una máquina de salto debe estar muy bien configurada, prestando especial atención a su securización (hardening). Se pueden implementar tanto en Windows como en Linux.

Una **VPN (red privada virtual)** es una tecnología que permite añadir una extensión de nuestra red local sobre una red pública como Internet. El objetivo es poder establecer y proteger el tráfico generado entre nuestra red y el host situado en Internet.

- Ejemplos de tecnologías utilizadas:
  - Internet Protocol Security (IPSec).
  - SSL/TLS.
- Ejemplos de software VPN:
  - Linux: OpenVPN, WireGuard.
  - Windows: L2TP/IPSec.

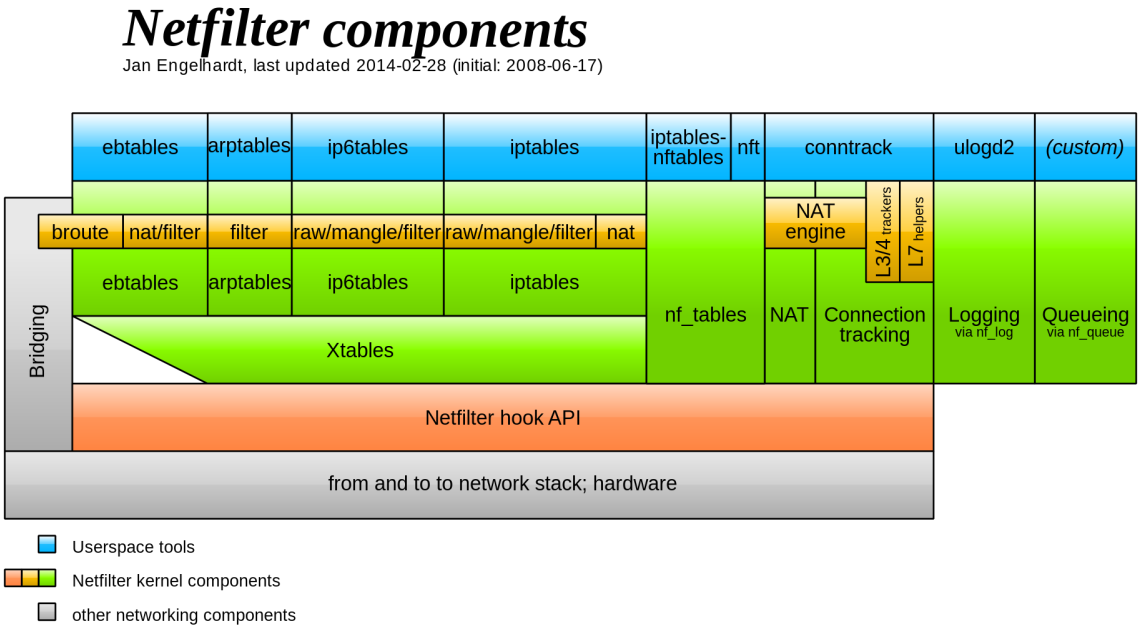
4. NetFilter

Netfilter es el framework del kernel de Linux que permite la interceptación y manipulación de paquetes de red. Las operaciones relacionadas con el networking pueden ser realizadas a través de *handlers* personalizados.

Entre las tareas básicas que Netfilter es capaz de hacer, destacan las siguientes:

- Filtrado de paquetes (cortafuegos).
- Traducción de direcciones y puertos de red (NAT).
- Manipulación de paquetes (packet mangling).

Netfilter define un conjunto de hooks en el kernel de Linux, permitiendo que determinados módulos se registren para ser llamados en determinados puntos de la pila de red. Estas funciones, generalmente en forma de reglas de filtrado y modificación, se invocan para cada paquete que atraviesa el hook respectivo dentro de la pila de red.



NetFilter se considera un cortafuegos capa 3 (nivel de red) pero con ciertas salvedades:

- No es un firewall de nivel de aplicación.
- Pero es capaz de controlar tráfico TCP/UDP y ciertas aplicaciones.
- Pero es capaz de controlar tráfico capa 2 (nivel de enlace).

4.1. Un poco de historia

Rusty Russell inició el proyecto netfilter/iptables en 1998, Rusty también había sido el desarrollador del proyecto anterior, *ipchains*. A medida que el proyecto crecía, se constituyó el grupo de desarrollo *Netfilter Core Team* (o simplemente *coreteam*) en 1999 para el desarrollo del framework *Netfilter*. Netfilter se liberó como Software Libre bajo la licencia GNU General Public License (GPL), y el 26 de agosto de 1999 se fusionó en el kernel de Linux, haciendo su primera aparición en la versión 2.4.0 del kernel Linux.

En agosto de 2003, Harald Welte se convirtió en presidente del *coreteam*. En abril de 2004, tras una ofensiva del proyecto contra quienes distribuían el software Netfilter integrado en routers sin cumplir la GPL, un tribunal alemán concedió a Welte una medida cautelar histórica contra Sitecom Germany, que se negaba a cumplir los términos de la GPL. En septiembre de 2007, Patrick McHardy, que dirigió el desarrollo durante los últimos años, fue elegido nuevo presidente del *coreteam*. Actualmente ese puesto recae sobre Pablo Neira Ayuso.

Antes de iptables, los programas más usados para crear cortafuegos en Linux eran *ipchains* en el núcleo Linux 2.2 e *ipfwadm* en el núcleo Linux 2.0, que a su vez se basaba en *ipfw* de BSD. Tanto *ipchains* como *ipfwadm* modificaban el código de red para poder manipular los paquetes, ya que no existía un framework general para la gestión de paquetes hasta la aparición de Netfilter.

iptables mantiene la idea básica introducida en Linux con *ipfwadm*: listas de reglas en las que se determina qué campos de la cabecera de un paquete deben ser evaluados y qué hacer con ese paquete si los campos de la cabecera coinciden con los criterios de la regla. *ipchains* agrega el concepto de cadenas de reglas (*chains*) e *iptables* extendió esto a la idea de tablas; se consultaba una tabla para decidir si había que NAT-ear un paquete, y se consultaba otra para decidir como filtrar un paquete. Adicionalmente, se modificaron los tres puntos en los que se realiza el filtrado en el viaje de un paquete, de modo que un paquete pase solo por un punto de filtrado.

Mientras que *ipchains* e *ipfwadm* combinan filtrado de paquetes y NAT (específicamente tres tipos de NAT, llamados masquerading o enmascaramiento de IP, port forwarding o redireccionamiento de puertos, y redirection o redirección), Netfilter hace posible por su parte separar las operaciones sobre los paquetes en tres partes: *packet filtering* (filtrado de paquetes), *connection tracking* (seguimiento de conexiones) y *Network Address Translation* (NAT o traducción de direcciones de red). Cada parte se conecta a las herramientas de Netfilter en diferentes puntos para acceder a los paquetes. Los subsistemas de seguimiento de conexiones y NAT son más generales y poderosos que los que realizaban *ipchains* e *ipfwadm*.

Esta división permite a *iptables*, a su vez, usar la información que la capa de seguimiento de conexiones ha determinado acerca del paquete: esta información estaba antes asociada a NAT. Esto hace a *iptables* superior a *ipchains*, ya que tiene la habilidad de monitorizar el estado de una conexión y redirigir, modificar o detener los paquetes de datos basados en el estado de la conexión y no solamente por el origen, destino o contenido del paquete. Un cortafuegos que utilice iptables de este modo se llama cortafuegos *stateful*, contrario a *ipchains* que solo permite crear un cortafuegos *stateless* (excepto en casos muy limitados). Se puede decir entonces que *ipchains* no está al tanto del contexto completo en el cual un paquete surge, mientras que *iptables* sí y por lo tanto puede tomar mejores decisiones sobre el futuro de los paquetes y las conexiones.

iptables, el subsistema NAT y el subsistema de seguimiento de conexiones son también extensibles, y muchas extensiones ya están incluidas en el paquete básico de iptables, tal como la extensión ya mencionada que permite la consulta del estado de la conexión. Extensiones adicionales se distribuyen junto a la utilidad iptables, como parches al código fuente del núcleo junto con una herramienta llamada patch-o-matic que permite aplicar los

parches. Todo esto se ha mejorado en gran medida con la llegada de **nft**.

En cuanto a las herramientas que los administradores de sistemas hemos tenido a nuestra disposición junto a Netfilter, tenemos:

- Primera generación.
  - Basada en **ipfw** de BSD. Portada por Alan Cox en 1994.
- Segunda generación.
  - Versión mejorada por Jos Vos (entre otros) para kernels 2.0. Introducción de la herramienta de espacio de usuario **ipfwadm**.
- Tercera generación.
  - Nueva versión para kernels 2.2. por Rusty Rusell, año 1998. Introducción de la herramienta **ipchains**.
- Cuarta generación.
  - Reescritura del código de cortafuegos por Rusty Rusell, año 1999. Introducción de la herramienta **iptables** para kernels 2.4.
- Quinta generación.
  - NFTables, año 2014, nuevo software y nueva herramientas, **nft** para kernels 3.13.

4.2. Conceptos importantes de NetFilter

4.2.1. Hooks

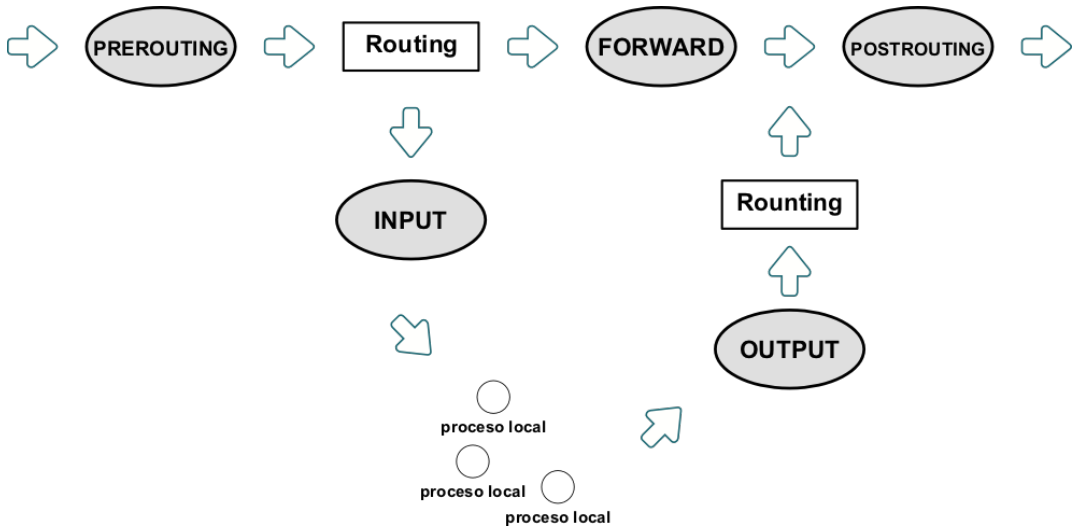
NetFilter está formado por un conjunto de *hooks*, un *hook* es un "punto de enganche" en la pila de protocolos.

Cada protocolo define sus propios *hooks*, puntos bien definidos en la travesía de un paquete por la pila de protocolos. Hay varios protocolos soportados, destacando IPv4 e IPv6.

Si nos centramos en un protocolo en concreto como IPv4, los *hooks* que se definen son los siguientes:

- PREROUTNG.
- INPUT.
- FORWARD.
- OUTPUT.
- POSTROUTING.

Gráficamente serían:



4.2.2. Reglas, cadenas y tablas

A la hora de definir qué trafico se permite o no en Netfilter, se utilizan los conceptos de **reglas**, **cadenas** y **tablas**.

Una **`regla`** define el conjunto de condiciones que debe cumplir el paquete.

Un conjunto de reglas forma una **`cadena`**.

Cada **`cadena`** se engancha a un **`hook`** determinado.

Para simplificar el manejo de todas las cadenas, existen un conjunto de **`tablas`** donde se pueden consultar y mantener. Una **`tabla`** está formada por un conjunto de cadenas.

Las tablas definidas son:

- Tabla **filter**, tabla por defecto. Usada en el filtrado.
- Tabla **nat**, para NAT (Network Address Translation), encargada de la traducción de direcciones de red, SNAT y DNAT.
- Tabla **mangle**, permite realizar modificaciones en los paquetes y tratamiento de usuario.

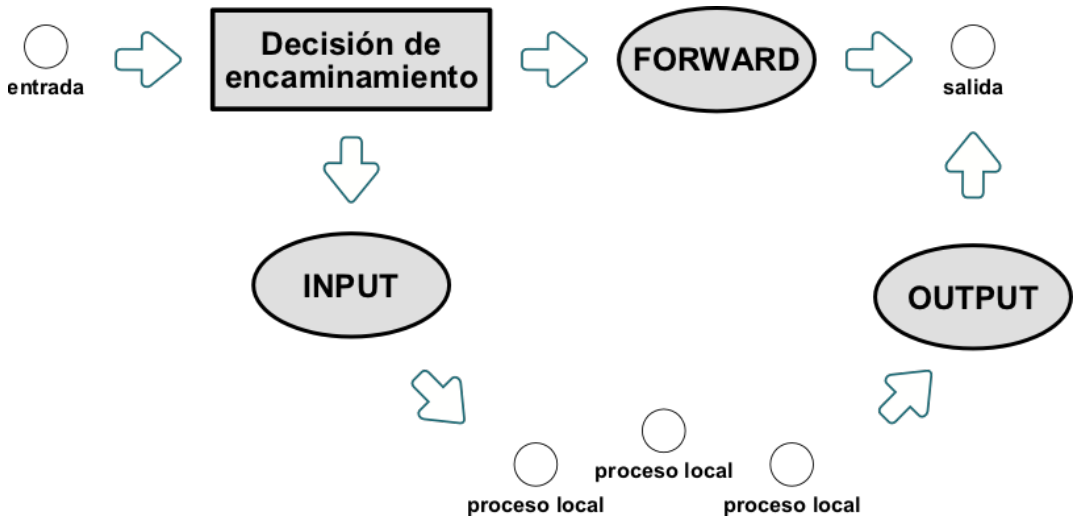
Funcionamiento básico:

Cuando un paquete alcanza un hook determinado se examinan las reglas de la cadena asociada para determinar la suerte (o destino) del paquete.

Si ninguna regla de la cadena se asocia con el paquete se determinará el **`destino`** (target) del paquete según la **`política`** por defecto de la cadena.

La tabla filter posee tres cadenas:

- INPUT.
- OUTPUT.
- FORWARD.



Estas tablas y cadenas venían predefinidas en los tiempos de **`iptables`**. **`Nftables`** **NO DEFINE** inicialmente ninguna tabla ni ninguna cadena.

4.3. iptables

Iptables *era* la herramienta de espacio de usuario utilizada para configurar **NetFilter**. Aunque se ha sustituido por la nueva herramienta **nft** en casi todas las distribuciones actuales, se sigue utilizando en determinadas versiones que aún cuentan con soporte (soporte extendido como en el caso de Red Hat o Ubuntu):

Distribución	Herramienta a utilizar
Ubuntu 18.04 LTS	iptables
Ubuntu 20.04 LTS	iptables
Ubuntu 22.04 LTS	nft
Fedora 37/38	nft (desde Fedora 32)
RHEL 7	iptables
RHEL 8	nft
RHEL 9	nft
Arch	nft

Aunque la herramienta oficial sea **nft**, se puede seguir utilizando **iptables** gracias a un módulo de conversión desarrollado para que la migración sea lo más sencilla posible.

4.4. ¿Qué es nftables?

Nftables es el actual framework de clasificación de paquetes del kernel de Linux. El framework Nftables integra en una única herramienta toda la infraestructura que proporcionaban anteriormente **`iptables`**, **`ip6tables`**, **`arptables`**, **`ebtables`** e **`ipset`**.

Nftables ofrece numerosas mejoras, características y rendimiento sobre las herramientas

anteriores de filtrado de paquetes, destacando principalmente:

- Disponible en el kernel de Linux desde la versión 3.13.
- Nueva herramienta de espacio de usuario en línea de comandos denominada **`nft`**, sintaxis muy distinta a la de **`iptables`**.
- Incorpora una capa de compatibilidad que permite ejecutar comandos **`iptables`** sobre el nuevo marco del kernel **`nftables`**.
- Tablas de búsqueda integradas en lugar de procesamiento lineal, lo que mejora el rendimiento en el procesamiento de las reglas.
- Un único framework que unifica los protocolos IPv4 e IPv6.
- Aplicaciones de reglas atómica, en lugar del procesamiento secuencial que hacía **`iptables`**.
- Soporte para depuración y rastreo en el conjunto de reglas (**`nftrace`**) y monitorización de eventos de rastreo (en la herramienta **`nft`**).
- Sintaxis más consistente y compacta, sin extensiones específicas de protocolo.
- Una API Netlink para aplicaciones de terceros.
- Proporciona una infraestructura de conjuntos genéricos que permite construir mapas y concatenaciones. Se pueden utilizar estas nuevas estructuras para organizar cualquier conjunto de reglas en un árbol multidimensional, reduciendo considerablemente el número de reglas que hay que inspeccionar hasta llegar a la acción final sobre un determinado paquete.

El framework *nftables* utiliza **tablas** para almacenar cadenas. Las **cadenas** contienen **reglas** individuales para realizar acciones. La utilidad **`nft`** reemplaza todas las herramientas de los marcos de trabajo anteriores de filtrado de paquetes. Además, podemos utilizar la librería **`libnftnl`** para interacción de bajo nivel con la API Netlink de nftables a través de la biblioteca **`libmnl`**.

A fecha 2024, las herramientas anteriores como *iptables* se siguen manteniendo, pero se debería migrar, ya con cierta urgencia, a la nueva herramienta. Existen herramientas que permiten la conversión de xtables (como iptables) a nftables, de forma semiautomática. Todas las distros importantes utilizan Nftables por defecto y otras herramientas de configuración de cortafuegos como **Firewalld** o **Ufw** también.

Para mostrar el conjunto de reglas, podemos utilizar el comando **`nft list ruleset`**. Hay que tener muy en cuenta que el uso de **`nft`** para interactuar con las tablas, cadenas, reglas y otros objetos de Nftables, puede afectar a los conjuntos de reglas que se hayan configurado con el comando iptables.

Por ejemplo, el comando **`nft flush ruleset`**, eliminará todas las reglas configuradas, tanto las definidas con **`nft`** como las definidas en **`iptables/ip6tables`**.

4.5. ¿Por qué nftables?

iptables es una herramienta que ha estado a nuestra disposición durante más de quince años, y probablemente seguirá activa durante unos años más en determinados despliegues.

Ha sido utilizada por muchos administradores de sistemas para filtrar tráfico, tanto por paquete como por flujo, registrar actividad de tráfico sospechoso, realizar NAT y muchas otras cosas. Además, ha venido acompañada de multitud de de extensiones que han sido aportadas durante la vida del proyecto.

Sin embargo, el framework iptables sufre de limitaciones que no se pueden solucionar fácilmente, y que **nftables** soluciona:

- Evitar la duplicación de código y las inconsistencias: muchas de las extensiones de **iptables** son específicas de protocolo, por lo que no hay una forma consolidada de hacer coincidir los campos de paquetes, en su lugar tenemos una extensión para cada protocolo que soporta. Esto sobrecarga el código base con código muy similar para realizar una tarea parecida: la correspondencia de la carga útil.
- Clasificación de paquetes más rápida gracias a una infraestructura mejorada de conjuntos genéricos y mapas.
- Administración simplificada de pila dual IPv4/IPv6, a través de la nueva familia **inet** que permite registrar cadenas base que ven tanto tráfico IPv4 como IPv6.
- Mejor soporte de actualizaciones dinámicas de conjuntos de reglas.
- Proporcionar una API Netlink para aplicaciones de terceros, al igual que hacen otros subsistemas de Linux como Linux Networking y Netfilter.
- Abordar las inconsistencias de sintaxis y proporcionar una sintaxis más agradable y compacta.

4.6. Principales diferencias con iptables

Desde el punto de vista del usuario, las principales diferencias entre **nftables** e **iptables** son:

- **nftables utiliza una nueva sintaxis**.
    - La herramienta de línea de comandos iptables utiliza un analizador sintáctico basado en **getopt_long()** en el que las palabras clave siempre van precedidas de un guión, como **--key** o **-p tcp**. En cambio, **nftables** utiliza una sintaxis compacta inspirada en **tcpdump**.
- **Las tablas y cadenas son totalmente configurables**.
    - **iptables** tiene múltiples tablas y cadenas base predefinidas, todas las cuales se registran aunque solamente se necesite una de ellas. Se han escrito informes indicando que incluso las cadenas base no utilizadas perjudican el rendimiento.
    - Con **nftables** no hay tablas o cadenas predefinidas. Cada tabla se define explícitamente y solo contiene los objetos (cadenas, conjuntos, mapas, tablas de flujo y objetos con estado) que se le añaden explícitamente. Con nftables sólo

hay que registrar las cadenas base que se necesitan. En **nftables** basta con elegir las tablas, cadenas y prioridades de los hooks de Netfilter que implementan eficientemente el pipeline de procesamiento de paquetes específicos.

- **Una única regla nftables puede ejecutar múltiples acciones**.
    - En lugar de los *macthes* y una única acción de destino utilizadas en **iptables**, una regla **nftables** consta de cero o más expresiones seguidas de una o más sentencias. Cada expresión comprueba si un paquete coincide con un campo de carga útil específico o con metadatos de paquete/flujo. Las expresiones múltiples se evalúan linealmente de izquierda a derecha: si la primera expresión coincide, entonces se evalúa la siguiente expresión y así sucesivamente.
    - Si llegamos a la última expresión, entonces el paquete coincide con todas las expresiones de la regla, y se ejecutan las sentencias de la regla. Cada sentencia realiza una acción, como establecer la marca del filtro de red, contar el paquete, registrar el paquete o emitir un veredicto, como aceptar o descartar el paquete o saltar a otra cadena.
    - Al igual que con las expresiones, las sentencias múltiples se evalúan linealmente de izquierda a derecha: una sola regla puede tomar múltiples acciones usando múltiples sentencias. hay que seguir teniendo en cuenta, que una sentencia de veredicto, por su naturaleza, finaliza la regla.
- **No hay contador incorporado por cadena y regla**.
    - En nftables los contadores son opcionales, se pueden activar según sean necesarios.
- **Mejor soporte para actualizaciones dinámicas de reglas**.
    - En contraste con el blob monolítico usado por **iptables**, los conjuntos de reglas de nftables se representan internamente en una lista enlazada. Ahora, al añadir o eliminar una regla, el resto del conjunto de reglas queda intacto, lo que simplifica el mantenimiento de la información de estado interna.
- **Administración simplificada de doble pila IPv4/IPv6**.
    - La familia **nftables** *inet* permite registrar cadenas base que ven tanto tráfico IPv4 como IPv6. Ya no es necesario depender de scripts para duplicar el conjunto de reglas.
- **Nueva infraestructura de conjuntos genéricos**.
    - Esta infraestructura se integra estrechamente en el núcleo de **nftables** y permite configuraciones avanzadas como mapas, mapas de veredicto e intervalos para lograr una clasificación de paquetes orientada al rendimiento. Lo más importante es que se puede utilizar cualquier selector compatible para clasificar el tráfico. Ya no se usa el comando **ipset** como se hacía con **iptables**.
- **Soporte para concatenaciones**.
    - Desde el kernel 4.1 de Linux, se puede concatenar varias claves y combinarlas con mapas y mapas de veredicto. La idea es construir una tupla cuyos valores hash para obtener la acción a realizar tengan un orden casi constante O(1).

- **Soportar nuevos protocolos sin necesidad de actualizar el kernel**.
    - Las actualizaciones del núcleo pueden ser una tarea larga y desalentadora, especialmente si tiene que mantener más de un único cortafuegos en la red. Los kernels que acompañan a las distribuciones suelen ir por detrás de la versión oficial. Con el nuevo enfoque de máquina virtual **`nftables`**, la compatibilidad con un nuevo protocolo no requiere obligatoriamente de un kernel nuevo, solamente una actualización relativamente sencilla del software de espacio de usuario **`nft`**.

4.7. Adopción actual

La adopción de **`nftables`** como el sistema predeterminado de filtrado de paquetes en Linux ha ido en aumento desde su introducción en 2014. Esta adopción se debe en gran parte a las mejoras significativas en términos de flexibilidad, eficiencia y simplicidad en comparación con su predecesor **`iptables`**.

En cuanto a las principales distribuciones de Linux, tenemos:

- **Debian y Ubuntu**: Debian, desde la versión 10 (Buster), y Ubuntu, desde la versión 22.04 LTS, han adoptado **`nftables`** como su sistema de firewall predeterminado, reemplazando a **`iptables`**.
- **Fedora**: Fedora fue una de las primeras distros en adoptar **`nftables`**, moviendo la configuración de firewall predeterminada a **`nftables`** desde Fedora 32. Esta adopción temprana refleja el compromiso de Fedora con las nuevas tecnoliogías que rodean a Linux.
- **Red Hat Enterprise Linux (RHEL)**: con el lanzamiento de RHEL 8, Red Hat también cambió de **`iptables`** a **`nftables`** como la solución de firewall predeterminada. Este cambio es significativo dado el uso extensivo de RHEL en entornos empresariales donde la seguridad y la estabilidad son críticas.
- **Arch Linux**: Arch Linux proporciona **`nftables`** en sus repositorios y ofrece amplia documentación sobre cómo migrar de **`iptables`** a **`nftables`**.

En cuanto a las herramientas más utilizadas como métodos de configuración de cortafuegos tenemos:

- **Firewalld**: **`firewalld`**, herramienta de gestión de firewall dinámica muy usada en distribuciones como RHEL o Fedora, ha incorporado soporte para **`nftables`**. Desde la versión 0.6.0, **`firewalld`** utiliza **`nftables`** como backend por defecto en lugar de **`iptables`**, lo que proporciona una mejor integración con las capacidades modernas de **`nftables`**.
- **Uncomplicated Firewall (ufw)**: aunque **`ufw`**, conocido por su simplicidad y facilidad de uso en distribuciones basadas en Debian como Ubuntu, todavía usa **`iptables`** como su backend por defecto, existe la posibilidad de configurar **`ufw`** para que funcione con **`nftables`**. Sin embargo, esta integración no es tan directa ni está tan avanzada como en otras herramientas.

La adopción generalizada de `nftables` en las principales distribuciones de Linux y herramientas de configuración de firewall indica una transición significativa hacia tecnologías más eficientes y modernas para la gestión de firewalls. `nftables` no solo simplifica la administración de las reglas de firewall, sino que también mejora el rendimiento y la capacidad de adaptación a las necesidades complejas de seguridad en la red.

Como cuadro resumen tenemos:

Distribución	Primera versión Nftables
Debian	Debian 10
Ubuntu	Ubuntu 22.04 LTS
Fedora	Fedora 32
RHEL	RHEL 8
Arch	Desde 2023

Este cambio hacia `nftables` refleja un enfoque más unificado y estandarizado en la configuración de seguridad en distribuciones de Linux, lo que beneficia tanto a los nuevos usuarios como a los administradores de sistemas más experimentados. La migración de `iptables` a `nftables` sigue en curso y en un estado muy avanzado.

Más información en cuanto al estado de adopción de Nftables en:

- https://wiki.nftables.org/wiki-nftables/index.php/What_is_nftables%3F#Adoption

5. Conceptos básicos: tablas, cadenas y reglas

Netfilter es una componente crucial del subsistema de filtrado de paquetes en Linux, y `nftables` es su evolución más reciente, diseñada para reemplazar sistemas anteriores como `iptables`. Para entender cómo funciona `nftables`, es esencial comprender tres conceptos fundamentales: tablas, cadenas y reglas.

Una **tabla** (table) en nftables es una estructura que organiza el conjunto de cadenas y reglas. Las tablas se utilizan para agrupar reglas por su funcionalidad y propósito. Pueden contener varias cadenas y pueden estar asociadas con uno o más *hooks* del kernel, los cuales determinan en qué punto del procesamiento de paquetes se evaluarán las reglas contenidas en las tablas. Existen diferentes tipos de tablas según el tipo de tráfico que gestionan, como pueden ser tablas de filtrado, enrutamiento o nat.

Las **cadenas** en *nftables* son conjuntos de reglas que se evalúan secuencialmente. Cada cadena está asociada a una tabla y se vincula a un *hook* específico en la pila de red del kernel.

Los *hooks* definen puntos específicos en el flujo de procesamiento de paquetes donde las cadenas pueden ser ejecutadas, como por ejemplo, `PREROUTING`, `INPUT`, `FORWARD`, `OUTPUT` y `POSTROUTING`. Las cadenas pueden ser de dos tipos: **base** y **regular**. Las cadenas base están directamente asociadas a los hooks del kernel y siempre se evalúan cuando el

paquete alcanza el *hook* correspondiente. Las cadenas regulares pueden ser llamadas desde otras reglas y no están vinculadas directamente a un *hook*.

Las **reglas** son las instrucciones individuales dentro de una cadena que especifican cómo manejar los paquetes. Cada regla consta de un conjunto de criterios de coincidencia (*matching*) y una acción a ejecutar si el paquete cumple con esos criterios.

Los criterios pueden incluir aspectos del paquete como direcciones IP de origen y destino, puertos, protocolo, etc. Las acciones pueden variar desde aceptar (permitir el paso del paquete), descartar (bloquear el paquete), modificar (alterar campos del paquete) hasta saltar a otra cadena para una evaluación adicional. Las reglas se evalúan en el orden en que aparecen en la cadena hasta que una de ellas coincide con el paquete o hasta que se alcanza el final de la cadena.

6. Migrando de iptables a nftables

Si nuestro firewall actual sigue utilizando **`iptables`** o tenemos varias reglas con la sintaxis anterior, el framework Nftables nos proporciona varias herramientas para que la transición sea un poco más sencilla.

6.1. Convirtiendo un conjunto de reglas en otro

Podemos utilizar las herramientas **`iptables-restore-translate`** e **`ip6tables-restore-translate`** para traducir los conjuntos de reglas de iptables/ip6tables a nftables.

Si tenemos configurado un cortafuegos con reglas **`iptables`** seguiremos los siguientes durante el proceso de migración:

1. Creamos el directorio **`/etc/nftables`**. Este directorio existe en RHEL9, pero no en Ubuntu por lo que en esta última tendremos que crear el directorio como paso previo antes de volcar las reglas:

```
# cd /etc/nftables
# iptables-save > /root/iptables.dump
# ip6tables-save > /root/ip6tables.dump
```

2. Convertimos las reglas de **`iptables`** a **`nftables`**:

```
# iptables-restore-translate -f /root/iptables.dump >
/etc/nftables/ruleset-migrated-from-iptables.nft
# ip6tables-restore-translate -f /root/ip6tables.dump >
/etc/nftables/ruleset-migrated-from-ip6tables.nft
```

3. Revisamos las reglas una a una.
4. Si todo es correcto podemos incluir ese conjunto de reglas desde el fichero principal,

añadiendo las siguientes directivas:

```
1 include "/etc/nftables/ruleset-migrated-from-iptables.nft"
2 include "/etc/nftables/ruleset-migrated-from-ip6tables.nft"
```

El fichero principal es distinto según la distribución: **En RHEL 9 se encuentra en `/etc/sysconfig/nftables`.** En Ubuntu 22/24 se encuentra en `/etc/nftables.conf`.

5. Deshabilitamos, en el caso de tenerlo habilitado, el servicio **`iptables`**:

```
# systemctl disable iptables
# systemctl stop iptables
```

6. Habilitamos e iniciamos el servicio **`nftables`**:

```
# systemctl enable nftables
# systemctl start nftables
```

7. Verificamos que las reglas se encuentran activas:

```
# nft list ruleset
```

6.2. Convirtiendo una sola regla de iptables a nftables

Además de los anteriores comandos para realizar una migración completa del conjunto de reglas, Nftables también proporciona los comandos **`iptables-translate`** e **`iptables-translate`** para convertir una sola regla.

Por ejemplo, si tenemos los siguientes comandos de iptables que nos definen dos reglas:

```
# iptables -A INPUT -i $eth0 -p tcp --dport 22 -m state --state NEW -j ACCEPT
# iptables -A INPUT -i eth0 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Podemos traducirlas al equivalente de Nftables con estos comandos:

```
# iptables-translate -A INPUT -i eth0 -p tcp --dport 22 -m state --state NEW -j
ACCEPT
nft 'add rule ip filter INPUT iifname "eth0" tcp dport 22 ct state new counter
accept'

# iptables-translate -A INPUT -i eth0 -m state --state ESTABLISHED,RELATED -j
ACCEPT
nft 'add rule ip filter INPUT iifname "eth0" ct state related,established
```

```
counter accept'
```

Si lo intentamos con una regla sin traducción, el resultado es el siguiente:

```
# iptables-translate -A INPUT -j CHECKSUM --checksum-fill
nft # -A INPUT -j CHECKSUM --checksum-fill
```

7. Comparación de comandos usuales

Comando	iptables	nftablesHerramienta a utilizar
Mostrar todas las reglas	iptables-save	nft list ruleset
Mostrar las reglas de una tabla	iptables -L	nft list table ip filter
Mostrar las reglas de la tabla filter y cadena	iptables -L INPUT	nft list chain ip filter INPUT
Mostrar las reglas de una tabla y cadena	iptables -t nat -L PREROUTTING	nft list chain ip nat PREROUTING

Nftables no crea ninguna tabla ni cadena por defecto. Solo existen las que el usuario crea específicamente.

8. Trabajando con tablas

Una tabla es un *namespace* que contiene una colección de cadenas, reglas, *sets* y otros objetos.

Cada tabla debe tener asignada una familia de direcciones (*address family*). La familia define los tipos de paquetes que la tabla es capaz de procesar. Al crear una tabla hay que especificar una familia de entre las siguientes que NetFilter soporta:

- **ip**: capaz de manejar paquetes IPv4. Esta es la familia por defecto si no se especifica ninguna a la hora de crear la tabla.
- **ip6**: permite manejar paquetes IPv6.
- **inet**: permite manejar tanto paquetes IPv4 como IPv6.
- **arp**: coincide con paquetes del protocolo de resolución de direcciones IPv4 (ARP).
- **bridge**: coincide con paquetes que pasan a través de un dispositivo bridge.
- **netdev**: permite gestionar los paquetes que pasan por *ingress*. Es decir paquetes que

acaban de llegar al dispositivo de red y aún no han sido procesados por el *stack* de red del kernel.

Podemos añadir una tabla de dos formas, según el formato de script que hayamos seleccionado:

```
# Formato nativo:
table <table_address_family> <table_name> {

}

# Formato shell script:
nft add table <table_address_family> <table_name>
```

Por ejemplo, podemos definir una tabla *filter* para procesar paquetes Ipv4 así:

```
# nft add table ip filter
```

Podemos mostrar todas las tablas con el comando:

```
# nft list tables
```

Podemos eliminar una tabla con el comando:

```
# nft delete table ip table_name
```

Podemos eliminar todas las reglas que pertenecen a una tabla con el siguiente comando, el comando elimina todas las reglas de todas las cadenas, pero los conjuntos definidos permanecen:

```
# nft flush table ip filter
```

9. Trabajando con cadenas

Una tabla consiste en un conjunto de cadenas (*chains*). Cada cadena está formada por un conjunto de reglas.

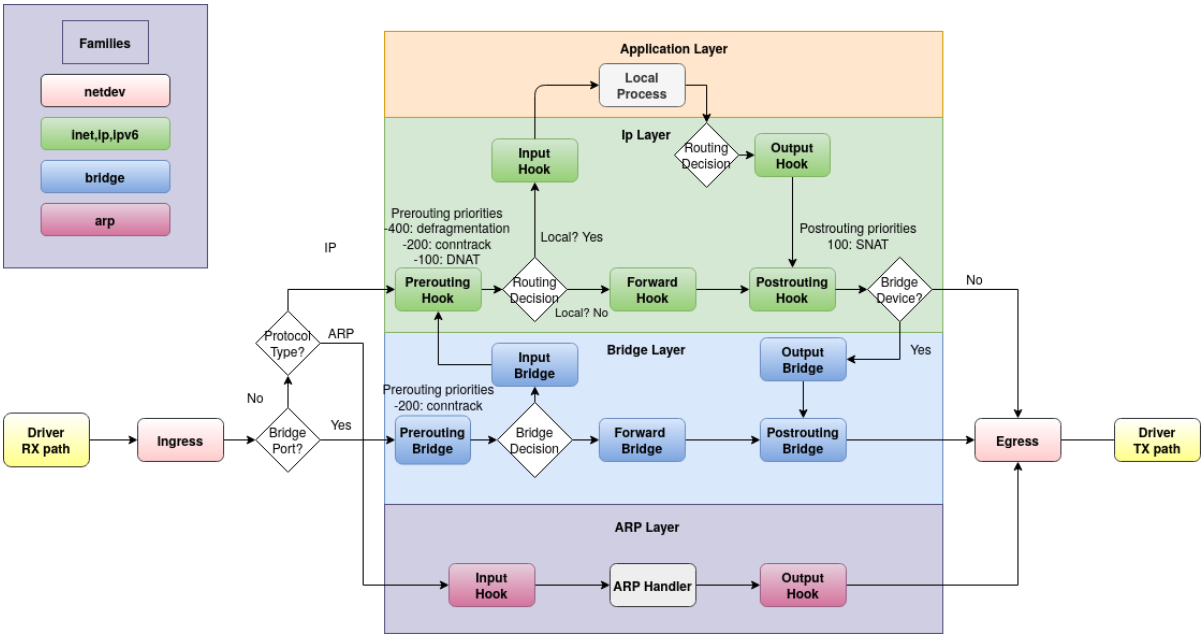
Existen dos tipos de cadenas:

- **Cadena base** (*base chain*):
    - Estas cadenas se usan como punto de entrada de los paquetes desde el *stack* de red. Al registrarse son capaces de "ver" el flujo de paquetes a través de la pila TCP/IP del kernel de Linux.
- **Cadena regular** (*regular chain*):

- Estas cadenas se usan como objetivo de salto (*jump target*) para una mejor organización de las reglas.

De la misma forma que se hacía en **iptables**, las reglas se insertan en cadenas. Pero en **nftables** no hay cadenas predefinidas como las que existían en **iptables**, no existen las cadenas INPUT, OUTPUT, FORWARD, etc. En **nftables** para filtrar paquetes en un determinado paso de procesamiento, hay que crear una cadena base con cualquier nombre a elegir, y enlazar dicha cadena a un *hook* de NetFilter. Esto aumenta la flexibilidad a la hora de configurar reglas, al mismo tiempo que no hay penalización en el rendimiento por culpa de cadenas que no son necesarias.

Los posibles *hooks* que se pueden utilizar al configurar una cadena base en **nftables** son:



- **ingress**.
  - Disponible solo en la familia **netdev** desde la versión 4.2 de Linux, y en la familia **inet** desde la 5.10.
  - Este *hook* ve los paquetes inmediatamente después de que son entregados por el driver de la tarjeta de red (interfaz NIC), incluso antes del prerouting. Esto permite ofrecer una alternativa a **tc** (Traffic Control).
- **prerouting**.
  - Ve todos los paquetes entrantes antes de que se tome cualquier decisión de enrutamiento. Los paquetes pueden estar dirigidos al propio sistema local o a procesos remotos.
- **input**.
  - Ve los paquetes entrantes que están dirigidos y han sido enrutados al sistema local y a los procesos que se ejecutan allí.
- **forward**.
  - Ve los paquetes entrantes que no están dirigidos al sistema local.
  - Son los paquete que un máquina que actúe como router debe procesar.

- **output**.
    - Ve los paquetes originados por procesos en la máquina local.
- **postrouting**.
    - Ve todos los paquetes después del enrutamiento, justo antes de que salgan del sistema local.

Estos **hooks** (ganchos) proporcionan puntos de interceptación para manejar paquetes a lo largo de diferentes etapas del flujo de red, permitiendo implementar reglas de filtrado, enrutamiento y manipulación de NAT en el punto más adecuado del proceso de manejo de paquetes.

Podemos añadir una cadena base a una tabla de dos formas, dependiendo del formato de script que estemos utilizando:

```
# Formato nativo:
table <table_address_family> <table_name> {
    chain <chain_name> {
        type <type> hook <hook> priority <priority>; policy <policy>;
    }
}

# Formato shell script:
nft add chain <table_address_family> <table_name> <chain_name> { type <type>
hook <hook> priority <priority> \; policy <policy> \; }    (1)
```

(1) El carácter **** es obligatorio para escapar los puntos y comas y que la shell no los procese.

Al definir una cadena hay que especificar de qué tipo es y a qué *hook* queremos enlazarla. en la siguiente tabla se muestran los tipos de cadena que existen, así como las familiar que los permiten y los hooks de enlace:

Tipo	Familia	Hooks	Descripción
filter	todas	todos	Cadena estándar
nat	ip, ip6, inet	prerouting, input, output, postrouting	Las cadenas de este tipo realizan una traducción de direcciones nativa basada en las entradas de seguimiento de conexiones. Solo el primer paquete atraviesa este tipo de cadena.
route	ip, ip6	output	Los paquetes aceptados que atraviesan este tipo de cadena provocan una nueva búsqueda en la tabla de enrutamiento si alguno de los campos relevantes de la cabecera IP ha cambiado.

Tal como vemos en la tabla, los tipos de cadena son tres:

- **filter**:
  - Se utiliza para filtrar paquetes.
  - Este tipo es compatible con las familias de tablas **`arp`**, **`bridge`**, **`ip`**, **`ip6`** e **`inet`**.
- **route**:
  - Se utiliza para redirigir paquetes si se modifica cualquier campo relevante de la cabecera IP o la marca del paquete.
  - Este tipo de cadena proporciona la misma funcionalidad equivalente a la tabla 'mangle` en **`iptables`**, pero solo para el hook **`output`**, para otros hooks hay que utilizar el tipo **`filter`**.
  - Este tipo es compatible con las familias de tablas **`ip, ip6`** e **`inet`**.
- **nat**:
  - Se utiliza para NAT (Network Address Translation).
  - Solo el primer paquete de un flujo dado atraviesa esta cadena, los paquetes subsiguientes la omiten. Por lo tanto, nunca se debe usar esta cadena para filtrar.
  - El tipo de cadena **`nat`** es compatible con las familias de tablas **`ip`**, **`ip6`** e **`inet`**.

Cada tipo de cadena está diseñado para una función específica dentro del manejo de paquetes, lo que permite a los administradores de sistemas y desarrolladores configurar de manera eficiente y precisa cómo se procesan los paquetes en la red.

Además del tipo, al definir una cadena hay que indicar la prioridad. La prioridad especifica el orden en el que los paquetes atraviesan las cadenas en el mismo *hook*. Para definir la prioridad se puede utilizar un valor entero o usar un nombre de prioridad. La siguiente table muestra los nombres de las prioridades estándar así como las familias y hooks en los que se pueden utilizar:

Nombre	Valor numérico	Familia	Hooks
raw	-300	ip, ip6, inet	todos
mangle	-150	ip, ip6, inet	todos
dstnat	-100	ip, ip6, inet	prerouting
dstnat	-300	bridge	prerouting
filter	0	ip, ip6, inet, arp, netdev	todos
filter	-200	bridge	todos
security	50	ip, ip6, inet	todos
srcnat	100	ip, ip6, inet	postrouting
srcnat	300	bridge	postrouting
out	100	bridge	output

A cada cadena base **nftables** se le asigna una prioridad que define su orden entre otras cadenas base, flowtables y operaciones internas de Netfilter en el mismo *hook*. Por ejemplo, una cadena en el *hook* prerouting con prioridad -300 se colocará antes que las operaciones de seguimiento de conexión (*connection tracking*).

Si un paquete es aceptado y existe otra cadena, con el mismo tipo de *hook* y con una prioridad posterior, entonces el paquete atravesará posteriormente esta otra cadena. Por lo tanto, un veredicto de aceptación ya sea por medio de una regla o de la política de cadena por defecto, no es necesariamente definitivo. Sin embargo, no ocurre lo mismo con los paquetes que son objeto de un veredicto *drop*. En su lugar, *drops* tienen efecto inmediato, sin que se evalúen más reglas o cadenas.

El siguiente conjunto de reglas demuestra esta diferencia de comportamiento:

```
table inet filter {
        # This chain is evaluated first due to priority
        chain services {
                type filter hook input priority 0; policy accept;

                # If matched, this rule will prevent any further evaluation
                tcp dport http drop

                # If matched, and despite the accept verdict, the packet
proceeds to enter the chain below
                tcp dport ssh accept

                # Likewise for any packets that get this far and hit the
default policy
        }

        # This chain is evaluated last due to priority
        chain input {
                type filter hook input priority 1; policy drop;
                # All ingress packets end up being dropped here!
        }
}
```

Si la prioridad de la cadena **input** anterior se cambiara a -1, la única diferencia sería que ningún paquete tendría la oportunidad de entrar en la cadena **services**. De cualquier manera, este conjunto de reglas resultará en que todos los paquetes de entrada serían descartados.

En resumen, los paquetes atravesarán todas las cadenas dentro del ámbito de un *hook* dado hasta que sean descartados o no existan más cadenas base. Sólo se garantiza que un veredicto de aceptación sea definitivo en el caso de que no haya ninguna cadena posterior con el mismo tipo de *hook* que la cadena en la que entró originalmente el paquete.

En cuanto a las políticas, la política por defecto de la cadena define qué tiene que hacer nftables si las reglas definidas no especifican ninguna acción.

En una cadena se pueden establecer las siguientes políticas por defecto:

- **accept**. El paquete se acepta, es la política por defecto.
- **drop**. El paquete se descarta.

La política por defecto es la acción (veredicto) que se aplicará a los paquetes que lleguen al final de la cadena (es decir, sin más reglas contra las que evaluarse).

El veredicto **`accept`** significa que el paquete seguirá atravesando la pila de red (por defecto), mientras que el veredicto **`drop`** significa que el paquete es descartado si el paquete alcanza el final de la cadena base.

Si no se selecciona explícitamente ninguna política, se utilizará la política por defecto **`accept`**.

9.1. Ejemplo

Por ejemplo, podemos añadir la cadena **`input`** a la tabla previamente definida **`filter`** con el siguiente comando:

```
# nft 'add chain ip filter input { type filter hook input priority 0 ; }'
```

Si queremos evitar el tener que escapar gran parte de los caracteres que se usan en **`nft`**, podemos ejecutar el programa de forma interactiva a través del comando **`nft -i`**`

El comando **`add chain`** anterior, registra la cadena **`input`**, asignándola al hook **`input`**, de tal forma que la nueva cadena sea capaz de "ver" y procesar los paquetes que van dirigidos a los procesos locales de la máquina.

La prioridad es importante ya que determina el orden de las cadenas, de tal forma, que si se tienen varias cadenas en el hook **`input`**, la prioridad determina qué cadena verá los paquetes antes que otras. Por ejemplo, las cadenas **`input`** con prioridades -12, -1, 0, 10 serían consultadas exactamente en ese orden. Es posible dar a dos cadenas base la misma prioridad, pero no hay un orden de evaluación garantizado de cadenas base con idéntica prioridad que estén adjuntas al mismo hook de Netfilter.

Si se quiere usar nftables para filtrar el tráfico de una máquina Linux de escritorio (que no reenvíe tráfico), se puede registrar también la cadena de salida a través de este comando:

```
# nft 'add chain ip filter output { type filter hook output priority 0 ; }'
```

De esta forma, se está preparado para filtrar tráfico, tanto el tráfico de entrada (*incoming*) dirigido a procesos locales de la máquina, como el tráfico de salida (*outgoing*) generado por los procesos locales.

Si no se incluye la configuración entre llaves, estaremos creando una cadena regular que no será capaz de ver ningún paquete (simlar a las cadenas creadas con **`iptables -N`**).

Si además queremos definir una política por defecto para la cadena, el comando a ejeutar sería el siguiente:

```
# nft 'add chain ip filter output { type filter hook output priority 0 ; policy
accept; }'
```

9.2. Cadenas regulares

Se pueden crear, además de las cadenas base, las denominadas cadenas normales o regulares (*regular chains*) de forma análoga a como se hacía en iptables con las cadenas de usuario. El comando básico es:

```
# nft -i
nft> add chain [family] <table_name> <chain_name> [{ [policy <policy> ;]
[comment "text comment about this chain" ;] }]
```

Se puede seleccionar cualquier cadena arbitraria para el nombre de la cadena.

Es importante tener en cuenta que cuando se añade una cadena regular en **`nftables`**, no se incluye la palabra clave **`hook`**. Dado que una cadena regular no está vinculada a ningún *hook* de Netfilter, por sí misma no será capaz de ver ningún tráfico directamente. Sin embargo, una o más cadenas base pueden incluir reglas que realicen saltos o vayan a estas cadenas (*jump/goto*), por lo que una cadena regular puede acabar procesando los paquetes exactamente de la misma manera que la cadena base que la llamó. Esto puede ser muy útil para organizar el conjunto de reglas en un árbol de cadenas base y regulares mediante el uso de las acciones *jump* y/o *goto*.

Además, existe otra característica de nftables, los *vmaps*, que ofrecen un mecanismo mucho más potente de construir conjuntos de reglas ramificados altamente eficientes. Los *vmaps* permiten mapear valores de campos de paquetes a acciones específicas, lo que puede simplificar significativamente la implementación y mejorar la eficiencia de las reglas en escenarios complejos.

Como ejemplo de cadena regular, podríamos tener la siguiente, denominada **`input_LAN`**:

```
table ip filter {
    chain input_LAN {
        tcp dport ssh  counter  accept comment "Accept SSH     (TCP 22)"
        udp dport 3394 counter  accept comment "Accept OpenVPN (UDP 3394)"
    }
```

```
    chain input {
        type filter hook input priority filter; policy drop;

        iif lo                                  accept comment "Accept any
localhost traffic"
        ct state invalid                        drop   comment "Drop invalid
connections"
        ct state established,related            accept comment "Accept traffic
originated from us"

        meta l4proto icmp            counter accept comment "Accept ICMP"
        ip protocol igmp             counter accept comment "Accept IGMP"

        iifname enp1s0 ct state new  jump input_LAN comment "Connections from
private IP address ranges"
    }
}
```

En la cadena *input* se define como última regla que todo lo que llegue a la máquina como trafico de conexión de entrada que se procese en la cadena **`input_LAN`**. De esta forma se agrupan las reglas y se muestra de una forma más visibla qué trafico de entrada permite la máquina.

Podemos crear una tabla de filtrado para una máquina de escritorio (workstation), de tal forma que filtre el tráfico que entra y que sale del host a través de estos comandos (suponiendo conectividad 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 ; }'
```

Una vez definida la tabla y sus cadenas, solo queda empezar a adjuntar reglas a estas dos cadenas base. Al tratarse de una máquina final, no hace falta la cadena **`forward`**, ya que la máquina no hará de router.

En nftables, las cadenas regulares (no base) se procesan de manera secuencial. Esto significa que las reglas en una cadena se evalúan en el orden en que están definidas. En cuanto a la definición y la llamada tenemos que tener en cuenta:

- **Definición de cadenas regulares**:
    - Las cadenas regulares se definen dentro de una tabla y se utilizan para organizar las reglas de filtrado de tráfico. Se pueden definir múltiples cadenas regulares para diferentes propósitos y referenciarlas desde otras cadenas.
    - Es una gran ayuda a la hora de estructurar y ordenar las reglas.
- **Llamada a cadenas regulares**:
    - Una cadena regular puede ser llamada desde una cadena base utilizando la acción **`jump`** o **`goto`**.

- La diferencia entre **jump** y **goto** es que **jump** volverá a la cadena original después de procesar la cadena regular, mientras que **goto** transferirá el control a la cadena regular sin volver.

9.3. Borrado de cadenas

En cuanto a la eliminación de cadenas, se puede realizar a través del siguiente comando:

```
# nft delete chain [family] <table_name> <chain_name>
```

Siempre y cuando la cadena esté vacía, es decir, no contenga ninguna regla, por ejemplo, podemos borrar la cadena **input** de la tabla **mytable** así:

```
# nft flush chain mytable input
```

Podemos borrar todas las reglas de una tabla a través del comando:

```
# nft flush table mytable
```

10. Trabajando con reglas

Las reglas ejecutan acciones sobre los paquetes de red, aceptándolos o descartándolos al atravesar la cadena. Estas decisiones se basan en si cumplen o no, los criterios que se especifican.

Cada regla consta de cero o más expresiones seguidas por una o más declaraciones. Cada expresión comprueba si un paquete coincide con un campo específico de los datos o metadatos del paquete o flujo. Las expresiones múltiples se evalúan linealmente de izquierda a derecha, si la primera expresión coincide, entonces se evalúa la siguiente expresión y así sucesivamente.

Si llegamos a la expresión final, entonces el paquete coincide con todas las expresiones en la regla, y se ejecutan las declaraciones de la regla.

Cada declaración toma una acción, como establecer la marca de Netfilter, contar el paquete, registrar (logging) el paquete o emitir un veredicto como aceptar/descartar el paquete o saltar a otra cadena.

Al igual que con las expresiones, las declaraciones múltiples se evalúan linealmente de izquierda a derecha: una sola regla puede tomar múltiples acciones usando varias declaraciones. Cabe destacar que una declaración de veredicto por su naturaleza termina la regla.

La forma de añadir una cadena depende de si seguimos el formato nativo:

```
table <table_address_family> <table_name> {
    chain <chain_name> {
         type <type> hook <hook> priority <priority> ; policy <policy> ;
         <rule>
    }
}
```

O de si usamos el formato script:

```
nft add rule <table_address_family> <table_name> <chain_name> <rule>
```

Si usamosd **`ntf add`** la regla se añadirá al final de la cadena, si queremos insertar la regla al principio de la cadena utilizaremos el comando **`nft insert`**.

10.1. Añadiendo reglas

Para agregar nuevas reglas, hay que especificar la tabla correspondiente y la cadena donde añadir la regla, como por ejemplo:

```
1 # 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 añade una regla para "coincidir" (hacer _match) con todos los paquetes que la cadena "output" ve y cuya dirección IP destino sea la **`8.8.8.8`**; en caso de coincidencia, actualiza los contadores de la regla.

Al contrario que en **`iptables`**, los contadores en **`nftables`** son opcionales.

Agregar una regla es equivalente a ejecutar el comando **`iptables -A`**.

10.2. Mostrando reglas

Creamos la tabla filter y añadimos la cadena input y output junto con un par de reglas:

```
# nft flush ruleset

# 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 ; }'
# nft add rule filter output ip daddr 8.8.8.8 counter
# nft add rule filter output tcp dport ssh counter
```

Podemos mostrar las reglas contenidas en la tabla con los siguientes comandos:

```
# nft list tables
```

```
table ip filter

# nft -n list table filter
table ip filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }

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

También podemos mostrar las reglas de una cadena en concreto:

```
# nft -n list chain filter output
table ip filter {
    chain output {
        type filter hook output priority 0; policy accept;
        ip daddr 8.8.8.8 counter packets 0 bytes 0
        tcp dport 22 counter packets 0 bytes 0
    }
}
```

10.3. Probando reglas definidas

Podemos probar las reglas generando tráfico para esa regla en concreto y comprobar cómo se modifican los contadores de paquetes y bytes:

```
# nft -n list table filter
table ip filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
        ip daddr 8.8.8.8 counter packets 0 bytes 0          (1)
        tcp dport 22 counter packets 0 bytes 0
    }
}


# ping -c 4 8.8.8.8                                        (2)
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=11.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=12.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=116 time=12.2 ms
```

```
64 bytes from 8.8.8.8: icmp_seq=4 ttl=116 time=12.9 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 11.765/12.216/12.865/0.408 ms

# nft -n list table filter
table ip filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
        ip daddr 8.8.8.8 counter packets 4 bytes 336       ❸
        tcp dport 22 counter packets 0 bytes 0
    }
}
root@fw:~#
```

❶ Aún no se ha generado tráfico ICMP desde la máquina.

❷ Lanzamos cuatro paquetes ICMP *echo request* al servidor de Google.

❸ Comprobamos que se han incrementado los contadores de la regla de Nftables.

10.4. Insertando reglas

Podemos insertar una regla al principio de una cadena a través del comadno **`insert`**, de la misma forma que hacíamos con el comanod **`iptables -I`**:

```
# nft -n list ruleset
table ip filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
        ip protocol 6 counter packets 13 bytes 1528
        ip saddr 127.0.0.1 ip daddr 127.0.0.6 drop
    }
}
# nft insert rule filter output ip daddr 192.168.1.1 counter
# nft -n list ruleset
table ip filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
        ip daddr 192.168.1.1 counter packets 0 bytes 0     ❶
```

```
        ip protocol 6 counter packets 26 bytes 3608
        ip saddr 127.0.0.1 ip daddr 127.0.0.6 drop
    }
}
```

❶ La regla se inserta en primera posición.

Si queremos insertar una regla en una posición determinada, debemos usar los identificadores de las reglas (*handlers*) como referencias, para ello mostramos las reglas con la opción **-a**:

```
# nft -n -a list ruleset
table ip filter { # handle 5
    chain input { # handle 1
        type filter hook input priority 0; policy accept;
    }

    chain output { # handle 2
        type filter hook output priority 0; policy accept;
        ip protocol 6 counter packets 136 bytes 13756 # handle 3
        ip saddr 127.0.0.1 ip daddr 127.0.0.6 drop # handle 4
    }
}
```

Si quisiéramos añadir una regla **después** de la regla con *id* 3, ejecutamos el siguiente comando:

```
# nft add rule filter output position 3 ip daddr 127.0.0.8 drop

# nft -n -a list ruleset
table ip filter { # handle 5
    chain input { # handle 1
        type filter hook input priority 0; policy accept;
    }

    chain output { # handle 2
        type filter hook output priority 0; policy accept;
        ip protocol 6 counter packets 154 bytes 16172 # handle 3
        ip daddr 127.0.0.8 drop # handle 5
        ip saddr 127.0.0.1 ip daddr 127.0.0.6 drop # handle 4
    }
}
```

Si hubiéramos querido insertra la regla **antes** de la regla con *id* 3, el comando hubiera sido:

```
# nft insert rule filter output position 3 ip daddr 127.0.0.8 drop
```

10.5. Eliminando reglas

Hay dos formas de eliminar una regla, la má sencilla es eliminarla por su identificador. Si tenemos las siguientes reglas:

```
# nft -n -a list ruleset
table ip filter { # handle 6
    chain input { # handle 1
        type filter hook input priority 0; policy accept;
    }

    chain output { # handle 2
        type filter hook output priority 0; policy accept;
        ip protocol 6 counter packets 31 bytes 2984 # handle 3
        ip saddr 127.0.0.1 ip daddr 127.0.0.6 drop # handle 4
    }
}
```

Podemos borrar la regla 4 con este comando:

```
# nft delete rule filter output handle 4

# nft -n -a list ruleset
table ip filter { # handle 6
    chain input { # handle 1
        type filter hook input priority 0; policy accept;
    }

    chain output { # handle 2
        type filter hook output priority 0; policy accept;
        ip protocol 6 counter packets 130 bytes 12356 # handle 3
    }
}
```

Podemos borrar todas las reglas de una cadena con el comadno:

```
# nft flush chain filter output
```

Y todas las reglas de una tabla con el comando:

```
# nft flush table filter
```

Y borrarlo todo con el comando:

```
# nft flush ruleset
```

10.6. Sustituyendo una regla por otra

Se puede reemplazar cualquier regla mediante el comando **replace** indicando el identificador de la regla, si tenemos una regla para contar todos los paquetes TCP y queremos sutituirla por una regla para contar TODOS los paquetes, los comandos serían:

```
# nft -a list ruleset
table ip filter { # handle 10
    chain input { # handle 1
        type filter hook input priority filter; policy accept;
        ip protocol tcp counter packets 89 bytes 5928 # handle 2
    }
}
# nft replace rule filter input handle 2 counter

# nft -a list ruleset
table ip filter { # handle 10
    chain input { # handle 1
        type filter hook input priority filter; policy accept;
        counter packets 7 bytes 504 # handle 2
    }
}
```

10.7. Utilizando expresiones en las reglas

A la hora de definir reglas podemos utilizar los siguientes operadores que **nftables** incluye:

Operador	Descripción
eq	Igual. También se puede utilizar **==** o no indicarlo (ej.: **dport 22**)
ne	Distinto. También se puede utilizar **!=**
lt	Menor que. También se puede utilizar **<**
gt	Mayor que. También se puede utilizar **>**
le	Menor o igual que. También se puede utilizar ⇐
ge	Mayor o igual que. También se puede utilizar **>=**

Ciertos caracteres, como **<** o **>** se interpretan por ciertas shells como Bash, en esos casos es necesario escaparlos, como por ejemplo **\>**.

Ejemplos:

- Podemos seleccionar todo el tráfico de entrada TCP a puertos que no sean el de SSH así:

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

- O filtrar tráfico TCP a puertos superiores al 1024 así:

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

Existe una gran multitud de selectores que podemos utilizar en nuestras reglas para poder seleccionar paquetes y aplicarles una acción.

Los selectores más importantes agrupados por categorías son:

- Selectores para metainformación del paquete.
- Selectores para las interfaces.
- Selectores paquetes IP.
- Selectores de estado de conexión.
- Selectores para marcas de paquete y clase de enrutamiento.
- Selectores para usuarios/grupos.
- Selectores para tiempo.
- Selectores de seguridad.
- Otros selectores.
- Selectores Ethernet y VLAN.
- Selectores ICMP.

10.7.1. Selectores para metainformación del paquete

Selector	Descripción	Observaciones
pkttype	Tipo de paquete a nivel de enlace	Identifica si el paquete es unicast, broadcast o multicast
length	Longitud del paquete	Longitud total del paquete en bytes
protocol	Número de protocolo en la capa de red	Por ejemplo, ICMP (1), TCP (6), UDP (17)
nfproto	Protocolo manejado por netfilter	Identifica el protocolo específico de netfilter (IPv4, IPv6, ARP, etc.)
l4proto	Protocolo de la capa 4	Protocolo de transporte, como TCP, UDP, etc

10.7.2. Selectores para las interfaces

Selector	Descripción	Observaciones
iif	Índice de la interfaz de entrada	Número de índice de la interfaz por la cual entra el paquete
iifname	Nombre de la interfaz de entrada	Nombre de la interfaz por la cual entra el paquete
iiftype	Tipo de interfaz de entrada	Tipo de la interfaz (por ejemplo, **bridge**, **vlan**, etc.)
iifkind	Tipo específico de interfaz de entrada	Específico para tipos particulares como **bridge** o **vlan**
iifgroup	Grupo de la interfaz de entrada	Grupo al que pertenece la interfaz de entrada
oif	Índice de la interfaz de salida	Número de índice de la interfaz por la cual sale el paquete
oifname	Nombre de la interfaz de salida	Nombre de la interfaz por la cual sale el paquete
oiftype	Tipo de interfaz de salida	Tipo de la interfaz (por ejemplo, **bridge**, **vlan**, etc.)
oifkind	Tipo específico de interfaz de salida	Específico para tipos particulares como **bridge** o **vlan**
oifgroup	Grupo de la interfaz de salida	Grupo al que pertenece la interfaz de salida
ibrname	Nombre de la interfaz puente de entrada	Nombre del puente al que pertenece la interfaz de entrada
obrname	Nombre de la interfaz puente de salida	Nombre del puente al que pertenece la interfaz de salida
ibrvproto	Protocolo de la VLAN del puente de entrada	Identificador del protocolo VLAN en el puente de entrada
ibrpvid	ID de la VLAN principal del puente de entrada	ID de la VLAN principal asociada con el puente de entrada
sdif	Índice de la interfaz de entrada secundaria	Índice de la interfaz secundaria de entrada (en caso de interfaces combinadas)
sdifname	Nombre de la interfaz de entrada secundaria	Nombre de la interfaz secundaria de entrada

10.7.3. Selectores paquetes IP

Selector	Descripción	Observaciones
`ip saddr`	Dirección IP de origen	Puede ser una dirección IP específica o un rango de direcciones
`ip daddr`	Dirección IP de destino	Puede ser una dirección IP específica o un rango de direcciones
`ip protocol`	Protocolo de la capa IP	Por ejemplo, TCP (6), UDP (17), ICMP (1)
`ip length`	Longitud total del paquete IP	Permite filtrar paquetes por su tamaño total
`ip ttl`	Time to Live del paquete IP	Permite filtrar paquetes por su valor TTL
`ip tos`	TOS/DCP del paquete IP	Utilizado para la clasificación de servicio
`ip frag`	Información de fragmentación del paquete IP	Permite filtrar paquetes basados en si están fragmentados
`ip id`	Identificador del paquete IP	Útil para rastrear paquetes específicos
`ip flags`	Banderas del encabezado IP	Por ejemplo, `DF` (Don't Fragment) y `MF` (More Fragments)
`tcp dport`	Puerto de destino TCP	Permite filtrar por el puerto de destino TCP
`tcp sport`	Puerto de origen TCP	Permite filtrar por el puerto de origen TCP
`tcp flags`	Banderas del encabezado TCP	Por ejemplo, `SYN`, `ACK`, `FIN`, `RST`, `PSH`, `URG`
`tcp option`	Opciones del encabezado TCP	Permite filtrar por opciones específicas en el encabezado TCP
`udp dport`	Puerto de destino UDP	Permite filtrar por el puerto de destino UDP
`udp sport`	Puerto de origen UDP	Permite filtrar por el puerto de origen UDP

10.7.4. Selectores de estado de conexión

Selector	Descripción	Observaciones
`ct state`	Estado de la conexión	Valores típicos incluyen `new`, `established`, `related`, `invalid`

Selector	Descripción	Observaciones
`ct status`	Estado detallado de la conexión	Incluye estados como `expected`, `seen-reply`, `assured`, entre otros
`ct direction`	Dirección de la conexión relativa	`original` o `reply` dependiendo del flujo de la conexión
`ct mark`	Marca asociada a la conexión	Usada para etiquetar conexiones para procesamiento especial
`ct expire`	Tiempo restante antes de que expire la conexión	Útil para identificar y manejar conexiones que están por cerrarse

10.7.5. Selectores para marcas de paquete y clase de enrutamiento

Selector	Descripción	Observaciones
`mark`	Marca del paquete	Utilizada para identificar o clasificar paquetes dentro de las reglas
`priority`	Prioridad del paquete	Puede influir en el procesamiento de paquetes, como en la cola de paquetes
`rtclassid`	Clase de enrutamiento (ID)	Identificador de la clase de enrutamiento utilizado para políticas de tráfico

10.7.6. Selectores para usuarios/grupos

Selector	Descripción	Observaciones
`skuid`	UID de usuario del socket	Identifica al usuario del proceso que originó el paquete
`skgid`	GID de grupo del socket	Identifica al grupo del proceso que originó el paquete

10.7.7. Selectores para tiempo

Selector	Descripción	Observaciones
`time`	Timestamp de la recepción del paquete	ns epoch o string formato ISO
`day`	Día del mes	Rango de 1 a 31, según el mes
`hour`	Hora del día	Formato de 24 horas, rango de 0 a 23

10.7.8. Selectores de seguridad

Selector	Descripción	Observaciones
`cpu`	CPU por la que el paquete fue procesado	Útil para sistemas con múltiples CPUs o en debugging
`cgroup`	ID del cgroup del proceso que originó el paquete	Permite aplicar reglas basadas en los cgroups de Linux
`secmark`	Marca de seguridad asignada al paquete	Utilizada en conjunto con políticas SELinux o similares
`ipsec`	Información del estado IPSec del paquete	Permite filtrar por estado de cifrado/descifrado IPSec

10.7.9. Otros selectores

Selector	Descripción	Observaciones
`nftrace`	Bit de depuración de nftrace	Permite activar el seguimiento del paquete para propósitos de debugging
`random`	Generar un valor aleatorio	Utilizado para tomar decisiones basadas en un porcentaje aleatorio, útil para balanceo de carga o muestreo

10.7.10. Selectores Ethernet y VLAN

Selector	Descripción	Observaciones
`ether saddr`	Dirección MAC de origen del paquete	Utilizada para identificar o filtrar por la dirección MAC del remitente
`ether daddr`	Dirección MAC de destino del paquete	Utilizada para identificar o filtrar por la dirección MAC del destinatario
`ether type`	Tipo de protocolo de la capa de enlace de datos	Permite filtrar paquetes basados en el tipo de protocolo Ethernet, como IP, ARP, etc
`vlan type`	Tipo de encapsulamiento VLAN	Identifica el protocolo de encapsulamiento VLAN, como 802.1Q
`vlan id`	ID de la VLAN	Número de identificación de la VLAN, permitiendo filtrar por VLAN específica
`vlan cfi`	DEI, antes CFI	Canal de Fidelidad de VLAN (Canonical Format Indicator), indica si la dirección MAC escanónica (0) o no canónica (1).
`vlan pcp`	PCP	Valor de prioridad de la VLAN (0-7), útil para QoS

10.7.11. Selectores ICMP

Selector	Descripción	Observaciones
`icmp type`	Tipo de mensaje ICMP	Identifica el tipo de mensaje ICMP, como **`echo-request`** o **`destination-unreachable`**
`icmp code`	Código del mensaje ICMP	Proporciona el código específico del mensaje ICMP, que varía según el tipo
`icmp id`	Identificador de mensaje ICMP para tipos específicos	Usado principalmente en mensajes como **`echo-request`** y **`echo-reply`**
`icmp seq`	Número de secuencia en mensajes ICMP	Utilizado en mensajes que requieren seguimiento de secuencia, como **`echo-request`**

En el caso de ICMP podemos obtener más información a través de los siguientes comandos:

```
# nft describe icmp type
payload expression, datatype icmp_type (ICMP type) (basetype integer), 8 bits

pre-defined symbolic constants (in decimal):
        echo-reply                                      0
        destination-unreachable                         3
        source-quench                                   4
        redirect                                        5
        echo-request                                    8
        router-advertisement                            9
        router-solicitation                            10
        time-exceeded                                  11
        parameter-problem                              12
        timestamp-request                              13
        timestamp-reply                                14
        info-request                                   15
        info-reply                                     16
        address-mask-request                           17
        address-mask-reply                             18
# nft describe icmp code
payload expression, datatype icmp_code (icmp code) (basetype integer), 8 bits

pre-defined symbolic constants (in decimal):
        net-unreachable                                 0
        host-unreachable                                1
        prot-unreachable                                2
        port-unreachable                                3
        net-prohibited                                  9
        host-prohibited                                10
        admin-prohibited                               13
        frag-needed                                     4
```

Ejemplos:

```
# nft add rule filter input  icmp type echo-request counter drop
# nft add rule filter output icmp code frag-needed  counter drop
```

11. Scripting

Históricamente las reglas de cortafuegos con 'iptables` se han agrupando y lanzado desde un script de **Bash** (o cualquier otra shell). Esto ha proporcionado varias ventajas, como la posibilidad de añadir comentarios, agrupar las reglas, definir variables o utilizar estructuras como bucles y condicionales.

Sin embargo, el uso de scripts de shell es problemático debido a que rompen la atomicidad al aplicar el conjunto de reglas, por lo tanto, la política de filtrado se aplica de manera inconsistente durante el tiempo de "ejecución" del script.

Afortunadamente, nftables ofrece un entorno de scripting nativo para abordar estos problemas. Un script de Nftables permite, entre otras cosas incluir otros ficheros de conjunto de reglas, definir variables y añadir comentarios.

Para crear un script de nftables, basta con añadir el siguiente encabezado en el fichero (*shebang*):

```
#!/usr/sbin/nft -f
```

11.1. "Ejecutando" scripts de Nftables

Hay dos formas de ejecutar un script de Nftables, supongamos que tenemos un script en **/etc/nftables/main.nft**. Podemos lanzar el script de dos formas:

1. A través del propio comando **nft** con la opción **-f**:

   ```
   # nft -f /etc/nftables/main.nft
   ```

2. Directamente, para ello basta con que tenga definido el shebnag así como permisos de ejecución:

   ```
   # cd /etc/nftables
   # ./main.nft
   ```

   - Previamente damos permisos de ejecución con:

     ```
     # chmod u+x /etc/nftables/main.nft
     ```

11.2. Scripting por defecto en Ubuntu

Por defecto, desde Ubuntu 22.04, se incluye un script para configuraión básica de Nftables, **`/etc/nftables.conf`** con el siguiente contenido:

```
---
#!/usr/sbin/nft -f
```

flush ruleset

table inet filter { chain input { type filter hook input priority 0; } chain forward { type filter hook forward priority 0; } chain output { type filter hook output priority 0; } } ---

El script tan solo:

- Define una tabla **`filter`** para el filtrado, del tipo **`inet`** por lo que incluye tráfico tanto IPv4 como tráfico IPv6.
- Define las cadenas **`input`**, **`output`** y **`forward`** enganchjadas a los hooks correspondientes de NetFilter.
- Define **`accept`** como política por defecto, por lo que se acepta todo el tráfico ya que no se define ninguna regla adicional.

11.3. Estructuras a utilizar

En un script de Nftables podemos:

- Incluir comentarios.
- Definir variables.
- Incluir otros scripts de Nftables.

11.3.1. Añadiendo comentarios

Se pueden añadir comentarios con el carácter **`. Todo lo que se indique tras un carácter`** se ignorará. Ejemplo:

```
#!/usr/sbin/nft -f

#
# table declaration
#
add table filter    # Create a table

#
# 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
```

11.3.2. Incluyendo otros ficheros

Se puede dividir la configuración del cortafuegos en varios ficheros e incluirlos dede un fichero principal a través de sentencia **`include`**. Los directorios que se utilizarán para buscar ficheros a inclir se pueden especificar a través de las opciones **`-I`**/**`--includepath`**. Se puede anular este comportamiento anteponiendo **`./`** a la ruta para forzar *includes* de ficheros a partir del directorioa actual (rutas relativas) o el carácter **`/`** para la ubicación de ficheros expresando la ruta absoluta.

Si no se especifica **`-I/--includepath`**, entonces el comando **`nft`** utilizará como directorio por defecto el que se especificó en el momento de la compilación. Se puede obtener este fichero a través de las opciones **`-h`**/**`--help`**. Ejemplo en Ubuntu 22.04:

```
# nft --help
Usage: nft [ options ] [ cmds... ]

Options (general):
  -h, --help                        Show this help
  -v, --version                     Show version information
  -V                                Show extended version information
...
  -I, --includepath <directory>   Add <directory> to the paths searched for
include files. Default is: /etc
...
```

Tanto en Ubuntu 22/24 como en RHEL 9 el directorio por defecto es **`/etc`**. Una sentencia como **`include example.nft`** cargará el fichero **`/etc/example.nft`**.

Las sentencias **`include`** permiten el uso de símbolos comunes de comodines de la shell como **`*`**, **`?`** y **`[]`**. No es un error no tener ningún fichero a incluir en una declaración de inclusión si se usan estos caracteres comodín.

Esto permite tener directorios de inclusión potencialmente vacíos para declaraciones como **`include "/etc/firewall/rules/"`**. Las coincidencias de comodines se cargan en orden alfabético.

Los ficheros que comienzan con un punto (**`.`**) no son coincidencias por las declaraciones de inclusión al usar caracteres comodín.

Ejemplo:

```
#!/usr/sbin/nft -f
```

```
# include a single file using the default search path
include "ipv4-nat.ruleset"

# include all files ending in *.nft in the default search path
include "*.nft"

# include all files in a given directory using an absolute path
include "/etc/nftables/*"
```

11.3.3. Definiendo variables

Se puede usar la palabra clave **define** para definir variables. En una variable se pueden almacenar tanto valores simples como conjuntos anónimos. Para valores más complejos se usan *sets* (conjuntos) y *verdict maps* (mapas de veredicto). Para utilizar la variable se le antepone el carácter **$**.

El siguiente ejemplo muestra el conjunto de reglas necesarias para contabilizar el tráfico de salida hacia el servidor DNS de Google 8.8.8.8.

```
#!/usr/sbin/nft -f

define google_dns = 8.8.8.8

flush ruleset

add table filter
add chain filter output { type filter hook input priority 0; }
add rule filter output ip saddr $google_dns counter
```

De la misma forma, también se pueden usar variables para definir *sets* anónimos (conjuntos). Para definir un conjunto anónimo se utilizan las llaves (**{ … }**):

```
#!/usr/sbin/nft -f

# DNS servers:
#  Google
#  Cloudflare
#  OpenDNS
define dns_servers = { 8.8.8.8, 8.8.4.4, 1.1.1.1, 208.67.222.123,
208.67.220.123 }

flush ruleset

add table filter
add chain filter output { type filter hook input priority 0; }
add rule filter output ip saddr $dns_servers counter
```

Es importante recordar que las llaves tienen un significado especial cuando se utilizan desde las reglas, ya que indican que esta variable representa un

> conjunto.

Por lo tanto, hay que tratar de evitar definiciones como esta…

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

Ya que es excesivo definir un conjunto que solo almacena un único elemento, en vez de definir la variable con un valor simple como:

```
define google_dns = 8.8.8.8
```

11.4. Formatos de ficheros NFT

El comando **`nft -f <filename>`** acepta 2 formatos, el primero es el formato visto en la salida del comando **`nft list ruleset`**. El segundo utiliza la misma sintaxis de llamar al binario **`nft`** varias veces, pero de manera atómica.

Si tomamos el ejemplo anterior de DNS, podríamos definir el script de dos formas:

1. Formato de salida de nftables:

```
#!/usr/sbin/nft -f

define dns_servers = { 8.8.8.8, 8.8.4.4, 1.1.1.1, 208.67.222.123,
208.67.220.123 }

flush ruleset

table ip filter {
        chain output {
                type filter hook input priority filter; policy accept;
                ip saddr { 1.1.1.1, 8.8.4.4, 8.8.8.8, 208.67.220.123,
208.67.222.123 } counter packets 0 bytes 0
        }
}
```

2. Comandos a ejecutar de forma atómica:

```
#!/usr/sbin/nft -f

define dns_servers = { 8.8.8.8, 8.8.4.4, 1.1.1.1, 208.67.222.123,
208.67.220.123 }

flush ruleset

add table filter
add chain filter output { type filter hook input priority 0; policy
accept; }
```

```
add rule filter output ip saddr $dns_servers counter
```

Traducir de un formato a otro es muy sencillo, ya que la sintaxis es prácticamente la misma, solo difiere la organización de las reglas. Dependiendo del caso de uso, optaremos por usar un formato u otro en la construcción del firewall.

11.5. Construyendo un fichero *nft* desde un script

Aunque no necesariamente recomendado, siempre se puede usar cualquier lenguaje de script para construir uno varios ficheros para procesarlos posteriormente con **`nft`**. Siempre se puede usar este método si se está coordinando la configuración de **`nft`** con la de otros subsistemas o simplemente si se prefiere el uso de cualquier lenguaje de scripting como Bash, Python o Ruby.

La misma funcionalidad proporcionada por la utilidad de línea de comandos **`nft`** está disponible desde programas en Python a través de la biblioteca de alto nivel **`libnftables`**. Más información al respecto en cualquiera de estos enlaces:

- https://github.com/aborrero/python-nftables-tutorial
- https://ral-arturo.org/2020/11/22/python-nftables-tutorial.html

11.6. Cargando las reglas durante el inicio del sistema

Tanto Ubuntu como RHEL definen una unidad de Systemd para cargas las reglas de cortafuegos durante el inicio del sistema. Para habilitar el servicio basta con ejecutar los siguientes comandos:

```
# systemctl enable nftables
# systemctl start nftables
```

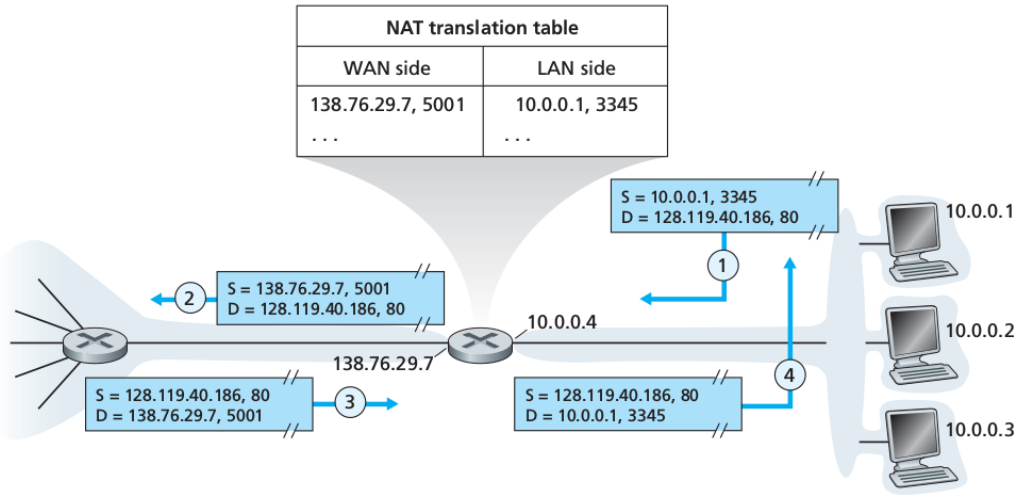
Todos los scripts se almacenan en el directorio **`/etc/nftables`**, el fichero principal es distinto según la distribución:

- En el caso de RHEL 9 se encuentra en **`/etc/sysconfig/nftables`**.
- En el caso de Ubuntu 22/24 se encuentra en **`/etc/nftables.conf`**.

12. NAT y Nftables

12.1. ¿Qué es NAT?

NAT (Network Address Translation) es un proceso consistente en alterar de alguna manera los orígenes o destinos de los paquetes.



Usamos NAT, y posiblemente abusamos de su uso, porque nos permite ahorrar direcciones IP públicas, nos proporciona un nivel de seguridad adicional, nos permite compartir conexiones a Internet y nos permite simplificar la administración de redes:

- **Escasez de direcciones IP**.
  - El uso de NAT permite a una organización utilizar un conjunto limitado de direcciones IP públicas para traducir múltiples direcciones IP privadas utilizadas en su red interna. Esto es especialmente importante debido a la escasez de direcciones IP públicas en IPv4.
- **Seguridad**.
  - NAT proporciona una capa adicional de seguridad al ocultar las direcciones IP privadas detrás de una dirección IP pública. Cuando los dispositivos de una red interna se conectan a Internet a través de una dirección IP pública, es más difícil para los atacantes directamente acceder a los dispositivos internos.
- **Conexiones compartidas**.
  - En entornos domésticos y pequeñas empresas, NAT se utiliza para compartir una conexión a Internet entre múltiples dispositivos. Los dispositivos de la red interna se traducen a una sola dirección IP pública, lo que permite el acceso a Internet para todos los dispositivos sin requerir direcciones IP públicas adicionales.
- **Simplificación de configuraciones**.
  - El uso de NAT simplifica la administración de redes al reducir la necesidad de configurar y administrar direcciones IP públicas únicas para cada dispositivo interno. Esto facilita el mantenimiento de la red y reduce la complejidad de la configuración.
- **Varios servidores**.
  - Si tenemos una única dirección IP válida utilizando un rango interno no válido, NAT nos permitirá poder ofrecer varios servicios al exterior desde varios servidores.
- **Transición a IPv6**.
  - NAT también se utiliza durante la transición de IPv4 a IPv6. Permite a las redes

internas utilizar direcciones IPv6 internamente y realizar la traducción a direcciones IPv4 para acceder a la Internet IPv4.

- **Conexiones a Internet a través de un ISP**.
    - La mayoría de ISP que dan acceso a Internet dan una sólo dirección IP que hay que compartir con varios hosts.

No son todo ventajas, NAT también presenta sus propios problemas:

- **Limitaciones de conectividad**.
    - El uso de NAT puede dificultar ciertos tipos de conexiones de red, como las conexiones punto a punto o las comunicaciones de extremo a extremo, ya que las direcciones IP internas se traducen a una dirección IP pública única. Esto puede complicar la configuración de ciertas aplicaciones o servicios que requieren una conectividad directa: FTP (modo activo), IRC, IPSec, VoIP, etc.
- **Dificultades en la monitorización y el registro de actividades**.
    - Debido a que NAT oculta las direcciones IP internas, la monitorización y el registro de actividades se vuelven más complicados. Identificar el origen exacto de un paquete o rastrear actividades específicas puede requerir herramientas o configuraciones adicionales.
- **Dependencia de puertos específicos**.
    - Para permitir la traducción correcta de direcciones IP y puertos, NAT a menudo se basa en asignar y mantener un seguimiento de los puertos utilizados en las comunicaciones. Esto puede generar problemas si se necesitan muchos puertos simultáneamente o si se utilizan protocolos que no son compatibles con NAT.
- **Complejidad en entornos empresariales**.
    - En redes empresariales más grandes y complejas, la implementación de NAT puede volverse complicada y requerir una planificación cuidadosa. La configuración de reglas de traducción y el manejo de múltiples direcciones IP públicas pueden aumentar la complejidad y la carga administrativa.
    - El router que realiza el NAT es un único punto de fallo.
- **Limitaciones para la autenticación y el cifrado de extremo a extremo**.
    - La traducción de direcciones IP en NAT puede interferir con la autenticación y el cifrado de extremo a extremo. Esto se debe a que las direcciones IP internas no son visibles más allá del router NAT, lo que dificulta la autenticación o el cifrado basado en direcciones IP.

Existen dos tipos básicos de NAT:

- Source NAT (**SNAT**).
    - Se produce cuando alteramos el origen de los paquetes.
    - Se hace siempre después del encaminamiento, justo antes de que el paquete salga.
- Destination NAT (**DNAT**).

- Se produce cuando alteramos el destino de los paquetes.
- Se hace siempre antes del encaminamiento, justo cuando el paquete llega.

12.2. NAT en Nftables

Desde el punto de vista de Nftables, NAT (Network Address Translation) se refiere a la técnica utilizada para mapear una dirección IP de origen o destino en un paquete IP a otra dirección IP.

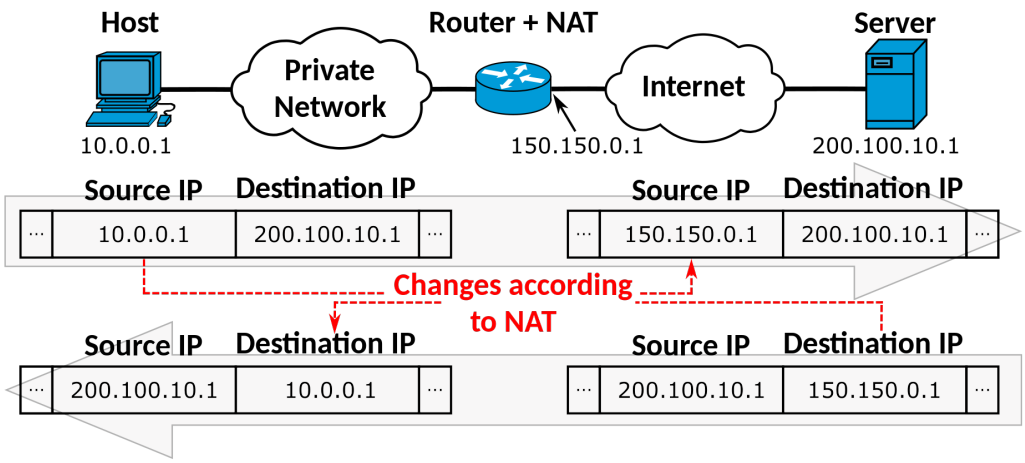
En nftables, las operaciones de NAT se realizan a través de cadenas específicas que se enganchan en puntos determinados del ciclo de vida de un paquete dentro de la arquitectura de filtrado del kernel de Linux. Los principales tipos de NAT en Nftables son:

- **Masquerading**.
  - No es más que un SNAT en la que la dirección IP fuente es dinámica (asignada por DHCP).
- Source NAT, **SNAT**.
- Destination NAT, **DNAT**.
- **Redirect**.

Masquerading y SNAT

Tanto SNAT como *masquerading*, que no es más que un tipo específico de SNAT, se utilizan para cambar la dirección IP fuente de los paquetes.

El típico ejemplo es el de los ISP, cuando contratamos Internet, nuestro proveedor nos proporciona un router, en un lado tiene configurada una dirección IP pública y válida, internamente nos configura un rango de red privado, como 192.168.1.0/24. Estas direcciones IP no son válidas en Internet, por lo para poder alcanzar Internet desde la red interna, uno de las tareas que tiene que hacer el router es mapear todas las direcciones privadas a una (o varias) de las direcciones públicas que nuestro router tenga asignadas.



SNAT y *masquerading* son muy similares entre sí, las principales diferencias son:

- *Masquerading* usa de forma automática la dirección IP que la interfaz de salida tiene

configurada en ese momento. Es la configuración que se recomienda si al router se le asigna la IP por DHCP.

- SNAT utiliza una direcció IP configurada previamente, por lo que no necesita comprobar cuál es la dirección de la interfaz. Es la configuración a utilizar cuando la IP de salida es fija. SNAT es más rápido.

DNAT

Destination NAT es un tipo de NAT que se utiliza para reescribir la dirección destino y puerto de los paquetes que llegan al router.

Por ejemplo, si tenemos un servidor web que no es accesible desde Internet, pero lo tenemos detrás de un router que sí lo es, podemos fijar una regla DNAT para que el router reenvíe los paquetes al servidor web.

Redirect

Es un tipo específico de DNAT en el que se redirigen paquetes a la propia máquina dependiendo del hook de la cadena.

Por ejemplo, si un servicio se ejecuta escuchando en un puerto que no es estándar, podemos redirigir el tráfico del puerto estándar al puerto en el que escucha el servicio.

Se pueden mezclar reglas de filtrado y reglas de NAT, tan solo hay que tener en cuenta:

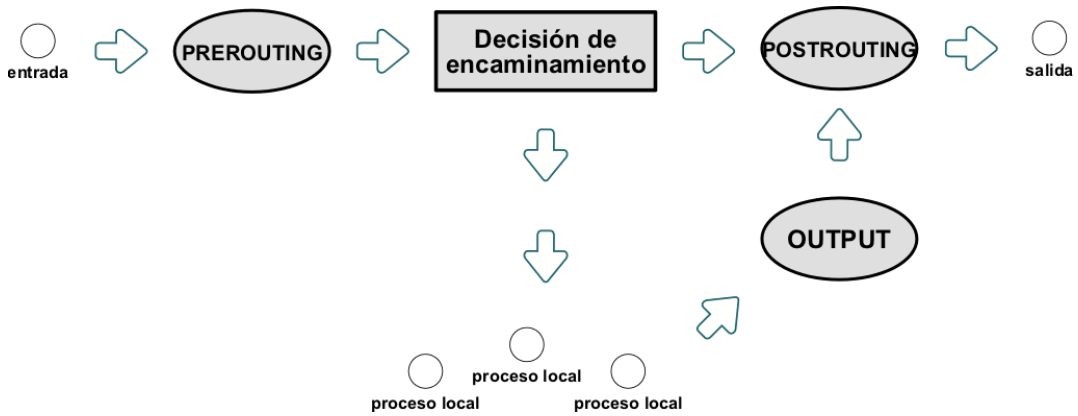
- Hay que diseñar el filtrado de paquetes ignorando completamente lo que se haga con NAT.
- Los orígenes y destinos vistos por el filtrado de paquetes serán los reales.

Merece la pena repetirlo: **se pueden mezclar reglas de filtrado y reglas de NAT, tan solo hay diseñar el filtrado de paquetes ignorando completamente lo que se haga con NAT, ya que los orígenes y destinos vistos por el filtrado de paquetes serán los reales**.

12.3. Configuración de SNAT/DNAT en nftables

En cuanto a NAT, hay tres *hooks* importantes:

- PREROUTING.
- POSTROUTING.
- OUTPUT.



Masquerading

Masquerading permite que un *router* pueda cambiarse la dirección IP de los paquetes que lo atraviesan de forma dinámica. Esto implica que si un *router* cambia su dirección IP, *nftables* empezará a usar la nueva dirección en los mapeos de los paquetes.

Si queremos cambiar la dirección IP fuente de todos los paquetes que salgan de la máquina a través de la interfaz **`eth0`** utilizando la dirección que la interfaz tenga configurada en ese momento, ejecutamos los siguientes comandos:

1. Creamos una nueva tabla, con el nombre `nat.

```
# nft add table
```

2. Añadimos las cadenas **`prerouting`** y **`postrouting`** a la nueva tabla.

```
# nft add chain nat postrouting { type nat hook postrouting priority 100
\; }
# nft -- add chain nat prerouting { type nat hook prerouting priority -100
\; }
```

Aunque no se añada una regla a la cadena de **`prerouting`**, el marco de trabajo de nftables requiere que esta cadena coincida con las respuestas de los paquetes entrantes.

La opción **`--`** se indica para evitar que la shell interprete el valor de prioridad negativa como una opción del comando **`nft`**.

3. Finalmente, agregamos la regla a la cadena **`postrouting`** encargada de reemplazar la dirección IP de origen de los paquetes salientes conla IP de la interfaz **`eth0`**:

```
# nft add rule nat postrouting oifname "eth0" masquerade
```

SNAT

En un *router*, **Source NAT** (SNAT) nos permite cambiar la IP de los paquetes enviados a través de una interfaz a una dirección IP específica. Se crea la tabla y las cadenas de la

misma forma que en *masquerading*, pero la regla es ligeramente diferente:

```
# nft add table
# nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
# nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
# nft add rule nat postrouting oifname "eth0" snat to _MY_PUBLIC_IP_GOES_HERE_
```

DNAT

Destination NAT (DNAT) nos permite redirigir el tráfico desde un router que recibe el tráfico, a un host que no es directamente accesible desde Internet.

Por ejemplo, con DNAT un router puede redirigir el tráfico entrante enviado a los puertos 80 y 443, a un servidor web interno con la dirección IP 192.168.100.1. Para ello seguimos los siguientes pasos:

1. Creamos la tabla necesaria y las cadenas, de la misma forma que en *masquerading* y SNAT, además deberemos configurar SNAT o *masquerading*, supongamos este último:

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

2. Agregamos una nueva regla en la cadena **`prerouting`** para redirigir el tráfico que llega al router a los puertos 80 y 443, al servidor web con la dirección IP 192.168.100.1:

   ```
   # nft add rule nat prerouting iifname "eth0" tcp dport { 80, 443 } dnat to
   192.168.100.1
   ```

3. Dependiendio del entorno, es probable que tengamos que añadir una regla SNAT o de *masquerading* para cambiar la dirección de origen de los paquetes que regresan del servidor web al remitente. En el caso de masquerading, la regla sería:

   ```
   # nft add rule nat postrouting oifname "eth0" masquerade
   ```

Habilitar el enrutamiento

Siempre que configuremos NAT, nuestra máquina estará haciendo de router, por lo que tendremos que habilitar el reenvío de paquetes.

Un router es una máquina que interconecta varias redes, es capaz de procesas paquetes dirigidos a ella misma, pero cuya dirección destino NO ES ella misma. A bajo nivel, un router está procesadno paquetes IP cuya dirección destino está en otra red, pero que están encapsulados en tramas

> cuya dirección MAC sí es la del router.

Para habilitar el reenvío de paquetes, basta con editar el fichero **`/etc/sysctl.conf`** descomentando la siguiente línea:

```
...
# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
...
```

...y ejecutando los siguientes comandos:

```
# cat /proc/sys/net/ipv4/ip_forward
0
# sysctl -p
net.ipv4.ip_forward = 1
# cat /proc/sys/net/ipv4/ip_forward
1
```

En cualquier momento podemos consultar el estado del reenvío de paquetes a través del comando:

```
# cat /proc/sys/net/ipv4/ip_forward
1                                    (1)
```

(1) Si devuelve "1" el reenvío está activo, "0" si está inactivo.

Los comandos anteriores activan el reenvío de forma permanente, si queremos que el cambio sea solo temporal hasta que la máquina se reinicie, el comando a ejecutar es:

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

Redirect

La función de *redirect* (redirección) es un caso especial de DNAT que redirige los paquetes a la máquina local dependiendo del hook de la cadena.

Por ejemplo, podemos redirigir el tráfico que llega a una máquina al puoerto 2222 a otro puerto como el 22. Para ello ejecutamos los siguientes comandos:

```
# nft add table nat
# nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
# nft add rule nat prerouting tcp dport 2222 redirect to 22
```

Para las redirecciones, no es necesario que el renvío de puertos esté activo, ya que se trata de conexiones a la propia máquina.

13. Sets

El framework Nftables soporta *sets* (conjuntos) de forma nativa. Se pueden utilizar *sets* cuando una regla debe coincidir con múltiples direcciones IP, varios puertos, varias interfaces o cualquier otro criterios de coincidencia múiltiple.

13.1. Anonymous Sets

Un *anonymous set* (conjunto anónimo), contiene valores separados por comas encerrados en llaves, como por ejemplo **`{ 22, 80, 443 }`**. Dichos *sets* se pueden usar directamente en una regla.

Se puede usar estos conjuntos para cualquier otro criterio de coincidencia como direcciones IP. La principal desventaja de los conjuntos anónimos es que, si tenemos que cambiar el conjunto, tenemos que reemplazar la regla.

Para una solución dinámica, hay que utilizar *named sets* (conjuntos con nombre), tal como se describe en la próxima sección.

Supongamos que queremos aceptar todo el tráfico de entrada en los puertos 22, 80 y 443, las reglas serían las siguientes:

```
# nft flush ruleset
# nft add table ip filter
# nft 'add chain ip filter input { type filter hook input priority 0 ; policy
drop; }'
# nft 'add chain ip filter output { type filter hook output priority 0 ; policy
accept; }'
# nft add rule filter input tcp dport { 22, 80, 443 } accept      (1)
# nft add rule filter input ct state established,related accept
```

(1) Con una sola regla estamos abriendo el tráfico de entrada a los puertos 22, 80 y 443.

Mostramos todas las reglas:

```
# nft list ruleset
table ip filter {
    chain input {
        type filter hook input priority filter; policy drop;
        tcp dport { 22, 80, 443 } accept
        ct state established,related accept
    }

    chain output {
        type filter hook output priority filter; policy accept;
```

```
    }
}
```

13.2. Named sets

El framework nftables soporta *named sets*, estos conjuntos con nombre son conjuntos mutables. Un *named set* es una lista o rango de elementos que se pueden usar en múltiples reglas dentro de una tabla. Otra vanteja de estos conjuntos frente a los anónimos es que podemos actualizar un *named set* sin reemplazar ni modificar las reglas que usan el conjunto.

Al crear un conjunto con nombre es necesario especificar el tipo de elementos que el conjunto contiene. Los tipos soportados son los siguientes:

- **ipv4_addr**:
  - Conjuntos que contienen direcciones IPv4 o rangos, como `192.0.2.1` o `192.0.2.0/24`.
- **ipv6_addr**:
  - Conjuntos que contienen direcciones IPv6 o rangos, como `2001:db8:1::1` o `2001:db8:1::1/64`.
- **ether_addr**:
  - Conjuntos que contienen una lista de direcciones de control de acceso al medio (MAC), como `52:54:00:6b:66:42`.
- **inet_proto**:
  - Conjuntos que contienen una lista de tipos de protocolos de internet, como `tcp`.
- **inet_service**:
  - Conjuntos que contienen una lista de servicios de internet, como `ssh`.
- **mark**:
  - Conjuntos que contienen una lista de marcas de paquetes. Las marcas de paquetes pueden ser cualquier valor entero positivo de 32 bits (0 a 2147483647).

Ejemplo, podemos crear una conjunto de este tipo siguiendo estos pasos:

1. Creamos un conjunto vacío, los siguientes ejemplos crean un conjunto para direcciones IPv4:
   - Para crear un conjunto que pueda almacenar múltiples direcciones IPv4 individuales, el formato del comando es:

```
# nft add set inet example_table example_set { type ipv4_addr \; }
```

- Para crear un conjunto que pueda almacenar rangos de direcciones IPv4 para las

anteriores reglas ejecutamos el siguiente comando, hay que tener en cuenta que el conjunto se crea asociado a una tabla:

```
# nft add set ip filter my_set { type ipv4_addr \; flags interval \; }
```

1. Optativamentem, creamos una regla que use el conjunto. Por ejemplo, el siguiente comando agrega una regla a la cadena **`input`** de la tabla **`filter`** que descartará todos los paquetes de direcciones IPv4 que pertenezcan al conjunto **`my_set`**:

```
# nft add rule ip filter input ip saddr @my_set drop      (1)
```

(1) La regla aún no tiene efecto ya que el conjunto está vacío.

2. Agregamos direciones IPv4 al conjunto **`my_set`**:
    - Si hemos creado un conjunto para direcciones individuales el comando es:

```
# nft add element ip filter my_set { 192.168.122.10, 192.168.122.11 }
```

    - Si hemos creado un conjunto para rangos de direcciones, podemos añadir una red completa con cualquiera de los siguientes comandos:

```
# nft add element ip filter my_set { 192.168.122.0-192.168.122.255 }
# Rango de direcciones
# nft add element ip filter my_set { 192.168.122.0/24 }
# Notación CIDR
```

13.3. Creando Sets

Podemos crear listas de direcciones IP para permitir/denegar usando las herramientas y bases de datos que proporcionan webs como:

- Country IP blocks.
    - https://www.countryipblocks.net/acl.php
- Proofpoint Emerginig Threats Rules.
    - https://rules.emergingthreats.net/fwrules/
    - https://rules.emergingthreats.net/fwrules/emerging-Block-IPs.txt
- FireHOL IP Lists.
    - https://iplists.firehol.org/
    - https://iplists.firehol.org/files/firehol_level1.netset
- My IP Address, black list IP Check.
    - https://myip.ms/

    - https://myip.ms/files/blacklist/csf/latest_blacklist.txt
- AbuseIPDB.
    - https://www.abuseipdb.com/
- I-blockList.
    - https://www.iblocklist.com/lists
- Export all attacker-IPs from the last 48 hours.
    - https://www.blocklist.de/en/export.html
- IPdeny IP country CIDR blocks.
    - https://www.ipdeny.com/ipblocks/
- RADb WHOIS Service.
    - https://www.radb.net/support/developer/whois.html
    - Servidor WHOIS en: whois.radb.net

A partir de estas fuentes podemos crear *sets* en nftables para modelar el tráfico de nuestro cortafuegos, tal como se muestra en las siguientes secciones.

13.4. Ejemplo: permitiendo tráfico Google Meet

Supongamos que tenemos un router/cortafuegos y queremos autorizar el tráfico de Google Meet desde la red interna:

- Web, TCP, puertos 80 y 443.
- DNS, UDP y TCP, puerto 53. Solo los DNS de Google y de OpenDNS:
    - DNS Google: 8.8.8.8 y 8.8.4.4.
    - OpenDNS: 208.67.222.123 y 208.67.220.123.
- Google Meet, según la documentación oficial hay que abrir los siguientes puertos:
    - https://support.google.com/a/answer/1279090?hl=es
    - Salida: puertos UDP 443, 3478, 19302-19309. TCP 443.

El problema con Google Meet es que se especifican URL y dominios, y eso siempre supone un problema a la hora de configurar un cortafuegos. Como solución se podrían abrir esos puertos a toda Internet, pero eso supone un problema en cuanto a la seguridad. Una solución de compromiso podría ser abrir esos puertos solo hacia la red de Google. Pero, ¿cuál es la red de Google? Google no tiene solamente una red o un conjunto de redes, Google tiene un AS completo, concretamente el AS15169.

La solución entonces es sencilla:

1. Enumeramos todas las redes que conforman el AS de Google.
2. Construimos un conjunto con todas las redes de Google.
3. Modificamos nuestras reglas para permitir el tráfico TCP/UDP solo a la red de Google, enumeradas en el *set* GOOGLE_NET.

Para ello necesitamos un par de scrips, uno que dado un AS nos enumere las redes y otro que las incluya en nuestro cortafuegos.

El script que dado un AS nos enumera sus redes sería:

Listado 1. get_AS.sh

```
#!/bin/bash

WHOIS_SERVER=whois.radb.net


if [ $# -ne 1 ]
then
    echo "Uso: `basename $0` AS" >&2
    echo -e "\tSpecify the AS number in the form ASX, where X is the AS number.
E.g.: AS15169 (Google)"
    exit 1
else
    ASN=$1
fi

if [[ ! "$ASN" =~ ^AS[0-9]+$ ]]; then
    echo "Specify the AS number in the form ASX, where X is the AS number.
E.g.: AS15169 (Google)"
    exit 1
fi

if ! command -v whois &> /dev/null
then
    echo "Command \"whois\" is not installed, please install it using your
favourite package manager"
    exit 2
fi

networks=$(whois -h $WHOIS_SERVER -- "-i origin $ASN" | grep ^route | cut -d':'
-f 2- | sed -e 's/ *//')

for net in $networks
do
    # We're only interested in IPv4 addresses
    if [[ $net =~ ^([0-9]{1,3}\.){3}[0-9]{1,3}/([0-9]|[12][0-9]|3[0-2])$ ]]
    then
        echo $net
    fi
done
```

El script que añade todas las redes de un conjunto a otro sería este:

Listado 2. load_set.sh

```
#!/usr/bin/env bash

# Check two paramenters, an string and a file
```

```
if [ $# -ne 2 ]; then
    echo "Usage: $0 IPSet_name acl_file"
    exit 1
fi

# Check if the file exists
if [ ! -f $2 ]; then
    echo "File $2 does not exist"
    exit 1
fi

# Check if the file is readable
if [ ! -r $2 ]; then
    echo "Can't read $2 file"
    exit 1
fi

#echo "Creating Nftables Set '$1'"
#nft add set ip filter $1 { type ipv4_addr \; flags interval \; }

echo "Adding network addresses from $2"
# Process the file line by line

while read line; do
  nft add element ip filter $1 { $line }
done < $2

echo "Done!!!!"
```

Para configurar nuestro cortafuegos seguimos los siguientes pasos:

1. Creamos un directorio de trabajo, **/root/nftables/** en el que copiamos ambos scripts.
2. Partimos de un cortafuegos como este, descrito en **/root/nftables/nftables.conf**:

```
#!/usr/sbin/nft -f

define WAN_IF = enp1s0
define WAN_IP = 192.168.122.254
define WAN_NET = 192.168.122.0/24

define LAN_IF = enp10s0
define LAN_IP = 10.0.100.254
define LAN_NET = 10.0.100.0/24

flush ruleset

table ip6 filter {
    chain INPUT {
        type filter hook input priority filter; policy drop;
    }
```

```
    chain OUTPUT {
        type filter hook output priority filter; policy drop;
    }

    chain FORWARD {
        type filter hook forward priority filter; policy drop;
    }
}

table ip filter {

    chain input_LAN {
        ct state new tcp dport ssh   counter accept comment "Accept SSH
(port 22)"
    }

    chain input {
        type filter hook input priority filter; policy drop;

        iif lo                               accept comment "Accept any
localhost traffic"
        ct state invalid                     drop   comment "Drop invalid
connections"
        ct state established,related         accept comment "Accept
traffic originated from us"

        meta l4proto icmp            counter accept comment "Accept ICMP"
        ip protocol igmp             counter accept comment "Accept IGMP"

        ip saddr { $LAN_NET, $WAN_NET }      jump input_LAN comment
"Connections from private IP address ranges"
    }

    chain forward {
        # Drop everything, we DON'T forward, we are NOT a router
        type filter hook forward priority filter; policy drop;

        ct state invalid             drop   comment "Drop invalid
connections"
        ct state established,related accept comment "Accept traffic
originated from us"

        iif $LAN_IF oif $WAN_IF ip saddr $LAN_NET ct state new tcp dport {
http, https } counter accept comment "Accept HTTP (ports 80, 443)"
        iif $LAN_IF oif $WAN_IF ip saddr $LAN_NET ct state new udp dport {
53 }          counter accept comment "Accept UDP DNS"
        iif $LAN_IF oif $WAN_IF ip saddr $LAN_NET ct state new tcp dport {
53 }          counter accept comment "Accept TCP DNS"
    }

    chain output {
        # Accept every outbound connection
        type filter hook output priority filter; policy accept;
    }
```

```
}

table nat {
    chain masquerading {
        type nat hook postrouting priority srcnat;
        oifname $WAN_IF masquerade;
    }
}
```

3. Creamos un fichero Nftables para crear el conjunto den nuestra tabla principal **`filter`**, **`sets.nft`**:

```
#!/usr/sbin/nft -f

table ip filter {
    set GOOGLE_NET {
        type ipv4_addr
        flags interval
    }
}
```

4. Modificamos el fichero de Nftables principal para que incluye el fichero anterior, añadimos las reglas que nos permiten el tráfico Google Meet:

```
#!/usr/sbin/nft -f

flush ruleset

include "/root/nftables/sets.nft"
...
    chain forward {
        ...
        iif $LAN_IF oif $WAN_IF ip saddr $LAN_NET ip daddr $GOOGLE_NET ct
state new tcp dport { 443 } counter accept comment "Google Meet TCP"
        iif $LAN_IF oif $WAN_IF ip saddr $LAN_NET ip daddr $GOOGLE_NET ct
state new udp dport { 443, 3478, 19302-19309 } counter accept comment
"Google Meet UDP"
    }
...
```

5. Cargamos las nuevas reglas:

```
# nft -f nftables.conf
```

6. Ejecutamos el siguiente script para obtener las redes de AS de Google y añadirlas al conjunto:

```
# bash get_AS.sh AS15169 > google.acl
```

```
# bash load_set.sh GOOGLE_NET google.acl
```

7. Volcamos la configuración de nuestro cortafuegos actual al fichero principal:

```
# nft --stateless list ruleset > /etc/nftables.conf
```

¡¡¡¡¡¡¡¡¡¡¡¡¡ POR AQUÍ LA REVISIÓN !!!!!!!!!!!!!!! Hay ya un cortafuegos principal en CYBER.FW, a falta de incluir tráfico de paises

13.5. GeoIP

GeoIP es una tecnología que permite determinar la ubicación geográfica de un usuario en Internet a partir de su dirección IP. Los métodos que usan los servicios de posicionamiento por IP utilizan bases de datos que contienen direcciones IP asociadas con ciudades, países, coordenadas geográficas, códigos postales, y otros datos relacionados. Los datos de GeoIP se utilizan para una variedad de aplicaciones, incluyendo:

- Personalización de contenido.
- Control de acceso y seguridad.
- Análisis y segmentación de mercado.
- Cumplimiento normativo.
- Enrutamiento y rendimiento de red.

En nuestro caso podemos utilizarlo para prohibir el acceso desde determinados países a nuestra red. Siempre podemos obtener y actualizar la lista de direcciones de red asociadas a países de forma manual, pero desde la documentación oficial de Nftables se nos invita a que clonemos un proyecto que implemanta de forma fácil los maps por países para Nftables.

Para generar los mapeos de IP/país seguimos los siguientes pasos:

1. Comprobamos que tenemos instalado Python 3.9 e instalamos las librerías necesarias:

```
# apt install python3-requests python-is-python3
```

2. Clonamos el proyecto de GitGub Nftables-GeoIP

```
$ git clone https://github.com/pvxe/nftables-geoip.git
```

3. Para generar los mapeos de las direcciones IPv4 e IPv6, descargamos los datos geográficos desde **db-ip.com** guardando la salida en la carpeta actual del proyecto de GitHub:

```
$ cd nftables-geoip
```

```
$ ./nft_geoip.py --file-location location.csv --download
```

4. Tras la ejecución comprobamos que se nos han descargado los siguientes ficheros:

```
-rw-r--r-- 1 root root 25905336 abr 23 14:32 dbip.csv
-rw-r--r-- 1 root root       15 abr 23 14:32 geoip-def-antarctica.nft
-rw-r--r-- 1 root root      461 abr 23 14:32 geoip-def-oceania.nft
-rw-r--r-- 1 root root      810 abr 23 14:32 geoip-def-europe.nft
-rw-r--r-- 1 root root      808 abr 23 14:32 geoip-def-asia.nft
-rw-r--r-- 1 root root      902 abr 23 14:32 geoip-def-americas.nft
-rw-r--r-- 1 root root     8419 abr 23 14:32 geoip-def-all.nft
-rw-r--r-- 1 root root      956 abr 23 14:32 geoip-def-africa.nft
-rw-r--r-- 1 root root 12081109 abr 23 14:32 geoip-ipv4.nft
-rw-r--r-- 1 root root 17449593 abr 23 14:32 geoip-ipv6.nft
```

   En cuanto a los ficheros descargados tenemos:

   - **`geoip-def-all.nft`** define variables para cada uno de los países que aparecen en el fichero en location.csv usando su nombre ISO (ISO 3166-1 alfa2) de 2 caracteres. Para cada uno delos países define su valor numérico ISO. Por ejemplo, para Canadá: **`define $CA = 124`**.
   - **`geoip-ipv4.nft`** define el mapa GeoIP4 (**`@geoip4`**).
   - **`geoip-ipv6.nft`** define el mapa GeoIP6 (**`@geoip6`**).

5. Por defecto se crean los *maps* para todos los países, se puede filtrar por países si en la ejecución del script añadimos el parámetro **`-c/--country-filter`**. Por ejemplo:

```
$ ./nft_geoip.py --download -c es,fr,pt          (1)
```

   (1) Ver siguiente apartado para una lista de países probablemente problemáticos

6. En el caso de que queramos "contar" todos los paquetes del tráfico de entrada de un determinado país como España, añadiríamos las siguientes reglas a Nftables:

```
#!/usr/sbin/nft -f

table inet geoip {
    include "geoip-def-all.nft"
    include "geoip-ipv4.nft"
    include "geoip-ipv6.nft"

    chain geoip-mark-input {
        type filter hook input priority -1; policy accept;

        meta mark set ip saddr map @geoip4
        meta mark set ip6 saddr map @geoip6
    }

    chain input {
```

```
        type filter hook input priority 0; policy accept;

        meta mark $ES counter
    }
}
```

Se puede susitituir el fichero **`geoip-def-all.nft`** por otro fichero más concreto como **`geoip-def-europe.nft`**, o incluso copia la línea **`DEFINE ES = ` al fichero `nft`** principal.

Se necesita alrededor de unos 300 MB de memoria para ejecutar el script y cargar los mapas completos de direcciones Ipv4 e IPv6. Lo normal es que queramos filtrar el tráfico de determinados países… Pero, ¿por qué querríamos hacer eso?

13.5.1. Cortando el tráfico origen de ciertos países

Desde la perspectiva de cortafuegos y de ciberseguridad (cyberwarfare), especialmente en contextos de seguridad nacional o en entornos de alta seguridad, puede haber consideraciones específicas para bloquear o restringir tráfico de ciertos países. Sin embargo, es importante señalar que tales decisiones deben estar basadas en una evaluación detallada de las amenazas, la legislación y la ética, y no deben ser vistas como medidas generalizadas o discriminatorias hacia países o regiones específicas.

Entre las razones para considerar restricciones geográficas tenemos:

1. **Alto volumen de ataques cibernéticos**:
    - Algunos países son conocidos por tener un alto número de entidades (ya sean estatales o no estatales) que participan en actividades maliciosas en línea como ataques DDoS, phishing, y otras formas de ciberataques. Si una organización experimenta ataques reiterados que pueden ser geográficamente rastreados a ciertas regiones, podemos optar por bloquear o restringir el acceso para mitigar estos riesgos.
2. **Cumplimiento normativo y sanciones**:
    - En algunos casos, las organizaciones están sujetas a cumplir con sanciones internacionales o regulaciones que prohíben la interacción con ciertos países. En estos casos, el bloqueo de tráfico de ciertas regiones puede ser legalmente necesario.
3. **Protección de información sensible**:
    - En entornos donde se maneja información especialmente sensible, como datos gubernamentales o de defensa, puede ser prudente limitar el acceso a usuarios de regiones consideradas de alto riesgo para proteger contra el espionaje o la intrusión.

Hay que tener además en cuenta ciertas consideraciones:

- **Efectividad**: bloquear el tráfico por región puede no ser siempre efectivo, especialmente porque los usuarios malintencionados pueden utilizar VPN, servidores

proxy o botnets para ocultar su ubicación real.

- **Impacto en usuarios legítimos**: las restricciones geográficas también pueden afectar a usuarios legítimos en las regiones bloqueadas. Esto puede tener implicaciones negativas para la reputación de la organización y para individuos que no están involucrados en actividades maliciosas.
- **Mantenimiento y precisión de las listas de bloqueo**: las direcciones IP pueden cambiar y los datos de geolocalización IP deben mantenerse actualizados para asegurar que las restricciones sean precisas y efectivas.

En el contexto de ciberseguridad nacional, algunos gobiernos han identificado países específicos como "actores de amenazas persistentes avanzadas" (APT, por sus siglas en inglés). Estos países, que pueden incluir naciones conocidas por su ciberespionaje o ciberguerra, a menudo son objeto de mayor escrutinio y medidas preventivas más estrictas. Entre países ejemplo para considerar cortar el tráfico tendríamos:

- **Rusia (RU)**: ampliamente conocida por sus capacidades avanzadas en ciberoperaciones, Rusia ha sido implicada en numerosos incidentes de ciberespionaje y ataques disruptivos.
- **China (CN)**: también ha sido señalada por su extensa red de operaciones de ciberespionaje dirigidas tanto a objetivos gubernamentales como corporativos para obtener ventajas económicas y estratégicas.
- **Irán (IR)**: en años recientes, Irán ha mostrado una creciente capacidad en ciberataques, frecuentemente dirigidos contra infraestructura crítica y entidades gubernamentales, especialmente de países considerados adversarios.
- **Corea del Norte (KP)**: este país ha sido asociado con varios ataques cibernéticos de alto perfil, muchos de los cuales parecen estar motivados por la necesidad de obtener fondos y evadir sanciones internacionales.
- **Estados Unidos (US)**: aunque Estados Unidos es líder en tecnología y ciberdefensa, también lleva a cabo operaciones de ciberespionaje, según se ha revelado en diversas filtraciones de documentos. Si que es ciuerto que muchos de los ataques provienen de servidores AWS, por lo que en vez de limitar Estados Unidos completamehte, merece la pensa de cortar el tráfico proveniente de AWS.
- **India (IN)**, en el contexto de ciberseguridad, es un actor complejo. Si bien no se menciona con tanta frecuencia como el resto en términos de amenazas cibernéticas globales, India tiene capacidades significativas en tecnología y ciberseguridad.

Si queremos configurar Nftables para "cortar" el tráfico de entrada de determinados países, los pasos a seguir serían los siguientes:

1. Determinar los países, un buen punto de partida sería hacerlo para los siguientes: Rusia, China, Irán, Corea del Norte e India.
2. Usamos GeoIP para generar la lista de direcciones, pero solo generamos los países de la lista. Usamos el parámetro **`-c/--country`** para generar los ficheros **`geoip-ipv{4,6}-interesting.nft`**:

```
# ./nft_geoip.py --download -c ru,cn,ir,kp,in
```

3. Creamos el fichero **geoip.nft** con el siguiente contenido, las reglas se pueden añadir tanto al *hook* **input** como al *hook* **forward**, pero si queremos aumentar el rendimiento y/o queremos aplicarlas tanto al tráfico que va hacia el cortafuegos, como al tráfico que va hacia las redes internas, podemos hacerlo en el *hook* **ingress**:

```
#!/usr/sbin/nft -f

define WAN_IF = enp1s0

table inet geoip {
    include "./geoip-def-all.nft"
    include "./geoip-ipv4-interesting.nft"
    #include "./geoip-ipv6-interesting.nft"

    chain geoip-mark-ingress {
        type filter hook ingress device $WAN_IF priority filter -1; policy
accept;

        meta mark set ip saddr map @geoip4
        #meta mark set ip6 saddr map @geoip6
    }

    chain ingress {
        type filter hook ingress device $WAN_IF priority filter; policy accept;

        meta mark $RU counter drop comment "Drop Russia"
        meta mark $CN counter drop comment "Drop China"
        meta mark $IR counter drop comment "Drop Iran"
        meta mark $KP counter drop comment "Drop North Korea"
        meta mark $IN counter drop comment "Drop India"
    }
}
```

Esta configuración se integra perfectamente con cualquier otra configuración de Nftables que tengamos, si queremos ver la configuración de GeoIP, tan siolo tenemos que ejecutar este comando:

```
# nft list table inet geoip
```

14. Operaciones en el conjunto de reglas

Nftables soporta ciertas operaciones aplicables al conjunto de reglas completo. Podemos mostrar todas las reglas con el siguiente comando:

```
# nft list ruleset
```

De la misma forma se puede mostrar la lista de reglas por familia:

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

Para borrar todas las reglas ejecutamos:

```
# nft flush ruleset
```

De la misma forma que antes, también podemos hacerlo por familia:

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

14.1. Backup/restore, operaciones atómicas

Podemos hacer una copia de todas nuestras reglas así:

```
# echo "flush ruleset" > backup.nft
# nft list ruleset >> backup.nft       (1)
```

(1) Podemos mostrar las reglas sin los contadores de paques y bytes añadiendo la opción **`--stateless`**.

El fichero resultante es una fichero de texto, completamente editable. Una vez tengamos la copia, podemos restaurar el estado de nuestro cortafuegos así:

```
# nft -f backup.nft
```

De forma análoga podemos volcar todas nuestra reglas en formato JSON:

```
# nft --json list ruleset > ruleset.json
```

O en pantalla con un formato más visible a través del comando **`jq`**:

```
# nft --json list ruleset | jq
```

14.2. Sustitución de reglas atómico

Con el uso de **iptables**, era muy habitual configurar un firewall mediante un script de Bash que incluía múltiples comandos de **iptables**. Este método tenía una desventaja significativa: no era atómico. Esto significaba que mientras se ejecutaba el script, durante solo unas fracciones de segundo, el firewall quedaba en un estado de configuración incompleto.

Nftables mejora esta situación con la introducción de una opción para reemplazar reglas de manera atómica, usando el parámetro -f. A diferencia de los scripts de Bash, nftables procesa todos los archivos de configuración a la vez, crea un objeto de configuración en memoria que combina la configuración nueva con la existente y, posteriormente, realiza un intercambio atómico de la configuración vieja por la nueva. Este proceso asegura que en ningún momento el firewall se encuentre parcialmente configurado.

Para hacerlo basta ejecutar este comando y proporcionar un fichero con las reglas

```
# nft -f myfile
```

Podemos almacenar todas las reglas en un ficheo como hemos visto anteriormente, o solo las reglas de una tabla o de una cadena:

```
# nft list ruleset              > all_rules.nft
# nft list table filter         > filter_table.nft
# nft list chain filter output > output_rules.nft
```

Cosas a tener en cuenta:

- **Reglas Duplicadas**:
    - Si se antepone la línea **flush table filter** o **fush ruleset** al principio del archivo con las reglas, se producirá un reemplazo de conjunto de reglas atómico equivalente a lo que proporcionaría un **iptables-restore**. El kernel maneja los comandos de reglas en el archivo en una sola transacción, así que básicamente el vaciado y la carga de las nuevas reglas ocurren en un solo paso. Si se opta por no vaciar las tablas, entonces aparecerán reglas duplicadas cada vez que se recargue la configuración.
- **Vaciado de Conjuntos**:
    - El comando **flush table filter** no vaciará ningún *set* definido en esa tabla. Para vaciar por completo las reglas hay que utilizar **flush ruleset** o elimina los *sets* de forma explícita.
- **¿Qué sucede al incluir 2 archivos teniendo ambos una declaración para la tabla de filtro?**:
    - Si se tienen dos archivos con declaraciones para la tabla de filtro, pero uno añade una regla que permite el tráfico desde 192.168.1.1 y el otro permite el tráfico desde 192.168.1.2, entonces ambas reglas serán incluidas en la cadena,

incluso si uno o ambos archivos contienen una declaración de vaciado.

- **¿Qué pasa con las declaraciones de vaciado en uno, otro o en ninguno de los archivos?**:
  - Si hay comandos de vaciado en cualquier archivo incluido, estos se ejecutarán en el momento en que se realice el cambio de configuración, no en el momento en que se cargue el archivo. Si no se incluye una declaración de vaciado en ningún archivo incluido, aparecerán reglas duplicadas. Si se incluyes una declaración de vaciado, no aparecerán reglas duplicadas y la configuración de los dos ficheros será incluida.

15. Monitorización de reglas

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

Nftables esa capaz de mostrar notificaciones cuando se produzcan actualizaciones en las reglas, para ello abrimos un terminal y ejecutamos el comando:

```
# nft monitor
```

El comando anterior se queda a la espera ya que **`nft`** queda suscrito a cualquier tipo de actualización del conjunto de reglas. Se pueden filtrar eventos por tipo de:

- Objetos: tablas, cadenas, reglas, conjuntos y elementos.
- Eventos: creación y eliminación de reglas.

El formato de salida puede ser:

- Texto plano (es decir, formato nativo de nft).
- XML.
- JSON (opción **`-j`**).

El siguiente ejemplo muestra cómo seguir el rastro de las actualizaciones de reglas solamente:

1. Desde una terminal lanzamos el comando:

   ```
   # nft monitor
   ```

2. Desde otra terminal ejecutamos los siguientes comandos:

   ```
   # nft add table inet monitor_this
   # nft add chain inet monitor_this forward
   # nft add rule inet monitor_this forward counter accept
   # nft flush chain inet monitor_this forward
   # nft delete chain inet monitor_this forward
   ```

```
# nft delete table inet monitor_this
```

En la terminal en la que lanzamos la monitorización, nos irán apareciendo los siguientes mensajes:

```
# nft monitor
add table inet monitor_this
# new generation 22 by process 1942 (nft)
add chain inet monitor_this forward
# new generation 23 by process 1955 (nft)
add rule inet monitor_this forward counter packets 0 bytes 0 accept
# new generation 24 by process 1971 (nft)
delete rule inet monitor_this forward handle 2
# new generation 25 by process 1981 (nft)
delete chain inet monitor_this forward
# new generation 26 by process 1997 (nft)
delete table inet monitor_this
# new generation 27 by process 2007 (nft)
```

16. Debugging Rules

Desde las versiones 4.6 del kernel de Linux, Nftables ofrece soporte para la depuración/trazado del conjunto de reglas. Esto es equivalente al antiguo método de iptables **`-J TRACE`**, pero con algunas mejoras significativas.

Los pasos para habilitar a depuración/trazado son los siguientes:

1. Añadimos el soporte de trazao en el conjunto de reglas, para ello establecemos **`nftrace`** en cualquier regla den nuestro conjunto.
2. Monitorizamos los eventos generados desde el comando **`nft`**.

Ejemplo:

1. Partimos de un cortafuegos ya configurado.
2. Creamos una cadena **`trace_chain`** dentro de nuestra tabla **`filter`**:

```
# nft add chain ip filter trace_chain { type filter hook prerouting
priority -1\; }
```

3. Creamos una regla **`nftrace`** dentro de la cadena que acabamos de definir:

```
# nft add rule ip filter trace_chain meta nftrace set 1
```

4. Ejecutamos el comando:

```
# nft monitor trace
```

5. Mientras abrimos otra consola y ejecutamos el comando **`ping -c 8.8.8.8`**.
6. Detenemos la monitorización de Nftables pulsando **`Ctrl + C`** y observamos los eventos que ha generado la ejecución del comando **`ping`**:

```
trace id 337116dd ip filter trace_chain packet: iif "enp1s0" ether saddr
52:54:00:c5:e5:dc ether daddr 52:54:00:76:3f:9e ip saddr 8.8.8.8 ip daddr
192.168.122.254 ip dscp cs0 ip ecn not-ect ip ttl 115 ip id 0 ip length 84
icmp type echo-reply icmp code net-unreachable icmp id 2 icmp sequence 1
@th,64,96 0xfe3566600000000db3b0500
trace id 337116dd ip filter trace_chain rule meta nftrace set 1 (verdict
continue)                  (1)
trace id 337116dd ip filter trace_chain verdict continue
trace id 337116dd ip filter trace_chain policy accept
trace id 337116dd ip filter input packet: iif "enp1s0" ether saddr
52:54:00:c5:e5:dc ether daddr 52:54:00:76:3f:9e ip saddr 8.8.8.8 ip daddr
192.168.122.254 ip dscp cs0 ip ecn not-ect ip ttl 115 ip id 0 ip length 84
icmp type echo-reply icmp code net-unreachable icmp id 2 icmp sequence 1
@th,64,96 0xfe3566600000000db3b0500
trace id 337116dd ip filter input rule ct state established,related accept
comment "Accept traffic originated from us" (verdict accept)
```

(1) Es posible que en los eventos del comando **`ping`** aparezcan otros eventos, en ese caso podemos filtrar por el idenificador. En nuestro caso **`337116dd`**.

El comando **`nft monitor trace`** mostrará los eventos en tiempo real, permitiéndonos observar cómo los paquetes son procesados por el conjunto de reglas. Esto es muy útil para depurar problemas y comprender mejor cómo se aplican las reglas a los paquetes.

16.1. Depurando el tráfico no controlado

Cómo ya hemos comentado, al configurar un cortafuegos tenemos que decidir cómo queremos implementarlo, podemos hacerlo a través de una política por defecto ACEPTAR, o con una política por defecto DENEGAR.

Desde el punto de vista de la seguridad es preferible aplicar una política por defecto DENEGAR, y a partir de ahí, empezar a autorizar el tráfico que permitimos en nuestra red.

A pesar de ser una forma mucho más segura de configurar un cortafuegos, plantea ciertos retos:

- Más difícil de cosntruir las reglas.
- Más difícil depurar reglas.
- Aparecerán problemas con los usuarios, algunas aplicaciones dejarán de funcionar y llevarñá tiempo ajustar nuestro cortafuegos al tŕafico que tenemos.

Una forma de depurar esto es añadir una regla **`LOG`** al final de la cadena a depurar, para que

se loguee todo el tráfico que estamos cortando. Esto presenta varias ventajas:

- Auditoría y monitorización:
  - Permite registrar información sobre los paquetes que están siendo descartados por la política por defecto **DROP**. Esto es útil para entender qué tipo de tráfico de nuestra red está siendo bloqueado.
  - En caso de un incidente de seguridad, los registros pueden ser utilizados para realizar un análisis forense y entender mejor la naturaleza del ataque o del tráfico no deseado.
- Depuración y diagnóstico:
  - Ayuda a diagnosticar problemas de red al proporcionar información sobre el tráfico que no está siendo permitido. Si hay un problema de conectividad, los registros pueden mostrar si el tráfico legítimo está siendo bloqueado inadvertidamente.
  - Permite verificar que las reglas del cortafuegos están funcionando como se espera. Si un paquete que debería ser aceptado está siendo bloqueado, los registros pueden ayudar a identificar y corregir las reglas con problemas.
- Seguridad proactiva:
  - Puede ayudar a detectar patrones de tráfico sospechoso o inusual que podrían indicar intentos de intrusión, escaneos de puertos u otros comportamientos maliciosos.
  - Los sistemas de monitorización pueden estar configurados para generar alertas en tiempo real basadas en los registros del cortafuegos, permitiendo una respuesta rápida a posibles amenazas.
- Cumplimiento Normativo:
  - Muchas regulaciones y estándares de seguridad requieren la implementación de medidas de auditoría y registro de eventos. Los registros del cortafuegos pueden ayudar a cumplir con estos requisitos proporcionando evidencia de las medidas de seguridad implementadas.

Para añadir este tipo de reglas podemos seguir los siguientes pasos:

1. Editamos la configyuraxci´ñion de **syslog`para que los *logs* de Nftables vayan a un fichero aparte. Editamos el fichero `/etc/rsyslog.d/50-default.conf** realizando estas modificaciones:

```
#
# First some standard log files.  Log by facility.

:msg, contains, "Unmatched traffic:" /var/log/nftables-unmatched.log
& stop
#
```

1. Añadimos la siguiente regla al final de la cadena que queramos depurar/monitorizar:

```
table ip filter {
    chain input {
        ...

        log prefix "Unmatched traffic INPUT: " level info
    }

    chain forward {
        ...

        log prefix "Unmatched traffic FORWARD: " level info
    }
}
```

1. Tan solo nos queda revisar de forma periódica iy/o puntual los logs que Nftables vaya generando.

Hay que tener en consideración ciertos aspectos importantes de estos logs:

1. **Cantidad de Tráfico**: la generación de logs para todos los paquetes descartados puede resultar en un gran volumen de logs.
2. **Frecuencia de Log**: se recomienda usar límites como `limit rate 10/second` para reducir la cantidad de logs.
3. **Sobrecarga del sistema**: el registro de un alto volumen de tráfico puede consumir recursos significativos del sistema.
4. **I/O de disco**: un gran número de logs puede causar sobrecarga de E/S en disco.
5. **Rotación de logs**: la configuración correcta de la rotación de estos logs es obligatoria.
6. **Relevancia del tráfico**: loguear solo el tráfico relevante, y posiblemente solo de forma temporal y/o puntual.
7. **Prefijos y etiquetas*** es necesario utilizar prefijos claros y etiquetas en los logs para facilitar su identificación.
8. **Acceso a logs**: hay que proteger los ficheros de logs contra accesos y modificaciones no autorizados.

17. Frontends de Netfilter

La decisión de utilizar de utilizar Nftables frente a otras herramientas de configuración de cortafueegos puede ser no tan sencilla como podría parecer a primera vista.

Es una decisión importante para administradores de sistemas y técnicos seguridad ya que puede afectar significativamente a la seguridad, eficiencia y facilidad de gestión de la infraestructura de red.

Mientras que `nftables` ofrece una plataforma moderna y eficiente con capacidades avanzadas de filtrado y manipulación de paquetes, herramientas como `ufw` y `firewalld` proporcionan interfaces más sencillas y amigables para la gestión de políticas de firewall, lo

que puede ser especialmente valioso para aquellas personas menos familiarizadas con la complejidad de la seguridad de red.

Cada una de estas opciones tiene sus propias ventajas y limitaciones, y la elección depende tanto de las necesidades específicas del entorno como del nivel de comodidad y experiencia del usuario con las tecnologías de firewall de Linux.

Como breve guía sobre cuándo utilizar una herramienta u otra tendríamos:

- firewalld o ufw: utilizaremos una herramienta de este tipo para casos de uso simples y fáciles de configurar. El ejemoplo típico es el de un servidor, si instalamos un servidor que proporcione un servicio único como MariaDB o Nginx, es muy sencillo utilizar este tiop de herramientas para corrtar todo el tráfico y permitir solamente paquetes desde una red hacia un puerto en concreto.
- nftables: utilizaremos nftables para configurar firewalls complejos y críticos en términos de rendimiento. Si tenemos que configurar uno de los cortafuegos principales, usaremos **`Nftyables`** por todas las ventajas que ofrece.

En cuanto a **`iptables`** la recomendación actual es dejar de utilizarlo, así como todas las herramientas que sigan dependiendo de él.

Para evitar que los diferentes servicios relacionados con el firewall (firewalld, ufw, nftables o iptables) interfieran entre sí, se recomienda usar/ejecutar solamente uno de ellos, y deshabilitar todos los demás.

17.1. ufw

Uncomplicated Firewall (UFW) es el software que permite gestionar el cortafuegos Netfilter de una forma fácil y sencilla. Utiliza un programa en línea de comandos que consiste en un pequeño número de subcomandos simples, utilizando iptables para la configuración.

UFW está disponible por defecto en todas las instalaciones de Ubuntu desde 8.04 LTS, así como en todas las instalaciones de Debian desde 10.4.

Entre sus características destacan:

- Política predeterminada para tráfico de entrada (permitir/negar).
- Soporte IPv6
- Estado.
- Registro (activado/desactivado).
- Framework extensible.
- Soporte Python.
- Integración de aplicaciones.
- Permite limitar la tase de tráfico IP a través del comando **`limit`**.
- Internacionalización.

- Reglas de entrada para múltiples puertos.
- Debconf/preseeding.
- Política predeterminada de tráfico de entrada (DENY).
- Reglas de entrada DENY.
- Inserción de reglas.
- Niveles de registro (log levels).
- Logging por regla.
- Filtrado de tráfico de salida (al mismo nivel que el de entrada).
- Filtrado por interfaz.
- Autocompletado para Bash.
- Soporte para Systemd.
- Informes mejorados.
- Comando de restablecimiento.
- Soporte para Rsyslog.
- Eliminación por número de regla.
- Informe "Show listening" (mostrar escucha).
- Mayor soporte de protocolos (AH, ESP, IGMP, GRE, ...).
- Informe "Show added" (mostrar añadidos).
- Hooks de extensibilidad antes/después.
- Filtrado de paquetes enrutados (FORWARD).
- Mayor soporte de protocolos (igmp, gre).
- Soporte de Snappy para Ubuntu Core.
- Comentarios por regla.

Los comandos más usuales de **`ufw`** son:

Acción	Comando
Ver el estado del cortafuegos	ufw status ufw status verbose
Habilitar/deshabilitar cortafuegos	ufw enable ufw disable
Resetear el cortafuegos a los valores por defecto	ufw reset
Recargar reglas	ufw reload

Acción	Comando
Políticas por defecto	ufw default deny incoming ufw default allow outgoing
Permitir SSH	ufw allow ssh ufw allow 22/tcp
Borrar la anterior regla de acceso a SSH	ufw delete allow ssh
Borrar regla numerada	ufw status numbered ufw delete [number]
Permitir rangos de puertos	ufw allow 1000:2000/tcp
Permitir conexiones desde una máquina concreta	ufw allow from 192.168.1.2
Permitir desde una red a un puerto en concreto TCP/UDP	ufw allow from 192.168.0.0/24 to any port 22
Solo TCP 22	ufw allow proto tcp from 192.168.0.0/24 to any port 22
Permitir todas las conexiones a MySQL desde una interfaz	ufw allow in on eth1 to any port 3306
Activar logging	ufw logging on

Ejemplo de uso de **`ufw`**:

```
# ufw status
# ufw app list
# ufw allow 'Nginx HTTP'    # También ufw allow 'Apache Full'
# ufw allow OpenSSH
# ufw enable
# ufw status
# ufw status verbose
```

17.2. firewalld

`firewalld` proporciona un cortafuegos gestionado dinámicamente con soporte para zonas de red/cortafuegos para definir el nivel de confianza de las conexiones o interfaces de red. Tiene soporte para configuraciones de cortafuegos IPv4, IPv6 y para bridges Ethernet, además de una separación de opciones de configuración en tiempo de ejecución y permanente. También admite una interfaz para que los servicios o aplicaciones añadan reglas de cortafuegos directamente.

Usado en distribuciones RHEL (RHEL y derivadas, así como en Fedora) sustituyendo al

anterior modelo de cortafuegos con **`system-config-firewall/lokkit`**. El anterior modelo era estático y cada cambio requería un reinicio completo del cortafuegos. Esto incluía también descargar los módulos del núcleo del netfilter del cortafuegos y cargar los módulos necesarios para la nueva configuración. La descarga de los módulos rompía el estado del cortafuegos y todas las conexiones establecidas.

Por otro lado, el demonio del cortafuegos gestiona el cortafuegos de forma dinámica y aplica los cambios sin tener que reiniciar de forma completa todo el cortafuegos. Por lo tanto, no es necesario recargar los módulos del kernel. Pero usar un demonio de cortafuegos requiere que todas las modificaciones del cortafuegos se hagan con ese demonio para asegurarse de que el estado en el demonio y el cortafuegos en el núcleo están sincronizados. El demonio cortafuegos no puede analizar reglas de cortafuegos añadidas por las herramientas de línea de comandos como ip*tables y ebtables.

El demonio proporciona información sobre la configuración actual del cortafuegos activo a través de D-BUS y también acepta cambios a través de D-BUS utilizando métodos de autenticación PolicyKit.

Entre sus características principales destacan:

- **Zonas de firewall**.
    - Firewalld organiza las reglas de firewall en zonas. Cada zona define un conjunto de reglas de filtrado que se aplican a un determinado conjunto de interfaces de red o conexiones. Esto permite una configuración más granular y flexible del firewall según las necesidades específicas de la red.
- **Gestión basada en servicios**.
    - En lugar de configurar reglas de firewall individuales para puertos y protocolos, Firewalld utiliza servicios predefinidos para facilitar la gestión del firewall. Los servicios agrupan las reglas necesarias para permitir el acceso a aplicaciones y servicios específicos.
- **Administración de estados y conexiones**.
    - Firewalld tiene un enfoque orientado a conexiones y estados, lo que significa que puede rastrear y administrar el estado de las conexiones establecidas. Esto permite que los paquetes relacionados con una conexión establecida sean permitidos automáticamente, facilitando la configuración de reglas de firewall.
- **Cambios en tiempo real**.
    - Firewalld permite realizar cambios en tiempo real en las configuraciones del firewall sin interrumpir las conexiones existentes. Esto significa que los cambios en las reglas de firewall se aplican de inmediato sin la necesidad de reiniciar el servicio o desconectar las conexiones activas.
- **Integración con NetworkManager**.
    - Firewalld se integra estrechamente con NetworkManager, el administrador de redes de Linux. Esto permite que las zonas de firewall se cambien automáticamente según la configuración de red actual, como cambiar de una red pública a una red doméstica o de trabajo.

- **Interfaz de línea de comandos y GUI**.
    - Firewalld proporciona una interfaz de línea de comandos (utilizando el comando firewall-cmd) y una interfaz gráfica (llamada firewall-config) para administrar y configurar el firewall. Esto facilita la administración del firewall tanto para usuarios de línea de comandos como para usuarios con preferencia por una interfaz gráfica.

Entre los comandos más interesantes destacan:

Acción	Comando
Iniciar cortafuegos	systemctl start firewalld.service
Ver estado	firewall-cmd --state
Ver listado de zonas	firewall-cmd --get-zones
Ver zona seleccionada	firewall-cmd --get-default-zone
Ver zonas activas	firewall-cmd --get-active-zones
Ver configuración de las zonas o de una zona en concreto zona	firewall-cmd --list-all-zones firewall-cmd --zone=public --list-all
Transferir interfaz a una zona	firewall-cmd --zone=home --change-interface=eth0
Fijar una zona como zona por defecto	firewall-cmd --set-default-zone=public
Mostrar servicios	firewall-cmd --get-services
Añadir un servicio (HTTP)	firewall-cmd --permanent --add-service=http firewall-cmd --zone=public --permanent --add -service=http
Eliminar un servicio (HTTP)	firewall-cmd --permanent --remove-service=http firewall-cmd --zone=public --permanent --remove -service=http
Añadir un puerto	firewall-cmd --zone=public --permanent --add-port 15672/tcp
Eliminar un puerto	firewall-cmd --zone=public --permanent --remove -port 15672/tcp
Mostrar servicios de una zona	firewall-cmd --zone=public --permanent --list -services
Mostrar los puertos de una zona	firewall-cmd --zone=public --permanent --list-ports
Recargar reglas	firewall-cmd --reload

Ejemplo del uso de **firewalld**:

```
# firewall-cmd --state
# firewall-cmd --get-default-zone
# firewall-cmd --zone=public --list-all
# firewall-cmd --zone=public --permanent --remove-service=cockpit
# firewall-cmd --zone=public --permanent --remove-service=dhcpv6-client
# firewall-cmd --reload
# firewall-cmd --zone=public --permanent --add-service=http
# firewall-cmd --zone=public --permanent --add-service=https
# firewall-cmd --reload
# firewall-cmd --zone=public --list-all
```

18. Ejemplos de cortafuegos

Los siguientes scripts de Nftabkles son solo ejemplos para aprender y describen un escenario con requisitos muy específicos. Aunque se puedan utilizar como base para otros scripts más avaznados hay que tebner siempre en cuenta que los scripts de firewall dependen en gran medida de la infraestructura de la red y los requisitos de seguridad.

18.1. Firewall simple para una workstation

Si queremos configurar un cortafuegos de forma rápida en la que se deniegue TODO el tráfico de entrada y permitamos solo el tráfico generado en nuestra propia máquina, el cortafuegos sería este:

```
#!/usr/sbin/nft -f

flush ruleset

table ip filter {
     chain input {
          type filter hook input priority 0; policy drop;

          # accept traffic originated from us
          ct state established,related accept

          # accept any localhost traffic
          iif lo accept
     }
}
```

Activamos el cortafuegos a través del comando **nft -f FILENAME**.

18.2. Firewall para una workstation

El ejemplo anterior es demasiado simple, si queremos configurar una *wokstation* de tal forma que solo se permite tráfico IP4 y que no se permita ningún tráfico de entrada (más allá de ICMP e IGMP), el cortafuegos sería el siguiente:

```
#!/usr/sbin/nft -f

flush ruleset

table ip6 filter {
    chain INPUT {
        type filter hook input priority filter; policy drop;
    }

    chain OUTPUT {
        type filter hook output priority filter; policy drop;
    }

    chain FORWARD {
        type filter hook forward priority filter; policy drop;
    }
}

table ip filter {

    chain input {
        type filter hook input priority filter; policy drop;

        iif lo                      accept comment "Accept any localhost
traffic"
        ct state invalid            drop   comment "Drop invalid connections"
        ct state established,related accept comment "Accept traffic originated
from us"

        meta l4proto icmp           counter accept comment "Accept ICMP"
        ip protocol igmp            counter accept comment "Accept IGMP"
    }

    chain forward {
        # Drop everything, we DON'T forward, we are NOT a router
        type filter hook forward priority filter; policy drop;
    }

    chain output {
        # Accept every outbound connection
        type filter hook output priority filter; policy accept;
    }
}
```

18.3. Firewall para un servidor

Un servidor también debe tener configurado un cortafuegos básico. En la configuración solo se debe permitir tráfico de entrada para los servicios que ofrece y desde la red a la que ofrece los servicios. Por ejemplo, supongamos un servidor que solo ofrece SSH a las redes **`192.168.122.0/24`y `10.0.200.0/24`**, sion permitir tráfico IP6. El cortafuegos sería el siguiente:

```
#!/usr/sbin/nft -f

flush ruleset

table ip6 filter {
    chain INPUT {
        type filter hook input priority filter; policy drop;
    }

    chain OUTPUT {
        type filter hook output priority filter; policy drop;
    }

    chain FORWARD {
        type filter hook forward priority filter; policy drop;
    }
}

table ip filter {

    set LAN {
        type ipv4_addr
        flags interval

        elements = { 192.168.122.0/24, 10.0.200.0/24 }
    }

    chain input_LAN {
        tcp dport ssh  counter  accept         comment "Accept SSH (port 22)"
    }

    chain input {
        type filter hook input priority filter; policy drop;

        iif lo                                 accept comment "Accept any
localhost traffic"
        ct state invalid                       drop   comment "Drop invalid
connections"
        ct state established,related           accept comment "Accept traffic
originated from us"

        meta l4proto icmp            counter accept comment "Accept ICMP"
        ip protocol igmp             counter accept comment "Accept IGMP"

        ct state new ip saddr @LAN             jump input_LAN comment
```

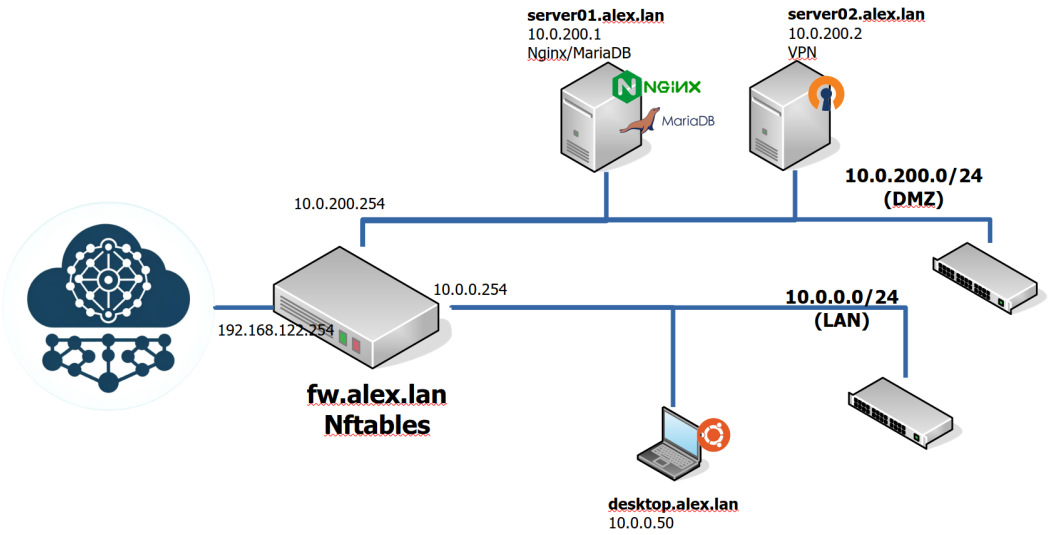
```
"Connections from private IP address ranges"
    }

    chain forward {
        # Drop everything, we DON'T forward, we are NOT a router
        type filter hook forward priority filter; policy drop;
    }

    chain output {
        # Accept every outbound connection
        type filter hook output priority filter; policy accept;
    }
}
```

18.4. Router/firewall con una red LAN y una DMZ

Tenemos la siguiente infraestructura:



Con las siguientes restricciones:

- El router está conectado a las siguientes redes:
  - Internet a través de la interfaz **`enp1s0`**
  - LAN interna a través de la interfaz **`enp7s0`**
  - DMZ conectada a través de la interfaz a través de **`enp8s0`**.
- La política por defecto será DENEGAR.
- El cortafuegos hará MASQUERADING de ambas redes hacia Internet.
- Desde Internet a la DMZ:
  - Las peticiones que reciba el firewall a HTTP/HTTPS se redirigirán a la máquina server01.
  - Las peticiones que reciba el firewall al puerto 3394 (OpenVPN) se redirigirán a la máquina server02.

    - Las peticiones que reciba el firewall al puerto 2222 (SSH) se redirigirán a la máquina server02 al puerto 22.
- Las máquinas de la red DMZ pueden:
    - Navegar.
    - Se permita el tráfico DNS a los servidores de Google y de OpenDNS.
- Las máquinas de la red LAN pueden:
    - Navegar.
    - Visitar la web que proporciona el servidor server01.
    - Se permita el tráfico DNS a los servidores 8.8.8.8 y 8.8.4.4.
    - La máquina 10.0.0.50 puede accerder por SSH a todas las máquinas de la red DMZ.
- En cuanto al propio firewall:
    - Solo aceptará peticiones SSH, en el puerto 1313 desde la red externa y al 22 desde la red LAN. El servicio SOLAMENTE escuchará en el puerto 22.
    - El firewall no tiene restricciones en cuanto a navegación (puertos 80, 443 y 53).

Las reglas para Nftables serían las siguientes:

```
#!/usr/sbin/nft -f

define WAN_IF = enp1s0
define WAN_IP = 192.168.122.254

define LAN_IF  = enp7s0
define LAN_IP  = 10.0.0.254
define LAN_NET = 10.0.0.0/24

define DMZ_IF  = enp8s0
define DMZ_IP  = 10.0.200.254
define DMZ_NET = 10.0.200.0/24

define WAN_DNS = { 172.20.254.81, 8.8.8.8, 8.8.4.4, 208.67.222.123,
208.67.220.123 } # Google and OpenDNS servers

define SSH_SERVER = 10.0.200.2
define VPN_SERVER = 10.0.200.2
define WEB_SERVER = 10.0.200.1

define IT_HOSTS   = 10.0.0.50

flush ruleset

table ip6 filter {
    chain INPUT {
        type filter hook input priority filter; policy drop;
    }
```

```
    chain OUTPUT {
        type filter hook output priority filter; policy drop;
    }

    chain FORWARD {
        type filter hook forward priority filter; policy drop;
    }
}

table ip filter {

    chain LAN_to_FW {
        ip saddr $IT_HOSTS tcp dport 22 counter accept comment "Accept SSH"
    }

    chain FW_to_WAN {
        ip daddr $WAN_DNS tcp dport 53              counter accept comment "DNS
from FW (TCP)"
        ip daddr $WAN_DNS udp dport 53              counter accept comment "DNS
from FW (UDP)"
                          tcp dport { http, https } counter accept comment "Web
traffic from FW"
    }

    chain LAN_to_WAN {
        ip daddr $WAN_DNS tcp dport 53              counter accept comment "DNS
from LAN (TCP)"
        ip daddr $WAN_DNS udp dport 53              counter accept comment "DNS
from LAN (UDP)"
                          tcp dport { http, https } counter accept comment "Web
traffic from LAN"
    }

    chain LAN_to_DMZ {
        ip daddr $WEB_SERVER tcp dport { http, https } counter accept comment
"Access web servers from LAN"
        ip saddr $IT_HOSTS   tcp dport 22              counter accept comment
"Access to DMZ Servers (SSH) from IT admins"
    }

    chain DMZ_to_WAN {
        tcp dport 53              counter accept comment "DNS from DMZ (TCP)"
        udp dport 53              counter accept comment "DNS from DMZ (UDP)"
        tcp dport { http, https } counter accept comment "Web traffic from DMZ"

    }

    chain WAN_to_DMZ {
        ip daddr $WEB_SERVER tcp dport { 80, 443 }  counter accept comment
"Accept WEB"
        ip daddr $SSH_SERVER tcp dport 22   counter accept comment "Accept SSH
(port 22)"
        ip daddr $VPN_SERVER udp dport 3394 counter accept comment "Accept
OpenVPN (port 3394)"
```

```
    }

    chain input {
        type filter hook input priority filter; policy drop;

        iif lo                      accept         comment "Accept any
localhost traffic"

        ct state invalid                    drop   comment "Drop invalid
connections"
        ct state established,related        accept comment "Accept traffic
originated from us"
        meta l4proto icmp           counter accept comment "Accept ICMP"
        ip protocol igmp            counter accept comment "Accept IGMP"

        iif $LAN_IF ip daddr $LAN_IP              ct state new         jump
LAN_to_FW comment "Connections from LAN to FW"
        iif $WAN_IF ip daddr $WAN_IP tcp dport 22 ct state new counter accept
comment "Direct connection from WAM (debug in virtual)"
    }

    chain forward {
        type filter hook forward priority filter; policy drop;

        ct state invalid            drop           comment "Drop invalid
connections"
        ct state established,related accept        comment "Accept traffic
originated from us"
        meta l4proto icmp           counter accept comment "Accept ICMP"

        iif $LAN_IF oif $WAN_IF ip saddr $LAN_NET                   ct state
new jump LAN_to_WAN
        iif $LAN_IF oif $DMZ_IF ip saddr $LAN_NET ip daddr $DMZ_NET ct state
new jump LAN_to_DMZ

        iif $DMZ_IF oif $WAN_IF ip saddr $DMZ_NET                   ct state
new jump DMZ_to_WAN
        iif $WAN_IF oif $DMZ_IF                   ip daddr $DMZ_NET ct state
new jump WAN_to_DMZ
    }

    chain output {
        # Accept every outbound connection
        type filter hook output priority filter; policy drop;

        oif lo                      accept         comment "Accept any
localhost traffic"
        ct state invalid                    drop   comment "Drop invalid
connections"
        ct state established,related        accept comment "Accept traffic
originated from us"
        meta l4proto icmp           counter accept comment "Accept ICMP"
        ip protocol igmp            counter accept comment "Accept IGMP"
```

```
        oif $WAN_IF ct state new jump FW_to_WAN
    }
}

table nat {
    chain masquerading {
        type nat hook postrouting priority srcnat;
        oif $WAN_IF masquerade;
    }

    chain port_forwarding {
        type nat hook prerouting priority dstnat;
        iif $WAN_IF tcp dport { 80, 443 } dnat to $WEB_SERVER
        iif $WAN_IF tcp dport 1313        dnat to $SSH_SERVER:22
        iif $WAN_IF udp dport 3394        dnat to $VPN_SERVER
    }
}
```

19. Enlaces

- **Configuring firewalls and packet filters. RHEL 9 Documentation**. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_firewalls_and_packet_filters/
- **Firewalld**. https://firewalld.org/
- **Firewalld, Fedora project**. https://fedoraproject.org/wiki/Firewalld
- **Netfilter**. https://www.netfilter.org/
- **Nftables Arch Wiki**. https://wiki.archlinux.org/title/nftables
- **¿Qué es una DMZ?**. https://www.incibe.es/empresas/blog/dmz-y-te-puede-ayudar-proteger-tu-empresa
- **RHEL 9 Configuring firewalls and packet filters**. https://access.redhat.com/documentation/es-es/red_hat_enterprise_linux/9/html/configuring_firewalls_and_packet_filters/getting-started-with-nftables_firewall-packet-filters
- **Uncomplicated firewall**. https://wiki.ubuntu.com/UncomplicatedFirewall
- Wikipedia:
    - **Netfilter**. https://es.wikipedia.org/wiki/Netfilter
- **Nftables Wiki**. https://wiki.nftables.org/wiki-nftables/index.php/Main_Page
    - **Configuring chains**. https://wiki.nftables.org/wiki-nftables/index.php/Configuring_chains
    - **Configuring tables**. https://wiki.nftables.org/wiki-nftables/index.php/Configuring_tables
    - **Netfilter hooks**. https://wiki.nftables.org/wiki-nftables/index.php/Netfilter_hooks

- **Simple rule management**. https://wiki.nftables.org/wiki-nftables/index.php/Simple_rule_management
- **What is nftables**. https://wiki.nftables.org/wiki-nftables/index.php/What_is_nftables%3F