Cortafuegos

Departamento de Sistemas Telemáticos y Computación (GSyC)

http://gsyc.urjc.es

Abril de 2014

Cortafuegos

Un cortafuegos (*firewall*) es un dispositivo que filtra el tráfico que intenta salir o entrar de una red: analiza cada paquete y decide, en base a ciertas reglas, si lo acepta o si lo rechaza.

- Puede considerar cabeceras, cargas o ambas cosas.
- Un cortafuegos con varias interfaces (*multi-homed*) también puede hacer encaminamiento, de forma que el resultado de aplicar las reglas sea enviar un datagrama a uno u otro segmento de red (o descartarlo)

- Es solo uno de los elementos necesarios para que el sistema sea seguro
- Implementa las políticas de seguridad de un organismo, si son incorrectas, o se implementan mal, el sistema no será seguro.
- Los cortafuegos hoy son impresencibles, hay mucho tráfico malicioso, sobre todo generado automáticamente.

Que NO es un cortafuegos

- NO es un sistema de autenticación, que solicite nombre de usuario y contraseña, que haga análisis biométricos o confirme la validez de certificados
  - Algunos cortafuegos incluyen alguna funcionalidad de este estilo, pero no es el enfoque más recomendable.
- NO es un RAS, *Remote Acces Server*
- NO es un elemento capaz de ver el tráfico cifrado.
  - En algunos casos excepcionales, como el modo transporte de IPsec, puede ver las cabeceras. Pero en modo túnel, no

Si necesitamos examinar todo el tráfico, incluido el cifrado, habrá que colocar el cortafuegos antes del cifrado (en el tráfico saliente) y después del descifrado (en el tráfico entrante). O incluso prohibir todo tráfico cifrado

Que NO es un cortafuegos (II)

- No es un antivirus. Un cortafuegos puede tener docenas ¿centenas? de reglas. Un antivirus, busca millones de firmas de virus
  - Algunos cortafuegos incluyen funcionalidad de este tipo, pero no es el mejor enfoque

  Un cortafuegos detiene mucho tráfico generado por virus, pero porque el tráfico incumple las normas, no porque identifique al virus
- No es un IDS, (*Intrusion Detection System*)
  Un IDS analiza el tráfico interno, buscando patrones conflictivos. Un cortafuegos solo analiza el tráfico que entra o que sale

Que NO es un cortafuegos (III)

- Obviamente no es un elemento de protección contra ataques por ingeniería social, ataques físicos, sabotaje, etc
- No es una defensa contra amenazas en dispositivos extraibles (cdrom, pendrives...)
- No es una defensa contra spam, ni contra correos con anexos maliciosos

Cosas que SÍ puede hacer un cortafuegos

- Puede delimitar zonas dentro de un mismo organismo. P.e separar desarrollo de producción
- Puede implementar NAT
  - Atención, cada *puerto abierto* es un riesgo potencial
- Puede actuar como proxy
  - Problema: hay que configurar explícitamente las aplicaciones
- Puede llevar un registro (*log*) de eventos. Normalmente eventos especiales
- Puede recabar información para análisis estadístico del tráfico

Clasificación de los cortafuegos

Según sus prestaciones y facilidad de uso

- De uso personal (doméstico o pequeña oficina)
- De uso corporativo

Según el tipo de *hardware* y SO (Sistema Operativo) donde se ejecute

- Cortafuegos software
  Aplicación ejecutándose sobre ordenador y SO de propósito general
- Cortafuegos software sobre SO adaptado
  - Distro Linux orientada a cortafuegos: SmoothWall, IPCop, IPFire
  - Distro de FreBSD: m0n0wall, pfSense
- Cortafuegos hardware
  Software corriendo sobre hardware específico

Tipos de filtrado: filtrado estático

Filtrado estático, aka sin estado (*stateless*) Mira cada paquete,sin considerar relación con anteriores ni posteriores. Se centra sobre todo en el nivel 3 (red) , también nivel 4 (transporte)

- Las reglas analizan dirección IP y puerto, tanto de origen como de destino
- Campo *protocolo* de la cabecera IP
- Código ICMP
- Flags de fragmentación
- Opciones IP

Ventajas del filtrado estático:

- Es rápido y eficiente

Inconvenientes:

- No es fácil distinguir peticiones de respuestas
- Sabemos a qué puerto se dirige el datagrama, pero no a qué aplicación
- Podemos provocar problemas en otras máquinas, enviando unos paquetes sí y otros no de la misma *conexión*
  - Llegando incluso a un ataque DOS involuntario
- Hay ataques basados en datagramas que individualmente son inofensivos, pero en secuencia resultan dañinos

Tipos de filtrado: filtrado dinámico

Filtrado dinámico *aka* con estado (*stateful*) aka orientado a la sesión

Relaciona un paquete con paquetes precedentes

- Típicamente trabaja en el nivel 4 (transporte)
- Comprueba que se sigan las normas de un protocolo. P.e el *three-way handshake* de TCP (SYN, SYN/ACK, ACK)
  - Aunque hoy, un cortafuegos dinámico que solo analice TCP es muy pobre

Pseudoconexiones

TCP tiene verdaderas conexiones, en UDP e ICMP se pueden considerar *pseudoconexiones*:

- Para UDP, un datagrama desde un socket A hasta un socket B, al que sigue otro datagrama desde el socket B hasta el socket A
- Un mensaje de error ICMP contiene la cabecera del paquete que provocó el error: en una pseudoconexión correcta esta referencia debe ser consistente con un paquete que realmente haya sido enviado

Ventajas del filtrado dinámico (I)

Puede analizar capas de sesión, presentación y aplicación Admite semántica como p.e.

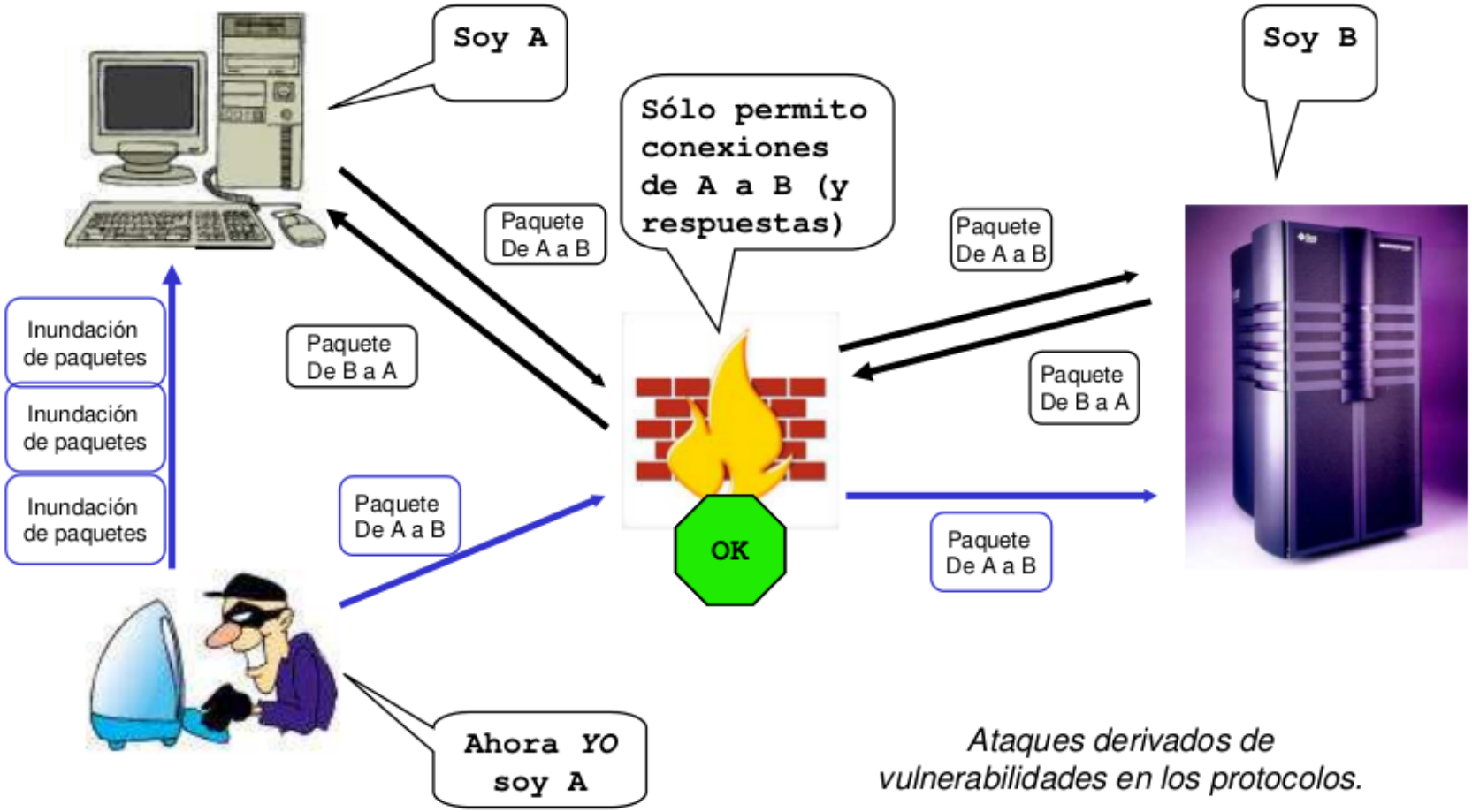
- *acepta el tráfico al puerto 80 pero solo si es HTTP*
- *Descarta las peticiones con inyección de SQL peligroso*
  Esto suele ser un tipo de cortafuegos especializado, *Application firewall*

Ventajas del filtrado dinámico (II)

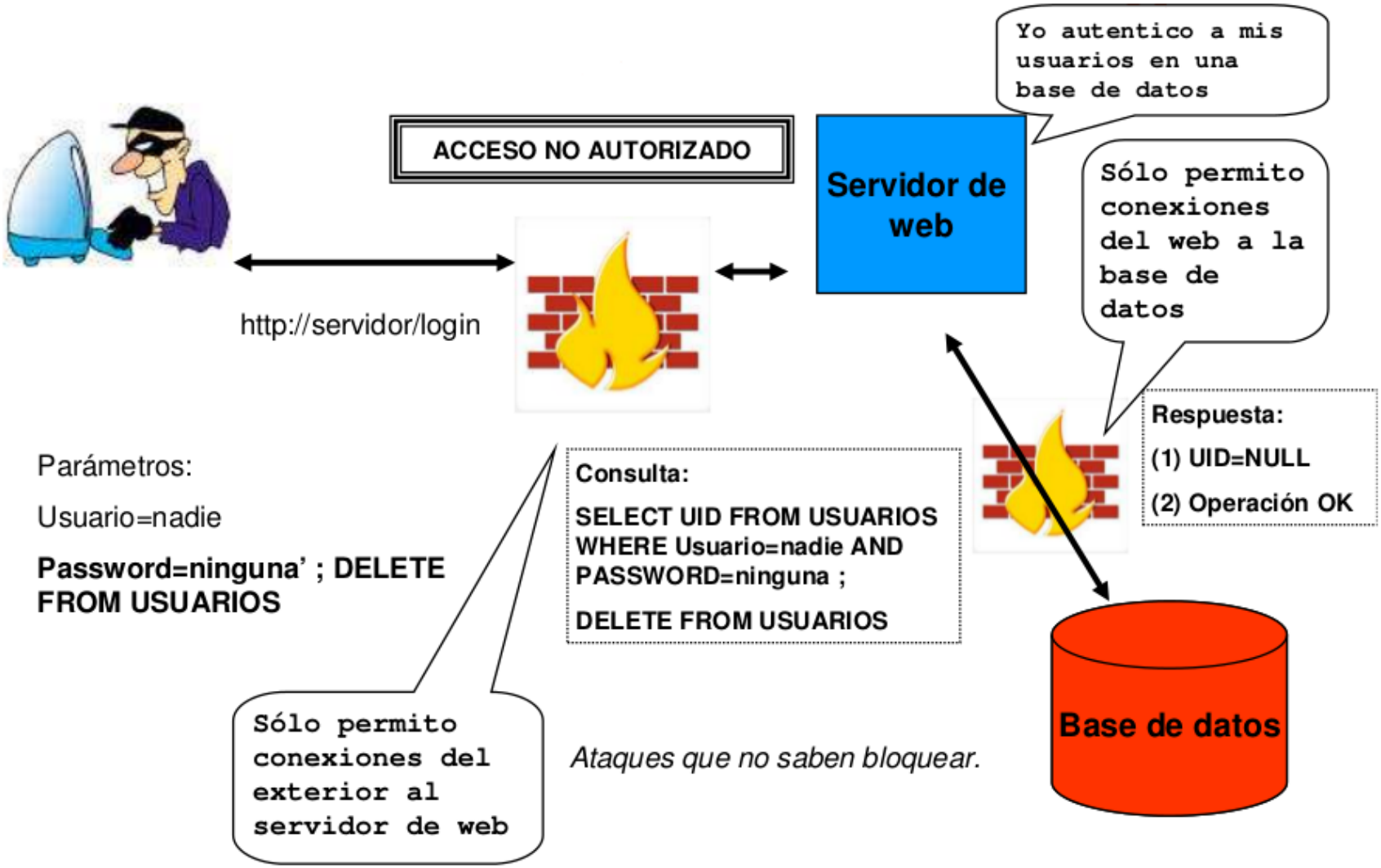
- Filtrado por contenido
  Analiza nombres de dominio, URL, nombre de fichero, extensión de fichero, etc
- Admite un tráfico relacionado con otro. P.e. FTP (que establece una conexión de control y otra de datos)
- Maneja el concepto de *sesión*
  - Conexiones y pseudoconexiones
  - Analiza que un paquete esté abriendo una nueva sesión de forma correcta (si abrir nuevas sesiones es admisible)
  - Cuando llega un paquetes, se busca si pertenece a una sesión abierta correctamente y entonces se acepta. En otro caso, se se descarta

La vinculación de un paquete con una sesión puede ser más o menos precisa.

- Un cortafuegos básico puede considerar solo la dirección IP y el puerto
  - Un atacante puede burlarlo, generando paquetes dañinos, aparentemente perteneciendo a la misma sesión
- Un análisis más riguroso (y caro) es mirar número de secuencia y números de asentimiento
  - También puede falsificarse, pero es más complicado



Robo de sesión TCP



Inyección de código SQL

Inconvenientes del filtrado dinámico

A partir de cierto caudal de tráfico, mantener estado es muy costoso en memoria y tiempo

- Es complicado guardar el estado, o transferirlo a otro cortafuegos que esté ofreciendo redundancia o equilibro de carga
- Hace al cortafuegos especialmente vulnerable a los ataque de DOS
- Transcurrido cierto tiempo, habrá que olvidar el estado de cada paquete concreto ¿cuándo? ¿cuál?
  - Es muy posible que un paquete se retrase un poco y llegue cuando el estado de la conexión ya se haya perdido. Así que es típico que a las reglas dinámicas se añadan reglas estáticas para suplir (en parte) este caso

Como hemos visto, por si falla el filtro dinámico conviene añadir un filtro estático.
Pero además una técnica habitual y recomendable es

- Poner en una primero línea de defensa un filtro estático, básico y barato
- En segunda línea un filtro dinámico, más fino pero más costoso

Estrategias de seguridad

- Prohibición por omisión
- Sistemas *fail-secure*, no *fail-safe*
- Segmentación de la red
- Otras estrategias

Prohibición por omisión

Para empezar, todo está prohibido
Todo lo que no está permitido explícitamente, está prohibido (*deny by default, allow by exception*)

- El enfoque más razonable es que por omisión todo el tráfico se rechace, y que luego vayamos indicando qué excepciones sí están permitidas
- Esto es exactamente lo contrario que desearían los usuarios y posiblemente la dirección de la empresa/organismo
- En ocasiones se aplica el criterio opuesto: Todo lo que no está prohibido explícitamente, está permitido. En principio da menos trabajo, pero es mucho más peligroso

Sistema fail-secure, no fail-safe

Los términos *fail-secure* vs *fail-safe* NO significan que no haya fallos, ni que los fallos sean muy poco probables
Significan que, en caso de fallo, el sistema resulte *safe* o resulte *secure*
Problema de los hispanoparlantes: En inglés *safe* y *secure* son dos palabras relacionadas pero distintas. En español, no tenemos ese matiz, usamos *seguro* para ambas

- *safe*: Que no hace daño, inofensivo
  En el ámbito de control de acceso, cerraduras, etc: un sistema que si se desconecta, se queda abierto
- *secure*: Protege contra un peligro
  En el ámbito de control de acceso, cerraduras, etc: un sistema que si se desconecta, se queda cerrado

¿Cómo debe ser una puerta de emergencia? ¿Fail-safe o fail-secure?
¿Y una puerta de entrada?
¿Y un cortafuegos? [1]

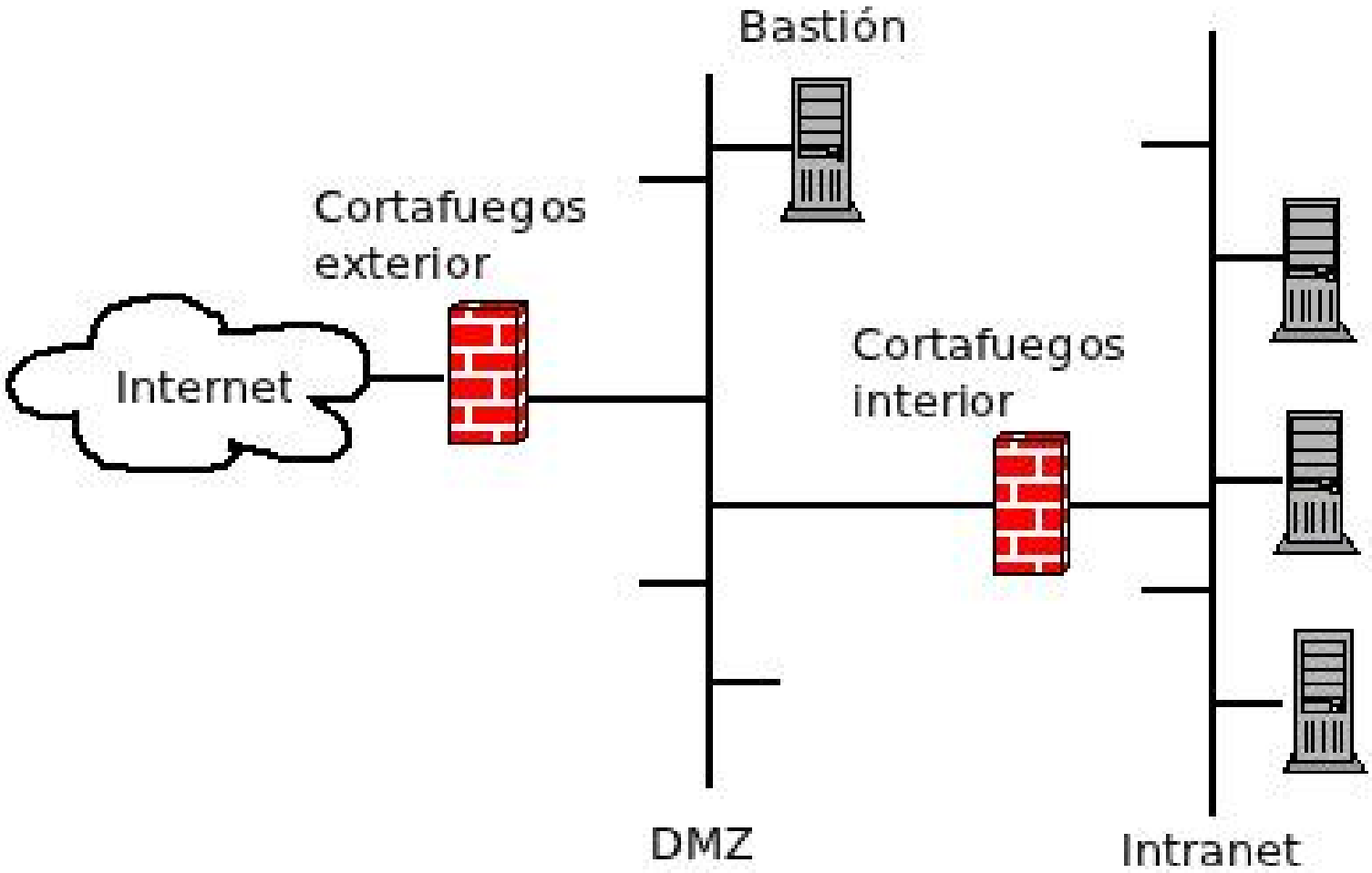
[1] Un cortafuegos fuera de servicio debe cerrar el tráfico. En terminología de puertas y cerraduras, esto es fail-secure. Aunque en el ámbito de los cortafuegos, a veces se le da un significado distinto y se llama fail-safe

Segmentación de la red

En un sistema de prestaciones intermedias, normalmente se divide la red en tres zonas

- Internet. Zona muy insegura
- Zona demilitarizada (DMZ, *De-Militarized Zone*), aka Red perimetral
  Seguridad intermedia. Aquí se coloca un máquina llamada *bastión*, ofreciendo servicio al exterior (HTTP, SMTP, DNS etc)
- Intranet. Zona relativamente segura

Entre la DMZ e internet se coloca el cortafuegos exterior
Entre la DMZ y la LAN, el cortafuegos interior


Bastión
Cortafuegos
exterior
Internet
Cortafuegos
interior
DMZ
Intranet

- Se pueden usar configuraciones más sencillas, con un único cortafuegos y segmento de red
- Se pueden usar configuraciones más complejas, con varios segmentos, varios bastiones, etc

Otras estrategias de seguridad (I)

- Privilegios mínimos
  Ofrecer a cada entidad (usuario, máquina, proceso, conexión) los privilegios mínimos imprescindibles. (No siempre es posible)
- Defensa redundante
  Usar varias barreras, que haya que romper una Y otra.
  (Mucho cuidado con que no sea una U otra)
  Idealmente, diversidad en la defensa. P.e. con equipo de distinto fabricante (auque esto es muy caro y no garantiza el éxito)
- Cuello de botella
  Que *todo* el tráfico pase por el cortafuegos. Mucho cuidado con dispositivos de acceso a internet no autorizados
- Análizar y reforzar el sistema globalmente. Una cadena es tan fuerte como el eslabón más débil

Otras estrategias de seguridad (II)

- Los logs son muy importantes. Un atacante intentará borrar sus huellas. Los logs deberían estar fuera de su alcance
- Un buen cortafuegos puede generar alarmas. Por un canal alternativo a la red ordinaria. (Canal que sea difícil de bloquear por el atacante)
- Usar direcciones IP siempre que sea posible. Procurar no confiar nunca en el DNS
- Simplicidad: Una configuración compleja puede tener errores no evidentes
- Seguridad por diseño. No confiamos en la seguridad por la oscuridad, pero tampoco hacemos públicos nuestros detalles de configuración

Consideraciones sobre el filtrado

A continuación desarrollaremos las siguientes consideraciones sobre el filtrado

- Conocimiento de los protocolos
- Bidireccionalidad del tráfico
- Rechazo de paquetes

Conocimiento de los protocolos

- Es necesario especificar qué protocolos estarán autorizados a superar el cortafuegos
- Es necesario conocer con cierto detalle qué conexiones y a qué puertos usa cada protocolo
  - Los protocolos más habituales están bien documentados
  - Otros protocolos tienen escasa o nula documentación. Hay dos posibilidades
    1. Prohibirlos (si procede)
    2. Analizarlos aplicando ingeniería inversa

Bidireccionalidad del tráfico

Los protocolos suelen ser bidireccionales.

- Aunque para hacer daño, puede bastar un paquete en un único sentido
- Atención al significado de las palabras *entrada* y *salida*, se puede considerar el sentido del servicio o el sentido del paquete
  Ejemplo: Un cliente fuera de la red hace una petición http y un servidor dentro de la red responde
  1. Puede que estemos usando el criterio del servicio. En este caso, todo este tráfico es de entrada
  2. Puede que estemos usando el criterio del sentido del paquete. En este caso, la petición es de entrada y la respuesta de salida.

  Lo habitual y lo recomendable es (2), pero en toda documentación conviene indicarlo explícitamente. Y estar atento a un posible uso de (1)

Rechazo de paquetes

Cuando se decide que un paquete no debe superar el cortafuegos, hay varias opciones

- Enviar mensaje de error ICMP genérico, del grupo *destination unreachable*: *host unreachable* o *network unreachable*
- Enviar mensaje de error ICMP más específico, del grupo *destination administratively unreachable*: *host administratively unreachable* o *network administratively unreachable*
- Descartar el paquete sin devolver ningún mensaje de error

Lo más habitual es descartar el paquete sin más

- Para no dar pistas al atacante.
- Para no generar más tráfico
- Para evitar ser cómplices involuntarios de un DDOS

A pesar de que esto perjudica a un usuario legítimo, que tendrá que hacer más reintentos

Tratamiento de datagramas en núcleo sin netfilter

Soy una máquina con Linux. El tratamiendo de un datagrama depende de si soy un router o no

- ¿Cómo se que si soy un router?
  Consultando el valor de `/proc/sys/net/ipv4/ip_forward`
- ¿Cómo hacer que me convierta en router?
  `echo 1 > /proc/sys/net/ipv4/ip_forward`
  Esto no es persistente, suele incluirse en un script de arranque

Previamente debe haber tablas de encaminamiento, creadas a mano o mediante algoritmos de encaminamiento

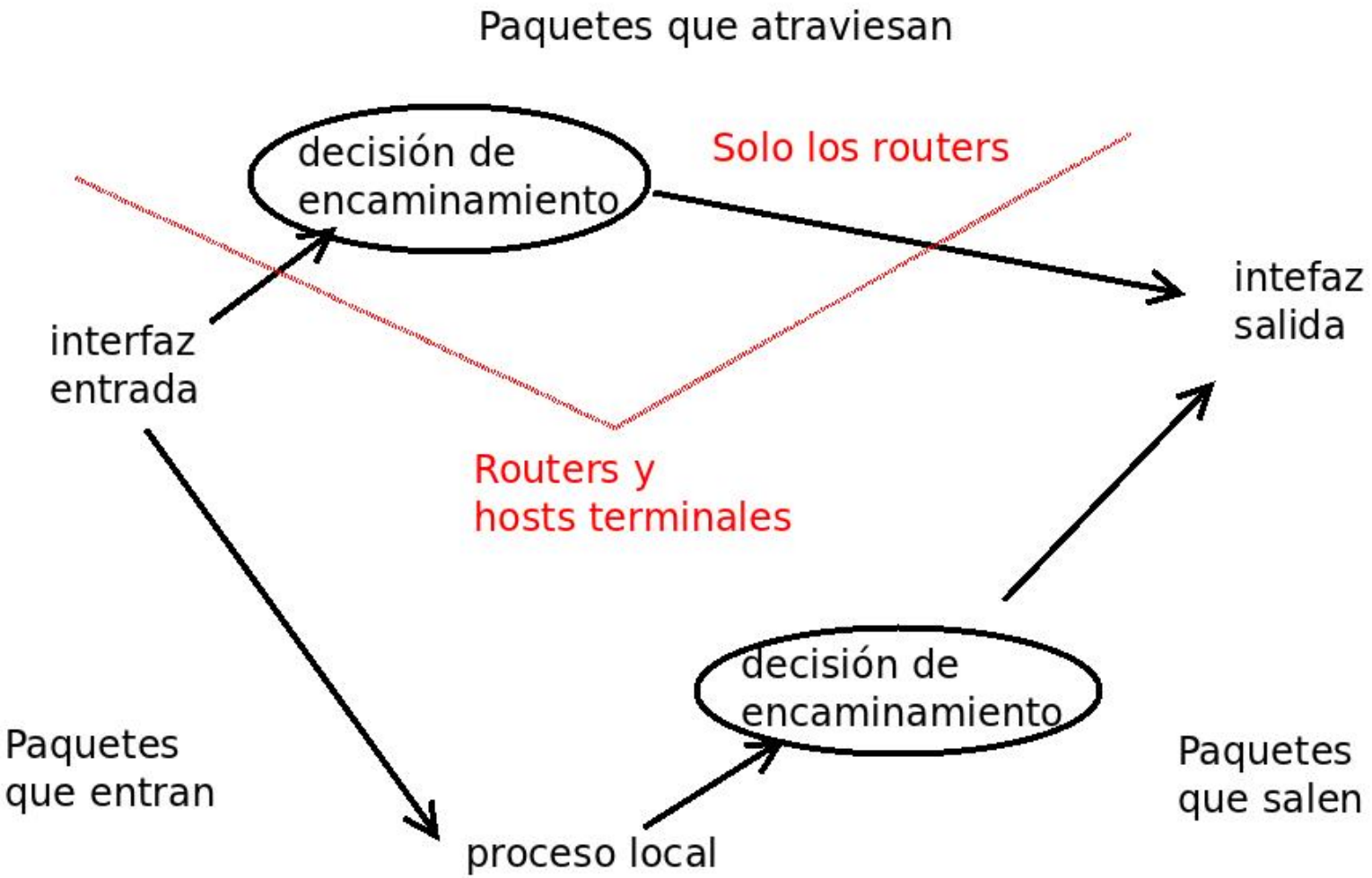
- Decidir el encaminamiento: consultar mi tabla de encaminamiento y averiguar por cuál de mis interfaces debo enviar el datagrama
- Encaminar: enviar el datagrama por el interfaz adecuado

Soy una máquina conectada a la red. No tengo iptables
Recibo un datagrama

```
Si es para mí, me lo quedo
Si no es para mí
- Si soy un router, decido su encaminamiento y lo encamino
- Si no soy un router, tiro el paquete
```

Envío un datagrama

```
- Decido su encaminamiento y lo encamino
```


Paquetes que atraviesan
decisión de encaminamiento
Solo los routers
intefaz salida
interfaz entrada
Routers y hosts terminales
decisión de encaminamiento
Paquetes que entran
proceso local
Paquetes que salen

Netfilter - iptables

- Netfilter es un framework de Linux que permite interceptar y modificar paquetes IP
- `iptables` es la herramienta de que permite administrar Netfilter
- En rigor, el cortafuegos es Netfilter, pero normalmente se le llama iptables
- El proyecto netfilter ha desarrollado una herramienta más avanzada, *nftables*, cuyo objetivo es reemplazar a iptables. La primera versión estable es de enero de 2014
- Permite definir reglas aplicables a los paquetes IP que entran y/o salen de una máquina para :
  - Filtrado de paquetes *(packet filtering)*.
  - Seguimiento de conexiones *(connection tracking)*.
  - Traducción de direcciones IP y puertos (NAT, *Network Address Translation*).

  Esto solo puede hacer el usuario root

iptables solo trabaja con paquetes IPv4
Herramientas similares son

- ip6tables, para IPv6
- ebtables, para ethernet
- arptables, para ARP

Hay 3 conceptos básicos en `iptables` :

- reglas
- cadenas
- tablas

Debemos prestar atención al uso de mayúsculas y minúsculas, iptables es *case-sensitive*

Reglas

- regla = IF **condición** THEN **acción**:
  - **condición**: características que debe cumplir un paquete para que la regla le sea aplicable. P.e.

    `-p tcp --dport 80`: el protocolo es TCP y el puerto destino es 80

    `-s 13.1.2.0/24`: la dirección de origen es de la subred 13.1.2.0/24.

  - **acción**: qué hacer con el paquete si cumple la condición. P.e.

    `ACCEPT`: el paquete se acepta
    `DROP`: el paquete se descarta
    `SNAT --to-source 13.1.2.1`: se cambia la IP origen del paquete
    `LOG`: el paquete se registra

    Una regla puede no tener acción: se limita a contar el paquete
- Una lista ordenada de reglas es una cadena
- Un grupo de cadenas es una tabla

Cadenas

Hay dos tipos de cadenas:

1. Predefinidas
   - PREROUTING la recorren todos los paquetes que entran en la máquina
   - INPUT la recorren los paquetes que entran en la máquina con destino a la propia máquina
   - OUTPUT la recorren los que salen (procedentes de la propia máquina)
   - FORWARD la recorren los que atraviesan la máquina
   - POSTROUTING la recorren todos los que salen de la máquina
2. Definidas por el usuario, aka subcadenas

El administrador las añade a cualquier otra cadena. P.e.

- MI_CADENA.

Desde cualquier cadena se puede llamar a una subcadena

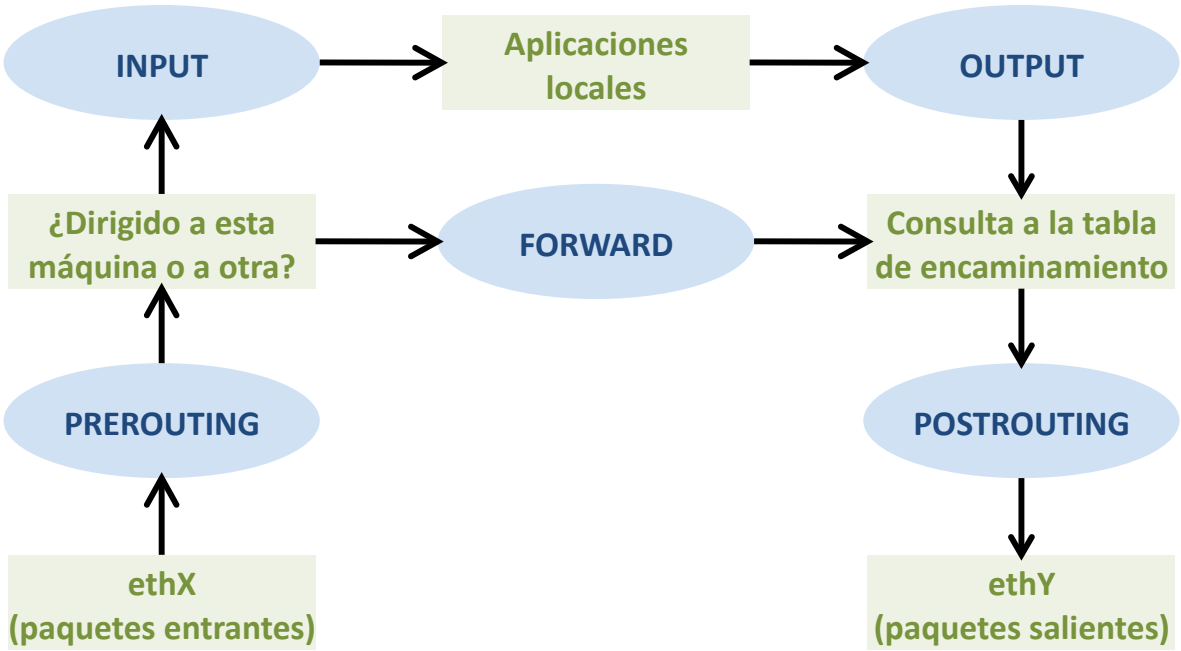
Las cadenas se nombran con mayúsculas

```
SI un paquete cumple cierta condición
ENTONCES hacer algo con el paquete
        - DROP. Descartarlo silenciosamente
        - REJECT. Descartarlo y enviar mensaje ICMP a origen
        - ACCEPT. Superar el cortafuegos
        - Enviarlo a otra cadena (subcadena)
        - ...
```

- Cuando se ejecuta una acción DROP, REJECT o ACCEPT ya no se evalúan más reglas
- Las acciones se escriben con mayúsculas
- Si el paquete no cumple la condición, sigue avanzando por la cadena
  - Una vez recorrida una cadena predefinida, se aplica la política por defecto de esa cadena: ACCEPT o DROP [2]

[2] Si no indicamos política por defecto, se entiende que es ACCEPT

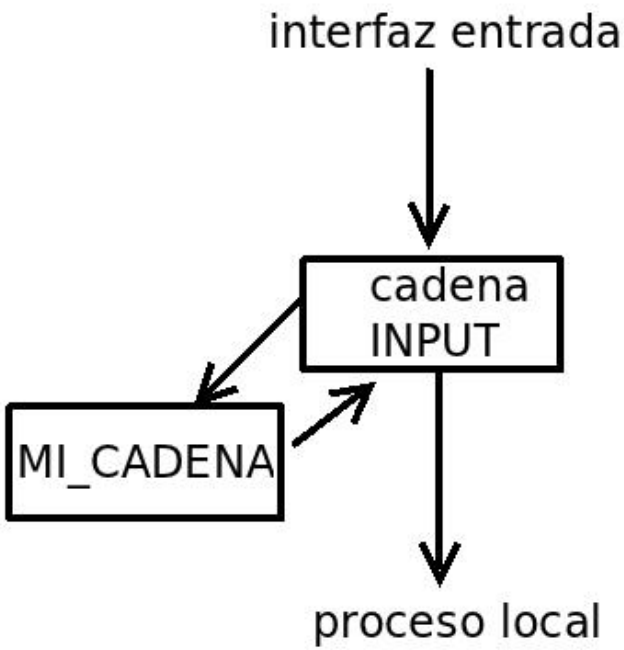
Tratamiento de datagramas en router con netfilter



- Los datagramas que entran en la máquina, dirigidos a la máquina, pasan por PREROUTING e INPUT, acaban en una aplicación local
- Los datagramas que atraviesan un router (entran pero van dirigidos a otra máquina), pasan por PREROUTING, FORWARD y POSTROUTING
  (Recordatorio: si la máquina no es un router, ningún paquete la atraviesa)
- Los datagramas generados por una aplicación local con destino al exterior, pasan por OUTPUT y POSTROUTING
- Los datagramas generados por una aplicación local para otra aplicación local pasan por OUTPUT, POSTROUTING, PREROUTING e INPUT

- Cadena `PREROUTING`:
  - Reglas que se aplican a los paquetes que llegan a la máquina. Esta cadena se ejecuta antes de comprobar si el paquete es para la propia máquina o hay que reenviarlo.
- Cadena `INPUT`:
  - Reglas que se aplican a los paquetes destinados a la propia máquina. Esta cadena se ejecuta justo antes de entregarlos a la aplicación local.
- Cadena `FORWARD`:
  - Reglas que se aplican a los paquetes que han llegado a la máquina pero van destinados a otra y hay que reenviarlos. Esta cadena se ejecuta antes de consultar la tabla de encaminamiento.
- Cadena `OUTPUT`:
  - Reglas que se aplican a los paquetes creados por la propia máquina. Esta cadena se ejecuta justo después de que la aplicación le pase los datos a enviar al *kernel* del sistema operativo y antes de consultar la tabla de encaminamiento.
- Cadena `POSTROUTING`:
  - Reglas que se aplican a los paquetes que salen de la máquina, tanto los creados por ella como los que se reenvían. Esta cadena se ejecuta después de consultar la tabla de encaminamiento.

Subcadenas



Ejemplo de subcadena llamada desde la cadena INPUT

- Desde cualquier cadena se puede llamar a una subcadena
- Al datagrama se le van aplicando las reglas de la subcadena
  - Si alguna condición se satisface, se ejecuta la acción
    - Tras acciones ACCEPT, DROP o REJECT, no se evalúan más reglas
    - Tras acciones LOG o salto a subcadena, sí se siguen evaluando
  - Si no se cumple ninguna condición, el datagrama retorna a la cadena anterior

Recordemos:

- Una vez recorrida una subcadena, el flujo vuelve a la cadena desde donde fue llamada
- Una vez recorrida una cadena predefinida, se aplica la política por defecto de esa cadena: ACCEPT o DROP

Es un esquema análogo a las funciones y subfunciones de cualquier lenguaje de programación

- Se pueden anidar subcadenas, desde una subcadena se puede llamar a otra subcadena

Las subcadenas no tienen política por defecto, siempre devuelven el datagrama a la cadena desde donde fueron llamadas

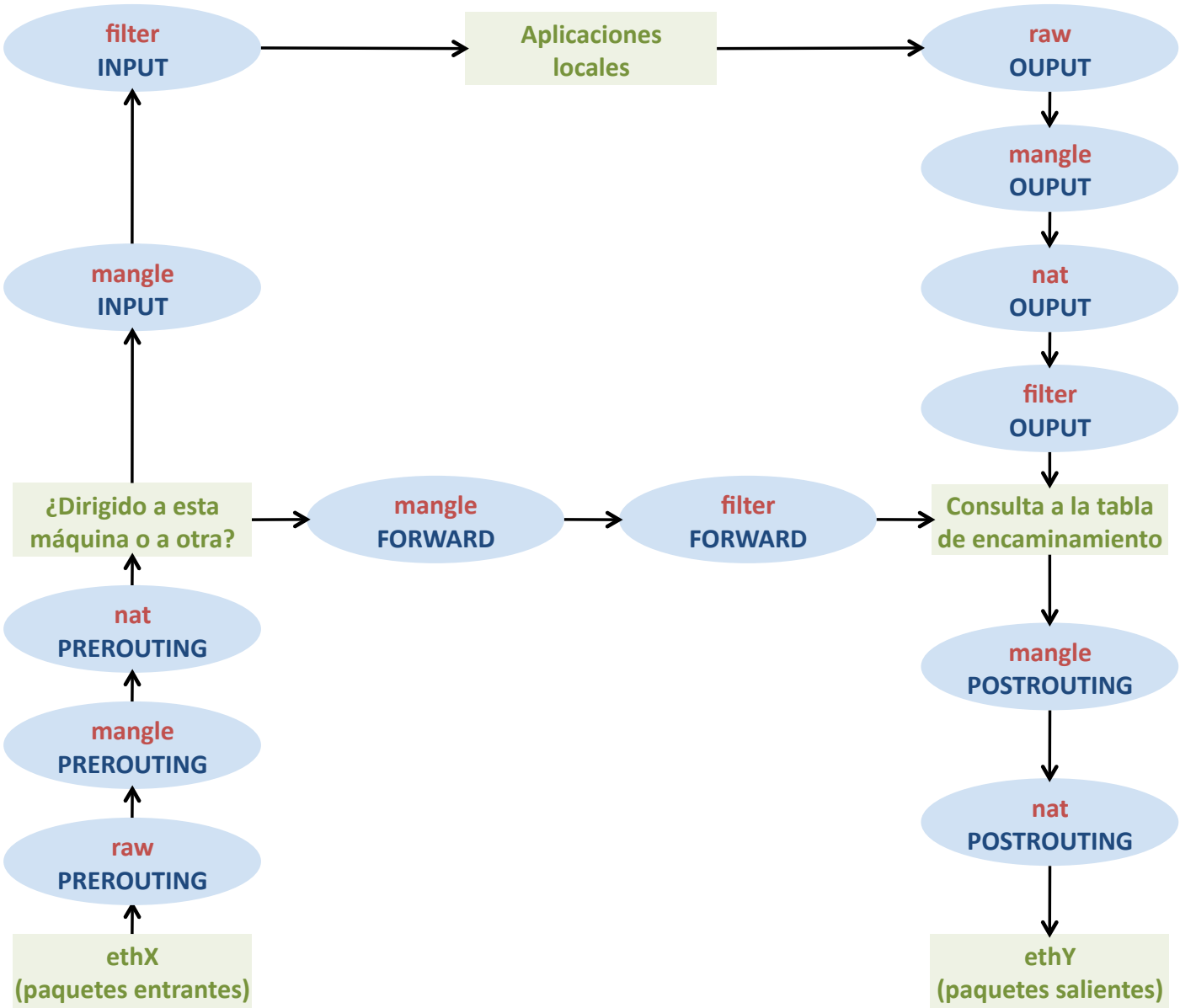
Tablas

- Una tabla contiene un conjunto de cadenas, tanto predefinidas como de usuario.
- Las tablas se nombran con minúsculas
- Una tabla concreta engloba las reglas (agrupadas en cadenas) relacionadas con un tipo de procesamiento de los paquetes.
- `Netfilter` define las tablas:
  - `filter`: engloba las reglas de filtrado, esto es, las que deciden que un paquete continúe su camino o sea descartado.
  - `nat`: engloba las reglas de modificación de direcciones IP y puertos de los paquetes
  - `mangle`: engloba las reglas de modificación de algunos campos de las cabeceras del paquete. Ejemplo: TTL
  - `raw`: engloba las reglas que permiten marcar excepciones al seguimiento que hace el *kernel* de las conexiones y *pseudoconexiones*

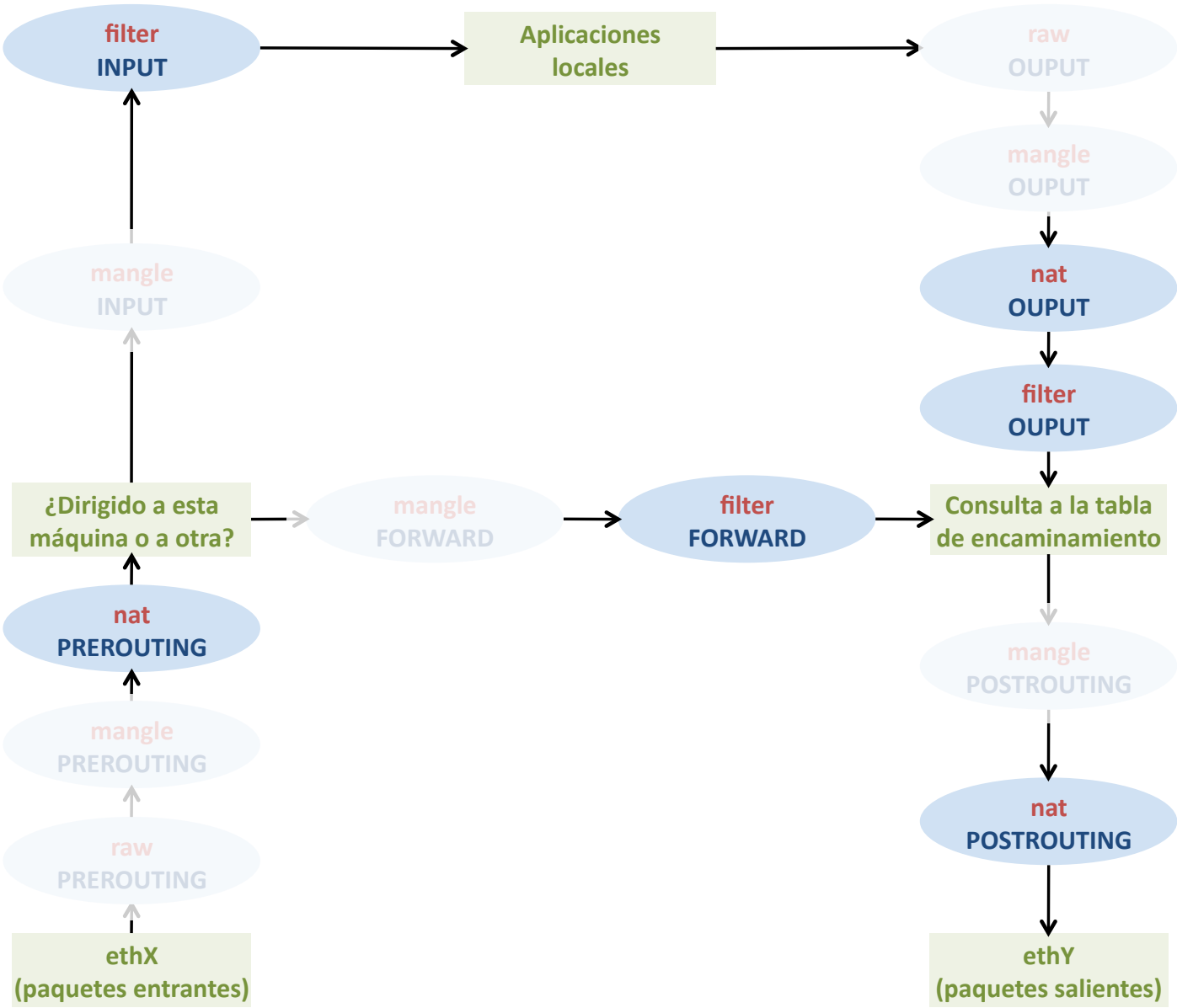
Cadenas predefinidas de cada tabla

- La tabla `filter` incluye las cadenas:
  - `FORWARD`
  - `INPUT`
  - `OUTPUT`
- La tabla `nat` incluye las cadenas:
  - `PREROUTING`
  - `OUTPUT`
  - `POSTROUTING`
- La tabla `mangle` incluye las cadenas:
  - `PREROUTING`
  - `FORWARD`
  - `INPUT`
  - `OUTPUT`
  - `POSTROUTING`
- La tabla `raw` incluye las cadenas:
  - `PREROUTING`
  - `OUTPUT`

Movimiento de los paquetes por tablas y cadenas



Movimiento de los paquetes (versión simplificada)



Uso de iptables

- Las reglas se borran al reiniciar la máquina. Así que hay que cargarlas en un script en el arranque. Es conveniente cargarlas antes de activar los interfaces de red
- La mayoría de las opciones pueden especificarse de dos formas alternativas, equivalentes
  1. Con un guión y letra mayúscula de opción
  2. Con dos guiones y el nombre de la opción (en minúsculas)

```
iptables -A INPUT -p tcp -s 193.186.1.6 -j DROP
iptables --append INPUT --protocol tcp --source 193.186.1.6 --jump DROP
```

Instalación de las reglas en el cortafuegos

El administrador siempre debería configurar el cortafuegos desde la consola local, no a tráves de ssh, ni mucho menos VNC, X Window, etc
Problema: con frecuencia esto es imposible. Corremos el riesgo de perder el acceso al cortafuegos
Soluciones:

- Probar siempre los scripts en un entorno local lo más similar posible, para detectar al menos errores sencillos
- Mientras depuramos los scripts, usar ACCEPT como política por omisión.
- Programar en un cron la desactivación periódica del cortafuegos. Tal vez cada dos minutos, o dos horas, algunos días...

Muy importante: una vez el sistema esté en producción, es imprescindible cerrar todas las puertas traseras

Sintaxis de iptables

`iptables [-t <tabla>] <mandato> [<condición>] [<acción>]`

Si no se especifica una tabla se utilizará por defecto la tabla `filter`.

Mandato es la traducción al español de la palabra *command*. También se puede decir *orden*. Casi siempre se traduce como *comando*, pero no es correcto

```
comando.  (De comandar).
1. m. Mil. Mando militar.
2. m. Pequeño grupo de tropas de choque, destinado a hacer
      incursiones ofensivas en terreno enemigo.
3. m. Grupo armado de terroristas.

mandato.  (Del lat. mandatum).
2. m. Orden dada a un aparato para que realice una determinada
      operación.
```

Mandatos más habituales

- `-L` `--list` Listar reglas
- `-A` `--append` Añadir reglas
- `-F` `--flush` Borrar reglas
- `-P` `--policy` Fijar política por omisión
- `-N` `--new-chain` Crear subcadena

Política por defecto

- `-P` `--policy` Fija la política por defecto de la cadena
  Puede ser ACCEPT o DROP (REJECT no es política válida)

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP
```

Al reiniciar la máquina, todas las políticas por defecto vuelven a ACCEPT

Listado de las reglas

- `-L  --list` Listar las reglas de una(s) cadena(s)
  - `-n  --numeric` Esta opción muestra las direcciones como números, sin intentar resolver nombres

`iptables -n L`	Listar todas las cadenas
`iptables -nL`	Forma abreviada
`iptables -nL FORWARD`	Listar la cadena FORWARD

Eliminación de reglas

- `-F` `--flush` Borrar todas las cadenas predefinida
- `-X` `--delete-chain` Borrar todas las cadenas de usuario

Ejemplos:

- `iptables -F`
  Borrar todas las cadenas predefinidas de la tabla *filter*
- `iptables -F INPUT`
  Borrar todas las reglas de la cadena INPUT de la tabla *filter*
- `iptables -t nat -t mangle -F`
  Borrar todas las cadenas predefinidas de las tablas *nat* y *mangle*
- `iptables -X`
  Borrar todas las cadenas de usuario de la tabla *filter*
- `iptables -t nat -t mangle -X`
  Borrar todas las cadenas de usuario de las tablas *nat* y *mangle*

Eliminar las reglas no cambia las últimas políticas fijadas

Modificación de reglas individuales

Hay opciones para borrar reglas sueltas, para insertar reglas en cierta posición y para reordenar las reglas

- Pero ya que hay que cargar las reglas mediante un script, suele ser más práctico que el script borre todas las reglas y luego fije todas las políticas e inserte las reglas adecuadas en el orden adecuado

Condiciones

- `-s --source` Dirección IP origen
- `-d --destination` Dirección IP destino
- `-i --in-interface` Interfaz de entrada
- `-o --out-interface` Interfaz de salida
- `!` negación

`-s 193.147.71.1`	Paquete proveniente de 193.147.71.1
`-d ! localhost`	Paquete no destinado a localhost
`-i lo`	Paquete que entra por el interfaz lo
`-o ! eth0`	Paquete que no sale por eth0

Todas las condiciones se escriben con minúsculas

- `-p` `--protocol` Protocolo del paquete
- `--sport` `--source-port` Puerto de origen
- `--dport` `--destination-port` Puerto de destino

Obviamente, no basta decir *el puerto 80*. ¿El puerto 80 TCP? ¿El puerto 80 UDP?

La opción `--protocol` debe preceder a `--dport/--sport`

`-p tcp`	Paquete de protocolo TCP
`--dport 80`	Paquete destinado al puerto 80
`--sport 1:1023`	Paquete con origen en los puertos del 1 al 1023
`--dport 1024:`	Paquete con destino al puerto 1024 o superior
`--sport :3000`	Paquete con origen en el puerto 3000 o inferior

El fichero /etc/services describe los *well-known* port numbers

Acciones

- `-j --jump`

```
-j DROP         Enviar el paquete a DROP (descartarlo, sin más)
-j REJECT       Rechazar el paquete y enviar un error al origen
-j ACCEPT       Aceptar el paquete. No se evalúan más reglas
-j RETURN       El paquete abandona la cadena actual
                -Si está en una subcadena, vuelve a la cadena principal
                -Si está en cadena predefinida, se le aplica política
                 por defecto
-j MI_CADENA    Enviar el paquete a MI_CADENA
-j LOG          Registrar traza del paquete
```

Ejemplos básicos

```
iptables -F
iptables -A INPUT -p icmp -s 127.0.0.1 -j DROP

root@mazinger:~# iptables -Ln
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  127.0.0.1           0.0.0.0/0

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination


iptables -A INPUT -p icmp -s 192.168.1.15 --j DROP

iptables -A INPUT -p tcp -s 192.168.1.15  --dport 21 -j DROP

iptables -A INPUT -i lo --source 127.0.0.1 --j ACCEPT
```

Logs

Podemos indicar que la acción de una regla sea trazar el paquete

- `iptables -A FORWARD -s 10.0.3.3 -p tcp --dport 80 -j LOG`

- También se puede añadir una cadena para la traza y/o el nivel de importancia de la traza

```
iptables -A FORWARD -s 10.0.3.3 -j LOG --log-prefix "bla bla "\\
                                       --log-level 7
```

El *log-level* es un entero entre 0 y 7, según el estándar de syslogd / rsyslogd

Despues de ejecutarse esta acción, se evalúa la siguiente regla de la cadena

Adición de subcadenas

`-N --new-chain` Crea una nueva cadena

```
iptables --new_chain MI_CADENA
```

Especificación de flags TCP

Cuando en la condición se ha especificado `-p tcp`, se puede indicar también que flags deben estar activos y cuáles inactivos mediante la opción `--tcp-flags`

`--tcp-flags bit1,bit2,bit3 bit_activo1,bit_activo2`

Recibe dos argumentos, separados por un espacio. Cada argumento es una lista, separada por comas

- Primer argumento: cuáles son los flags a comprobar (en mayúsculas)
- Segundo argumento: cuáles de los flags comprobados tienen que estar activos. (El resto tiene que estar desactivado)

Ejemplo:

`-p tcp --tcp-flags SYN,RST,ACK,FIN SYN`

Se consideran los flags `SYN,RST,ACK y FIN`. El flag `SYN` tiene que estar activo, y los demás (`RST,ACK y FIN`), desactivados

- `--syn` equivale al ejemplo anterior:
  `--tcp-flags SYN,RST,ACK,FIN SYN`
- `! --syn` es la negación de esta condición, esto es
  `syn== 0 OR RST == 1 OR ACK==1 OR FIN==1`

iptables: condiciones

- Condiciones:

Protocolo	`-p <protocolo>`
Dirección IP	`-s <dirIP[/máscara]>`: dirección origen `-d <dirIP[/máscara]>`: dirección destino
Puerto	`--sport <puerto|puertoInicio:puertoFin>`: puerto origen `--dport <puerto|puertoInicio:puertoFin>`: puerto destino
Interfaz	`-i <interfaz>`: interfaz de entrada `-o <interfaz>`: interfaz de salida
Estado de la conexión[3]	`-m state --state <estado>` situación de un paquete con respecto a la conexión a la que pertenece. Estado: `INVALID`: no pertenece a una conexión existente `ESTABLISHED`: es parte de una conexión existente con paquetes en ambos sentidos `NEW`: es parte de una nueva conexión que aún no está establecida `RELATED`: está relacionado con otra conexión ya existente Ejemplo: un mensaje ICMP de error
Flags TCP	`--syn`: segmento SYN
	`--tcp-flag <flagsAComprobar> <flagsQueDebenEstarActivados>` flags: `SYN`, `FIN`, `ACK`, `RST`, `PSH`, `URG`, `ALL`, `NONE` Ejemplo: `-p tcp --tcp-flags ALL SYN,ACK` (deben estar activados `SYN`, `ACK` y desactivados `FIN`, `RST`, `PSH`, `URG`)

- La negación de una condición se expresa anteponiendo el caracter `!` al valor de la condición. Ejemplos:

`-p tcp --sport ! 80`	protocolo TCP y puerto origen distinto del 80
`-p ! icmp`	protocolo distinto de icmp

[3] "conexión" en sentido amplio

iptables: acciones

La acción se especifica empezando con `-j`

Tabla `filter`	`-j ACCEPT` se acepta el paquete
	`-j DROP` se descarta el paquete
	`-j REJECT [--reject-with <tipo>]` se rechaza el paquete, informando al origen con un ICMP, se puede especificar el tipo de ICMP, por defecto `icmp-port-unreachable`
Tabla `nat`	`-j SNAT --to-source [<dirIP>][:<puerto>]` Realiza *Source NAT* sobre los paquetes salientes (es decir, se cambia dirección IP y/o puerto origen). Sólo se puede realizar en la cadena `POSTROUTING`. **NOTA: esta regla hace que también se cambie automáticamente la dirección de destino del tráfico entrante de respuesta al saliente de la misma "conexión".**
	`-j DNAT --to-destination [<dirIP>][:<puerto>]` Realiza *Destination NAT* sobre los paquetes entrantes (es decir, se cambia dirección IP y/o puerto destino). Sólo se puede realizar en la cadena `PREROUTING`. Esta regla sólo es necesaria para "abrir puertos", es decir, permitir tráfico entrante nuevo.
Tabla `mangle`	`-j TTL --ttl-set <valorTTL>` Modifica el valor de TTL
Todas las tablas	`-j LOG` se guarda información de ese paquete en el fichero de log y **se continúa con la siguiente regla de la cadena**
	`-j <cadena-de-usuario>` Salta a aplicar al paquete las reglas de una cadena definida por el usuario. Si termina esa cadena sin cumplirse la condición de ninguna de sus reglas, continuará en la cadena desde la que se saltó, por la regla siguiente a la que hizo la llamada.

Si en una regla **no se especifica ninguna acción** (no hay cláusula `-j`), si se cumple la condición se actualizan los contadores de paquetes y bytes para la regla, pero **se continúa aplicando la seiguiente regla de la cadena para ese paquete**.

Seguimiento de "conexiones"

- Las conexiones o pseudoconexiones de las comunicaciones TCP, UDP, ICMP que atraviesan una máquina se pueden monitorizar a través del módulo `conntrack` de `iptables`.
- Para visualizar dichas conexiones hay que mostrar el contenido del fichero `/proc/net/ip_conntrack`:

```
r1:~# cat /proc/net/ip_conntrack
tcp      6 117 SYN_SENT src=192.168.1.6 dst=192.168.1.9 sport=32775 \
     dport=22 [UNREPLIED] src=192.168.1.9 dst=192.168.1.6 sport=22 \
     dport=32775 use=2
```

Filtro con estado

`-m <STATE>`

El valor de `<STATE>` puede ser

- NEW
  Equivale al SYN de TCP o a un primer paquete UDP
- ESTABLISHED
  Equivale al ACK de TCP o a una respuesta UDP
- RELATED
  Mensaje ICMP referido a tráfico que hemos enviado
- INVALID Mensaje ICMP de tráfico que no es nuestro, como respuesta ping que no hemos pedido

Más ejemplos

- Borrar las reglas y reiniciar los contadores:

```
iptables  -F
iptables  -Z
```

- Definir las políticas por defecto: Descartar cualquier cosa salvo paquetes de salida:

```
iptables  -P INPUT DROP
iptables  -P FORWARD DROP
iptables  -P OUTPUT ACCEPT
```

- Permitir el reenvío de todos los paquetes que se reciben en un router a través de una interfaz (eth0) para que se envíen a través de otra interfaz (eth1) (por ejemplo, permitir tráfico saliente de una organización):

```
iptables -t filter -A FORWARD -i eth0 -o eth1 -j ACCEPT
```

- Permitir el reenvío paquetes entrantes que pertenezcan a "conexiones" ya existentes:

```
iptables  -A FORWARD -i eth1 -o eth0
         -m state --state RELATED,ESTABLISHED -j ACCEPT
```

- Permitir el paso de segmentos TCP de establecimiento de conexión de entrada dirigidos a una dirección IP de la red interna (10.0.0.10) y a un puerto (80).

```
iptables  -A FORWARD -p tcp -d 10.0.0.10 --dport 80 --syn -j ACCEPT
```

- Permitir el paso de datagramas UDP de entrada dirigidos a una dirección IP de la red interna (10.0.0.10) y a un puerto (80).

```
iptables  -A FORWARD -p udp -d 10.0.0.10 --dport 80 -j ACCEPT
```

NAT

Si queremos hacer un NAT básico, sin especificar ningún parámetro especial, bastaría con

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

(Donde `eth0` es el interfaz que nos conecta a Internet)

Ejemplos de traducción de direcciones IP y puertos con iptables

- Borrar las reglas y reiniciar los contadores:

```
iptables -t nat -F
iptables -t nat -Z
```

- Modificar la dirección IP origen de los datagramas IP al salir de una red privada (10.0.0.0/24) a través de la interfaz de salida (eth0) de un router NAT. Todos los datagramas llevarán la dirección IP pública del router NAT (200.0.0.1):

```
iptables -t nat -A POSTROUTING -s 10.0.0.0/24 -o eth0
         -j SNAT --to-source 200.0.0.1
```

- Modificar la dirección IP destino y puerto destino de los segmentos TCP al entrar dentro de una red privada (10.0.0.0/24). Los segmentos van dirigidos inicialmente a la dirección IP del router NAT (200.0.0.1) y puerto 8080, recibiéndose en su interfaz (eth1). Antes de comprobar la tabla de encaminamiento (PREROUTING) se modificará su dirección IP destino a 10.0.0.10 y puerto destino 80.

```
iptables -t nat -A PREROUTING -p tcp -i eth1 -d 200.0.0.1
         --dport 8080 -j DNAT --to-destination 10.0.0.10:80
```

Política de las reglas

Una vez presentada la sintaxis de iptables, algunas cuestiones a tener en cuenta sobre la política de las reglas son

- Rechazo de conexiones entrantes
- Combinaciones de flags TCP inválidas
- Inseguridad relativa de UDP
- Número de puerto y seguridad
- Filtrado básico sobre origen y destino
- Tráfico con origen y destino en el mismo host
- Ejemplo: DNS
- Ejemplo: Correo electrónico
- Otros servicios

Rechazo de conexiones entrantes

Lo más habitual en internet es el modelo cliente-servidor, donde el cliente inicia la conexión y el servidor responde

- Buena parte de los ataques consisten en accesos no autorizados a servicios locales, explotando vulnerabilidades en servicios locales
- Es habitual configurar el cortafuegos para que de cierta libertad a sus clientes locales para que soliciten servicios del exterior (permitir conexiones salientes)
- Es habitual que las reglas sean mucho más estricta en lo relativo a clientes externos solicitando servicios a servidores internos (restringir las conexiones entrantes, las más peligrosas)

Para inhabilitar las conexiones TCP entrantes, basta aplicar cualquiera de de estas dos restricciones (son equivalentes)

1. Prohibir el tráfico entrante con el flag SYN activo y el resto de flags desactivo
2. Exigir que todo el tráfico entrante tenga el flag ACK activo

Recordatorio: apertura de conexión en TCP:

```
1                    --   SYN   -->
2                   <-- SYN/ACK --
3                    --   ACK   -->
```

Cierre de conexión en TCP:

```
4                     --  FIN  -->
5                    <--  ACK  --

6                    <--  FIN  --
7                     --  ACK  -->
```

- Prohibir el tráfico entrante con el flag SYN activo prohibe la entrada de un paquete como el (1)
- Exigir que todo tráfico entrante tenga el flag ACK activo prohibe la entrada de un paquete como el (6)
  La pérdida del paquete (6) no es grave: el cliente ya ha cerrado su parte de la conexión. Aunque no reciba el FIN, cuando pase cierto tiempo cerrará la conexión igualmente
  - En la práctica, esta situación no llega a darse, porque lo más común en unir los paquetes (5) y (6) en uno solo, incluyendo tanto FIN como ACK
  - E incluso en los casos en que (5) y (6) no puedan unirse (por ejemplo porque el servidor tenga que enviar datos después de que el cliente haya cerrado), lo normal es que el flag ACK esté siempre activo en todos y cada uno de los paquetes, excepto en el primero

Combinaciones de flags TCP inválidas

Ciertas combinaciones de los flags TCP jamás se darán en una secuencia legítima, por lo que pueden filtrarse segmentos con:

- Ningún flag activo
- SYN y FIN simultáneamente
- SYN y RST
- FIN y RST
- FIN sin ACK
- PUSH sin ACK
- URG sin ACK

```
Apertura correcta
 --   SYN   -->
<-- SYN/ACK --
 --   ACK   -->
```

```
Cierre correcto
 --  FIN  -->
<--  ACK  --


<--  FIN  --
 --  ACK  -->
```

Inseguridad relativa de UDP

Comparemos la seguridad de UDP frente a la de TCP:
TCP es inseguro

- A menos que emplee IPsec, no podemos saber con certeza quién envía un segmento
- Un atacante puede robar la sesión de A, silenciando a A, y enviando tráfico TCP con los números de secuencia adecuados

TCP se limita a comprobar

- Que las conexiones se abran correctamente, distinguiendo el extremo que estaba escuchando del extremo que abre la conexión
- Que cada segmento enviado pertenezca a una conexión abierta
- Que no haya segmentos repetidos
- Que no *sobren* segmentos

Pero UDP es mucho más inseguro, carece de cualquier control de cualquier clase

- Cada paquete es independiente de los demás
- Para suplantar la identidad de una víctima, basta poner su dirección como origen del datagrama
- Si faltan paquetes (del orígen legítimo), es imposible detectarlo
- Si hay paquetes adicionales (añadidos por el atacante), es imposible detectarlo

Por su insegurdad relativa, es muy habitual filtrar por completo todo el tráfico UDP, lo que dificulta mucho el uso de servicios que lo necesitan, p.e. streaming de audio y vídeo, juegos, NFS, DNS, etc

- El *descontrol* de UDP lo puede suplir en todo o en parte el protocolo ubicado encima de UDP, (ya sea estándar o propio de una aplicación concreta)
  Pero esto queda fuera del ámbito del cortafuegos

Número de puerto y seguridad

Un criterio fundamental para filtrar los servicios TCP y UDP es considerar el número de puerto
¿Qué puerto corresponde a cada servicio?

- Muchos puertos, por debajo y por encima de 1024, están asignados oficialmente por IANA (*Internet Assigned Numbers Authority*)
  Lo mejor es consultar directamente el web del IANA, otras fuentes pueden estar desfasadas
- Hay servicios que siempre usan el mismo puerto, pero sin reconocimiento oficial
- Algunos protocolos antiguos como IRC y FTP pueden usar cualquier puerto

Problema fundamental

- No hay ninguna garantía garantía de que el tráfico que circula por cierto puerto realmente use el protocolo asignado a ese puerto
  Cliente y servidor pueden acordar romper el convenio oficial/habitual

Para conseguir alguna certidumbre, es necesario analizar dinámicamente el tráfico

- Con el consiguiente coste
- Y también sin garantía, pueso que el atacante puede *ofuscar* el tráfico, aparentando emplear un protocolo distinto

Filtrado básico sobre origen y destino

Un cortafuegos completo debería incluir reglas para filtrar:

- Tráfico entrante cuya dirección de origen sea del interior (si el cortafuegos es el límite entre internet y una intranet, serán direcciones privadas)
- Tráfico saliente cuya dirección de origen sea del exterior (si el cortafuegos es el límite entre internet y una intranet, serán direcciones públicas)
- Tráfico con dirección origen broadcast o multicast, salvo que tal tráfico esté permitido y las direcciones sean correctas
- Tráfico con encaminamiento en origen o con opciones de IP
- Tráfico de tipo ICMP pero excesivamente grande (más de unos pocos kilobytes)

Tráfico con origen y destino en el mismo host

Cualquier host conectado a internet necesita el interfaz de loopbak con la dirección 127.0.0.1 y otro interfaz adicional con su propia dirección
Ejemplo:

- Interfaz `lo` , dirección 127.0.0.1
- Interfaz `eth0` , dirección 193.147.71.64

Todo el trafico desde la máquina hasta la propia máquina circulará por el interfaz `lo`, no importa que la dirección en el datagrama sea 193.147.71.64, esos paquetes se comportan como si tuvieran la dirección 127.0.0.1

Por tanto:

- Es necesario permitir el tráfico del interfaz lo
- Deberíamos filtrar el tráfico que entre por eth0 pero con dirección de origen 193.147.71.64 o 127.0.0.1
- Deberíamos filtrar el tráfico que entre por eth0 con dirección de destino 127.0.0.1
- Deberíamos filtrar todo el tráfico que salga al exterior pero con dirección de origen 127.0.0.1

Ejemplo: DNS

El DNS es un objetivo muy valioso de cualquier atacante

- Manipular el DNS permite engañar a las víctimas y redirigirlas a máquinas del atacante
- El mapa de DNS de una red es una información muy sensible

El protocolo DNS incluye tráfico cliente-servidor y entre servidores

- Las consultas cliente-servidor normalmente se hacen mediante UDP, al puerto 53. A menos que la respuesta no quepa en un datagrama UDP, entonces se usa TCP
- El tráfico entre servidor primario y servidores secundarios va sobre TCP

Para que funcione el DNS hay que permitir

- Consultas de un cliente local por UDP

```
ip_interna:cualq_alto   -->  servidor_externo_dns:53    UDP
servidor_externo_dns:53   --> ip_interna:cualq_alto   UDP
```

(Suponiendo un servidor externo)

- Consultas de un cliente local por TCP

```
ip_interna:cualq_alto  -->  servidor_externo_dns:53   TCP
servidor_externo_dns:53  --> ip_interna:cualq_alto   TCP, flag ACK
```

(Suponiendo un servidor externo)

- Consultas un servidor local por UDP

```
mi_servidor:53   -->  cualquier_host:53   UDP
cualquier_host:53   --> mi_servidor:53   UDP
```

`cualq_alto`: cualquier puerto alto (1024-65535)

- Petición *zone transfer* de servidor local a servidor remoto

```
mi_servidor:cualq_alto  -->  servidor_primario:53   TCP
servidor_primario:53  --> mi_servidor:cualq_alto TCP, flag ACK
```

- Consultas de un cliente remoto por UDP

```
cualquier_ip:cualq_alto   -->  mi_servidor:53   UDP
mi_servidor:53  --> cualquier_ip:cualq_alto   UDP
```

- Consultas de un cliente remoto por TCP

```
cualquier_ip:cualq_alto   -->  mi_servidor:53  TCP
mi_servidor:53  --> cualquier_ip:cualq_alto TCP
```

- Petición *zone transfer* de servidor remoto (autorizado)

```
servidor_remoto:cualq_alto   --> mi_servidor:53   TCP
mi_servidor:53  --> servidor_remoto:cualq_alto  TCP
```

Ejemplo: Correo electrónico

- Los clientes envían mensajes a los servidores mediante SMTP (TCP puerto 25)
- Entre servidores se emplea SMTP tanto para recibir como para enviar
- Los clientes emplean diversos protocolos para la recepción
  - POP3 (TCP 110, TCP 995 si es sobre SSL)
  - IMAP (TCP 143, TCP 993 si es sobre SSL)
  - Microsoft Exchange Server TCP, puertos 389 o 390 (LDAP), 25 o 465 o 691 (SMTP), 636 (LDAP sobre SSL), 379 (Site Replication Service), 3268 o 3269 (*active directory global catalog*), 143 o 993 (IMAP), 119 o 563 (NNTP), 80 o 443 (HTTP), 102 (X.400) 135 (MS-RPC), 522 (ULS, *User Locator Service*).
    Puertos 53 TCP y UDP (DNS)
  - IBM Lotus Notes TCP 1352 (*NRPC, Notes Remote Procedure Call*)
- En el caso de clientes de correo sobre web, el trafico es https convencional

Para que funcione el correo hay que permitir

- Envío de correo de un cliente local o un servidor local a un servidor remoto

```
cualq_local:cualq_alto  --> cualq_remoto:25  TCP
cualq_remoto:25 --> cualq_local:cualq_alto TCP, flag ACK
```

- Servidor local recibe correo de servidor remoto

```
cualq_remoto:cualq_alto --> mi_servidor:25 TCP
mi_servidor:25 TCP --> cualquier_remoto:cualq_alto TCP, flag ACK
```

- Cliente local pide correo POP3 a servidor remoto

```
cualq_local:cualq_alto --> cualq_remoto:110 TCP
cualq_remoto:110 --> cualq_local:cualq_alto TCP, flag ACK
```

(Suponiendo que permitamos el uso de cualquier servidor remoto)

- Cliente remoto pide correo a servidor local mediante POP3

```
cualq_remoto:cualq_alto --> servidor_local:110 TCP
servidor_local:110 --> cualq_remoto:cualq_alto TCP, flag ACK
```

- Cliente local pide correo IMAP a servidor remoto

```
cualq_local:cualq_alto --> cualq_remoto:143 TCP
cualq_remoto:143 --> cualq_local:cualq_alto TCP, flag ACK
```

- Cliente remoto pide correo a servidor local mediante IMAP

```
cualq_remoto:cualq_alto --> servidor_local:143 TCP
servidor_local:143 --> cualq_remoto:cualq_alto TCP, flag ACK
```

Otros servicios

- Las conexiones HTTP se hacen mediante TCP, desde puertos altos del cliente hasta el puerto 80 del servidor.
  En HTTPS se usa el puerto 443
- *ping* no es un servicio basado en TCP ni UDP, son mensajes ICMP.
  traceroute está basado en *ping*, pero sí depende de UDP, puerto 33434
- DHCP emplea mensajes broadcast y los puertos 67 y 68 UDP
- NTP, *Network Time Protocol* funciona sobre el puerto 123 UDP

Naturalmente, el aumento de la complejidad de los servicios exige el empleo de más puertos

Ejemplo: Windows Live Messenger 8.1 en Windows Vista

- Iniciar sesión en Messenger: TCP 80, 443, 1863
- Detección de la red: TCP 7001 UDP 9, 7001
- Audio: TCP 80, 443, 1863 TCP/UDP 30000 - 65535
- Chat de video: TCP 80 TCP/UDP 5000 - 65535
- Transferencia de archivos: TCP 443, 1863 TCP/UDP 1025 - 65535
- Carpetas compartidas: TCP 1863 TCP/UDP 1025 – 65535
- Pizarra:TCP 1503
- Asistencia remota: TCP 3389 TCP/UDP 49152 – 65535
- Windows Live Call: TCP 443, 5061 UDP 5004 - 65525
- Juegos: TCP 80, 443, 1863 TCP/UDP 1025 - 65535

Referencias

Bibliografía:

- J. Michael Stewart
  Network Security, Firewalls, and VPNs
- Elizabeth D. Zwicky, Simon Cooper, D. Brent Chapman
  Building Internet Firewalls, 2nd Edition
- Steve Suehring; Robert Ziegler
  Linux Firewalls, Third Edition. Ed. Novell, 2005

Tutoriales:

- Pello Altadill. iptables manual práctico
- linux 2.4 filtrado de paquetes COMO teve Suehring; Robert Ziegler