Comandos de IPTables

Comando	**-A**, **--append**
Ejemplo	**iptables -A INPUT ...**
Explicación	Agrega una regla a una cadena especificada

Comando	**-D**, **--delete**
Ejemplo	**iptables -D INPUT --dport 80 -j DROP**, **iptables -D INPUT 1**
Explicación	Borra una regla de una cadena especificada. Se le puede pasar como argumento toda la regla la cual deberá coincidir exactamente con la existente o el número de linea que ocupa en la cadena.

Comando	**-R**, **--replace**
Ejemplo	**iptables -R INPUT 1 -s 192.168.0.1 -j DROP**
Explicación	Reemplaza una regla. Se le pasa como argumento el numero de linea dentro de la cadena y la nueva regla

Comando	**-I**, **--insert**
Ejemplo	**iptables -I INPUT 1 --dport 80 -j ACCEPT**
Explicación	Inserta una regla en el lugar de la cadena que le pasemos como argumento. Recordemos que in IPTables es de suma importancia el orden en que están las regla dentro de las cadenas.

Comando	**-L**, **--list**
Ejemplo	**iptables -L INPUT**
Explicación	Muestra las reglas que contiene la cadena que le pasemos como argumento.

Comando	**-F**, **--flush**
Ejemplo	**iptables -F INPUT**
Explicación	Borra todas las reglas de una cadena.

Comando	**-Z**, **--zero**
Ejemplo	**iptables -Z INPUT**
Explicación	Pone en cero todos los contadores de una determinada cadena

Comando	**-N**, **--new-chain**
Ejemplo	**iptables -N allowed**
Explicación	Permite al usuario crear su propia cadena. En este ejemplo la cadena se llamará “allowed”

Comando	**-X**, **--delete-chain**
Ejemplo	**iptables -X allowed**
Explicación	Borra la cadena especificada. Si se escribe -X solamente, borrará todas las cadenas creadas en esa tabla

Comando	**-P**, **--policy**
Ejemplo	**iptables -P INPUT DROP**
Explicación	Explicita al Kernel que hacer con los paquetes que no coinciden con ninguna regla.

Comando	**-E**, **--rename-chain**
Ejemplo	**iptables -E allowed disallowed**
Explicación	Cambia el nombre de una cadena.

Condiciones de IPTables (IPTables Matches)

Una regla pude tener varias condiciones. Existen cinco categorías de condiciones:

- *Condiciones genericas: Pueden ser usados en cualquier regla*
- *Condiciones TCP: Solo pueden ser usados en paquetes TCP*
- *Condiciones UDP: Solo pueden ser usados en paquetes UDP*
- *Condiciones ICMP: Solo pueden ser usados en paquetes ICMP*
- *Otras condiciones: Como condiciones de estado, etc.*

Condiciones genéricas

Condición	**-p**, **--protocol**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -A INPUT -p tcp**
Explicación	El paquete debe ser del protocolo especificado después de **-p**. Cada protocolo se corresponde a un entero ej: ICMP es equivalente a 0. Si usamos la opción -p ALL son los tres (ICMP, TCP; UDP). Todos lo protocolos soportado están en */etc/protocols*

Condición	**-s**, **--src**, **--source**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -A INPUT -s 192.168.1.1**
Explicación	Permite seleccionar paquetes en función de la dirección origen. Podemos especificar un host o un rango de IP's agregando la máscara de subred. (ej: 192.168.0.0/24). Si queremos seleccionar todos los paquetes que no vengan de un determinado host o red podemos usar la negación (ej: iptables -A INPUT -s **!** 192.168.1.0/24)

Condición	**-d**, **--dst**, **--destination**
Kernel	2.3, 2.4, 2.5 and 2.6

Ejemplo	**iptables -A INPUT -d 192.168.1.1**
Explicación	IDEM anterior pero con dirección o rango de direcciones de destino

Condición	**-i**, **--in-interface**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -A INPUT -i eth0**
Explicación	Permite seleccionar paquetes que vienen de una determinada interface. Esta opción solo es legal en las siguientes cadenas **INPUT**, **FORWARD** y **PREROUTING,** y retornaría un error en cualquier otra. Pude usarse **eth+**, por ejemplo para especificar que todas las interfaces ethernet. También puede usarse la negación (ej: **-i ! eth0** sería todas las interfaces excepto la eth0).

Condición	**-o**, **--out-interface**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -A FORWARD -o eth0**
Explicación	Permite seleccionar paquetes que salen de una determinda interface. Esta opción solo es legal en las siguientes cadenas OUTPUT, FORWARD y POSTROUTING y retornaría un error en cualquier otra. Pude usarse **eth+**, por ejemplo para especificar que todas las interfaces ethernet. También puede usarse la negación (ej: **-i ! eth0** sería todas las interfaces excepto la eth0).

Condición	**-f**, **--fragment**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -A INPUT -f**
Explicación	Permite seleccionar las segundas o terceras partes de paquetes fragmentados. Estos paquetes son peligrosos. Como nuestro interés es determinar la primera parte podemos usar esta opción así: iptables -A INPUT **!** -f

Condiciones TCP

Estas condiciones son específicas del protocolo TCP, por lo tanto solo se pueden usar con paquetes TCP. Es por ello que previo a especificar cualquiera de estas condiciones se debe especificar la condición **-p tcp.**

Condición	**--sport**, **--source-port**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -A INPUT -p tcp --sport 22** **--source-port 22:80** **--source-port :80** (puerto 0 a 80) **--source-port ! 6**:**1024** (todos los puertos menos el rango de 6 a 1024)
Explicación	Permite seleccionar o excluir paquetes de un determinado puerto o rango de puertos tcp origen. El puerto puede escribirse también con el nombre del servicio (en linux estan en /etc/services), pero esto supone una pequeña sobrecarga que se hace considerable con gran cantidad de reglas.

Condición	**--dport**, **--destination-port**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -A INPUT -p tcp --dport 22**
Explicación	IDEM pero con puerto tcp destino

Condiciones UDP

Para seleccionar paquetes a través de los puertos origen y destino, se hace de la misma forma que con TCP, la diferencia que se debe anteponer **-p udp** en lugar de -p tcp

Condiciones ICMP

Condición	**--icmp-type**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -A INPUT -p icmp --icmp-type 8**
Explicación	Evita el tipo de paquetes icmp que especifiquemos. El del ejemplo es peligroso ya que puede redirigir a lugares peligrosos.

Condiciones explicitas

Las condiciones explicitas se deben cargar con **-m** o –**match.**

Condiciones AH/ESP

Condición	**--ahspi**
Kernel	2.5 and 2.6
Ejemplo	**iptables -A INPUT -p 51 -m ah --ahspi 500**
Explicación	Permite seleccionar paquetes del protocolo AH de IPSEC en base a su SPI (Security Parameter Index). El SPI es usado en conjunción con la dirección destino, la dirección origen y la clave secreta para establecer una asociación segura. Se puede seleccionar un rango de SPI's haciendo un intervalo.

Condición	**--espspi**
Kernel	2.5 and 2.6
Ejemplo	**iptables -A INPUT -p 50 -m esp --espspi 500**
Explicación	IDEM pero con el protocolo ESP de IPSEC.

Condiciones de rangos IPS

Condición	**--src-range**
Kernel	2.4, 2.5 and 2.6
Ejemplo	**iptables -A INPUT -p tcp -m iprange --src-range 192.168.1.13-192.168.2.19**
Explicación	Selecciona paquetes de ese rango de IPS origen, pudiendo también usarse en forma invertida (ej: **iptables -A INPUT -p tcp -m iprange ! --src-range 192.168.1.13-192.168.2.19**)

Condición	**--dst-range**
Kernel	2.4, 2.5 and 2.6
Ejemplo	**iptables -A INPUT -p tcp -m iprange --dst-range 192.168.1.13-192.168.2.19**
Explicación	IDEM pero con IPS destino

Condiciones MAC

Condición	**--mac-source**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -A INPUT -m mac --mac-source 00:00:00:00:00:01**
Explicación	Permite seleccionar paquetes en función a su dirección MAC origen. Esta regla solo puede ser usada en las cadenas PREROUTING, FORWARD and INPUT.

Condiciones de Marca

La marca es un entero de 32 bits que se le puede realizar a ciertos paquetes para luego poder tomar decisiones con los mismos de acuerdo a estas.
La marca es un campo especial mantenido en el kernel que es asociado a un conjunto de paquetes y solo es valido en el transcurso en que los paquetes están dentro de la computadora.

Condición	**--mark**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -t mangle -A INPUT -m mark --mark 1**
Explicación	Permite seleccionar paquetes en función a marcas previamente realizadas dentro de la misma computadora.

Condiciones de dueño

Permite seleccionar paquetes en de acuerdo al UID, GID, PID o SID que los haya creado.
La condición de dueño (**owner** match) solo se puede usar en la cadena OUTPUT por rasones obvias.

Condición	**--uid-owner**
Kernel	2.3, 2.4, 2.5 and 2.6

Ejemplo	**iptables -A OUTPUT -m owner --uid-owner 500**
Explicación	Permite seleccionar paquetes creados por un determinado usuario. Un uso posible podría ser bloquear el usuario `http` para mandar paquetes.

Condición	**--gid-owner**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -A OUTPUT -m owner --gid-owner 0**
Explicación	Permite seleccionar paquetes de un determinado grupo en función de su `Group ID` (GID). Puede usarse para bloquear la salida a internet a todos los usuarios excepto un grupo o solo permitirles a los miembros del grupo `http` crear paquetes que salen por el por el puerto `http`.

Condición	**--pid-owner**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -A OUTPUT -m owner --pid-owner 78**
Explicación	Permites seleccionar paquetes que un determinado proceso creó en función a su Process ID (PID). Se complica su uso si el proceso posee muchos hilos y en este caso se debe manejar con un script que lea los PID del comando **ps**.

Condición	**--sid-owner**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -A OUTPUT -m owner --sid-owner 100**
Explicación	Permite seleccionar paquetes en base a una sesión de un determinado programa en cuestión a través del Session ID. El valor del SID de un proceso, es el PID del proceso y todos los procesos resultantes de este tendrán el mismo SID.

Condiciones de estado

Condición	**--state**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -A INPUT -m state --state RELATED,ESTABLISHED**
Explicación	Nos permite seleccionar paquetes de acuerdo al estado de su conexión Existen cuatro tipos de estados en los que puede estar una conexión y sus paquetes asociados. Estos son: ● **ESTABLISHED** Significa que el paquete es parte de una conexión existente, que manda paquetes en ambas direcciones. ● **NEW** Significa que el paquete iniciará una conexión nueva o es asociado a una conexión a la que todavía no se le ha visto paquetes en ambas direcciones. ● **RELATED** Significa que el paquete esta creando una nueva conexión asociada a otra existente conocida. (Ejs: Una transferencia FTP iniciada por una TCP o un paquete ICMP como resultado de un error en TCP o UDP). ● **INVALID** Significa que el paquete no se puede asociar a ninguna conexión y lo que generalmente se hace es un DROP

Condiciones de TOS

Condición	**--tos**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -A INPUT -p tcp -m tos --tos 0x16**
Explicación	Permite seleccionar paquetes en función al campo **TOS** de la cabecera IP Esto se puede usar conjuntamente con **iproute2** y las marcas previamente descriptas para hacer un ruteo avanzado (Ej: Se podría marcar paquetes con un determinado TOS y luego con iproute2 routerlos por una interfaz mas veloz.) Los type of services pueden escribirse como números en hexadecimal o escribirse como nombres sacados de **iptables -m tos -h**

Condiciones TTL

Condición	**--ttl**
Kernel	2.3, 2.4, 2.5 and 2.6
Ejemplo	**iptables -A OUTPUT -m ttl --ttl 60**
Explicación	Permite seleccionar paquetes de un determinado tiempo de vida. Esto se usa en muchos caso para debuguear el funcionamiento de una LAN

Targets y Jumps

Los target y jumps deciden que se hará con un paquete que cumpla todas las condiciones que le hayamos especificado.

Jumps

Con **-j** podemos hacer saltos a una cadena, que debe previamente haber sido creada si no existe.

ej:
iptables -A INPUT -p tcp -j tcp_packets

*Podríamos estar en la cadena **INPUT** y si el paquete cumple por ejemplo la condición **-p tcp** saltar a una cadena previamente creada por nosotros donde hagamos un filtrado refinado de los distintos paquetes TCP. En este caso se pueden dar dos situaciones: que el paquete se aceptado dentro de la cadena tcp_packets o bien que alcance el final de la misma sin haber encontrado ninguna condición que se iguale a la del paquete. En este caso el mismo volverá al la cadena desde la que vino a la próxima linea donde la abandonó..*

Targets

La opción -j también puede usarse para especificar una acción con un paquete que cumple todas condiciones. Las dos acciones mas comunes son: **DROP** or **ACCEPT** (eliminarlo o aceptarlo

respectivamente), que paran el paquete evitando que siga recorriendo el resto de la cadena y las restantes si aún quedan.
Existen otro tipo de targets que modifican el paquete y el mismo sigue recorriendo las reglas que quedan como por ejemplo: **TTL TOS.**

Target ACCEPT

Un vez que el paquete cumple la todas las condiciones que hayamos especificado, si usamos la opción -j ACCEPT el mismo será aceptado y no recorrerá mas reglas de la cadena actual y de ninguna otra de esa misma tabla. Esto es importante aclararlo ya que el paquete puede ser eliminado en otra tabla.
La forma en que usamos este target es **-j ACCEPT**.

Target DROP

Un vez que el paquete cumple la todas las condiciones que hayamos especificado, si usamos la opción -j DROP el mismo será bloqueado y no sera procesado en ningún otra cadena ni tabla.
Esto puede traer como inconveniente dejar sockets muertos y en muchos casos conviene usar el target REJECT

Target DNAT (Destination Network Address Translation)

Este target es usado para reescribir el IP de destino. Por lo tanto, si un paquete cumple todas las condiciones y este es el target de la regla, este paquete y todos los subsecuentes del mismo flujo, le serán reescrita su dirección de destino y por lo tanto ruteados a la correspondiente red o host.
Esto es muy útil cuando tenemos un servidor dentro de una LAN con una IP no publicable, ya que le decimos al firewall que todos los paquetes que reciba él, en el puerto que escucha el servidor, los reenvie a el IP real y no publicable del servidor. De esta forma desde afuera de la LAN (internet), pareciera que el servidor tiene como IP, el valor de IP que en realidad tiene nuestro firewall.
Cabe destacar que se puede "dnatear" a un rango de IPS y el firewall tomará una decisión aleatoria a la hora de decidir a quien enviarle el flujo de paquetes.

El target DNAT solo se puede usar dentro de las cadenas **PREROUTING** y **OUTPUT** de la tabla NAT**.**

Opción	**--to-destination**
Ejemplo	**iptables -t nat -A PREROUTING -p tcp -d 15.45.23.67 --dport 80 -j DNAT --to-destination 192.168.1.1-192.168.1.10**
Explicación	Reescribe la dirección de destino de un paquete y todos los subsecuentes del mismo flujo. también es posible reescribir el puerto de destino a otro puerto o a un rango ej: **--to-destination 192.168.1.1:80** ó **--to-destination 192.168.1.1:80-100**

Un buen ejemplo para entenderlo (Esta en ingles)

Since **DNAT** requires quite a lot of work to work properly, I have decided to add a larger explanation on how to work with it. Let's take a brief example on how things would be done normally. We want to publish our website via our Internet connection. We only have one IP address, and the HTTP server is located on our internal network. Our firewall has the external IP address **$INET_IP**, and our HTTP server has the internal IP address **$HTTP_IP** and finally the firewall has the internal IP address **$LAN_IP**. The first thing to do is to add the following simple rule to the PREROUTING chain in the nat table:

```
iptables -t nat -A PREROUTING --dst $INET_IP -p tcp --dport 80 -j DNAT \
```

```
--to-destination $HTTP_IP
```

Now, all packets from the Internet going to port 80 on our firewall are redirected (or **DNAT**'ed) to our internal HTTP server. If you test this from the Internet, everything should work just perfect. So, what happens if you try connecting from a host on the same local network as the HTTP server? It will simply not work. This is a problem with routing really. We start out by dissecting what happens in a normal case. The external box has IP address **$EXT_BOX**, to maintain readability.

1. Packet leaves the connecting host going to **$INET_IP** and source **$EXT_BOX**.
2. Packet reaches the firewall.
3. Firewall **DNAT**'s the packet and runs the packet through all different chains etcetera.
4. Packet leaves the firewall and travels to the $HTTP_IP.
5. Packet reaches the HTTP server, and the HTTP box replies back through the firewall, if that is the box that the routing database has entered as the gateway for **$EXT_BOX**. Normally, this would be the default gateway of the HTTP server. (Recordemos que el paquete llega al server con destino $HTTP y origen $EXT_BOX. El server responde con destino $EXT_BOX y origen $HTTP. Como el firewall suele generalmente ser el gateway del server, el paquete de respuesta llega de vuelta al firewall)
6. Firewall Un-**DNAT**'s the packet again, so the packet looks as if it was replied to from the firewall itself.
7. Reply packet travels as usual back to the client **$EXT_BOX**.

Now, we will consider what happens if the packet was instead generated by a client on the same network as the HTTP server itself. The client has the IP address **$LAN_BOX**, while the rest of the machines maintain the same settings.

1. Packet leaves **$LAN_BOX** to **$INET_IP**.
2. The packet reaches the firewall.
3. The packet gets **DNAT**'ed, and all other required actions are taken, however, the packet is not **SNAT**'ed, so the same source IP address is used on the packet.
4. The packet leaves the firewall and reaches the HTTP server.
5. The HTTP server tries to respond to the packet, and sees in the routing databases that the packet came from a local box on the same network, and hence tries to send the packet directly to the original source IP address (which now becomes the destination IP address).
6. The packet reaches the client, and the client gets confused since the return packet does not come from the host that it sent the original request to. Hence, the client drops the reply packet, and waits for the "real" reply.

The simple solution to this problem is to **SNAT** all packets entering the firewall and leaving for a host or IP that we know we do **DNAT** to. For example, consider the above rule. We SNAT the packets entering our firewall that are destined for **$HTTP_IP** port 80 so that they look as if they came from **$LAN_IP**. This will force the HTTP server to send the packets back to our firewall, which Un-**DNAT**'s the packets and sends them on to the client. The rule would look something like this:

```
iptables -t nat -A POSTROUTING -p tcp --dst $HTTP_IP --dport 80 -j SNAT \
--to-source $LAN_IP
```

Remember that the POSTROUTING chain is processed last of the chains, and hence the packet will already be **DNAT**'ed once it reaches that specific chain. This is the reason that we match the packets based on the internal address.

You think this should be enough by now, and it really is, unless considering one final aspect to this whole scenario. What if the firewall itself tries to access the HTTP server, where will it go? As it looks now, it will unfortunately try to get to its own HTTP server, and not the server residing on **$HTTP_IP**. To get around this, we need to add a **DNAT** rule in the OUTPUT chain as well. Following the above example, this should look something like the following:

```
iptables -t nat -A OUTPUT --dst $INET_IP -p tcp --dport 80 -j DNAT \
--to-destination $HTTP_IP
```

Adding this final rule should get everything up and running. All separate networks that do not sit on the same net as the HTTP server will run smoothly, all hosts on the same network as the HTTP server will be able to connect and finally, the firewall will be able to do proper connections as well. Now everything works and no problems should arise.

Target SNAT

El target SNAT es usado para hacer reescribir la dirección de destino. Esto es muy usado cuando queremos que varios host compartan una sola conexión. De esta forma, todos salen a Internet con la misma dirección de destino. Cabe aclarar que para hacer esto es necesario activar el bit de forwardeo en le kernel del host que va a compartir la conexión a internet.
El target SNAT solo es valido dentro de la tabla **nat** en la cadena POSTROUTING y puede usarse con la siguiente opción:

Opción	**--to-source**
Ejemplo	**iptables -t nat -A POSTROUTING -p tcp -o eth0 -j SNAT --to-source 194.236.50.155-194.236.50.160:1024-32000**
Explicación	Reescribe la dirección de destino de los paquetes que cumplan con la condiciónes especificadas, a alguna de las direcciones y puertos seleccionados, dentro de los rangos especificados, en forma aleatoria.

Target MASQUERADE

El target **MASQUERADE** es usado de la misma forma que SNAT pero no requiere –**to-source** opción ya que el mismo fue hecho especialmente para trabajar con conexiones donde se recibe un IP dinámico. (Ej: Dial-up, ADSL, etc).
En el caso que tengamos un IP estático es mejor trabajar con SNAT ya que MASQUERADE genera mas carga en el procesador.
Cuando se hace MASQUERADE en una conexión estaremos fijando como dirección de destino la dirección IP que posee la interfaz por donde están saliendo los paquetes. Por lo tanto usar MASQUERADE solo es válido en la cadena POSTROUTING. El mismo puede usarse con la siguiente opción ya explicada en el target SNAT.

Opción	**--to-ports**
Ejemplo	**iptables -t nat -A POSTROUTING -p TCP -j MASQUERADE --to-ports 1024-31000**

Target MARK

Este es usado para asociar paquetes a marcas. El mismo solo es valido en la tabla **Mangle.**
Las marcas son usadas para poder hacer un ruteo mas avanzado y organizado. Cabe aclarar que la marca solo es mantenida dentro del host y no se debe esperar que la misma permanezca en un paquete enviado a otro host, ya que es una asociación que se hace dentro del kernel.

Opción	**--set-mark**
Ejemplo	**iptables -t mangle -A PREROUTING -p tcp --dport 22 -j MARK --set-mark 2**
Explicación	A los paquetes que cumplen las condiciones especificadas se les setea un determinado valor de marca. El valor puede ser un número de 32 bits (tenemos para tirar marcas al techo!!)

Target TOS

El target **TOS** se usa dentro de la tabla MANGLE solo puede tomar una opción y es la descripta a continuación

Opción	**--set-tos**
Ejemplo	**iptables -t mangle -A PREROUTING -p TCP --dport 22 -j TOS --set-tos 0x10**
Explicación	Setea el valor de **TOS**, el mismo se pasa en numeración hexadecimal

Target TTL

El target **TTL** se usa dentro de la tabla MANGLE y sus opciones son:

Opción	**--ttl-set**
Ejemplo	**iptables -t mangle -A PREROUTING -i eth0 -j TTL --ttl-set 64**
Explicación	Setea un determinado valor de TTL en los paquetes que cumplen la condición especificada.

Opción	**--ttl-dec**
Ejemplo	**iptables -t mangle -A PREROUTING -i eth0 -j TTL --ttl-dec 1**
Explicación	Decrementa un determinado valor de TTL en los paquetes que cumplen la condición especificada.

Opción	**--ttl-inc**
Ejemplo	**iptables -t mangle -A PREROUTING -i eth0 -j TTL --ttl-inc 1**
Explicación	Incrementa un determinado valor de TTL en los paquetes que cumplen alguna condición.