Cuando configuramos un cortafuegos y queremos impedir determinado tráfico, normalmente tenemos dos formas principales de hacerlo:

- Descartar silenciosamente el paquete con **DROP** o
- Rechazarlo explícitamente con **REJECT**.

Aunque ambas acciones impiden que la conexión llegue a establecerse, desde el punto de vista del equipo que intenta comunicarse con nosotros el comportamiento es completamente diferente. Esto afecta a los tiempos de espera, al funcionamiento de las aplicaciones, a los escaneos de puertos y, en determinados casos, incluso al consumo de recursos.

Qué hace DROP

Cuando aplicamos DROP, el cortafuegos simplemente **elimina el paquete sin responder absolutamente nada al equipo que lo ha enviado**. Podemos imaginarlo como si el paquete nunca hubiese llegado a su destino. Por ejemplo, con iptables podríamos tener una regla como esta:

```
iptables -A INPUT -p tcp --dport 22 -j DROP
```

Si alguien intenta establecer una conexión TCP contra nuestro puerto 22, recibiremos el paquete SYN pero el cortafuegos lo descartará. El cliente no recibe:

- un SYN/ACK;
- un RST;
- un mensaje ICMP;
- ni ninguna otra indicación de que hemos rechazado la conexión.

El cliente solamente sabe que **ha enviado un paquete y nadie ha contestado**. Esto es importante porque TCP no interpreta inmediatamente la ausencia de respuesta como un rechazo. En su lugar, el sistema operativo realiza retransmisiones y espera hasta que finalmente vence el timeout. Una conexión bloqueada mediante DROP puede, por tanto, tardar varios segundos o incluso bastante más tiempo en devolver un error a la aplicación.

Qué hace REJECT

REJECT también bloquea el tráfico, pero **informa inmediatamente al equipo remoto de que la comunicación ha sido rechazada**. Dependiendo del protocolo y de cómo configuremos la regla, el cortafuegos puede responder mediante TCP RST o mediante un mensaje ICMP indicando que el destino, puerto o comunicación no está disponible. Por ejemplo:

```
iptables -A INPUT -p tcp --dport 22 -j REJECT --reject-with tcp-reset
```

En este caso, cuando alguien intenta establecer una conexión TCP contra el puerto 22, el cortafuegos responde inmediatamente mediante un paquete RST. El equipo remoto sabe entonces que la conexión no puede establecerse y puede finalizar el intento prácticamente de inmediato. Con nftables podríamos conseguir un comportamiento equivalente con:

```
nft add rule inet filter input tcp dport 22 reject with tcp reset
```

Para otros tipos de tráfico podemos utilizar respuestas ICMP:

```
nft add rule inet filter input reject with icmpx type admin-prohibited
```

La diferencia fundamental

La diferencia puede resumirse de esta forma:

ACCIÓN	DROP	REJECT
Bloquea el paquete	Sí	Sí
Responde al origen	No	Sí
El cliente conoce inmediatamente el rechazo	No	Sí
Puede provocar retransmisiones	Sí	Normalmente no
Tiempo hasta mostrar error	Generalmente largo	Generalmente inmediato
Genera tráfico de respuesta	No	Sí

Qué ocurre con TCP

Las consecuencias son especialmente fáciles de observar cuando trabajamos con TCP. Para iniciar una conexión TCP, el cliente envía un paquete SYN y espera una respuesta. Si el cortafuegos utiliza DROP, ocurre algo parecido a esto:

```
Cliente  -> SYN -> Cortafuegos
                     |
                   DROP

Cliente  -> SYN -> Cortafuegos
                     |
                   DROP

Cliente  -> SYN -> Cortafuegos
                     |
                   DROP

Cliente  -> timeout
```

El cliente continúa retransmitiendo porque no sabe si el paquete se ha perdido por congestión, si existe un problema en la red o si el destino simplemente no responde. Con REJECT utilizando TCP RST ocurre algo muy distinto:

```
Cliente  -> SYN -> Cortafuegos
Cliente  <- RST <- Cortafuegos

Conexión rechazada inmediatamente
```

Desde el punto de vista de rendimiento y experiencia de usuario, REJECT suele ser mucho más limpio.

Qué ocurre con UDP

Con UDP la situación es diferente porque el protocolo no establece conexiones y no dispone de un equivalente directo al TCP RST. Si descartamos un datagrama UDP mediante DROP, el cliente simplemente no recibe ninguna respuesta. Dependiendo de la aplicación, esto puede provocar múltiples reintentos hasta alcanzar un timeout.

Con REJECT normalmente podemos enviar una respuesta ICMP, por ejemplo indicando que el puerto no está disponible. Esto permite que el sistema remoto determine rápidamente que la comunicación no puede realizarse.

Cómo afecta DROP a los escaneos de puertos

Una razón habitual para utilizar DROP es dificultar ligeramente el reconocimiento realizado contra nuestro sistema. Por ejemplo, cuando nmap realiza un escaneo TCP SYN, un puerto que responde con RST normalmente puede identificarse como cerrado. Si el cortafuegos elimina silenciosamente los paquetes, nmap puede clasificar el puerto como **filtered** porque no puede determinar si existe un servicio detrás del cortafuegos.

De forma simplificada:

```
SYN -> SYN/ACK     = puerto abierto

SYN -> RST         = puerto cerrado

SYN -> sin respuesta
                   = normalmente filtered
```

Esto no significa que DROP haga invisible nuestro equipo. Existen muchas otras técnicas para determinar la existencia de un host, identificar un cortafuegos o deducir su política de filtrado. Además, determinadas respuestas ICMP utilizadas por REJECT también permiten que herramientas como nmap clasifiquen el puerto como filtrado en lugar de cerrado. Por tanto, **DROP no debe considerarse una medida de seguridad adicional significativa por sí misma**. Su principal diferencia es que no proporciona una respuesta directa al paquete bloqueado.

DROP provoca más esperas y retransmisiones

Una consecuencia importante de DROP que a veces ignoramos es que **el equipo remoto puede continuar transmitiendo paquetes**. Supongamos que una aplicación intenta conectarse a un servidor que hemos bloqueado. Si utilizamos DROP, la aplicación puede esperar varios segundos mientras TCP realiza retransmisiones. Si esa operación se produce repetidamente, tendremos:

- Conexiones esperando innecesariamente;
- Procesos bloqueados esperando un timeout;
- Retransmisiones adicionales por la red;
- Mayor tiempo hasta que una aplicación detecta un fallo;
- Peor experiencia para los usuarios.

Con REJECT, el fallo se comunica inmediatamente y la aplicación puede reaccionar sin esperar. Esto puede ser especialmente importante dentro de nuestra propia infraestructura.

Por qué normalmente tiene sentido utilizar REJECT en redes internas

Dentro de una LAN, una VPN o una infraestructura controlada por nosotros, normalmente nos interesa que los errores sean detectados lo antes posible. Supongamos que un servidor intenta conectarse a otro servidor mediante TCP/5432, pero hemos bloqueado accidentalmente ese tráfico. Con DROP podríamos encontrarnos con aplicaciones esperando durante bastante tiempo antes de indicar que PostgreSQL no está disponible. Con REJECT el error aparecerá inmediatamente. Esto simplifica además el diagnóstico porque podemos diferenciar claramente entre **una comunicación explícitamente rechazada** y una máquina que simplemente no responde. Por este motivo, en muchas redes internas resulta razonable utilizar REJECT como política para tráfico que sabemos que queremos bloquear.

Por qué podemos preferir DROP en interfaces expuestas a Internet

En una interfaz directamente expuesta a Internet podemos decidir utilizar DROP para tráfico no autorizado. Por ejemplo:

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

    ct state established,related accept
    iif "lo" accept
    tcp dport 22 ip saddr 203.0.113.10 accept
  }
}
```

Con esta configuración utilizamos una política DROP para todo aquello que no hayamos permitido expresamente. Un escáner remoto que intente conectarse contra otros puertos no recibirá respuesta del cortafuegos. Esto tiene varias consecuencias:

- Reduce las respuestas que enviamos a tráfico no solicitado;
- Puede ralentizar determinados tipos de escaneo;
- Evita proporcionar una respuesta explícita a cada intento;
- Hace que muchos puertos aparezcan como filtrados.

El inconveniente es precisamente el mismo: los clientes legítimos afectados por una regla incorrecta también tendrán que esperar al timeout.

REJECT también consume tráfico de salida

Mientras que DROP únicamente elimina el paquete, REJECT requiere construir y enviar una respuesta. En condiciones normales la diferencia es prácticamente irrelevante, pero puede importar cuando recibimos grandes cantidades de tráfico hostil. Si recibimos millones de paquetes destinados a puertos bloqueados y respondemos a todos ellos, estaremos generando también millones de respuestas. DROP evita completamente ese tráfico de retorno.

Además, cuando trabajamos con UDP o ICMP debemos tener cuidado de no convertir determinadas respuestas en un mecanismo que pueda contribuir a ataques de reflexión o amplificación. Los sistemas operativos modernos aplican diferentes limitaciones y controles sobre las respuestas ICMP, pero sigue siendo conveniente tener presente que **REJECT genera tráfico mientras que DROP no genera ninguno**.

REJECT no significa necesariamente que el puerto aparezca como cerrado

También debemos diferenciar entre los distintos tipos de REJECT. Si rechazamos una conexión TCP mediante TCP RST:

```
reject with tcp reset
```

Desde el punto de vista de un escáner, el comportamiento puede parecerse al de un puerto TCP cerrado. Sin embargo, si respondemos mediante determinados errores ICMP:

```
reject with icmpx type admin-prohibited
```

...estamos indicando explícitamente que existe una política administrativa impidiendo la comunicación. Una herramienta de reconocimiento puede utilizar esa información para deducir que existe un cortafuegos. Por tanto, cuando elegimos REJECT también debemos decidir **qué tipo de rechazo queremos enviar**.

DROP tampoco oculta que existe un cortafuegos

Es frecuente encontrar la idea de que debemos utilizar siempre DROP porque así un atacante no puede saber que nuestra

máquina existe. Esto es una simplificación excesiva. Un atacante puede obtener información mediante:

- Otros puertos que sí estén accesibles;
- Respuestas ARP dentro de una red local;
- ICMP;
- Servicios publicados;
- DNS;
- Otras máquinas de nuestra infraestructura;
- Diferencias de comportamiento durante el filtrado;
- Análisis de rutas y saltos de red.

Incluso el hecho de que un determinado conjunto de puertos no responda mientras otras direcciones sí lo hacen puede aportar información. DROP puede reducir la información directa proporcionada por el equipo, pero **no debemos utilizarlo como sustituto de una política de seguridad correctamente diseñada**.

Cuándo utilizar DROP y cuándo utilizar REJECT

No existe una regla universal, pero podemos utilizar como criterio general el siguiente:

SITUACIÓN	OPCIÓN HABITUAL
Tráfico no solicitado procedente de Internet	DROP
Puertos que no queremos exponer públicamente	DROP
Tráfico claramente hostil	DROP
Comunicación entre máquinas de nuestra LAN	REJECT
VPN y redes administradas por nosotros	REJECT
Reglas destinadas a impedir determinadas conexiones de usuarios internos	REJECT
Situaciones donde queremos que una aplicación falle inmediatamente	REJECT

Una estrategia bastante razonable consiste en **utilizar DROP para tráfico externo no solicitado y REJECT para tráfico interno que queremos prohibir explícitamente**. Por ejemplo, nuestro cortafuegos podría aceptar únicamente los servicios públicos necesarios y descartar silenciosamente el resto del tráfico procedente de Internet, mientras que entre diferentes VLAN o segmentos internos podríamos utilizar REJECT para que las aplicaciones detecten inmediatamente las comunicaciones que nuestra política de seguridad no permite.

La diferencia práctica que debemos recordar es sencilla: **DROP bloquea y guarda silencio; REJECT bloquea y avisa**.