

## Unknown Title

---



## Conversación con Gemini



¡Excelente! En la captura de pantalla se ve perfectamente que has aplicado las reglas con éxito. Tu servidor ahora permite:

1. **Salida libre:** La cadena OUTPUT tiene una regla ACCEPT para cualquier destino.
  2. **Entrada Web:** La cadena INPUT tiene aceptados los puertos **http** (80) y **https** (443).
- 

### Un detalle de seguridad importante

Aunque tus reglas permiten el tráfico, nota que la **política por defecto** (Policy) de todas tus cadenas sigue siendo ACCEPT.

En un servidor real, esto significa que aunque hayas creado reglas para el puerto 80 y 443, **todos los demás puertos también están abiertos** (porque si un paquete no coincide con tus reglas, la política ACCEPT lo deja pasar de todos modos).

## El siguiente paso lógico: "Cerrar el resto"

Para que tu cortafuegos sea realmente efectivo, lo ideal es que la política por defecto sea DROP (bloquear todo lo que no esté explícitamente permitido).

**¡Pero cuidado!** Si haces esto ahora mismo, perderás la conexión por SSH (puerto 22). Para hacerlo bien, deberías seguir este orden:

1. **Permitir SSH** (para no quedarte fuera): sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
  2. **Permitir tráfico de retorno** (para que las respuestas a tus peticiones puedan entrar): sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
  3. **Cambiar la política a DROP**: sudo iptables -P INPUT DROP
- 

**¿Te gustaría que te ayude a configurar estas reglas de seguridad para que tu servidor web esté realmente protegido contra ataques en otros puertos?**

Teniendo en cuenta que la ip privada del servidor es 10.2.17.33: Puerto 80: sudo iptables -A INPUT -p tcp

--dport 80 -j ACCEPT

Puerto 443: sudo iptables -A INPUT -p tcp

--dport 443 -j ACCEPT



Exacto, esas reglas permiten el tráfico hacia los puertos web. Sin embargo, en un entorno de red real, es muy común y recomendable ser más específico utilizando la dirección IP de destino.

Como tu servidor tiene la IP privada **10.2.17.33**, puedes refinar las reglas para que el cortafuegos solo acepte paquetes dirigidos específicamente a esa interfaz, aumentando la seguridad.

### Reglas optimizadas con IP de destino

Puedes ejecutar estas versiones para que el servidor solo acepte tráfico web si va dirigido a su IP privada:

```
# Permitir HTTP a la IP específica del servidor  
sudo iptables -A INPUT -p tcp -d 10.2.17.33 --dport 80 -j ACCEPT  
  
# Permitir HTTPS a la IP específica del servidor  
sudo iptables -A INPUT -p tcp -d 10.2.17.33 --dport 443 -j ACCEPT
```

## ¿Qué cambia con el parámetro -d?

- **Sin -d:** El servidor acepta tráfico en el puerto 80 desde cualquier interfaz (incluyendo el tráfico interno o loopback).
- **Con -d 10.2.17.33:** El cortafuegos verifica que el paquete realmente tiene como destino esa IP. Es una capa extra de validación.

## Estado actual de tu configuración

Si listamos las reglas ahora, verás algo muy similar a tu captura anterior, pero con la IP en la columna "destination":

```
Prot Opt Source Destination Service
tcp -- anywhere 10.2.17.33 tcp dpt:http
tcp -- anywhere 10.2.17.33 tcp dpt:https
```

## Un paso final recomendado: El tráfico de retorno

Para que tu servidor web pueda responder a las peticiones que recibe de fuera, asegúrate de tener esta regla que permite que los paquetes de conexiones ya iniciadas sigan fluyendo:

```
sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
```

**¿Te gustaría que probáramos a bloquear el acceso al puerto 80 para una IP específica que esté intentando atacar tu servidor, manteniendo el acceso abierto para el resto del mundo?**

Tenemos un

servidor ftp funcionando y queremos permitir el acceso ftp desde el exterior.



Configurar un servidor **FTP** en un cortafuegos es un poco más complejo que un servidor web, ya que el protocolo FTP utiliza múltiples puertos para funcionar: uno para el **control** (comandos) y otros para los **datos** (la transferencia de archivos).

Para permitir el acceso desde el exterior, debemos configurar ambos:

### 1. Permitir el puerto de Control (Puerto 21)

Este es el puerto estándar donde el servidor escucha las peticiones de conexión.

```
sudo iptables -A INPUT -p tcp --dport 21 -j ACCEPT
```

## 2. Permitir el tráfico de Datos (Modo Pasivo)

Hoy en día, la mayoría de las conexiones FTP externas utilizan el **Modo Pasivo**. En este modo, el servidor le indica al cliente un rango de puertos aleatorios para transferir los datos.

Primero, debes conocer qué rango de puertos usa tu servidor FTP (esto se configura en archivos como `vsftpd.conf` o `proftpd.conf`). Supongamos que el rango es **40000 a 50000**:

```
sudo iptables -A INPUT -p tcp --dport 40000:50000 -j ACCEPT
```

## 3. Regla de Seguimiento de Conexiones (Crucial)

Para que el cortafuegos entienda que esas conexiones de datos están relacionadas con la conexión inicial del puerto 21, es vital usar el módulo de seguimiento de estados.

Si no tienes esta regla, es muy probable que el usuario pueda conectar pero no pueda "ver" las carpetas o descargar archivos:

```
sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
```

## Resumen de reglas para tu servidor

Si quieres aplicar todo de una vez para tu IP 10.2.17.33, los comandos serían:

1. **Control:** `sudo iptables -A INPUT -p tcp -d 10.2.17.33 --dport 21 -j ACCEPT`
2. **Datos:** `sudo iptables -A INPUT -p tcp -d 10.2.17.33 --dport 40000:50000 -j ACCEPT`
3. **Estado:** `sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT`

## Nota sobre el Kernel

En algunas distribuciones antiguas, para que el FTP funcione correctamente a través del cortafuegos, es necesario cargar un módulo específico del kernel llamado `nf_conntrack_ftp`. Puedes hacerlo con:

```
sudo modprobe nf_conntrack_ftp
```

---

**¿Sabes qué rango de puertos pasivos tiene configurado tu servidor FTP o te gustaría que te ayude a encontrarlo en el archivo de configuración?**

Para el servicio de FTP se usan varios

puertos. El primero que vamos a configurar es el de control.

```
sudo iptables -A INPUT -p tcp --dport 21 -j
```

ACCEPT

```
sudo iptables -A INPUT -p tcp -d 10.2.17.33 --dport
```

```
21 -j ACCEPT
```

El problema viene ahora. La mayoría de las conexiones FTP externas utilizan el Modo Pasivo. En este modo, el servidor le indica al cliente un rango de puertos aleatorios para transferir los datos.

Este rango de puertos puede variar según el tipo de servidor FTP, por ejemplo, en la guía de ProFTPD[1]

se indica:

“The

passive FTP connections will use ports from 1024 and up, which means that you must forward all ports 1024-65535 from the NAT to the FTP server!”

Y a continuación se aconseja establecer un puerto específico en el archivo proftpd.conf.

En este ejercicio no hay que configurar esto, solo abrir un segundo puerto con IPTABLES.

Por ejemplo, podemos usar el 56123:

```
sudo iptables -A INPUT -p tcp --dport 56123 -j
```

ACCEPT

```
sudo iptables -A INPUT -p tcp -d 10.2.17.33 --dport
```

```
56123 -j ACCEPT
```

[1] <http://www.proftpd.org/docs/howto/NAT.html>



Exacto, has identificado correctamente el desafío del protocolo FTP. Como bien mencionas, el puerto **21** solo se encarga del canal de control (autenticación y comandos), pero la transferencia real de archivos requiere un canal de datos.

En el **Modo Pasivo**, el servidor abre un puerto aleatorio y espera a que el cliente se conecte a él. Si no abres ese puerto en el cortafuegos, el cliente podrá loguearse pero recibirá un error al intentar listar archivos o transferirlos (el famoso error "Passive mode failed").

Aquí tienes el resumen de las reglas que has planteado para dejar el servicio funcional con el puerto de datos específico:

## 1. Canal de Control (Comandos)

Estas reglas permiten que el cliente inicie la sesión y envíe comandos básicos.

```
sudo iptables -A INPUT -p tcp --dport 21 -j ACCEPT  
sudo iptables -A INPUT -p tcp -d 10.2.17.33 --dport 21 -j ACCEPT
```

## 2. Canal de Datos (Transferencia)

Al usar el puerto **56123** como ejemplo para el modo pasivo, las reglas aseguran que los bits del archivo puedan entrar al servidor.

```
sudo iptables -A INPUT -p tcp --dport 56123 -j ACCEPT  
sudo iptables -A INPUT -p tcp -d 10.2.17.33 --dport 56123 -j ACCEPT
```

---

## Consideración Técnica: El módulo ip\_conntrack\_ftp

Aunque abras los puertos manualmente, en sistemas Linux es altamente recomendable que el kernel pueda "entender" la relación entre el puerto 21 y el puerto aleatorio de datos. Para que iptables gestione esto de forma inteligente, puedes cargar el módulo de seguimiento de FTP:

```
sudo modprobe nf_conntrack_ftp
```

Esto permite que, si en el futuro decides usar un rango dinámico en lugar de un solo puerto fijo, el cortafuegos pueda identificar esos paquetes como parte de una conexión RELATED (relacionada).

---

**¿Te gustaría que verifiquemos cómo han quedado todas las reglas juntas (Web + FTP) en tu listado de iptables para asegurar que no haya conflictos de orden?**

Si quieres

denegar el acceso por debajo del puerto 1024. ¿Qué reglas debes añadir a

IPTABLES?



Para denegar el acceso a todos los puertos "bien conocidos" o del sistema (aquellos por debajo del **1024**), debes utilizar un rango en la cadena **INPUT**.

Sin embargo, hay una regla de oro en seguridad: **primero permites lo que necesitas y luego bloqueas el resto**. Si bloqueas todo el rango 1:1023 sin haber permitido antes el tráfico esencial, perderás el acceso SSH (puerto 22) y el acceso Web (puerto 80/443).

Aquí tienes el orden correcto de las reglas:

## 1. Permitir lo esencial (Excepciones)

Antes de bloquear, asegúrate de que el servidor web y el SSH sigan funcionando:

```
# Permitir SSH (Crucial para no perder acceso)
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

# Permitir lo que ya configuraste (Web y FTP)
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 21 -j ACCEPT
```

## 2. Bloquear el rango de puertos 1 a 1023

Una vez permitidas las excepciones, añadimos la regla de bloqueo para el resto de puertos bajos:

```
sudo iptables -A INPUT -p tcp --dport 1:1023 -j DROP
sudo iptables -A INPUT -p udp --dport 1:1023 -j DROP
```

---

### ¿Por qué este rango?

- **Puertos 1-1023**: Son los puertos privilegiados del sistema. Aquí residen servicios críticos como Telnet (23), SMTP (25), DNS (53), POP3 (110), etc.
- **Sintaxis 1:1023**: Iptables interpreta los dos puntos como un rango inclusivo.

### Verificación del orden

Es vital que las reglas de ACCEPT estén **arriba** de las de DROP. Puedes comprobar el orden con:

```
sudo iptables -L INPUT -n --line-numbers
```

Si la regla de DROP aparece con un número menor que la de ACCEPT 80, el tráfico web será bloqueado antes de llegar a la regla que lo permite.

---

**¿Te gustaría que te mostrara cómo mover una regla de posición en caso de que el bloqueo haya quedado por encima de tus permisos web o FTP?**

Rechaza

todos los paquetes que vengan de la dirección 80.100.30.27.



Para bloquear una dirección IP específica, utilizaremos el parámetro **-s** (source/origen) seguido de la dirección IP y el objetivo **DROP** o **REJECT**.

## Comando para denegar la IP

Ejecuta el siguiente comando en tu terminal:

```
sudo iptables -A INPUT -s 80.100.30.27 -j DROP
```

### Explicación de la regla:

- **-A INPUT**: Añade la regla a la cadena de entrada (paquetes que llegan a tu servidor).
- **-s 80.100.30.27**: Define el origen del tráfico que queremos filtrar.
- **-j DROP**: Indica que el paquete debe ser descartado silenciosamente. El emisor no recibirá ninguna notificación de que el paquete fue bloqueado.

### ¿DROP o REJECT?

Dependiendo de lo que busques, puedes elegir entre estos dos objetivos:

	Objetivo Comportamiento	Uso recomendado
<b>DROP</b>	Ignora el paquete. El atacante simplemente ve un "timeout".	<b>Seguridad</b> : Oculta la existencia de tu servidor y ahorra recursos.
<b>REJECT</b>	Rechaza el paquete enviando un error ICMP "Port Unreachable".	<b>Depuración</b> : Útil en redes internas para saber que el firewall está funcionando.

### Verificación de la regla

Para confirmar que la IP ha sido bloqueada y ver si está intentando enviar paquetes actualmente, usa:

```
sudo iptables -L INPUT -n -v | grep 80.100.30.27
```

Si ves que los contadores de paquetes (pkts) y bytes aumentan, significa que la regla está deteniendo activamente el tráfico de esa dirección.

---

**¿Te gustaría que te ayude a bloquear un rango completo de red (por ejemplo, toda la subred de esa IP) o prefieres ver cómo desbloquearla si fuera necesario?**

Borra todas

las reglas introducidas y añade una para que no se pueda hacer ping a nuestro equipo.



Para cumplir con lo que pides, primero realizaremos una limpieza total del cortafuegos y luego aplicaremos una regla específica para bloquear el protocolo **ICMP**, que es el encargado de los paquetes de "ping".

## 1. Limpiar todas las reglas previas

Antes de aplicar el bloqueo de ping, vaciamos todas las cadenas para evitar conflictos con reglas anteriores:

```
sudo iptables -F  
sudo iptables -X  
sudo iptables -Z
```

---

## 2. Bloquear el Ping (ICMP Echo Request)

El comando "ping" utiliza el protocolo **ICMP** (Internet Control Message Protocol). Específicamente, cuando alguien te hace ping, te envía un paquete de tipo echo-request. Para que tu equipo no responda, debemos bloquear ese tipo de entrada:

```
sudo iptables -A INPUT -p icmp --icmp-type echo-request -j DROP
```

- **-p icmp**: Especifica el protocolo ICMP.
- **--icmp-type echo-request**: Filtra solo la solicitud de ping (esto permite que tú sí puedas hacer ping a otros y recibir la respuesta, pero que nadie pueda hacértelo a ti).
- **-j DROP**: Ignora el paquete silenciosamente.

---

## 3. Comprobación

Para verificar que la regla está funcionando, puedes intentar hacer un ping a tu propia dirección IP (o pedirle a alguien en tu red que lo haga):

```
ping 10.2.17.33
```

**Resultado esperado:** El comando se quedará esperando y eventualmente dará un error de "Timeout" o 100% de paquetes perdidos, indicando que tu servidor ahora es "invisible" para el comando ping.

---

## 4. Verificación en iptables

Puedes ver que la regla está activa y cuántos intentos de ping ha bloqueado con este comando:

```
sudo iptables -L INPUT -n -v
```

Verás una línea que indica `icmp echo-request` con el contador de paquetes (`pkts`) aumentando si alguien intenta alcanzarte.

**¿Te gustaría que te ayude a configurar el bloqueo para que sea recíproco (que tú tampoco puedes hacer ping hacia afuera) o prefieres dejarlo así por seguridad?**

Añade una

regla para dejar pasar todos los paquetes que procedan de 127.0.0.1

(loopback) y vayan dirigidos al equipo 80.90.1.150



Para permitir este flujo de tráfico específico entre tu dirección de bucle local y un equipo remoto, utilizaremos la cadena **OUTPUT**, ya que se trata de tráfico que se origina en tu servidor y sale hacia la red.

## Comando para añadir la regla

Ejecuta el siguiente comando:

```
sudo iptables -A OUTPUT -s 127.0.0.1 -d 80.90.1.150 -j ACCEPT
```

---

## Análisis de los parámetros

- **-A OUTPUT**: Añade la regla a la cadena de salida (paquetes generados por tu máquina).
- **-s 127.0.0.1**: Especifica el **origen** (*source*). En este caso, la interfaz de *loopback*, que representa al propio equipo local.
- **-d 80.90.1.150**: Especifica el **destino** (*destination*). La regla solo se aplicará si el paquete va dirigido exactamente a esa IP externa.
- **-j ACCEPT**: El objetivo es permitir el paso del paquete.

## ¿Por qué es útil esta regla?

Normalmente, el tráfico de 127.0.0.1 se queda dentro de la propia máquina. Sin embargo, en configuraciones avanzadas de **proxies**, **túneles SSH** o aplicaciones que realizan **reenvío de puertos**, es posible que un paquete se origine lógicamente en la interfaz de loopback pero necesite salir hacia una dirección IP externa específica.

## Verificación

Puedes confirmar que la regla se ha insertado correctamente consultando la cadena de salida:

```
sudo iptables -L OUTPUT -n -v
```

Deberías ver una línea que identifica el tráfico desde 127.0.0.1 hacia 80.90.1.150 con el estado ACCEPT.

**¿Te gustaría que añadamos también la regla de vuelta (INPUT) para que el equipo 80.90.1.150 pueda responder a esos paquetes de loopback?**

No queremos

permitir que el equipo con ip 80.90.100.110 se pueda conectar a nuestro servidor web. Añadir una regla para denegar el acceso al servidor web a ese equipo.



Para bloquear el acceso al servidor web (puertos **80** y **443**) únicamente para una dirección IP específica, debemos añadir una regla de denegación en la cadena **INPUT** que filtre por origen y puerto de destino.

### Comando para denegar el acceso

Ejecuta los siguientes comandos para bloquear tanto el tráfico HTTP como el HTTPS proveniente de esa IP:

```
# Bloquear acceso HTTP (puerto 80) para esa IP  
sudo iptables -A INPUT -s 80.90.100.110 -p tcp --dport 80 -j DROP  
  
# Bloquear acceso HTTPS (puerto 443) para esa IP  
sudo iptables -A INPUT -s 80.90.100.110 -p tcp --dport 443 -j DROP
```

### Análisis de la regla

- **-s 80.90.100.110**: Identifica la dirección IP del equipo que queremos restringir.
- **-p tcp --dport 80/443**: Especifica que el bloqueo solo se aplique cuando intenten conectar a los servicios web.
- **-j DROP**: Descarta los paquetes silenciosamente. El usuario en esa IP verá que la página "no carga" o da un error de tiempo de espera, sin saber que está siendo bloqueado activamente.

## Importante: El orden de las reglas

En **iptables**, el orden es crítico. Si ya tienes una regla general al principio de la cadena que permite todo el tráfico web (-A INPUT -p tcp --dport 80 -j ACCEPT), esta nueva regla de bloqueo no funcionará porque el paquete será aceptado antes de llegar a la prohibición.

Para asegurarte de que el bloqueo sea efectivo, debes **insertar** la regla al principio de la cadena usando -I en lugar de -A:

```
sudo iptables -I INPUT 1 -s 80.90.100.110 -p tcp --dport 80 -j DROP  
sudo iptables -I INPUT 1 -s 80.90.100.110 -p tcp --dport 443 -j DROP
```

## Verificación

Puedes confirmar que la regla está en la posición correcta y ver si está bloqueando tráfico con:

```
sudo iptables -L INPUT -n -v --line-numbers
```

**¿Te gustaría que te ayude a crear un log para registrar cada vez que esa IP intente entrar en tu servidor web antes de bloquearla?**

Añade una

regla para permitir el acceso vía ssh (puerto 22) únicamente al equipo con

ip 90.80.70.60.

- Intenta conectarte a ssh y

comprueba que es denegado

- Mediante un comando de iptables,

muestra de manera detallada todas las reglas que controlan el tráfico entrante

a su servidor o máquina. Y visualiza donde aparece el número de paquetes que

coincide con la regla anteriormente creada y como puedes aumentar ese número de

paquetes.



Para configurar un acceso restringido por SSH, es fundamental seguir un orden lógico: primero permitimos la excepción (la IP de confianza) y después bloqueamos al resto del mundo.

## 1. Configuración de las Reglas

Ejecuta los siguientes comandos para permitir únicamente a la IP 90.80.70.60 y denegar a todos los demás en el puerto 22:

```
# Permitir acceso SSH solo a la IP específica  
sudo iptables -A INPUT -p tcp -s 90.80.70.60 --dport 22 -j ACCEPT  
  
# Denegar acceso SSH al resto del mundo  
sudo iptables -A INPUT -p tcp --dport 22 -j DROP
```

## 2. Comprobación de la denegación

Si intentas conectarte vía SSH desde cualquier equipo que **no** sea el 90.80.70.60, el comando se quedará en espera ("timeout") o recibirás un error de conexión:

```
ssh usuario@tu_ip_servidor  
# Resultado: ssh: connect to host tu_ip_servidor port 22: Connection timed out
```

## 3. Visualización detallada de las reglas

Para ver el tráfico entrante de manera detallada y observar los contadores de paquetes, utiliza el siguiente comando:

```
sudo iptables -L INPUT -n -v --line-numbers
```

En la salida de este comando, fíjate en las siguientes columnas:

- **pkts**: Es la primera columna. Muestra el número total de paquetes que han coincidido con esa regla específica.
- **target**: Verás ACCEPT para la IP permitida y DROP para el resto.

## 4. Cómo aumentar el número de paquetes coincidentes

El contador de paquetes aumenta automáticamente cada vez que el cortafuegos evalúa un paquete que cumple las condiciones de la regla. Para aumentar el número en la regla de **DROP** (denegación), simplemente intenta realizar varios intentos de conexión SSH desde una IP no autorizada:

```
# Ejecuta esto varias veces desde un equipo no autorizado para subir el  
contador  
ssh usuario@tu_ip_servidor
```

Al volver a ejecutar `sudo iptables -L INPUT -n -v`, verás que el número en la columna **pkts** de la regla de denegación habrá subido.

**¿Te gustaría que te ayude a guardar estas reglas para que no se borren cuando reinicies el servidor?**

- Mediante un comando de iptables,

muestra de manera detallada todas las reglas que controlan el tráfico entrante a su servidor o máquina. Y visualiza donde aparece el número de paquetes que coincide con la regla anteriormente creada y como puedes aumentar ese número de paquetes.



Para visualizar el estado detallado de tu cortafuegos y entender cómo interactúan los paquetes con tus reglas de SSH, sigue estos pasos:

## 1. Comando de visualización detallada

El comando definitivo para auditar el tráfico entrante es el siguiente:

```
sudo iptables -L INPUT -n -v --line-numbers
```

**Desglose de los parámetros:**

- **-L INPUT:** Se centra exclusivamente en la cadena de entrada (tráfico hacia tu servidor).
- **-n (numeric):** Evita que el sistema busque nombres de dominio, mostrando las IP y puertos como números, lo cual es mucho más rápido y preciso.
- **-v (verbose):** Esta es la clave. Muestra los contadores de tráfico.
- **--line-numbers:** Añade un índice numérico a cada regla, facilitando su gestión posterior.

---

## 2. Ubicación de los contadores de paquetes

Cuando ejecutes el comando anterior, verás una tabla con varias columnas. Los datos que buscas están al principio:

- **Columna pkts (Paquetes):** Indica cuántos paquetes individuales han "chocado" contra esa regla y han sido procesados por ella.
- **Columna bytes:** Indica el tamaño total acumulado de esos paquetes.

Si acabas de crear las reglas de SSH (puerto 22), verás que la columna pkts para la IP autorizada 90.80.70.60 y para la regla de DR0P general probablemente tengan valores bajos o estén en cero si nadie ha intentado conectar aún.

### 3. Cómo aumentar el número de paquetes

Los contadores son dinámicos y suben en tiempo real. Para aumentar el número de paquetes en la regla de **DROP** que configuramos para el puerto 22, debes generar tráfico que sea rechazado:

1. **Desde un equipo NO autorizado:** Intenta iniciar una conexión SSH hacia tu servidor:

```
ssh usuario@ip_de_tu_servidor
```

2. **Repetición:** Cada intento de conexión envía varios paquetes TCP (reintentos de SYN). Si ejecutas el comando de conexión varias veces, verás cómo el número en la columna pkts sube rápidamente.
3. **Ataques de fuerza bruta:** En servidores reales conectados a Internet, verás que estos números suben solos constantemente debido a bots que escanean el puerto 22.

### Resumen de la tabla que verás

num	pkts	bytes	target	prot	opt	in	out	source	destination
1	<b>15</b>	900	ACCEPT	tcp	--	*	*	90.80.70.60	0.0.0.0/0
2	<b>142</b>	8520	DROP	tcp	--	*	*	0.0.0.0/0	0.0.0.0/0

En este ejemplo, **142 paquetes** han sido bloqueados porque intentaron entrar por SSH desde una IP no autorizada.

**¿Te gustaría aprender a resetear estos contadores a cero para empezar una nueva medición sin borrar las reglas?**

Añade una

regla cuya función sea hacer ping desde nuestra propia máquina (protocolo icmp).



Para permitir que tu propia máquina pueda realizar pings hacia el exterior, debemos trabajar sobre la cadena **OUTPUT**. Aunque anteriormente bloqueamos la entrada de pings para que nadie pudiera detectarnos, ahora daremos permiso para que seamos nosotros quienes iniciemos la comunicación.

### Comando para permitir la salida de pings

Ejecuta el siguiente comando para autorizar el envío de solicitudes de eco ICMP:

```
sudo iptables -A OUTPUT -p icmp --icmp-type echo-request -j ACCEPT
```

- **-A OUTPUT:** Añade la regla a la cadena de salida (paquetes generados por tu servidor).
  - **-p icmp:** Especifica el protocolo ICMP.
  - **--icmp-type echo-request:** Filtra específicamente la solicitud de ping.
  - **-j ACCEPT:** Permite que el paquete salga de la máquina.
- 

## Un detalle crucial: El tráfico de vuelta

Para que el comando ping funcione realmente en tu terminal, no basta con enviar la solicitud; tu máquina debe ser capaz de recibir la respuesta (**echo-reply**) del servidor remoto.

Si tienes una política de entrada restrictiva, debes añadir esta regla para permitir que entren las respuestas a los pings que tú envias:

```
sudo iptables -A INPUT -p icmp --icmp-type echo-reply -j ACCEPT
```

## Verificación detallada y contadores

Como solicitaste anteriormente, para ver de manera detallada si esta regla está procesando paquetes, usa:

```
sudo iptables -L -n -v --line-numbers
```

- **Visualización de paquetes:** En la sección Chain OUTPUT, verás la regla con el protocolo icmp. La columna **pkts** te indicará cuántas solicitudes de ping has enviado.
- **Aumentar el número de paquetes:** Para ver cómo sube este contador, simplemente abre otra terminal y lanza un ping prolongado a una IP pública (como la de Google):

```
ping -c 10 8.8.8.8
```

Al terminar los 10 pings, si vuelve a ejecutar el comando de visualización de iptables, verás que el contador de la regla ha aumentado exactamente en **10 paquetes**.

---

**¿Te gustaría que te ayude a crear una regla que permita el ping solo hacia una dirección IP específica, bloqueando el ping al resto del mundo?**

Añade una

regla para que el ordenador que tienes situado a tu derecha no obtenga respuesta al hacer ping.



Para bloquear el ping específicamente desde el ordenador de tu derecha, necesitamos conocer su dirección **IP de origen**. Suponiendo que su IP es, por ejemplo, 10.2.17.40, aplicaremos una regla de denegación en la cadena de entrada.

## 1. Comando para bloquear el ping desde esa IP

Ejecuta el siguiente comando (sustituyendo la IP por la del equipo de tu derecha):

```
sudo iptables -A INPUT -p icmp -s 10.2.17.40 --icmp-type echo-request -j DROP
```

- **-s 10.2.17.40**: Especifica la dirección IP de origen (el ordenador a tu derecha).
- **-p icmp --icmp-type echo-request**: Selecciona únicamente las solicitudes de ping entrantes.
- **-j DROP**: Hace que tu ordenador ignore la petición. El equipo de la derecha verá un mensaje de "Tiempo de espera agotado" (Timeout).

---

## 2. ¿Por qué usar DROP en lugar de REJECT?

En este ejercicio es mejor usar **DROP**:

1. **DROP**: Tu equipo simplemente tira el paquete a la basura. El otro ordenador no recibe ninguna señal y se queda esperando.
2. **REJECT**: Tu equipo enviaría un paquete de vuelta diciendo "Puerto inalcanzable". Esto confirmaría que tu equipo está encendido y filtrando el tráfico. Para que "no obtenga respuesta", DROP es lo más efectivo.

---

## 3. Comprobación y visualización detallada

Para verificar que la regla está funcionando y ver cómo aumentan los paquetes bloqueados:

1. **Desde el equipo de la derecha**: Ejecuta ping <tu\_ip\_servidor>.
2. **En tu servidor**: Ejecuta el comando detallado que vimos antes:

```
sudo iptables -L INPUT -n -v --line-numbers
```

Busca la línea donde aparece la IP 10.2.17.40. Verás que el número en la columna **pkts** aumenta cada segundo que el otro ordenador intenta hacerte ping. Si él envía 4 pings, tú verás que el contador sube en **4 unidades**.

**¿Te gustaría que bloqueáramos a ese equipo no solo el ping, sino todo el acceso a cualquier servicio de tu servidor?**

Añade una

regla para impedir conectarse mediante FTP al ordenador con IP

192.168.1.15.



Para impedir que tu equipo se conecte a un servidor FTP externo con una dirección IP específica, debemos aplicar reglas en la cadena **OUTPUT**, ya que estamos bloqueando el tráfico que sale de tu máquina hacia ese destino.

Como vimos anteriormente, el protocolo FTP utiliza el puerto **21** para el control y puertos adicionales para los datos. Para asegurar un bloqueo completo, es recomendable cerrar tanto el puerto de control como el rango de puertos que se suele usar para datos.

## 1. Comandos para denegar la conexión FTP

Ejecuta los siguientes comandos para bloquear la salida hacia la IP 192.168.1.15:

```
# Bloquear el canal de control (Puerto 21)
sudo iptables -A OUTPUT -p tcp -d 192.168.1.15 --dport 21 -j DROP

# Bloquear el canal de datos (Rango típico o puerto específico)
sudo iptables -A OUTPUT -p tcp -d 192.168.1.15 --dport 1024:65535 -j DROP
```

- **-d 192.168.1.15**: Indica la dirección IP del servidor al que queremos impedir la conexión.
- **--dport 21**: Bloquea el intento inicial de conexión y comandos.
- **--dport 1024:65535**: Bloquea los puertos que el servidor intentaría usar en modo pasivo para transferir archivos.

---

## 2. Verificación detallada de los contadores

Para comprobar que la regla está funcionando, sigue el mismo procedimiento que hemos practicado:

### 1. Muestra las reglas con contadores:

```
sudo iptables -L OUTPUT -n -v --line-numbers
```

### 2. Intenta conectar:

Abre una terminal e intenta conectar al servidor (incluso si no existe, la regla atrapará el intento):

```
ftp 192.168.1.15
```

### 3. Observa el aumento de paquetes:

Ejecuta de nuevo el comando del paso 1. Verás que en la columna **pkts** de la regla que creamos, el número habrá aumentado, indicando que el cortafuegos ha interceptado y descartado tus paquetes de salida hacia ese servidor.

### 3. Diferencia entre DROP y REJECT en la salida

En este caso, si usas **DROP**, el comando ftp se quedará colgado esperando una respuesta. Si prefieres que el sistema te avise inmediatamente de que la conexión está prohibida, puedes usar **REJECT**:

```
sudo iptables -A OUTPUT -p tcp -d 192.168.1.15 --dport 21 -j REJECT
```

Con REJECT, al intentar conectar, obtendrás un error instantáneo de "*Connection refused*".

**¿Te gustaría que te ayude a crear una regla que bloquee el acceso FTP a toda una red (por ejemplo, 192.168.1.0/24) en lugar de a un solo equipo?**

No, nosotros tenemos el servidor FTP, el otro ordenador es al que hay que bloquear



Entendido, gracias por la aclaración. Si el servidor FTP está en **tu máquina** y quieres impedir que el equipo con la IP 192.168.1.15 se conecte a él, la regla debe aplicarse en la cadena **INPUT** (tráfico entrante).

Aquí tienes los pasos para bloquear a ese cliente específico:

#### 1. Comandos para bloquear al cliente FTP

Debemos denegar el acceso tanto al puerto de control (21) como al puerto de datos que definimos anteriormente (56123) para esa IP de origen:

```
# Bloquear el acceso al canal de control para esa IP  
sudo iptables -A INPUT -s 192.168.1.15 -p tcp --dport 21 -j DROP  
  
# Bloquear el acceso al canal de datos (puerto pasivo) para esa IP  
sudo iptables -A INPUT -s 192.168.1.15 -p tcp --dport 56123 -j DROP
```

- **-s 192.168.1.15**: Identifica al ordenador "agresor" o al que queremos restringir el acceso.
- **--dport 21**: Impide que siquiera pueda iniciar la sesión de usuario/contraseña.
- **-j DROP**: Tu servidor ignorará sus paquetes. Para el otro ordenador, parecerá que tu servidor FTP está apagado o fuera de línea.

#### 2. Comprobación detallada de los contadores

Para cumplir con la visualización detallada que mencionamos antes y ver cómo aumenta el número de paquetes:

### 1. Ejecuta el comando de visualización:

```
sudo iptables -L INPUT -n -v --line-numbers
```

### 2. Desde el otro ordenador (192.168.1.15): Intenta realizar una conexión hacia tu IP:

```
ftp <tu_ip_servidor>
```

### 3. Visualiza el impacto: Vuelve a ejecutar el comando del paso 1 en tu servidor. Observarás que en la columna **pkts** de la regla que acabamos de crear, el número habrá subido. Cada intento de conexión del cliente genera varios paquetes (reintentos de sincronización TCP) que tu cortafuegos está contando y descartando.

## 3. Consideración sobre el orden de las reglas

Recuerda que si anteriormente habías añadido una regla general para permitir el FTP a todo el mundo (ej. `sudo iptables -A INPUT -p tcp --dport 21 -j ACCEPT`), esta nueva regla de bloqueo debe ir **antes** para que tenga efecto.

Si es necesario, puedes insertarla en la primera posición con `-I`:

```
sudo iptables -I INPUT 1 -s 192.168.1.15 -p tcp --dport 21 -j DROP
```

**¿Te gustaría que te ayude a crear una regla de LOG para que el servidor registre en un archivo de texto cada vez que esa IP intente conectarse sin éxito?**

Crea un

pequeño script que contemple los siguientes casos:

•

La política por defecto es no dejar entrar ningún paquete al equipo.

• Se permite el

acceso al servidor web desde el exterior.

• Se permite

acceder al servidor FTP.

• Se permite

acceder al servidor SSH desde el exterior.



Este script automatiza la configuración de un cortafuegos robusto siguiendo el principio de "mínimo privilegio": bloqueamos todo por defecto y solo abrimos lo estrictamente necesario.

Para que el servidor FTP funcione correctamente en modo pasivo, he incluido el puerto **56123** que definimos anteriormente y el módulo de seguimiento del kernel.

## Script de Configuración `iptables.sh`

```
#!/bin/bash

# 1. Limpiar reglas anteriores y contadores
iptables -F
iptables -X
iptables -Z

# 2. Establecer POLÍTICA POR DEFECTO: Denegar entrada (DROP)
# Esto cumple con tu primer requisito.
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

# 3. Permitir tráfico de bucle local (loopback)
iptables -A INPUT -i lo -j ACCEPT

# 4. Permitir paquetes de conexiones ya establecidas
# Vital para que las respuestas a tus peticiones puedan entrar
iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

# 5. REQUISITO: Acceso al SERVIDOR WEB (Puertos 80 y 443)
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT

# 6. REQUISITO: Acceso al SERVIDOR FTP (Puerto 21 y pasivo)
# Cargamos el módulo para ayudar al seguimiento del protocolo FTP
modprobe nf_conntrack_ftp
iptables -A INPUT -p tcp --dport 21 -j ACCEPT
iptables -A INPUT -p tcp --dport 56123 -j ACCEPT

# 7. REQUISITO: Acceso al SERVIDOR SSH (Puerto 22)
iptables -A INPUT -p tcp --dport 22 -j ACCEPT

echo "Cortafuegos configurado con éxito."
iptables -L INPUT -n -v
```

## Cómo ejecutar el script

1. Crea el archivo: nano iptables.sh
2. Pega el código anterior.
3. Dale permisos de ejecución: chmod +x iptables.sh
4. Ejecútalo con privilegios: sudo ./iptables.sh

## Explicación de las políticas aplicadas

- **Política DROP:** Al establecer iptables -P INPUT DROP, cualquier paquete que no coincida exactamente con las reglas de Web, FTP o SSH será destruido automáticamente.
- **Seguridad SSH:** Aunque el script abre el puerto 22 para todos, recuerda que podrías restringirlo a una IP específica como hicimos antes cambiando la línea por: iptables -A INPUT -p tcp -s TU\_IP --dport 22 -j ACCEPT.
- **Conexiones Establecidas:** La regla ESTABLISHED,RELATED es técnica pero esencial; sin ella, si tú haces un apt update o descargas algo, los paquetes de respuesta serían bloqueados por la política DROP de la cadena INPUT.

**¿Te gustaría que añadiera una línea al script para que estas reglas se guarden automáticamente y persistan después de reiniciar el sistema?**

rehaz el script, pero teniendo en cuenta que las iptables están limpias



Entendido. Al tener las **iptables limpias** (sin reglas previas), el script debe ser extremadamente cuidadoso: si aplicamos la política DROP por defecto antes de permitir el tráfico de gestión, podrías perder la conexión con el servidor inmediatamente.

He diseñado este script para que se ejecute de forma segura, estableciendo primero las excepciones y, finalmente, cerrando el acceso al resto.

## Script de Configuración: seguridad\_servidor.sh

```
#!/bin/bash

# =====
# SCRIPT DE CONFIGURACIÓN IPTABLES (DESDE CERO)
# =====

# 1. Aseguramos que el tráfico interno (loopback) esté permitido
# Esto es vital para que los servicios del sistema hablen entre sí.
```

```

iptables -A INPUT -i lo -j ACCEPT

# 2. Permitir el tráfico de conexiones ya iniciadas (ESTABLISHED)
# Esto garantiza que no cortemos sesiones activas mientras aplicamos
reglas.
iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

# 3. REQUISITO: Acceso al SERVIDOR WEB
# Abrimos HTTP y HTTPS para todo el exterior.
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT

# 4. REQUISITO: Acceso al SERVIDOR FTP
# Abrimos el puerto de control (21) y el puerto de datos pasivo (56123).
iptables -A INPUT -p tcp --dport 21 -j ACCEPT
iptables -A INPUT -p tcp --dport 56123 -j ACCEPT

# 5. REQUISITO: Acceso al SERVIDOR SSH
# Abrimos el puerto 22 para permitir la administración remota.
iptables -A INPUT -p tcp --dport 22 -j ACCEPT

# 6. REQUISITO: POLÍTICA POR DEFECTO (EL CERROJO)
# Ahora que ya hemos permitido lo necesario, denegamos todo lo demás.
# Cualquier paquete que no cumpla las reglas anteriores será descartado.
iptables -P INPUT DROP
iptables -P FORWARD DROP

# La salida la dejamos permitida por defecto para que el servidor pueda
navegar.
iptables -P OUTPUT ACCEPT

echo "-----"
echo " Configuración aplicada con éxito."
echo " Puertos abiertos: 21, 22, 80, 443, 56123"
echo " Política de entrada: DROP (Todo lo demás bloqueado)"
echo "-----"

```

## Por qué este orden es el más seguro

En un entorno con las tablas limpias, el orden de los factores sí altera el producto:

- 1. Primero el tráfico local e hilos activos:** Evita que el servidor se "bloquee a sí mismo" en procesos internos.

2. **Apertura de puertos:** Al añadir las reglas de ACCEPT primero, nos aseguramos de que cuando llegue el comando de la política DROP, ya existan "puertas abiertas" para nosotros.
3. **La política DROP al final:** Actúa como un muro que solo deja pasar lo que definimos arriba.

## Verificación detallada

Una vez ejecutado el script, puedes ver el resultado con el comando que hemos practicado:

```
sudo iptables -L -n -v --line-numbers
```

En la parte superior verás Chain INPUT (policy DROP), lo cual confirma que tu primer requisito se está cumpliendo. Debajo, verás las reglas para los puertos 80, 443, 21, 56123 y 22.

**¿Te gustaría que probáramos a añadir una regla de "log" al final del script para que puedas ver en un archivo quién intenta entrar a puertos que no son los permitidos?**

y como hago para que este script se aplique incluso en caso de reinicio del servidor?



Para que las reglas de `iptables` sean persistentes, es decir, que se vuelvan a cargar automáticamente tras un reinicio, tienes varias opciones dependiendo de tu sistema operativo. Por defecto, `iptables` pierde toda la configuración cuando el servidor se apaga.

Aquí tienes los métodos más efectivos:

### 1. El método estándar (Paquete `iptables-persistent`)

Es la forma más sencilla y recomendada en sistemas basados en **Debian o Ubuntu**.

#### 1. Instala la herramienta:

```
sudo apt update  
sudo apt install iptables-persistent
```

*Durante la instalación, te preguntará si quieres guardar las reglas actuales. Dile que Sí.*

#### 2. Guardar cambios futuros: Si después de instalarlo haces cambios en el script o en la terminal y quieres que se graben, ejecuta:

```
sudo netfilter-persistent save
```

### 2. Guardado manual (En cualquier Linux)

Puedes exportar las reglas a un archivo y decirle al sistema que las lea al iniciar.

## 1. Exporta tus reglas actuales:

```
sudo iptables-save > /etc/iptables.rules
```

## 2. Carga automática mediante /etc/network/interfaces: Si usas el gestor de red tradicional, puedes añadir esta línea al final del archivo de configuración de tu interfaz (ej. eth0):

```
pre-up iptables-restore < /etc/iptables.rules
```

---

## 3. Modificación del Script (Añadiendo el guardado)

Si quieras que tu propio script se encargue de "hacer permanentes" las reglas cada vez que lo ejecutes, añade estas líneas al final de tu archivo .sh:

```
# ... (al final de tu script anterior)

# Guardar para sistemas con iptables-persistent
if [ -d /etc/iptables ]; then
    iptables-save > /etc/iptables/rules.v4
    echo "Reglas guardadas en /etc/iptables/rules.v4"
fi

# Guardar para sistemas RedHat/CentOS/Fedora
if [ -f /etc/sysconfig/iptables ]; then
    iptables-save > /etc/sysconfig/iptables
    echo "Reglas guardadas en /etc/sysconfig/iptables"
fi
```

---

## 4. Verificación tras el reinicio

Una vez que hayas aplicado uno de estos métodos, reinicia tu servidor con `sudo reboot`. Al volver a entrar, comprueba que todo sigue ahí:

```
sudo iptables -L -n -v
```

Si ves que la política sigue en **DROP** y tus puertos (21, 22, 80, etc.) siguen listados, ¡lo has configurado correctamente!

**¿Te gustaría que comprobemos si tu sistema utiliza netplan o systemd-networkd para darte una instrucción más precisa sobre dónde colocar la línea de carga?**

he hecho lo que se dice en el manual y no se guarda: Script Personalizado de Reglas

Otra opción para lograr la persistencia es crear un script personalizado donde escribiremos todas las reglas. Este método ofrece mayor control y flexibilidad en la gestión del firewall.

01

Crear el Script

Crear el archivo que contendrá

todas las reglas y guardarlo en

/etc (por ejemplo, /etc/reglas-  
iptables)

02

Asignar Permisos

Asignar permisos de ejecución

mediante el comando: chmod

+x /etc/reglas-iptables

03

Configurar Ejecución

Automática

Añadir la ruta del script en

/etc/rc.local para que se ejecute

al iniciar el sistema

```
#!/bin/sh -e
```

```
# rc.local
```

```
/etc/reglas-iptables
```

```
exit 0
```



El problema por el cual no se guarda o no se aplica es que, en las versiones modernas de Linux (como Ubuntu 20.04, 22.04 o Debian 11/12), el archivo **rc.local** está desactivado por defecto o incluso no existe, ya que ha sido reemplazado por systemd.

Para que tu método funcione, necesitamos asegurarnos de que **rc.local** tenga vida y los permisos correctos. Sigue estos pasos para solucionarlo:

## 1. Asegúrate de que el script tiene el "Shebang"

Tu archivo /etc/reglas-iptables **debe** empezar con esta línea en la parte superior, de lo contrario el sistema no sabrá cómo interpretarlo:

```
#!/bin/bash  
# Tus reglas de iptables aquí...
```

## 2. Verifica y habilita rc.local

Si el archivo /etc/rc.local no existía y lo creaste tú, es muy probable que el sistema no lo esté "escuchando". Hagamos lo siguiente:

### 1. Dale permisos de ejecución al propio rc.local:

```
sudo chmod +x /etc/rc.local
```

### 2. Activa el servicio de compatibilidad (Systemd): Ejecuta estos comandos para decirle a Linux que use rc.local al arrancar:

```
sudo systemctl enable rc-local  
sudo systemctl start rc-local
```

### 3. Comprueba el estado:

```
sudo systemctl status rc-local
```

*Si aparece en verde (active/running), tu script se ejecutará en el próximo reinicio.*

---

## 3. El error común en el contenido de rc.local

Revisa que tu /etc/rc.local se vea exactamente así. Es vital que el exit 0 esté al final y que la ruta a tu script sea correcta:

```
#!/bin/sh -e  
  
# Ejecutar el script de iptables  
/etc/reglas-iptables  
  
exit 0
```

---

## ¿Por qué sigue fallando a veces?

Si después de esto sigue sin funcionar, puede ser por dos razones:

- **Rutas absolutas:** Dentro de tu script /etc/reglas-iptables, intenta usar rutas completas para el comando iptables (ejemplo: /sbin/iptables en lugar de solo iptables). Puedes saber la ruta exacta con which iptables.
- **Conflictos con otros gestores:** Si tienes instalado ufw (Uncomplicated Firewall), este podría estar sobrescribiendo tus reglas al arrancar. Desactívalo con sudo ufw disable.

## Una alternativa más moderna (Systemd)

Si rc.local te sigue dando problemas, la forma "correcta" hoy en día es crear un pequeño servicio de systemd. Es más robusto:

1. Crea el archivo: sudo nano /etc/systemd/system/firewall.service
2. Pega esto:

```
[Unit]
Description=Mi Script de Iptables
After=network.target

[Service]
Type=oneshot
ExecStart=/etc/reglas-iptables
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

3. Actívalo: sudo systemctl enable firewall.service

**¿Te gustaría que probemos a crear este servicio de systemd juntos, ya que es mucho más fiable que el antiguo rc.local?**

C

Gemini puede cometer errores, incluso sobre personas, así que verifica sus respuestas. [Tu privacidad y GeminiSe abre en una ventana nueva](#)