

Lesson 01 — Packet Filtering

From iptables to nftables, beginner edition.

Linux Networking

In this lesson you will learn how the Linux kernel inspects and filters network packets using the **netfilter** framework, how to write and read **iptables** rules, what **conntrack** is and why it matters, and how the newer **nftables** tool improves on the classic toolchain.

What is a firewall on Linux?

- **A firewall = packet inspector + decider** — it looks at each packet and decides: allow, drop, or rewrite?
- **On Linux, the inspector lives in the kernel: netfilter** — a set of hooks deep inside the networking stack.
- **iptables/nftables are just front-ends** — they load rules into netfilter. The kernel does the actual filtering.
- No firewall daemon needs to run continuously — the rules are stored in kernel memory.

Plain English

Think of the Linux kernel as a post office. Every packet is a letter passing through. *Netfilter* is the sorting machine that can inspect, redirect, or shred any letter. *iptables* and *nftables* are the tools you use to program that sorting machine — they just write the instructions, they don't run them.

The 5 netfilter hooks

A packet's journey through netfilter



- **PREROUTING** — first hook, before routing decision; used for DNAT
- **INPUT** — packets destined for the local machine
- **FORWARD** — packets routed through this machine (router/gateway)
- **OUTPUT** — packets generated by the local machine

- **POSTROUTING** — last hook, after routing decision; used for SNAT/MASQUERADE
-

Tables vs chains

iptables organises rules into **tables** (purpose) and **chains** (hook attachment points).

Table	What it does	Typical chains	Example rule
<code>filter</code>	Accept, drop, or reject packets — the default firewall table	INPUT, FORWARD, OUTPUT	<code>-A INPUT -p tcp --dport 22</code> <code>-j ACCEPT</code>
<code>nat</code>	Rewrite source or destination IP/port (SNAT, DNAT, MASQUERADE)	PREROUTING, OUTPUT, POSTROUTING	<code>-t nat -A POSTROUTING -o</code> <code>eth0 -j MASQUERADE</code>
<code>mangle</code>	Modify packet headers (TTL, TOS, mark for routing) without rewriting addresses	All 5 chains	<code>-t mangle -A PREROUTING -j</code> <code>MARK --set-mark 1</code>
<code>raw</code>	Bypass conntrack for specific traffic (performance)	PREROUTING, OUTPUT	<code>-t raw -A PREROUTING -p</code> <code>udp --dport 53 -j NOTRACK</code>

Notes: The `filter` table is the default — when you omit `-t`, you are working in `filter`. Rules are evaluated top-to-bottom; first match wins.

Stateful inspection (conntrack)



- **NEW** — first packet of a connection (e.g. SYN); conntrack creates a tracking entry
- **ESTABLISHED** — packets belonging to an already-opened connection; allowed through without re-inspection
- **RELATED** — packets related to an existing connection (e.g. ICMP errors, FTP data channel)

- **INVALID** — packet doesn't match any known connection and looks malformed; should be dropped
-

iptables crash course

Three commands every sysadmin needs to know:

List all rules with packet/byte counters:

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

The `-v` flag shows interface and packet/byte counters. `-n` prevents slow DNS lookups — always use it.

Allow inbound SSH:

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

Appends (`-A`) a rule to the INPUT chain: TCP on port 22 → ACCEPT. The `-A` adds to the end; use `-I` to insert at a specific position.

Set default policy to drop all inbound:

```
iptables -P INPUT DROP
```

The policy is the fallback when no rule matches. Set this *after* adding ACCEPT rules for needed services, or you will lock yourself out.

Reading a rule like a sentence

Anatomy of an iptables rule

```
iptables -A INPUT -p tcp --dport 22 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
```

- `iptables` — the tool that writes rules into the kernel's netfilter tables
- `-A INPUT` — **Append** to the **INPUT** chain (packets destined for this machine)
- `-p tcp` — match only **TCP** packets
- `--dport 22` — match destination port **22** (SSH)
- `-m conntrack --ctstate NEW,ESTABLISHED` — load the conntrack module and match packets in state **NEW** or **ESTABLISHED**
- `-j ACCEPT` — **jump** to target ACCEPT (let the packet through)

Read it aloud: *"For all TCP packets arriving at port 22 that are new or part of an established connection, accept them."*

NAT in action

SNAT (outgoing): POSTROUTING rewrites SOURCE IP

src=10.0.0.5
dst=8.8.8.8

POSTROUTING

DNAT (incoming): PREROUTING rewrites DEST IP

PREROUTING

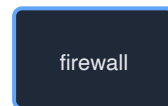
src=8.8.8.8 →
dst=203.0.113.10

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

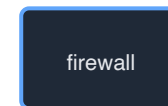
MASQUERADE is dynamic SNAT — it automatically uses the current IP of the outgoing interface.
Ideal for dynamic IP addresses (home routers, VMs sharing a host NIC).

DROP vs REJECT

DROP — silent void



REJECT — ICMP unreachable



DROP

Packet is silently discarded. The sender gets no response and must wait for a timeout. Harder to fingerprint your firewall, but slower failure for legitimate clients.

REJECT

Packet is discarded AND an ICMP "port unreachable" (or TCP RST) is sent back. Sender knows immediately the port is closed — faster failure, but reveals firewall presence.

```
iptables -A INPUT -p tcp --dport 23 -j DROP
```

```
iptables -A INPUT -p tcp --dport 23 -j REJECT \  
--reject-with tcp-reset
```

nftables — what changed and why

nftables replaced iptables as the recommended Linux firewall tool starting with kernel 3.13 (2014), and became the default in Debian 10 / RHEL 8.

- **Single binary** — `nft` replaces four tools: `iptables`, `ip6tables`, `arptables`, `ebtables`. One syntax for IPv4, IPv6, ARP, and bridge filtering.
- **Native sets** — you can match against a set of IPs, ports, or prefixes with a single rule instead of a loop of rules: `ip saddr @blocked_ips drop`. Performance scales sub-linearly.
- **Atomic reloads** — the entire ruleset is loaded as one transaction: `nft -f rules.nft`. No window where partial rules leave the machine exposed.
- **Cleaner syntax** — rules read more like prose; the table/chain/rule hierarchy is explicit and user-defined rather than baked into the tool.

Quick start

```
# List all tables and rules
nft list ruleset
```

```
# Load a complete ruleset from a file (atomic)
nft -f /etc/nftables.conf
```

iptables vs nftables — side by side

Task	iptables	nftables
Accept SSH (port 22)	<code>iptables -A INPUT -p tcp --dport 22 -j ACCEPT</code>	<code>tcp dport 22 accept</code>
Drop all ICMP	<code>iptables -A INPUT -p icmp -j DROP</code>	<code>ip protocol icmp drop</code>
NAT MASQUERADE	<code>iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE</code>	<code>ip saddr 10.0.0.0/8 oifname "eth0" masquerade</code>
Match a set of IPs	Requires multiple <code>-A</code> rules or ipset module	<code>ip saddr { 192.168.1.0/24, 10.0.0.0/8 } accept</code>
Reload entire ruleset atomically	Not natively supported; use iptables-restore	<code>nft -f /etc/nftables.conf</code>
IPv6 support	Separate <code>ip6tables</code> command	Same ruleset; use <code>inet</code> table for dual-stack

Both tools write rules into the same kernel netfilter subsystem — they can coexist but may conflict; pick one per production system.

Migration & gotchas

- `iptables-translate` — converts individual iptables rules to nftables syntax. Run `iptables-translate -A INPUT -p tcp --dport 22 -j ACCEPT` to get the equivalent nft line. Not 100% coverage, but a great starting point.
 - **Rule order matters** — iptables uses first-match; nftables uses first-match by default but supports `policy accept/drop` per chain. Moving from one to the other can silently change behaviour if chain policies differ.
 - **Default-allow vs default-deny** — fresh installations often have an ACCEPT policy on all chains. Always audit with `iptables -L -v -n` or `nft list ruleset` before assuming a host is hardened.
 - **conntrack on UDP** — UDP is stateless at the protocol level, but conntrack still tracks UDP "connections" (source/dest 4-tuple). Rules like `-m conntrack --ctstate ESTABLISHED` work for DNS, QUIC, and other UDP protocols.
 - **Persistence** — iptables rules are lost on reboot unless saved with `iptables-save` / `iptables-restore` or via `netfilter-persistent`. nftables rules are loaded from `/etc/nftables.conf` by systemd on boot if the service is enabled.
-

Recap

- **Netfilter is the kernel engine** — iptables and nftables are just front-end tools that write rules into it. Five hooks (PREROUTING → INPUT / FORWARD → OUTPUT → POSTROUTING) define where rules can intercept packets.
- **Conntrack makes rules smarter** — stateful matching (`-m conntrack --ctstate ESTABLISHED,RELATED`) lets you write a single inbound ACCEPT rule instead of one per response port, and is the foundation of NAT.
- **nftables is the future, but iptables is still everywhere** — understand both. Use `iptables-translate` to convert existing rules and `nft list ruleset` to audit nftables configuration.

Lab time

In the lab you will: add, list, and delete iptables rules on a live VM; observe conntrack state transitions with `conntrack -L`; write an equivalent nftables ruleset; and test DROP vs REJECT behaviour with `netcat`.

Coming up: Lesson 02 — How Linux Decides Where Packets Go (routing tables, ip route, policy routing, and Linux as a router).
