

Firewall & Access Control Policies

Educational Use Only: The firewall rules and access control configurations in this lesson are intended for learning purposes on systems you own or are authorized to manage. Misconfigured firewall rules can lock you out of servers or block legitimate traffic. Always test changes in a staging environment before applying them to production systems.

Firewall Fundamentals

A **firewall** is a network security device — hardware, software, or both — that monitors and controls incoming and outgoing network traffic based on predetermined security rules. Firewalls establish a barrier between trusted internal networks and untrusted external networks such as the internet. Understanding the three primary filtering approaches is essential for any blue team operator.

Packet Filtering

The simplest form of firewalling operates at the network and transport layers (OSI layers 3 and 4). A packet-filtering firewall examines each packet in isolation, comparing its source IP, destination IP, source port, destination port, and protocol against a set of rules. If a packet matches an allow rule, it passes; if it matches a deny rule or no rule at all, it is dropped. Packet filters are fast and lightweight, but they have no awareness of connection state or application-layer content. A classic example is a router ACL that permits TCP port 80 inbound while blocking everything else.

Stateful Inspection

Stateful firewalls improve on packet filtering by tracking the state of active connections. When an internal host initiates a TCP connection to an external server, the firewall records the connection in a state table. Return traffic from that server is automatically permitted because the firewall recognizes it as part of an established session. This eliminates the need for explicit rules allowing return traffic and prevents attackers from

injecting packets that do not belong to a legitimate connection. Modern Linux firewalls (iptables and nftables) are stateful by default when you use the `conntrack` module.

Application-Layer Filtering

Application-layer firewalls (also called proxy firewalls or deep packet inspection engines) operate at OSI layer 7. They understand the semantics of protocols like HTTP, DNS, and SMTP. This enables them to block SQL injection attempts inside HTTP POST bodies, prevent DNS tunneling, or reject email attachments with dangerous file types. Web Application Firewalls (WAFs) such as ModSecurity, AWS WAF, and Cloudflare WAF are the most common examples in modern infrastructure.

iptables and nftables

On Linux systems, the kernel's **Netfilter** framework provides the underlying packet processing engine. Two user-space tools interact with Netfilter: the legacy `iptables` and its modern replacement `nftables`.

iptables: Chains, Tables, and Rules

iptables organizes rules into **tables** (filter, nat, mangle, raw) and **chains** (INPUT, OUTPUT, FORWARD, plus user-defined chains). The `filter` table is the default and most commonly used. Each chain is an ordered list of rules; packets traverse the chain top-to-bottom until a matching rule is found.

```
# Set default policies: drop everything, then whitelist
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

# Allow loopback traffic
iptables -A INPUT -i lo -j ACCEPT

# Allow established and related connections (stateful)
iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

# Allow SSH from a specific management subnet
iptables -A INPUT -p tcp --dport 22 -s 10.0.1.0/24 -j ACCEPT

# Allow HTTP and HTTPS from anywhere
```

```
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT

# Allow ICMP ping (rate-limited to prevent flood)
iptables -A INPUT -p icmp --icmp-type echo-request -m limit --limit 5/sec -j ACCEPT

# Log dropped packets for analysis
iptables -A INPUT -j LOG --log-prefix "IPT-DROP: " --log-level 4
iptables -A INPUT -j DROP
```

Rule ordering matters: iptables evaluates rules sequentially. Place more specific rules before general ones. A broad ACCEPT rule placed too early will bypass all subsequent restrictions.

nftables: The Modern Replacement

nftables provides a cleaner syntax, better performance, and a unified framework for IPv4, IPv6, and ARP filtering. Most modern distributions ship with nftables as the default.

```
#!/usr/sbin/nft -f

flush ruleset

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

        # Allow loopback
        iif "lo" accept

        # Allow established/related connections
        ct state established,related accept

        # Allow SSH from management subnet
        ip saddr 10.0.1.0/24 tcp dport 22 accept

        # Allow HTTP and HTTPS
        tcp dport { 80, 443 } accept

        # Rate-limited ICMP
```

```
icmp type echo-request limit rate 5/second accept

# Log and drop everything else
log prefix "NFT-DROP: " drop
```

UFW: The Simplified Frontend

Uncomplicated Firewall (UFW) is a user-friendly command-line frontend for iptables, widely used on Ubuntu and Debian systems. It abstracts away the complexity of raw iptables commands while still generating proper Netfilter rules underneath.

```
# Enable UFW with default deny incoming, allow outgoing
ufw default deny incoming
ufw default allow outgoing
ufw enable

# Allow SSH from management subnet only
ufw allow from 10.0.1.0/24 to any port 22 proto tcp

# Allow HTTP and HTTPS from anywhere
ufw allow 80/tcp
ufw allow 443/tcp

# Allow a specific application profile
ufw allow "Nginx Full"

# Deny a specific IP address
ufw deny from 192.168.1.100

# Check current rules
ufw status verbose

# Delete a rule by number
ufw status numbered
ufw delete 3
```

UFW is excellent for single-server configurations, but for complex multi-zone networks or high-performance packet processing, direct nftables configuration is preferred.

Cloud Firewalls

Cloud providers implement firewalls as software-defined constructs that attach to virtual resources. The concepts are similar to on-premise firewalls, but the implementation details differ significantly.

AWS Security Groups

Security Groups are stateful virtual firewalls that control traffic at the instance level. They default to deny all inbound and allow all outbound. Rules specify only allow actions — there are no explicit deny rules.

```
{
  "GroupName": "web-server-sg",
  "Description": "Security group for web servers",
  "IpPermissions": [
    {
      "IpProtocol": "tcp",
      "FromPort": 80,
      "ToPort": 80,
      "IpRanges": [{"CidrIp": "0.0.0.0/0", "Description": "HTTP from anywhere"}],
    },
    {
      "IpProtocol": "tcp",
      "FromPort": 443,
      "ToPort": 443,
      "IpRanges": [{"CidrIp": "0.0.0.0/0", "Description": "HTTPS from anywhere"}],
    },
    {
      "IpProtocol": "tcp",
      "FromPort": 22,
      "ToPort": 22,
      "IpRanges": [{"CidrIp": "10.0.1.0/24", "Description": "SSH from mgmt subnet"}],
    },
  ],
  "IpPermissionsEgress": [
    {
      "IpProtocol": "-1",
```

Azure NSGs and GCP Firewall Rules

Azure Network Security Groups (NSGs) operate similarly to AWS Security Groups but support both allow and deny rules with explicit priority ordering. **GCP firewall rules** are applied at the VPC level and use tags or service accounts to target instances. Both support stateful connection tracking. The key difference across all three clouds is scope: AWS SGs are per-instance, Azure NSGs attach to subnets or NICs, and GCP rules apply across the VPC with tag-based targeting.

Access Control Lists (ACLs)

ACLs extend beyond firewalls into multiple domains of security. Understanding the different types is critical for a comprehensive defense posture.

Network ACLs

Network ACLs operate at the subnet level in cloud environments (AWS NACLs) or on routers and switches in traditional networks. Unlike Security Groups, NACLs are **stateless** — you must write rules for both inbound and outbound traffic. Rules are evaluated by priority number, lowest first.

Filesystem ACLs

POSIX ACLs and Windows NTFS ACLs control file and directory access beyond the basic owner/group/other permission model. On Linux, the `setfacl` and `getfacl` commands manage extended ACLs:

```
# Grant read access to a specific user on a file
setfacl -m u:analyst:r /var/log/auth.log

# Grant read/execute to a group on a directory (recursive)
setfacl -R -m g:security-team:rx /opt/security-tools/

# View ACLs on a file
getfacl /var/log/auth.log

# Remove a specific ACL entry
setfacl -x u:analyst /var/log/auth.log
```

Role-Based Access Control (RBAC)

RBAC assigns permissions to roles rather than individual users. Users are then assigned to roles based on their job function. This model simplifies permission management in organizations of any size. Kubernetes RBAC, AWS IAM policies, and Azure RBAC are all implementations of this pattern. The principle of **least privilege** dictates that each role should grant only the minimum permissions required for its function.

Zero Trust Architecture

Traditional network security follows a "castle and moat" model: everything inside the perimeter is trusted, and the firewall protects the boundary. **Zero Trust** rejects this assumption entirely. Its core principle is *"never trust, always verify."*

Zero Trust architecture is built on several pillars:

- **Verify explicitly:** Authenticate and authorize every request based on all available data points — identity, location, device health, data classification, and anomalies.
- **Least-privilege access:** Limit user and service access to the minimum required. Use just-in-time (JIT) and just-enough-access (JEA) policies.
- **Assume breach:** Design systems as if the network is already compromised. Segment access, encrypt end-to-end, and monitor continuously.
- **Micro-segmentation:** Divide the network into small zones with independent access controls. Lateral movement between zones requires explicit authentication.

In practice, Zero Trust means that a database server does not trust a web server simply because they share a VLAN. The web server must authenticate with a short-lived token, the connection must be encrypted, and the database firewall must restrict access to only the specific queries the web application needs.

AI-Assisted Firewall Management

Firewall rulesets in production environments often grow to hundreds or thousands of rules over time. Manual review becomes impractical, and misconfigurations accumulate. LLMs like Claude can serve as powerful assistants for firewall management tasks.

Reviewing Rules for Conflicts

You can paste a firewall ruleset into Claude and ask it to identify redundant rules, shadowed rules (rules that never match because a broader rule appears earlier), and overly permissive entries:

```
import anthropic

client = anthropic.Anthropic()

ruleset = """
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -s 10.0.1.0/24 -j ACCEPT
iptables -A INPUT -p tcp -j ACCEPT
iptables -A INPUT -p tcp --dport 3306 -s 10.0.2.0/24 -j ACCEPT
iptables -A INPUT -j DROP
"""

message = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=1024,
    messages=[{
        "role": "user",
        "content": (
            "Analyze this iptables ruleset for security issues. Identify "
            "shadowed rules, overly permissive rules, and rule ordering "
            f"problems. Suggest fixes.\n\n{ruleset}"
        )
    }]
)

print(message.content[0].text)
```

Key Insight: In the ruleset above, rule 2 (SSH from 10.0.1.0/24) is shadowed by rule 1 (SSH from anywhere), and rule 3 (accept all TCP) renders rule 4 (MySQL from 10.0.2.0/24) meaningless since all TCP is already accepted. Claude can spot these issues instantly.

Generating Policies from Natural Language

Instead of memorizing syntax, you can describe your requirements in plain English and have Claude generate the corresponding firewall rules for your target platform —

iptables, nftables, UFW, AWS Security Groups, or any other format. Always review and test generated rules before deployment.

Firewall Rule Ordering and Optimization

Rule ordering directly impacts both security and performance. Key principles include:

- **Most specific first:** Place narrow rules (specific IPs, specific ports) before broad rules.
- **Most frequently matched first:** For performance, place rules that match the most traffic near the top of the chain so packets do not traverse unnecessary rules.
- **Explicit deny last:** End every chain with a default deny (drop) rule. Never rely on implicit behavior.
- **Group by function:** Organize rules into logical sections (management access, application traffic, monitoring, logging) using comments or user-defined chains.

Common Mistakes

Even experienced administrators make firewall configuration errors. These are the most frequent and dangerous:

- **Overly permissive rules:** Rules like `iptables -A INPUT -p tcp -j ACCEPT` effectively disable the firewall for all TCP traffic. Every rule should specify the minimum required scope.
- **Forgotten temporary rules:** Rules added during debugging or maintenance ("just let everything through for now") that are never removed. Implement a process for tracking and expiring temporary rules.
- **Implicit allows:** Some platforms default to allow. If you add rules without changing the default policy, you may think traffic is filtered when it is not.
- **Missing egress filtering:** Many administrators focus exclusively on inbound rules and leave outbound traffic unrestricted. This allows compromised hosts to exfiltrate data or communicate with command-and-control servers.
- **Ignoring IPv6:** Configuring iptables for IPv4 while leaving ip6tables wide open. If your system has IPv6 enabled, it needs firewall rules too.

- **No logging:** Dropping packets silently makes troubleshooting impossible. Always log denied traffic, at minimum.

Key Takeaways

A firewall is only as strong as its ruleset. Well-ordered, minimal, regularly audited rules are the foundation of network defense. Zero Trust principles ensure that even if the perimeter is breached, lateral movement is contained.

- Understand the three filtering layers: packet filtering, stateful inspection, and application-layer filtering.
- Master iptables/nftables syntax and know when to use UFW for simpler deployments.
- Cloud firewalls (Security Groups, NSGs, GCP firewall rules) have different scoping and statefulness characteristics.
- ACLs apply to networks, filesystems, and identity systems — RBAC is the standard for scalable permission management.
- Zero Trust architecture eliminates implicit trust and requires continuous verification.
- Use AI to review, audit, and generate firewall rules — but always validate before deployment.
- Avoid common pitfalls: overly permissive rules, forgotten temporary rules, missing egress filtering, and ignored IPv6.

Self-check: Can you explain the difference between stateful and stateless firewalls? Can you write an nftables ruleset from scratch that allows only SSH, HTTP, and HTTPS? Can you describe three principles of Zero Trust architecture? Can you identify a shadowed rule in an iptables chain?

CTF Challenge Resources

Audit these configurations and fix all security issues. Use AI to help identify conflicts and generate corrected rules.

Download Files

Each file contains multiple intentional misconfigurations, policy violations, and security gaps to identify and remediate. All files are plain text (.txt) and safe to open.

↓ [iptables-rules.txt](#) ~3 KB

↓ [aws-security-groups.txt](#) ~6 KB

↓ [network-diagram.txt](#) ~7 KB

↓ [acl-audit.txt](#) ~7 KB