

Firewall & Access Control Policies

Building, auditing, and automating network defense policies with AI assistance — from iptables rules to Zero Trust architecture.

⚠ Educational Use Only

Firewall and access control configurations should only be modified on systems you own or are authorized to manage. Incorrect firewall rules can block legitimate traffic.

Firewalls

ACLs

Zero Trust

WAF

AI-Assisted Policy

Today's Agenda

1. **Firewall fundamentals** — packet filtering, stateful, application-layer
 2. **iptables / nftables** — rule writing with AI assistance
 3. **Cloud security groups** — AWS SGs, Azure NSGs, GCP firewall rules
 4. **Web Application Firewalls** — ModSecurity, AWS WAF, Cloudflare
 5. **Zero Trust architecture** — principles and implementation
 6. **AI-assisted policy generation** — natural language to rules
 7. **Network segmentation** — AI-optimized zone design
 8. **Access control models** — RBAC, ABAC, MAC
 9. **Policy auditing** — finding overly permissive rules with AI
 10. **CTF Challenge** — defend a network with firewall rules
-

Firewall Fundamentals

A firewall is a network security device that monitors and filters incoming and outgoing traffic based on a defined set of security rules. It acts as the first line of defense between trusted internal networks and untrusted external networks.

Three Generations of Firewalls

- **Packet Filtering (1st gen)** — inspects individual packets against rules based on IP, port, protocol. Stateless — no memory of previous packets.
 - **Stateful Inspection (2nd gen)** — tracks active connections in a state table. Understands that a reply packet belongs to an established connection.
 - **Application-Layer / NGFW (3rd gen)** — deep packet inspection (DPI), understands HTTP, DNS, TLS. Can block specific URLs, malware signatures, or application behaviors.
-

Packet Filtering

The simplest firewall type. Each packet is examined independently against an ordered list of rules. No awareness of connection state or application context.

Strengths

- Fast — minimal processing overhead
- Transparent to users and applications
- Works at network/transport layers

Weaknesses

- Cannot inspect payload content
- Vulnerable to IP spoofing
- No understanding of application protocols
- Cannot handle fragmented packets well

Notes: Classic example: early router ACLs. Still used as a fast first-pass filter in modern systems.

Stateful Inspection

Stateful firewalls maintain a **connection table** that tracks the state of every active session (NEW, ESTABLISHED, RELATED, INVALID).

State	Meaning	Example
NEW	First packet of a connection	SYN packet initiating TCP handshake
ESTABLISHED	Part of an active connection	Subsequent packets after SYN/ACK
RELATED	Associated with an existing connection	FTP data channel from control channel
INVALID	Doesn't match any known connection	Orphaned ACK, malformed packet

Why Stateful Matters

You can allow outbound traffic and only permit inbound packets that are part of ESTABLISHED or RELATED connections — dramatically reducing attack surface.

Application-Layer Firewalls & NGFW

Next-Generation Firewalls combine traditional firewall capabilities with deep packet inspection, intrusion prevention, and application awareness.

- **Deep Packet Inspection (DPI)** — examines packet payloads, not just headers
- **Application identification** — recognizes apps regardless of port (e.g., Slack on port 443)
- **TLS inspection** — decrypts, inspects, and re-encrypts HTTPS traffic
- **Integrated IPS** — blocks known exploits and attack patterns inline
- **User identity awareness** — applies policies per user/group, not just per IP

Notes: Leading NGFW vendors: Palo Alto, Fortinet, Cisco Firepower, Check Point. Open-source: OPNsense, pfSense with Suricata/Snort.

iptables: Linux Firewall Foundation

iptables is the traditional Linux packet filtering framework. Rules are organized into **tables** (filter, nat, mangle) and **chains** (INPUT, OUTPUT, FORWARD).

Rule Anatomy

```
# Structure: iptables -t TABLE -A CHAIN [match criteria] -j TARGET

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

# Allow established/related connections
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# Drop all other incoming traffic (default deny)
iptables -A INPUT -j DROP
```

Rules are evaluated **top-to-bottom**. First match wins. Order matters enormously.

Realistic iptables Ruleset: Web Server

```
#!/bin/bash
# Flush existing rules
iptables -F
iptables -X

# Default policies: deny all incoming, allow outgoing
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

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

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

nftables: The Modern Replacement

nftables replaces iptables with a cleaner syntax, better performance, and unified IPv4/IPv6 handling. It is the default on Debian 10+, RHEL 8+, and Ubuntu 20.04+.

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

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

    # Allow loopback
    iif "lo" accept

    # Track connections
    ct state established,related accept
    ct state invalid drop

    # SSH from management only
```

AI-Assisted iptables/nftables Rule Writing

Instead of memorizing complex syntax, describe your desired firewall behavior to an AI and have it generate rules.

Prompt Example

"Generate nftables rules for a database server that: allows PostgreSQL (5432) only from the app-server subnet 10.0.2.0/24, allows SSH from the bastion host 10.0.100.5, blocks all other inbound traffic, rate-limits ICMP, and logs dropped connections."

AI Workflow for Firewall Rules

1. **Describe intent** in natural language — what services, who accesses them
2. **AI generates rules** with proper syntax and ordering
3. **Review critically** — check for missing rules, ordering issues, overly broad matches

4. **Test in staging** — never deploy AI-generated rules directly to production

5. **Ask AI to audit** — "Are there any security gaps in these rules?"

AWS Security Groups

AWS Security Groups are **stateful** virtual firewalls attached to EC2 instances. They allow rules only (no explicit deny) — anything not allowed is implicitly denied.

```
{
  "GroupName": "web-server-sg",
  "Description": "Security group for production web servers",
  "IpPermissions": [
    {
      "IpProtocol": "tcp",
      "FromPort": 443,
      "ToPort": 443,
      "IpRanges": [
        { "CidrIp": "0.0.0.0/0", "Description": "HTTPS from anywhere" }
      ]
    },
    {
      "IpProtocol": "tcp",
      "FromPort": 80,
```

Notes: Key difference from NACLs: Security Groups are stateful (return traffic auto-allowed), NACLs are stateless.

Azure NSGs & GCP Firewall Rules

Azure NSGs

- Applied to **subnets** or **NICs**
- Explicit **allow and deny** rules
- Priority-based (100–4096, lower = higher priority)
- Service tags: `Internet`, `AzureLoadBalancer`, `VirtualNetwork`
- Application Security Groups (ASGs) for role-based grouping

GCP Firewall Rules

- Applied at the **VPC network** level
- Uses **network tags** or service accounts as targets
- Priority-based (0–65535, lower = higher)
- Supports **allow and deny** rules
- Firewall Insights for usage analytics

Cross-Cloud Pattern

All three clouds follow the same principle: **least privilege by default**. Only open what is explicitly needed, reference other security groups/tags instead of IPs when possible.

Cloud Firewall Comparison

Feature	AWS SG	Azure NSG	GCP Firewall
Stateful	Yes	Yes	Yes
Explicit Deny	No (implicit)	Yes	Yes
Attach To	ENI / Instance	Subnet / NIC	VPC (tag targets)
Rule Limit	60 inbound + 60 outbound	1000 per NSG	Varies by quota
Logging	VPC Flow Logs	NSG Flow Logs	Firewall Rules Logging
Identity-Based	SG references	ASGs	Service account targets

Web Application Firewalls (WAF)

A WAF operates at **Layer 7** (application layer), inspecting HTTP/HTTPS requests to block web-specific attacks that network firewalls cannot see.

What WAFs Protect Against

- **SQL injection** — malicious SQL in form fields and URL parameters
- **Cross-Site Scripting (XSS)** — injected JavaScript in user input
- **Command injection** — OS commands in HTTP requests
- **Path traversal** — `../../../../etc/passwd` in URLs
- **DDoS at Layer 7** — HTTP floods, slowloris attacks
- **Bot traffic** — credential stuffing, scraping, abuse

WAF vs. Network Firewall

A network firewall sees packets and ports. A WAF sees HTTP headers, cookies, POST bodies, and query strings. They are complementary — use both.

ModSecurity & OWASP Core Rule Set

ModSecurity is the most widely deployed open-source WAF. It runs as a module inside Apache, Nginx, or IIS and uses the **OWASP Core Rule Set (CRS)** for baseline protection.

```
# ModSecurity Configuration
SecRuleEngine On
SecRequestBodyAccess On
SecResponseBodyAccess Off
SecAuditEngine RelevantOnly
SecAuditLog /var/log/modsec_audit.log

# Block SQL injection in query string
SecRule ARGS "@detectSQLi" \
  "id:1001,\
  phase:2,\
  deny,\
  status:403,\
  msg:'SQL Injection Detected',\
  severity:'CRITICAL'."
```

Cloud WAF Services

AWS WAF

- Integrates with CloudFront, ALB, API Gateway
- **Managed rule groups** from AWS and marketplace
- Custom rules with JSON syntax
- Rate-based rules for DDoS protection
- Bot Control and Account Takeover Prevention

Cloudflare WAF

- Edge-based — blocks attacks before they reach origin
- Managed rulesets (OWASP, Cloudflare Specials)
- Custom firewall rules with Wireshark-like syntax
- Bot management and JS challenge
- Zero-day virtual patching

AWS WAF Rule Example

```
{  
  "Name": "BlockSQLi",  
  "Priority": 1,  
  "Action": { "Block": {} },  
}
```

```
"Statement": {
  "SqlMatchStatement": {
    "FieldToMatch": { "AllQueryArguments": {} },
    "TextTransformations": [
      { "Priority": 0, "Type": "URL_DECODE" },
      { "Priority": 1, "Type": "HTML_ENTITY_DECODE" }
    ]
  }
},
"VisibilityConfig": {
  "SampledRequestsEnabled": true
```

Zero Trust Architecture

"Never trust, always verify." Zero Trust assumes no implicit trust based on network location. Every request must be authenticated, authorized, and encrypted — regardless of origin.

Core Principles (NIST SP 800-207)

- **Verify explicitly** — authenticate and authorize every access request using all available data points (identity, location, device health, data classification)
- **Least privilege access** — limit user access with just-in-time and just-enough-access (JIT/JEA)
- **Assume breach** — minimize blast radius through microsegmentation, end-to-end encryption, and continuous monitoring

Zero Trust vs. Perimeter Security

Traditional: "Castle and moat" — trust everything inside the network. Zero Trust: every resource is behind its own access gate, regardless of network position.

Implementing Zero Trust

Key Components

Component	Purpose	Tools
Identity Provider	Strong authentication (MFA, SSO)	Okta, Azure AD, Google Workspace
Device Trust	Verify device health and compliance	Intune, Jamf, CrowdStrike
Microsegmentation	Isolate workloads at granular level	Illumio, VMware NSX, Calico
ZTNA / SDP	Replace VPN with per-app access	Cloudflare Access, Zscaler, Tailscale
Policy Engine	Real-time access decisions	Open Policy Agent (OPA), Cedar
Continuous Monitoring	Detect anomalies, revoke access	SIEM, UEBA, EDR

Notes: Zero Trust is a journey, not a product. Start with identity, then add device trust, then microsegmentation.

AI-Assisted Policy Generation

AI transforms how we create and maintain security policies. Instead of hand-crafting complex rule syntax, describe your security intent and let AI generate the implementation.

Example: Natural Language to AWS Security Group

Prompt

"Create an AWS security group for a three-tier web application: the web tier accepts HTTPS from the internet and talks to the app tier on port 8080; the app tier talks to the database tier on PostgreSQL 5432; the database tier only accepts connections from the app tier. All tiers allow SSH from the 10.0.100.0/24 management subnet."

AI generates:

- Three separate security group definitions with cross-references
- Proper ingress and egress rules per tier

- SG-to-SG references instead of IP ranges where possible
 - Descriptive comments for each rule
-

AI-Generated Three-Tier Security Groups

```
# AI-generated AWS CLI commands for three-tier architecture

# Web Tier SG
aws ec2 create-security-group --group-name web-tier-sg \
  --description "Web tier - accepts HTTPS, talks to app tier"

aws ec2 authorize-security-group-ingress --group-name web-tier-sg \
  --protocol tcp --port 443 --cidr 0.0.0.0/0
aws ec2 authorize-security-group-ingress --group-name web-tier-sg \
  --protocol tcp --port 22 --cidr 10.0.100.0/24

# App Tier SG
aws ec2 create-security-group --group-name app-tier-sg \
  --description "App tier - accepts from web tier, talks to DB tier"
```

Network Segmentation Strategies

Segmentation divides a network into isolated zones to limit lateral movement when a breach occurs. Each zone has its own firewall policies controlling inter-zone traffic.

Common Zone Architecture

Zone	Purpose	Trust Level
DMZ	Public-facing services (web, email, DNS)	Low
Application	Internal app servers, APIs, middleware	Medium
Database	Data stores, caches, message queues	High
Management	Admin access, monitoring, CI/CD	Highest
User	Workstations, BYOD devices	Low-Medium

Zone	Purpose	Trust Level
IoT / OT	Cameras, sensors, SCADA	Untrusted

AI-Optimized Zone Design

AI can analyze network traffic patterns and recommend optimal segmentation by identifying which services actually communicate with each other.

AI-Assisted Approach

1. **Collect flow data** — feed NetFlow/sFlow data or VPC Flow Logs to AI
2. **AI maps communication patterns** — identifies actual service dependencies
3. **Propose zone boundaries** — group services by communication patterns and trust level
4. **Generate inter-zone rules** — create policies that allow required flows, block everything else
5. **Simulate impact** — AI predicts which connections would break before you apply rules

AI Prompt for Segmentation

"Given these VPC Flow Logs showing traffic between 15 services, recommend network zones and firewall rules that minimize lateral movement risk while maintaining all required service communication."

Access Control Models

Access control determines **who** can access **what** resources and **under what conditions**. Three major models dominate enterprise security.

RBAC — Role-Based

Permissions assigned to **roles**, users assigned to roles. Simple, widely adopted.

- `admin` → full access
- `developer` → read/write code, deploy staging
- `viewer` → read-only

ABAC — Attribute-Based

Decisions based on **attributes** of user, resource, environment, and action.

- User: department, clearance, location
- Resource: classification, owner
- Environment: time, IP, device

MAC — Mandatory Access Control

System-enforced labels (e.g., TOP SECRET, SECRET, UNCLASSIFIED). Users cannot override — the system enforces rules based on security labels. Used in military/government (SELinux,

AppArmor).

RBAC Implementation Patterns

```
# RBAC Implementation Example
from functools import wraps

ROLES = {
    "admin":      {"read", "write", "delete", "deploy", "manage_users"},
    "developer": {"read", "write", "deploy_staging"},
    "analyst":    {"read", "run_queries"},
    "viewer":     {"read"},
}

def require_permission(permission):
    """Decorator to enforce RBAC on API endpoints."""
    def decorator(func):
        @wraps(func)
        def wrapper(user, *args, **kwargs):
```

ABAC with Open Policy Agent (OPA)

OPA uses the Rego policy language to make fine-grained, attribute-based access decisions.

```
# policy.rego - ABAC policy for API access
package api.authz

default allow = false

# Allow if user is in the same department as the resource owner
allow {
    input.user.department == input.resource.owner_department
    input.action == "read"
}

# Allow write only during business hours from corporate network
allow {
    input.action == "write"
    input.user.clearance >= input.resource.classification
}
```

Notes: OPA is widely used in Kubernetes (Gatekeeper), service meshes (Envoy), and API gateways for policy-as-code.

Mandatory Access Control: SELinux

SELinux enforces MAC on Linux by labeling every process and file with a security context. Even root cannot bypass SELinux policies.

```
# View SELinux context of a file
$ ls -Z /var/www/html/index.html
system_u:object_r:httpd_sys_content_t:s0 /var/www/html/index.html

# View SELinux context of a process
$ ps -eZ | grep httpd
system_u:system_r:httpd_t:s0 1234 ? 00:00:05 httpd

# Policy: httpd_t can read httpd_sys_content_t
# If an attacker compromises Apache, they STILL can't read
# files labeled with other types (e.g., user_home_t, etc_t)

# Set SELinux to enforcing mode
$ sudo setenforce 1
```

SELinux Prevents Lateral Movement

Even if an attacker gets a shell through a web vulnerability, SELinux confines them to the `httpd_t` domain — they cannot read `/etc/shadow`, write to `/tmp` arbitrarily, or connect to unexpected network ports.

Policy Auditing: Finding Overly Permissive Rules

The most dangerous firewall is one full of rules that were "temporary" three years ago. AI excels at finding security gaps humans overlook.

Common Findings AI Catches

- **0.0.0.0/0 on SSH (port 22)** — SSH exposed to the entire internet
 - **Overly broad port ranges** — `0-65535` allowed where only specific ports needed
 - **Stale rules** — rules for decommissioned servers still active
 - **Redundant rules** — later rules shadowed by earlier, broader rules
 - **Missing egress controls** — no outbound filtering, allowing data exfiltration
 - **Unused security groups** — old SGs with wide-open rules still attached
-

AI-Powered Firewall Audit

Audit Prompt

"Analyze these iptables rules and identify: 1) overly permissive rules, 2) rules that could be tightened, 3) missing best practices, 4) potential bypass techniques an attacker could exploit."

```
# Problematic ruleset for AI to audit
iptables -A INPUT -p tcp --dport 22 -j ACCEPT           # SSH open to world!
iptables -A INPUT -p tcp --dport 3306 -j ACCEPT         # MySQL open to world!
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
iptables -A INPUT -p tcp --dport 8080:8090 -j ACCEPT     # Broad port range
iptables -A INPUT -s 10.0.0.0/8 -j ACCEPT               # Entire /8 trusted
iptables -A INPUT -j DROP
```

AI Audit Findings

1. **Critical:** SSH (22) and MySQL (3306) open to all source IPs
2. **High:** Entire 10.0.0.0/8 trusted — overly broad, should use specific subnets

3. **Medium:** Port range 8080–8090 should be limited to specific ports in use
 4. **Low:** No rate limiting, no logging before the DROP rule
 5. **Missing:** No ESTABLISHED,RELATED rule — stateful tracking not leveraged
-

AI-Remediated Ruleset

After AI audit, the ruleset is tightened with specific source restrictions, rate limiting, logging, and proper connection tracking.

```
#!/bin/bash
# AI-remediated iptables ruleset
iptables -F && iptables -X
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

# Loopback
iptables -A INPUT -i lo -j ACCEPT

# Connection tracking (was missing)
iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -m conntrack --ctstate INVALID -j DROP

# SSH restricted to management subnet with rate limit
```

Automated Compliance Checking

Use AI to continuously verify firewall rules comply with organizational policies and regulatory frameworks.

```
"""AI-assisted firewall compliance checker."""
import json

COMPLIANCE_RULES = {
    "PCI-DSS-1.3.1": {
        "desc": "Restrict inbound traffic to only that which is necessary",
        "check": lambda r: r.get("source") != "0.0.0.0/0"
                    or r["port"] in [80, 443],
    },
    "PCI-DSS-1.3.2": {
        "desc": "Limit inbound internet traffic to DMZ IPs",
        "check": lambda r: r.get("zone") == "dmz"
                    if r.get("source") == "0.0.0.0/0" else True,
    },
    "CIS-9.2.1": {
```

Defense in Depth: Layered Firewall Strategy

No single firewall layer is sufficient. A robust architecture layers multiple types of filtering at different points in the network.

Layer	Technology	What It Catches
Edge / Perimeter	Cloud WAF, CDN firewall	DDoS, bot traffic, known bad IPs
Network Border	NGFW, IPS	Port scans, exploit attempts, C2 traffic
Subnet / VLAN	Security groups, NACLs	Lateral movement, unauthorized zone access
Host	iptables, Windows Firewall	Per-service access control, local policy
Application	WAF, API gateway	SQLi, XSS, API abuse, business logic attacks
Data	Encryption, DLP, MAC	Data exfiltration, unauthorized access

Notes: Each layer compensates for the weaknesses of other layers. An attacker must defeat ALL layers, not just one.

Firewall Logging & Monitoring

Firewall rules without logging are security theater. You must know what is being blocked and what is being allowed.

Essential Log Sources

- **Denied connections** — who is trying to reach what? Recon or misconfiguration?
- **Allowed connections** — sample logging to verify expected traffic patterns
- **State table overflows** — signs of DDoS or resource exhaustion
- **Rule hit counts** — identify unused rules (candidates for removal)
- **Configuration changes** — audit trail for who changed what and when

AI Integration with Firewall Logs

Feed firewall logs to AI to: detect anomalous traffic patterns, identify rules that are never hit (cleanup candidates), and correlate blocked connection attempts with known threat intelligence.

Firewall & ACL Best Practices

1. **Default deny** — start with blocking everything, then allow what is needed
 2. **Least privilege** — open only the ports, protocols, and source IPs required
 3. **Document every rule** — include who requested it, why, and an expiration date
 4. **Use connection tracking** — always leverage ESTABLISHED,RELATED states
 5. **Log before you drop** — but rate-limit logs to avoid disk exhaustion
 6. **Review regularly** — quarterly audits with AI assistance to find stale rules
 7. **Use security group references** — prefer SG-to-SG over IP-based rules in cloud
 8. **Separate management traffic** — SSH/RDP on a dedicated management network
 9. **Test before deploy** — use `iptables -C` to check, `nft --check` to validate
 10. **Version control your rules** — firewall-as-code in Git with PR reviews
-

CTF Challenge: Secure the Network

Scenario

"Design and implement firewall rules to protect this network while maintaining required services."

Network Layout

- **Web Server** (10.0.1.10) — serves HTTPS on port 443, HTTP on port 80
- **App Server** (10.0.2.10) — API on port 8080, receives from web server only
- **Database** (10.0.3.10) — PostgreSQL on 5432, app server only
- **Admin Bastion** (10.0.100.5) — SSH to all servers on port 22
- **DNS Server** (10.0.1.53) — serves DNS (53/udp, 53/tcp) to all internal hosts

Your Tasks

1. Write nftables rules for **each server** enforcing least privilege
 2. Create **AWS security group JSON** for the same architecture
 3. Write a **ModSecurity rule** protecting the web server from SQLi and XSS
 4. Use AI to **audit your rules** — fix any findings
 5. Document your **network segmentation** rationale
-

CTF Challenge: Hints & Evaluation

Key Questions to Ask Yourself

- Can the database server reach the internet? **It should not.**
- Can the web server connect to the database directly? **It should not.**
- Is SSH limited to the bastion host only? **It must be.**
- Are you logging dropped packets? **You should be.**
- Are you using connection tracking? **You must be.**
- Do you have egress rules? **Do not forget outbound filtering.**

Scoring Criteria

Criterion	Points
Default deny policy on all hosts	20
Correct inter-tier traffic flow (no shortcuts)	25

Criterion	Points
SSH restricted to bastion only	15
Connection tracking and logging	15
WAF rules for web server	15
Egress filtering on database tier	10

Lesson 05 Recap

- **Firewalls evolve** — from packet filtering to stateful inspection to application-aware NGFWs
- **iptables/nftables** — Linux host firewalls with powerful rule syntax; AI makes writing them easier
- **Cloud security groups** — AWS SGs, Azure NSGs, GCP rules all follow least-privilege principles
- **WAFs** — Layer 7 protection against SQLi, XSS, and other web attacks (ModSecurity, AWS WAF, Cloudflare)
- **Zero Trust** — never trust, always verify; microsegmentation replaces perimeter trust
- **Access control models** — RBAC for simplicity, ABAC for flexibility, MAC for mandatory enforcement
- **AI-assisted policy** — generate rules from natural language, audit for gaps, automate compliance
- **Defense in depth** — layer firewalls at edge, network, host, and application levels

Next Lesson

Lesson 06: Secrets & Crypto — protecting sensitive data with encryption, key management, and AI-assisted secret detection.
