

Configuration & Infrastructure Weaknesses

Why misconfiguration is consistently the #1 attack vector in modern infrastructure — and how AI accelerates both attack and defense.

What You'll Learn

- Cloud, Kubernetes, Docker, and network misconfigs
- AI-assisted configuration auditing
- Automated scanning tools with AI overlay

Hands-On

- Audit realistic misconfigured infrastructure
- Find escalation paths across cloud resources
- Use AI to triage and prioritize findings

Notes: This lesson bridges traditional infrastructure pentesting with AI-augmented approaches. Ensure students have access to sample config files.

Misconfiguration: The #1 Attack Vector

Year after year, misconfiguration dominates breach statistics. It is not exotic zero-days that compromise organizations — it is default passwords, open buckets, and overly permissive policies.

Source	Finding
OWASP Top 10 (2021)	Security Misconfiguration moved to #5 , up from #6
Verizon DBIR 2024	68% of breaches involved a human element — misconfig is the leading technical cause
IBM Cost of a Breach	Cloud misconfiguration avg. cost: \$4.0M per incident
NSA/CISA Top 10	Misconfigurations ranked #1 in joint advisory (2023)
Palo Alto Unit 42	80% of cloud exposures stem from misconfigs, not vulnerabilities

Notes: Emphasize that these are not theoretical — Capital One (2019), Microsoft (2023), and Toyota (2023) all suffered major misconfig-driven breaches.

Real-World Misconfig Breaches

- **Capital One (2019)** — SSRF exploited an overly permissive IAM role attached to a WAF. 100M+ customer records exposed via EC2 metadata endpoint.
- **Microsoft AI Research (2023)** — 38TB of internal data leaked via a misconfigured Azure SAS token with no expiration and full-account scope.
- **Toyota (2023)** — Cloud misconfiguration exposed vehicle location data of 2.15M customers for over 10 years.
- **Uber (2022)** — Hardcoded credentials in a PowerShell script on a network share led to full domain compromise.
- **Twitch (2021)** — Misconfigured server allowed access to the entire 125GB source code repository.

Common Thread

None of these required a zero-day. Each was a configuration mistake that went undetected — sometimes for years.

The Misconfiguration Attack Surface

Infrastructure misconfigurations span every layer of the stack. Attackers look for the weakest link across all of them simultaneously.

Cloud Layer

- Open storage buckets (S3, GCS, Azure Blob)
- Overly permissive IAM policies
- Exposed metadata endpoints
- Unencrypted data at rest

Container / Orchestration

- Privileged containers
- RBAC bypasses
- Exposed dashboards
- Secrets in environment variables

Network Layer

- Default credentials
- SNMP community strings

Application / Web Server

- Debug mode in production
- Directory listing enabled

- Open management ports
- Unencrypted protocols

- Default error pages with stack traces
 - Missing security headers
-

Cloud Misconfigs: Open S3 Buckets

Public S3 buckets remain one of the most common cloud exposures. A single misconfigured ACL or bucket policy can expose terabytes of sensitive data.

Misconfigured Bucket Policy

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "PublicRead",
    "Effect": "Allow",
    "Principal": "*",
    "Action": [
      "s3:GetObject",
      "s3:ListBucket"
    ],
    "Resource": [
      "arn:aws:s3:::company-backups",
      "arn:aws:s3:::company-backups/*"
    ]
  }]
}
```

What's Wrong

`"Principal": "*"` grants access to **anyone on the internet**. Combined with `ListBucket`, attackers can enumerate every object. The bucket name `company-backups` is also trivially guessable.

Cloud Misconfigs: Overly Permissive IAM

IAM policies that grant excessive permissions are the gateway to privilege escalation in cloud environments.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "*",
    "Resource": "*"
  }]
}
```

This policy grants **full administrator access** to every resource in the AWS account. It is more common than you would expect.

IAM Escalation Paths

- **iam:PassRole + lambda:CreateFunction** — Create a Lambda with an admin role, execute it
- **iam:CreatePolicyVersion** — Write a new policy version granting yourself admin
- **iam:AttachUserPolicy** — Attach AdministratorAccess to your own user

- **sts:AssumeRole** — Assume a more privileged cross-account role
- **ec2:RunInstances + iam:PassRole** — Launch an EC2 instance with an admin instance profile

Notes: Reference the Rhino Security Labs IAM privilege escalation research — they document 20+ escalation techniques.

Cloud Misconfigs: Metadata Endpoints

Cloud instance metadata services (IMDS) provide credentials and configuration to running instances. When accessible via SSRF or misconfigured network policies, they become a goldmine.

```
# AWS IMDSv1 — no authentication required
curl http://169.254.169.254/latest/meta-data/iam/security-credentials/

# Response reveals the IAM role name
curl http://169.254.169.254/latest/meta-data/iam/security-credentials/MyAppRole

# Returns temporary credentials:
# {
#   "AccessKeyId": "ASIA...",
#   "SecretAccessKey": "wJalr...",
#   "Token": "IQoJb3...",
#   "Expiration": "2026-04-09T18:00:00Z"
# }
```

Other Cloud Metadata Endpoints

Provider	Endpoint
AWS	<code>169.254.169.254/latest/meta-data/</code>
GCP	<code>metadata.google.internal/computeMetadata/v1/</code>
Azure	<code>169.254.169.254/metadata/instance?api-version=2021-02-01</code>
DigitalOcean	<code>169.254.169.254/metadata/v1/</code>

Defense: IMDSv2

AWS IMDSv2 requires a PUT request to obtain a session token first, blocking most SSRF attacks. However, many organizations still allow IMDSv1 fallback.

Kubernetes Security: RBAC Bypasses

Kubernetes RBAC is powerful but frequently misconfigured. Overly broad ClusterRoleBindings are the most common issue.

```
# DANGEROUS: Grants cluster-admin to all service accounts
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: permissive-binding
subjects:
- kind: Group
  name: system:serviceaccounts
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

Impact

Every pod in every namespace now has **full cluster-admin privileges**. A single compromised container can read all secrets, create privileged pods, and take over the entire cluster.

Common RBAC Mistakes

- **Wildcard verbs/resources** — `verbs: ["*"]` on `resources: ["*"]`
 - **Default service account abuse** — Pods using the default SA that has been granted extra permissions
 - **Namespace-scoped roles bound cluster-wide** — Using ClusterRoleBinding when RoleBinding was intended
-

Kubernetes: Exposed Dashboards & Privileged Pods

Exposed Dashboard

The Kubernetes Dashboard, when exposed without authentication, provides full cluster management via a web UI. Tesla's 2018 breach started with an unauthenticated dashboard.

```
# Dashboard exposed via LoadBalancer
apiVersion: v1
kind: Service
metadata:
  name: kubernetes-dashboard
  namespace: kubernetes-dashboard
spec:
  type: LoadBalancer # PUBLIC!
  ports:
    - port: 443
      targetPort: 8443
```

Privileged Pod

A privileged pod has full access to the host node, effectively breaking out of the container sandbox.

```
apiVersion: v1
kind: Pod
metadata:
  name: debug-pod
spec:
  hostNetwork: true
  hostPID: true
  containers:
    - name: shell
      image: ubuntu
      securityContext:
        privileged: true
      volumeMounts:
        - name: host-fs
          mountPath: /host
```



With `privileged: true`, `hostPID`, `hostNetwork`, and the host filesystem mounted, this pod has **complete control over the node**.

Kubernetes: Secrets in Environment Variables

Storing secrets as environment variables is a common anti-pattern. Environment variables are visible in process listings, logged by crash reporters, and exposed via the Kubernetes API.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-server
spec:
  template:
    spec:
      containers:
        - name: api
          image: myapp:latest
          env:
            - name: DB_PASSWORD
              value: "SuperSecret123!"          # Hardcoded!
            - name: AWS_SECRET_ACCESS_KEY
              value: "wJalrXUtnFEMI/K7MDENG" # AWS creds in plain text!
```

Why This Is Dangerous

- **Readable via API:** `kubectl get pod api-server -o yaml` exposes all env vars
- **Visible in process table:** `/proc/<pid>/environ` on the node
- **Logged in crash dumps:** Many frameworks dump environment on unhandled exceptions
- **Stored in etcd unencrypted:** Unless encryption-at-rest is configured, etcd stores these in plain text

Better Approach

Use Kubernetes Secrets (with encryption at rest), or external secret managers (Vault, AWS Secrets Manager) via CSI driver or operator.

Docker Misconfigs: Running as Root

By default, Docker containers run as root. If an attacker escapes the container or exploits a volume mount, they get root on the host.

```
# BAD: Running as root (default)
FROM python:3.11
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
# Container process runs as UID 0 (root)!
```

```
# BETTER: Non-root user
FROM python:3.11
RUN groupadd -r appuser && useradd -r -g appuser appuser
COPY --chown=appuser:appuser . /app
WORKDIR /app
RUN pip install -r requirements.txt
USER appuser
CMD ["python", "app.py"]
```

Docker Socket Exposure

```
# docker-compose.yml — NEVER do this
services:
  monitoring:
    image: monitoring-tool
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock # Full Docker API access!
```

Mounting the Docker socket gives the container **full control over the Docker daemon** — it can create privileged containers, access host filesystem, and effectively own the host.

Docker: Image Vulnerabilities & Misconfigs

Vulnerable Docker Compose

```
version: "3.8"
services:
  webapp:
    image: node:14          # EOL version with known CVEs
    ports:
      - "0.0.0.0:3000:3000" # Bound to all interfaces
    environment:
      - NODE_ENV=development # Debug mode in production
      - DATABASE_URL=postgres://admin:password123@db:5432/app
    privileged: true        # Unnecessary privilege
    network_mode: host      # No network isolation

  db:
    image: postgres:12
    environment:
```

Issues Found

- **EOL base image** (node:14) with unpatched vulnerabilities
 - **Credentials in environment** — database URL with plaintext password
 - **Privileged mode** with **host network** — no isolation
 - **Database port exposed** on all interfaces to the public internet
 - **Development mode** enabled — verbose errors, debug endpoints
-

Network Misconfigs: Default Credentials

Default credentials on network devices, management interfaces, and services remain shockingly common even in enterprise environments.

Device / Service	Default Username	Default Password
Cisco IOS	admin	admin / cisco
Tomcat Manager	tomcat	tomcat / s3cret
Jenkins	(no auth)	(open by default)
MongoDB	(no auth)	(binds 0.0.0.0 by default)
Redis	(no auth)	(no password by default)
Elasticsearch	(no auth)	(open by default <v8)
IPMI/BMC	ADMIN	ADMIN

SNMP Community Strings

```
# Default SNMP community strings allow reading device configs
snmpwalk -v2c -c public 10.0.0.1 system
snmpwalk -v2c -c private 10.0.0.1 # "private" = read-write access

# Common default community strings:
# public, private, community, manager, cisco, cable-docsis
```

Notes: SNMP v1/v2c sends community strings in plaintext. SNMPv3 adds authentication and encryption but is often not deployed.

Network Misconfigs: Open Management Ports

Management interfaces exposed to the internet provide direct access to administer critical infrastructure.

Port	Service	Risk
22	SSH	Brute force, key auth bypass
23	Telnet	Cleartext credentials
161/162	SNMP	Device enumeration, config read
445	SMB	EternalBlue, relay attacks
623	IPMI	BMC takeover, hash disclosure
2379	etcd	K8s secrets, cluster state
5900	VNC	Often weak/no auth
6443	K8s API	Cluster takeover

Port	Service	Risk
8080	HTTP Alt	Admin panels, debug endpoints
9090	Prometheus	Metrics, internal topology
10250	Kubelet	Pod exec, node takeover

Reconnaissance Pattern

Attackers scan for these ports using masscan/nmap, then fingerprint services to identify default or weak configurations before attempting exploitation.

AI-Assisted Config Auditing: The Approach

LLMs excel at configuration auditing because they can understand the **semantic intent** of configurations, not just pattern-match known-bad values.

Traditional Scanners

- Rule-based pattern matching
- Known-bad value detection
- Cannot understand context
- High false positive rate
- Misses novel misconfigs

AI-Augmented Auditing

- Understands configuration semantics
- Cross-references multiple files
- Identifies logical contradictions
- Explains risk in context
- Suggests specific remediations

AI Audit Workflow

1. **Collect** — Gather all config files (Terraform, K8s YAML, Docker, nginx, etc.)

2. **Scan** — Run traditional tools first for baseline findings
 3. **Analyze** — Feed configs + scanner output to AI for deeper analysis
 4. **Correlate** — AI identifies cross-service attack paths that tools miss
 5. **Prioritize** — AI ranks findings by actual exploitability, not just severity
-

AI Auditing: Terraform Configuration

Terraform files define infrastructure-as-code. Misconfigurations here propagate to every deployment.

```
# main.tf - Spot the issues
resource "aws_security_group" "web_sg" {
  name = "web-server-sg"

  ingress {
    from_port    = 0
    to_port      = 65535
    protocol     = "tcp"
    cidr_blocks = ["0.0.0.0/0"] # All ports open to internet!
  }

  egress {
    from_port    = 0
    to_port      = 0
    protocol     = "-1"
  }
}
```

Notes: Ask students to count all issues before revealing. There are at least 8 distinct security misconfigurations in this snippet.

AI Audit Prompt: Terraform Analysis

You are a cloud security auditor. Analyze the following Terraform configuration for security misconfigurations. For each finding:

1. Severity (CRITICAL / HIGH / MEDIUM / LOW)
2. Resource affected
3. The specific misconfiguration
4. Attack scenario – how an attacker would exploit this
5. Remediation – the exact Terraform change needed

Also identify any cross-resource attack chains where multiple misconfigurations combine to create a more severe vulnerability.

Rate the overall infrastructure security posture on a scale of 1-10.

--- BEGIN TERRAFORM ---

Sample AI Response (Abbreviated)

CRITICAL — Attack Chain Identified: The security group allows all inbound traffic + IMDSv1 is enabled + admin IAM profile attached = SSRF to full AWS account compromise. An attacker who

finds any web vulnerability can reach the metadata endpoint, steal admin credentials, and pivot to all AWS resources including the unencrypted, publicly accessible RDS database.

AI Auditing: Kubernetes YAML

```
# deployment.yaml - Production API server
apiVersion: apps/v1
kind: Deployment
metadata:
  name: payment-api
  namespace: default          # Using default namespace
spec:
  replicas: 1                  # Single replica for payment service?
  template:
    metadata:
      labels:
        app: payment-api
    spec:
      serviceAccountName: ""   # Uses default SA
      automountServiceAccountToken: true
```

A payment service with **root execution, hardcoded Stripe keys, no resource limits, mutable image tags, and direct internet exposure.**

AI Auditing: nginx / Apache Configs

```
# nginx.conf — Misconfigured reverse proxy
server {
    listen 80;
    server_name api.example.com;
    # Missing: return 301 https://$host$request_uri;

    location / {
        proxy_pass http://backend:3000;
        proxy_set_header X-Forwarded-For $remote_addr;
    }

    # Debug endpoint left in production
    location /debug {
        proxy_pass http://backend:3000;
        # No access restriction!
    }
}
```

Key Findings

- **No HTTPS redirect** — traffic sent in cleartext
- **Debug endpoint** exposed without authentication
- **Alias traversal** — `/static../etc/passwd` path traversal

- **Server version** disclosed via `server_tokens on`
 - **No security headers** — clickjacking, XSS, MIME sniffing all possible
-

AI-Assisted Infrastructure Scanning

Combine traditional scanning tools with AI analysis for comprehensive coverage. The AI layer interprets raw scan output, eliminates false positives, and identifies attack chains.

```
import anthropic
import subprocess
import json

def ai_scan_pipeline(target_configs: dict) -> str:
    """Run scanners, then feed results to AI for analysis."""

    # Step 1: Run traditional scanners
    results = {}
    results["trivy"] = subprocess.run(
        ["trivy", "config", "--format", "json", "./"],
        capture_output=True, text=True
    ).stdout
    results["kube_bench"] = subprocess.run(
        ["kube-bench", "run", "--iso...]
```

Tool: ScoutSuite — Multi-Cloud Auditor

ScoutSuite is an open-source multi-cloud security auditing tool that collects configuration data using cloud provider APIs and highlights risk areas.

```
# Install
pip install scoutsuite

# Run against AWS account
scout aws --profile my-profile

# Run against GCP project
scout gcp --user-account --project-id my-project

# Run against Azure subscription
scout azure --cli
```

What ScoutSuite Checks

AWS

- IAM policies and roles

AI Enhancement

- Parse ScoutSuite JSON report

- S3 bucket policies and ACLs
- Security groups and NACLs
- CloudTrail and logging
- RDS/EC2/Lambda configs

- Correlate findings across services
 - Identify privilege escalation paths
 - Generate executive summary
 - Prioritize by actual exploitability
-

Tool: Prowler — AWS/GCP/Azure Hardening

Prowler performs security assessments against CIS benchmarks and custom checks for cloud environments.

```
# Install Prowler
pip install prowler

# Run all checks against AWS
prowler aws

# Run specific check categories
prowler aws --category internet-exposed
prowler aws --service iam s3 ec2

# Output as JSON for AI processing
prowler aws --output-formats json -o ./prowler-results/
```

AI Analysis of Prowler Output

```
prompt = f"""You are analyzing Prowler security scan results for an AWS account.
```

```
Prowler found {len(findings)} issues:
```

```
- CRITICAL: {critical_count}  
- HIGH: {high_count}  
- MEDIUM: {medium_count}
```

```
For each CRITICAL finding, explain:
```

1. What the misconfiguration is
2. The realistic attack scenario
3. Immediate remediation steps (AWS CLI commands)
4. How this finding relates to other findings (attack chains)

```
Findings: {json.dumps(critical_findings)}"""
```

Notes: Prowler covers 300+ checks across AWS services. The JSON output is ideal for programmatic AI analysis.

Tool: kube-bench — CIS Kubernetes Benchmark

kube-bench checks Kubernetes clusters against the CIS Kubernetes Benchmark — the industry standard for K8s hardening.

```
# Run kube-bench on a cluster node
kube-bench run --targets master,node,policies

# Output as JSON for AI consumption
kube-bench run --json > kube-bench-results.json

# Run specific checks
kube-bench run --check 1.2.1,1.2.2,4.2.1
```

Key CIS Benchmark Areas

Section	Focus	Example Check
1.x	Control Plane	API server anonymous auth disabled

Section	Focus	Example Check
2.x	etcd	Client certificate auth enabled
3.x	Control Plane Config	Controller manager bind address
4.x	Worker Nodes	Kubelet authentication
5.x	Policies	Pod Security Standards enforced

AI + kube-bench

Feed kube-bench JSON output to Claude to get plain-English explanations of each failing check, specific `kubect1` commands to remediate, and identification of which failures create compound risks.

Tool: Trivy — Comprehensive Scanner

Trivy scans container images, filesystems, IaC files, and Kubernetes clusters for vulnerabilities, misconfigurations, and exposed secrets.

```
# Scan a container image for vulnerabilities
trivy image nginx:latest

# Scan Terraform/Kubernetes configs for misconfigurations
trivy config ./infrastructure/

# Scan a running Kubernetes cluster
trivy k8s --report summary cluster

# Scan filesystem for secrets
trivy fs --scanners secret ./

# JSON output for AI pipeline
trivy config --format json ./infrastructure/ > trivy-results.json
```

Trivy Scan Coverage

What It Scans

- Container images (OS & app packages)
- IaC (Terraform, CloudFormation, K8s)
- Filesystem for embedded secrets
- SBOM generation

AI Overlay Value

- Determine if CVEs are actually reachable
 - Prioritize by exploit availability
 - Map misconfigs to MITRE ATT&CK
 - Generate fix PRs from findings
-

Building an AI Audit Pipeline

A complete AI-augmented infrastructure audit combines multiple scanners into a unified analysis pipeline.

```
import anthropic, json, subprocess
from pathlib import Path

class InfraAuditor:
    def __init__(self):
        self.client = anthropic.Anthropic()
        self.findings = {}

    def collect_configs(self, path: str):
        """Gather all configuration files."""
        config_patterns = [
            "**/*.tf", "**/*.yaml", "**/*.yml",
            "**/Dockerfile", "**/docker-compose*.yml",
            "**/nginx.conf", "**/.env*"
        ]
```

AI Enumeration: Discovering Infrastructure

AI can assist in interpreting enumeration results, identifying patterns, and suggesting next steps during infrastructure reconnaissance.

```
def ai_enumerate(scan_output: str) -> str:
    """Use AI to analyze nmap/masscan output and plan next steps."""
    client = anthropic.Anthropic()
    response = client.messages.create(
        model="claude-sonnet-4-20250514",
        max_tokens=4096,
        messages=[{
            "role": "user",
            "content": f"""Analyze this network scan output from a
sanctioned penetration test. Identify:

1. All services and their versions
2. Services likely running with default configs
3. Potential misconfigurations based on port/service combos
4. Suggested next enumeration steps for each service"""
```

What AI Catches That Scanners Miss

- **Service correlation** — etcd (2379) + K8s API (6443) + Kubelet (10250) = likely a control plane node
 - **Version inference** — Specific SSH/TLS versions reveal OS and patch level
 - **Deployment patterns** — Consul + Nomad + Vault ports = HashiCorp stack, check for unsealed Vault
 - **Shadow IT** — Jenkins on non-standard port + Grafana + Prometheus = likely a CI/CD pipeline
-

Tool Comparison Matrix

Tool	Scope	Best For	AI Integration
ScoutSuite	AWS, GCP, Azure	Multi-cloud posture assessment	JSON report parsing
Prowler	AWS, GCP, Azure	CIS benchmark compliance	JSON finding analysis
kube-bench	Kubernetes	CIS K8s benchmark	Remediation generation
Trivy	Images, IaC, FS	Vulnerability + misconfig scanning	CVE reachability analysis
Checkov	IaC files	Terraform/CloudFormation audit	Policy-as-code generation
tfsec	Terraform	Terraform-specific checks	HCL remediation
Falco	Runtime	Runtime anomaly detection	Alert triage + correlation

The AI Multiplier

Each tool produces raw findings. AI transforms those findings into **actionable intelligence** by correlating across tools, eliminating duplicates, identifying attack chains, and generating specific remediation commands.

CTF Challenge: Infrastructure Audit

Scenario: You have been given read access to a company's infrastructure configuration repository. Your mission: find all security weaknesses and map escalation paths.

Challenge Files Provided

- `terraform/main.tf` — AWS infrastructure definition
- `k8s/deployment.yaml` — Kubernetes workload configs
- `k8s/rbac.yaml` — RBAC policies
- `docker/docker-compose.yaml` — Container orchestration
- `docker/Dockerfile` — Application container build
- `nginx/nginx.conf` — Reverse proxy configuration
- `ci/pipeline.yml` — CI/CD pipeline definition

Scoring Criteria

- **+10 pts** — Each unique misconfiguration identified

- **+25 pts** — Each multi-step attack chain mapped
 - **+15 pts** — Each remediation with specific code fix
 - **+30 pts** — Full attack narrative from initial access to data exfiltration
 - **Bonus +20 pts** — AI-assisted audit script that automates the analysis
-

CTF Challenge: Sample Attack Chain

The best CTF submissions map complete attack chains across infrastructure layers.

Example: Initial Access to Data Exfiltration

1. **Recon:** nginx exposes `/debug` endpoint without auth, leaking internal service topology
2. **Initial Access:** Docker Compose exposes database port (5432) to the internet with weak credentials
3. **Lateral Movement:** Database contains K8s service account tokens stored as application data
4. **Privilege Escalation:** Service account has overly broad RBAC — can list secrets across all namespaces
5. **Secret Access:** K8s secrets contain AWS credentials with IAM role that allows S3 and RDS access
6. **Data Exfiltration:** S3 bucket has no access logging; RDS snapshots are unencrypted and publicly restorable

AI-Assisted Chain Discovery

Prompt: "Given these 7 configuration files, identify every possible path from an external attacker to sensitive data exfiltration. For each path, list the specific misconfigurations exploited at each step and rate the overall difficulty (1-10)."

Defense: Hardening Checklist

Cloud

- Enforce IMDSv2 (block v1)
- Enable S3 Block Public Access at account level
- Use least-privilege IAM with permission boundaries
- Enable CloudTrail in all regions
- Encrypt all data at rest and in transit

Kubernetes

- Enforce Pod Security Standards (restricted)
- Disable automountServiceAccountToken
- Use NetworkPolicies (default deny)
- Enable audit logging
- Encrypt etcd at rest

Docker

- Run as non-root user
- Use read-only root filesystem
- Drop all capabilities, add only needed

Network

- Change all default credentials
- Use SNMPv3 with auth + encryption
- Restrict management ports to bastion/VPN

- Scan images in CI pipeline
- Never mount Docker socket

- Enable network segmentation
 - Monitor for unauthorized services
-

Educational Use Disclaimer

⚠ Educational Use Only

Infrastructure exploitation techniques are for authorized penetration testing and CTF competitions only. Unauthorized access to cloud resources or infrastructure is illegal.

Legal Framework

- **Computer Fraud and Abuse Act (CFAA)** — US federal law criminalizing unauthorized access to computer systems
- **Computer Misuse Act 1990** — UK equivalent covering unauthorized access and modification
- **GDPR Article 83** — Fines up to 4% of annual revenue for data protection violations
- **Cloud provider ToS** — Unauthorized scanning violates AWS/GCP/Azure terms of service

Always Obtain

- Written authorization (scope, timeline, methods)
 - Rules of engagement document
 - Emergency contact procedures
 - Cloud provider penetration testing approval (if required)
-

Key Takeaways

- **Misconfiguration is #1** — More breaches come from misconfigs than from zero-days or sophisticated exploits
- **Every layer matters** — Cloud, containers, orchestration, and network configs all need auditing
- **Defaults are dangerous** — Default credentials, permissive policies, and "it works" configurations are attacker gold
- **AI multiplies both sides** — Attackers use AI to find misconfigs faster; defenders must use AI to audit before attackers do
- **Attack chains > individual findings** — The real risk is how multiple misconfigs combine into escalation paths
- **Automate auditing** — Traditional scanners for breadth, AI analysis for depth and correlation

Next Lesson

Lesson 08 will build on infrastructure weaknesses to explore **privilege escalation** — moving from initial access to full control using the misconfigurations we identified today.

Notes: Remind students to complete the CTF challenge using both manual review and AI-assisted analysis. Compare the two approaches in the next session.
