

AI-Assisted Security Hardening

Using AI to systematically reduce attack surfaces, enforce compliance, and automate the hardening of infrastructure at scale.

What You Will Learn

- Core hardening principles
- CIS Benchmarks & STIG compliance
- Linux, Windows, container & K8s hardening
- AI-generated hardening scripts

CTF Challenge

Harden a vulnerable server configuration to pass all automated security checks.

Warning: Educational Use Only

Security hardening techniques should only be applied to systems you own or have authorization to configure. Improper hardening can cause service outages.

Why Security Hardening Matters

Default configurations are designed for ease of use, not security. Every unpatched default is an open door for attackers.

- **70%+ of breaches** exploit misconfigurations, not zero-days
- **CIS Top 18 Controls** — hardening is Control #4 (Secure Configuration)
- **Compliance mandates** — PCI-DSS, HIPAA, SOC 2 all require baseline hardening
- **Reduced attack surface** — fewer services, fewer ports, fewer vulnerabilities

The AI Advantage

AI can audit thousands of configuration files against hardening standards in seconds, generate remediation scripts, and explain *why* each change matters — turning a week-long audit into a one-hour workflow.

Hardening Principle 1: Defense in Depth

No single control is sufficient. Layer multiple defenses so that failure of one does not compromise the system.

- **Network layer** — firewalls, segmentation, IDS/IPS
- **Host layer** — OS hardening, endpoint protection, integrity monitoring
- **Application layer** — input validation, WAF, secure coding
- **Data layer** — encryption at rest and in transit, access controls
- **Identity layer** — MFA, least privilege, session management

Notes: Ask the AI: "Given this architecture diagram, identify which layers lack hardening controls and suggest additions for each gap."

Hardening Principle 2: Least Privilege

Every user, process, and service should operate with the minimum permissions required to function.

Before (Over-Privileged)

- App runs as `root`
- Database user has `GRANT ALL`
- Service account has admin API keys
- World-readable config files

After (Least Privilege)

- App runs as dedicated `appuser`
- DB user has `SELECT, INSERT` only
- Scoped API keys per service
- Config files `0640 root:appgroup`

Notes: AI prompt: "Audit this sudoers file and identify any entries that violate least privilege. Suggest specific replacements."

Hardening Principle 3: Attack Surface Reduction

Disable, remove, or restrict everything that is not explicitly required.

- **Remove unused packages** — `apt purge telnet rsh-client`
- **Disable unused services** — `systemctl disable avahi-daemon cups`
- **Close unnecessary ports** — default deny firewall rules
- **Remove default accounts** — guest, test, sample users
- **Strip development tools** from production images

AI-Assisted Surface Audit

Feed your system's `ss -tlnp` output and installed package list to the AI. Ask it to flag every listening service and package that is not in your approved baseline.

CIS Benchmarks & STIG Compliance

Industry-standard hardening baselines that define exactly how a system should be configured.

Standard	Publisher	Scope	Format
CIS Benchmarks	Center for Internet Security	OS, cloud, containers, apps	PDF + automated tools
DISA STIGs	US Department of Defense	OS, network devices, apps	XCCDF / SCAP
NIST 800-123	NIST	General server hardening	Guidelines document
Microsoft SCT	Microsoft	Windows, Edge, Office	GPO baselines

Notes: CIS Benchmarks have two profiles: Level 1 (practical, minimal disruption) and Level 2 (defense-in-depth, may impact usability).

Using AI to Audit Against CIS Benchmarks

Instead of manually reading a 300-page CIS PDF, use AI to map your configuration to specific benchmark controls.

```
# AI-assisted CIS benchmark audit
prompt = """You are a CIS Benchmark auditor for Ubuntu 22.04 (Level 1).
Analyze this sshd_config and report:
1. Each CIS control ID that applies
2. PASS or FAIL status
3. Remediation command for failures

sshd_config:
{config_content}

Output as a markdown table: | CIS ID | Description | Status | Remediation |"""

response = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=4096,
```

Notes: Always validate AI output against the official CIS benchmark document. AI may hallucinate control IDs.

Linux Hardening: SSH Configuration

SSH is the most critical remote access service. A hardened `sshd_config` is the first line of defense.

```
# /etc/ssh/sshd_config – Hardened Configuration
Protocol 2
Port 2222

# Authentication
PermitRootLogin no
MaxAuthTries 3
PubkeyAuthentication yes
PasswordAuthentication no
PermitEmptyPasswords no
AuthenticationMethods publickey

# Session Security
ClientAliveInterval 300
ClientAliveCountMax 2
```

Linux Hardening: Kernel Parameters (sysctl)

Kernel parameters control networking behavior, memory protection, and system security at the deepest OS level.

```
# /etc/sysctl.d/99-hardening.conf

# Network Security
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.all.log_martians = 1
net.ipv4.icmp_echo_ignore_broadcasts = 1
net.ipv4.tcp_syncookies = 1
net.ipv6.conf.all.accept_redirects = 0
net.ipv6.conf.all.accept_ra = 0

# Memory Protection
kernel.randomize_va_space = 2
```

Apply with: `sudo systemctl --system`

Linux Hardening: PAM & Sudoers

Pluggable Authentication Modules (PAM) control login policy. Sudoers defines privilege escalation rules.

PAM Password Quality (/etc/security/pwquality.conf)

```
minlen = 14
dcredit = -1
ucredit = -1
ocredit = -1
lcredit = -1
maxrepeat = 3
difok = 8
reject_username
enforce_for_root
```

Hardened Sudoers (/etc/sudoers.d/hardening)

```
# Require password re-entry after 5 minutes
Defaults        timestamp_timeout=5
Defaults        passwd_tries=3
Defaults        logfile="/var/log/sudo.log"
```

```
Defaults      log_input, log_output
Defaults      use_pty
Defaults      !visiblepw

# Specific command allowlists – never use ALL
%ops    ALL=(ALL) /usr/bin/systemctl restart nginx, \
        /usr/bin/systemctl restart postgresql
%deploy ALL=(deploy) NOPASSWD: /opt/deploy/run.sh
```

Linux Hardening: File Permissions & Filesystem

Critical system files must have strict ownership and permissions to prevent tampering.

```
#!/bin/bash
# AI-generated file permission hardening script

# Critical file permissions (CIS Benchmark 6.1)
chmod 644 /etc/passwd
chmod 000 /etc/shadow
chmod 644 /etc/group
chmod 000 /etc/gshadow
chmod 600 /etc/ssh/sshd_config
chmod 700 /root
chmod 600 /boot/grub/grub.cfg

# Find world-writable files (excluding expected paths)
find / -xdev -type f -perm -0002 \
-not -path '/proc/*' \
```

Linux Hardening: AppArmor & SELinux

Mandatory Access Control (MAC) systems confine processes to only the resources they need, even if they run as root.

AppArmor (Ubuntu/Debian)

```
# Check status
aa-status

# Set profile to enforce mode
aa-enforce /etc/apparmor.d/usr.sbin.nginx

# Example nginx profile
/usr/sbin/nginx {
    /etc/nginx/** r,
    /var/log/nginx/** w,
    /var/www/** r,
    /run/nginx.pid rw,
    network inet stream,
    deny /etc/shadow r,
    deny /home/** rwx.
```

SELinux (RHEL/CentOS)

```
# Check status
getenforce

# Set enforcing
setenforce 1

# View context labels
ls -Z /var/www/html/

# Fix context after file move
restorecon -Rv /var/www/html/

# Audit denials
ausearch -m avc -ts recent \
| audit2whv
```

Notes: Ask AI: "Generate an AppArmor profile for my custom Python web application running in /opt/myapp that needs access to /var/log/myapp and port 8080."

Windows Hardening: Group Policy

Group Policy Objects (GPOs) are the primary mechanism for enforcing security configuration across Windows environments.

```
# Export current security policy for AI analysis
secedit /export /cfg C:\Temp\secpol.cfg

# Key GPO hardening settings
# Computer Configuration > Windows Settings > Security Settings

# Account Policies > Password Policy
# - Minimum password length: 14 characters
# - Password complexity: Enabled
# - Maximum password age: 60 days
# - Enforce password history: 24 passwords

# Account Policies > Account Lockout Policy
# - Account lockout threshold: 5 invalid attempts
# - Account lockout duration: 30 minutes
```

Notes: Feed the exported secpol.cfg to AI and ask: "Compare this against CIS Windows Server 2022 Benchmark Level 1 and list all failures."

Windows Hardening: Defender & PowerShell Logging

Windows Defender configuration and comprehensive logging are essential for detection and forensics.

```
# Windows Defender hardening
Set-MpPreference -DisableRealtimeMonitoring $false
Set-MpPreference -MAPSReporting Advanced
Set-MpPreference -SubmitSamplesConsent SendAllSamples
Set-MpPreference -PUAProtection Enabled
Set-MpPreference -EnableNetworkProtection Enabled

# Attack Surface Reduction (ASR) rules
$rules = @(
    "BE9BA2D9-53EA-4CDC-84E5-9B1EEEE46550", # Block executable content from email
    "D4F940AB-401B-4EFC-AADC-AD5F3C50688A", # Block Office child processes
    "3B576869-A4EC-4529-8536-B80A7769E899", # Block Office creating executables
    "75668C1F-73B5-4CF0-BB93-3ECF5CB7CC84", # Block Office code injection
    "D3E037E1-3EB8-44C8-A917-57927947596D" # Block JS/VBS launching executables
)
```

Windows Hardening: Audit Policies

Advanced audit policies generate the events needed to detect intrusions, lateral movement, and privilege escalation.

```
# Configure advanced audit policies via auditpol
auditpol /set /subcategory:"Logon" /success:enable /failure:enable
auditpol /set /subcategory:"Logoff" /success:enable
auditpol /set /subcategory:"Account Lockout" /success:enable /failure:enable
auditpol /set /subcategory:"Special Logon" /success:enable
auditpol /set /subcategory:"Process Creation" /success:enable
auditpol /set /subcategory:"Sensitive Privilege Use" /success:enable /failure:enable
auditpol /set /subcategory:"Security Group Management" /success:enable
auditpol /set /subcategory:"User Account Management" /success:enable /failure:enable
auditpol /set /subcategory:"Authentication Policy Change" /success:enable

# Enable command-line logging in process creation events
reg add "HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System\Audit" `
    /v ProcessCreationIncludeCmdLine_Enabled /t REG_DWORD /d 1 /f
```

Feed audit policy output to AI: "Compare my auditpol configuration against MITRE ATT&CK data sources. Which techniques would I miss detecting with my current logging?"

Container Hardening: Dockerfile Best Practices

Containers inherit the host kernel but their images can be hardened to minimize risk.

```
# HARDENED Dockerfile — multi-stage, non-root, minimal
FROM python:3.11-slim AS builder
WORKDIR /build
COPY requirements.txt .
RUN pip install --no-cache-dir --prefix=/install -r requirements.txt

FROM python:3.11-slim
# Security: create non-root user
RUN groupadd -r appgroup && \
    useradd -r -g appgroup -d /app -s /sbin/nologin appuser

# Security: copy only what we need
COPY --from=builder /install /usr/local
COPY --chown=appuser:appgroup . /app
WORKDIR /app
```

Container Hardening: Seccomp & Read-Only FS

Seccomp profiles restrict which system calls a container can make. Read-only filesystems prevent runtime tampering.

```
{
  "defaultAction": "SCMP_ACT_ERRNO",
  "architectures": ["SCMP_ARCH_X86_64"],
  "syscalls": [
    {
      "names": [
        "read", "write", "open", "close", "stat", "fstat",
        "mmap", "mprotect", "munmap", "brk", "ioctl",
        "access", "pipe", "select", "poll", "socket",
        "connect", "accept", "bind", "listen", "clone",
        "execve", "exit", "exit_group", "futex",
        "getpid", "getuid", "getgid"
      ],
      "action": "SCMP_ACT_ALLOW"
    }
  ]
}
```

```
# Run container with hardening flags
docker run -d \
```

```
--read-only \  
--tmpfs /tmp:noexec,nosuid,size=64m \  
--security-opt no-new-privileges:true \  
--security-opt seccomp=hardened-profile.json \  
--cap-drop ALL \  
--cap-add NET_BIND_SERVICE \  
--memory 512m --cpus 1 \  
--pids-limit 100 \  
myapp:latest
```

Container Hardening Checklist

A comprehensive checklist for container security, suitable for AI-assisted auditing.

Category	Check	Severity
Image	Base image is minimal (alpine/distroless)	High
Image	No secrets baked into image layers	Critical
Image	Image is signed and from a trusted registry	High
Runtime	Container runs as non-root user	Critical
Runtime	All capabilities dropped, only needed ones added	High
Runtime	Read-only root filesystem	Medium
Runtime	Resource limits set (memory, CPU, PIDs)	Medium
Runtime	<code>no-new-privileges</code> flag enabled	High

Category	Check	Severity
Network	Host network mode disabled	High
Storage	Sensitive mounts are read-only	Medium

Kubernetes Hardening: Pod Security Standards

Kubernetes Pod Security Standards (PSS) replace the deprecated PodSecurityPolicy with three built-in profiles.

Profile	Description	Use Case
Privileged	No restrictions	System-level workloads (logging agents)
Baseline	Prevents known privilege escalations	Most workloads
Restricted	Best-practice pod hardening	Security-sensitive workloads

```
# Enforce restricted profile on a namespace
apiVersion: v1
kind: Namespace
metadata:
  name: production
  labels:
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/enforce-version: latest
```

```
pod-security.kubernetes.io/audit: restricted
```

```
pod-security.kubernetes.io/warn: restricted
```

Kubernetes: Hardened Pod Specification

A production-ready pod spec that passes the **Restricted** Pod Security Standard.

```
apiVersion: v1
kind: Pod
metadata:
  name: hardened-app
  namespace: production
spec:
  automountServiceAccountToken: false
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    runAsGroup: 1000
    fsGroup: 1000
    seccompProfile:
      type: RuntimeDefault
  containers:
```

Kubernetes: NetworkPolicies

By default, all pods can communicate with all other pods. NetworkPolicies implement microsegmentation.

```
# Default deny all ingress and egress
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
  namespace: production
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress
---
# Allow specific traffic: web -> api on port 8080
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
```

Notes: AI prompt: "Given these service dependencies, generate NetworkPolicies that implement a zero-trust microsegmentation model."

Kubernetes: RBAC Hardening

Role-Based Access Control in Kubernetes must follow least privilege. Overly broad ClusterRoles are a common attack vector.

```
# BAD: Overly permissive ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: too-permissive
rules:
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]    # NEVER DO THIS
---
# GOOD: Scoped Role for a deployment operator
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: deployment-operator
```

Export all roles with `kubectl get clusterroles,roles -A -o yaml` and ask AI to identify any that grant wildcard permissions, secrets access, or pod exec capabilities.

Kubernetes: Admission Controllers

Admission controllers intercept API requests before persistence, enforcing policies at the cluster gate.

Built-in Controllers

- **PodSecurity** — enforces Pod Security Standards
- **LimitRanger** — enforces resource defaults
- **ResourceQuota** — caps namespace resource usage
- **NodeRestriction** — limits kubelet API access

Policy Engines

- **OPA/Gatekeeper** — Rego-based policy
- **Kyverno** — K8s-native YAML policies
- **Kubewarden** — WebAssembly policies
- **AI-generated policies** from natural language

```
# Kyverno policy: require non-root containers
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-non-root
```

```
spec:
  validationFailureAction: Enforce
  rules:
    - name: check-non-root
      match:
        any:
          - resources:
              kinds: ["Pod"]
      validate:
```

AI-Generated Hardening Scripts

Use AI to generate comprehensive, auditable hardening scripts tailored to your specific environment.

```
hardening_prompt = """Generate a bash hardening script for Ubuntu 22.04 LTS  
that implements CIS Benchmark Level 1. The script must:
```

1. Be idempotent (safe to run multiple times)
2. Log every change to /var/log/hardening.log
3. Create backups before modifying any file
4. Include rollback instructions in comments
5. Cover these categories:
 - Filesystem configuration (mount options, permissions)
 - SSH hardening
 - Kernel parameters (sysctl)
 - Firewall rules (UFW)
 - Service disablement
 - Authentication (PAM, password policy)
 - Audit daemon configuration

Always Review Before Running

Never execute AI-generated hardening scripts without manual review. Test in a staging environment first. Incorrect changes can lock you out of the system.

AI-Generated Audit Checklists

Ask AI to produce structured, role-specific hardening checklists from your infrastructure context.

```
checklist_prompt = """Based on the following server role and configuration,  
generate a security hardening checklist in JSON format.
```

```
Server role: PostgreSQL database server
```

```
OS: Ubuntu 22.04
```

```
Network: Internal only, accessed from app tier
```

```
Compliance: SOC 2 Type II
```

```
For each item include:
```

- category (os, network, database, auth, logging)
- check: what to verify
- command: verification command
- expected: expected output
- remediation: fix command
- cis ref: CIS Benchmark reference if applicable

Notes: These checklists can be converted into automated test suites — each "command" and "expected" pair becomes an assertion in a compliance scanner.

Automated Compliance Scanning with AI Overlay

Combine traditional scanning tools with AI analysis for deeper, more actionable results.

```
import subprocess
import json

# Step 1: Run OpenSCAP baseline scan
result = subprocess.run(
    ["oscap", "xccdf", "eval",
     "--profile", "xccdf_org.ssgproject.content_profile_cis_level1_server",
     "--results-arf", "/tmp/scan-results.xml",
     "--report", "/tmp/scan-report.html",
     "/usr/share/xml/scap/ssg/content/ssg-ubuntu2204-ds.xml"],
    capture_output=True, text=True
)

# Step 2: Parse results to extract failures
failures = parse_arf_results("/tmp/scan-results.xml")
```

Compliance Scanning Architecture

A complete pipeline from scanning to AI-prioritized remediation and drift detection.

Pipeline Flow

- 1. **Collect** — Gather configs from hosts (Ansible facts, `oscap`, `auditpol`)
- 2. **Scan** — Run against CIS/STIG baselines (OpenSCAP, InSpec, Prowler)
- 3. **Analyze** — AI prioritizes failures by risk, blast radius, and dependencies
- 4. **Remediate** — AI generates idempotent fix scripts with rollback
- 5. **Verify** — Re-scan to confirm fixes, AI validates before/after delta
- 6. **Monitor** — Continuous drift detection alerts on regression

Tool	Target	Standards
OpenSCAP	Linux servers	CIS, DISA STIG

Tool	Target	Standards
InSpec	Any infrastructure	CIS, custom profiles
Prowler	AWS accounts	CIS AWS, PCI-DSS
kube-bench	Kubernetes clusters	CIS Kubernetes Benchmark
ScoutSuite	Multi-cloud	CIS, custom rules

Hardening Anti-Patterns

Common mistakes that make hardening efforts ineffective or counterproductive.

Do Not

- **Harden without testing** — changes can break applications silently
- **Copy-paste configs** — every environment needs tailored settings
- **Harden once, forget** — drift happens within days
- **Ignore application layer** — OS hardening alone is insufficient
- **Disable SELinux/AppArmor** because it is "annoying"

Best Practices

- **Test in staging** before production rollout
- **Version-control configs** and track changes in Git
- **Continuous compliance** scanning on schedule
- **Document exceptions** with risk acceptance sign-off
- **Use AI to explain** each setting to the team

Full-Stack Hardening Summary

Layer	Key Actions	AI Use
Linux OS	sysctl, PAM, SSH, file perms, MAC	Audit configs against CIS, generate scripts
Windows	GPO, Defender ASR, audit policies	Analyze secpol exports, map to ATT&CK
Containers	Non-root, read-only FS, seccomp, cap-drop	Review Dockerfiles, generate seccomp profiles
Kubernetes	PSS, NetworkPolicy, RBAC, admission	Audit RBAC, generate policies from intent
Compliance	CIS Benchmarks, STIGs, continuous scanning	Prioritize findings, explain risk, generate fixes

Key Takeaway

AI does not replace security expertise — it **amplifies** it. Use AI to handle the tedious configuration auditing at scale, while human judgment drives risk prioritization and exception handling.

CTF Challenge: Harden the Vulnerable Server

You are given a deliberately misconfigured server. Your mission: harden it to pass all automated security checks.

Scenario

A web server running Ubuntu 22.04 has been deployed with default settings. It runs SSH, Nginx, and a Python API. The compliance scanner currently reports **47 failures** against CIS Level 1. You must bring it to **zero failures**.

Vulnerabilities to Find and Fix

- **SSH** — root login enabled, password auth on, weak ciphers, default port
- **Kernel** — IP forwarding on, no SYN cookies, ASLR disabled
- **Permissions** — world-writable configs, SUID on unnecessary binaries

- **Services** — telnet running, cups enabled, avahi broadcasting
- **Firewall** — UFW disabled, no default deny rules
- **Containers** — app runs as root, no resource limits, host networking
- **Auth** — no password complexity, no account lockout, empty sudoers NOPASSWD

Notes: Students should use AI to analyze the initial configuration dump, generate a prioritized remediation plan, and produce hardening scripts — then verify with a compliance scanner.

CTF Challenge: Scoring Rubric

Category	Points	Criteria
SSH Hardening	20	All CIS SSH controls pass
Kernel Parameters	15	All sysctl hardening applied
File Permissions	15	Critical files properly secured
Service Minimization	10	Unnecessary services disabled
Firewall Configuration	10	Default deny, explicit allow rules
Container Hardening	15	Non-root, read-only, cap-drop, limits
Authentication	10	PAM, password policy, sudoers locked down
Bonus: AI Usage	5	Document how AI assisted your approach

Success Criteria

Run the provided `check_hardening.sh` script. A score of 90+ points means the server is hardened. Full marks require zero CIS Level 1 failures and a documented AI-assisted workflow.

Resources & Next Steps

- **CIS Benchmarks** — cisecurity.org/cis-benchmarks (free PDF registration)
- **DISA STIGs** — public.cyber.mil/stigs
- **OpenSCAP** — open-scap.org
- **kube-bench** — github.com/aquasecurity/kube-bench
- **Docker Bench Security** — github.com/docker/docker-bench-security
- **Lynis** — cisofy.com/lynis (Linux auditing tool)

Up Next: Lesson 05

Network traffic analysis and intrusion detection — using AI to identify anomalies in packet captures and flow data.

Warning: Educational Use Only

Security hardening techniques should only be applied to systems you own or have authorization to configure. Improper hardening can cause service outages.
