

AI-Assisted Security Hardening

Blue Team Mindset: Security hardening is the process of reducing a system's attack surface by eliminating unnecessary services, tightening configurations, and enforcing the principle of least privilege. In this lesson we combine established hardening frameworks with AI to accelerate auditing, script generation, and compliance verification.

Security Hardening Principles

Every hardening effort rests on three foundational principles:

- **Defense in Depth:** No single control can stop every attack. Layer multiple defenses — firewalls, host-based intrusion detection, file integrity monitoring, application whitelisting — so that if one layer fails, others still protect the system. Think of it as concentric rings of security, each independently slowing an attacker.
- **Least Privilege:** Every user, process, and service should operate with the minimum permissions required to perform its function. A web server does not need root access. A database service account does not need write access to `/etc`. Reducing privilege limits the blast radius of any compromise.
- **Attack Surface Reduction:** Every open port, running service, and installed package is a potential entry point. Remove what you do not need. Disable what you cannot remove. The smaller the surface, the fewer opportunities an attacker has.

Linux Hardening

Linux servers form the backbone of most infrastructure. Hardening them is a critical blue team skill.

SSH Configuration

SSH is often the primary remote access vector, making its configuration a top priority. A hardened `/etc/ssh/sshd_config` looks like this:

```
# /etc/ssh/sshd_config - Hardened configuration

# Disable root login over SSH
PermitRootLogin no

# Use only SSH protocol 2
Protocol 2

# Disable password authentication - use keys only
PasswordAuthentication no
PubkeyAuthentication yes
AuthorizedKeysFile .ssh/authorized_keys

# Limit login attempts and idle sessions
MaxAuthTries 3
ClientAliveInterval 300
ClientAliveCountMax 2

# Restrict SSH to specific users/groups
AllowGroups ssh-users

# Disable X11 forwarding and TCP forwarding unless needed
X11Forwarding no
AllowTcpForwarding no

# Use strong ciphers and MACs only
```

User Management and File Permissions

Lock down user accounts and filesystem permissions to prevent privilege escalation:

```
# Remove unnecessary user accounts
userdel -r games
userdel -r ftp

# Set strong password policies in /etc/login.defs
PASS_MAX_DAYS    90
PASS_MIN_DAYS    7
PASS_MIN_LEN     14
PASS_WARN_AGE    14

# Restrict permissions on sensitive files
chmod 600 /etc/shadow
```

```
chmod 644 /etc/passwd
chmod 700 /root
chmod 600 /etc/ssh/sshd_config

# Set SUID/SGID audit - find binaries with elevated bits
find / -perm /6000 -type f -exec ls -la {} \; 2>/dev/null
```

Kernel Parameters and Unnecessary Services

Harden the kernel via `sysctl` and disable services that expand your attack surface:

```
# /etc/sysctl.d/99-hardening.conf
# Disable IP forwarding (unless this is a router)
net.ipv4.ip_forward = 0

# Ignore ICMP redirects
net.ipv4.conf.all.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0

# Enable SYN flood protection
net.ipv4.tcp_syncookies = 1

# Disable source routing
net.ipv4.conf.all.accept_source_route = 0

# Enable reverse path filtering
net.ipv4.conf.all.rp_filter = 1

# Restrict core dumps
fs.suid_dumpable = 0

# Apply immediately
# sysctl -p /etc/sysctl.d/99-hardening.conf
```

```
# Disable and stop unnecessary services
systemctl disable --now avahi-daemon
systemctl disable --now cups
systemctl disable --now rpcbind
systemctl disable --now telnet.socket

# List all enabled services for review
systemctl list-unit-files --state=enabled --type=service
```

Windows Hardening

Windows environments require a different toolset but the same principles apply.

Group Policy is the primary mechanism for enforcing security baselines across a Windows domain. Key policies include password complexity requirements, account lockout thresholds, restricting local administrator access, and controlling software installation. The Security Compliance Toolkit (SCT) from Microsoft provides pre-built GPO baselines for Windows Server and Windows 10/11.

Windows Defender should be configured with real-time protection enabled, cloud-delivered protection active, controlled folder access turned on for ransomware mitigation, and attack surface reduction (ASR) rules enabled. ASR rules block common malware behaviors such as Office macro code execution, credential theft from LSASS, and untrusted process execution from USB drives.

Audit Policies should log account logon events, object access, privilege use, and process creation. Enable command-line logging in process creation events — this is invaluable for forensic analysis. Configure logs to forward to a central SIEM.

Service Accounts should use Group Managed Service Accounts (gMSAs) instead of shared password accounts. gMSAs rotate passwords automatically and eliminate the risk of stale credentials on service accounts.

CIS Benchmarks and Security Baselines

The **Center for Internet Security (CIS)** publishes comprehensive hardening benchmarks for virtually every major operating system, cloud platform, and application. CIS Benchmarks provide two profile levels:

- **Level 1:** Practical settings that can be applied broadly without significant performance impact or functionality loss. Suitable for most environments.
- **Level 2:** Deeper hardening intended for high-security environments. May restrict some functionality but provides stronger protection.

CIS Benchmarks are freely available at [cisecurity.org/cis-benchmarks](https://www.cisecurity.org/cis-benchmarks) and serve as the gold standard for compliance audits. Many regulatory frameworks (PCI-DSS, HIPAA, SOC 2) reference CIS benchmarks as an acceptable hardening standard.

AI-Assisted Hardening

Security hardening involves reviewing hundreds of configuration settings across dozens of files. This is exactly the kind of tedious, detail-oriented work where AI excels. Here are three high-value use cases:

1. Configuration Review

Feed a configuration file to Claude and ask it to identify deviations from security best practices. The AI can compare settings against CIS benchmarks and flag misconfigurations that a manual review might miss:

```
import anthropic

client = anthropic.Anthropic()

sshd_config = open("/etc/ssh/sshd_config").read()

message = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=2048,
    messages=[
        {
            "role": "user",
            "content": (
                "You are a Linux security hardening expert. Review the following\n"
                "sshd_config file against CIS Benchmark recommendations. For each\n"
                "finding, state the current setting, the recommended setting,\n"
                "CIS control ID if applicable, and the risk level (Critical, High,\n"
                "Medium, Low).\n\n"
                f"```\n{sshd_config}\n```"
            )
        }
    ]
)

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

2. Hardening Script Generation

Describe your environment and let Claude generate a hardening script tailored to your requirements. Always review generated scripts before execution — never run AI-

generated code blindly on production systems.

3. Compliance Auditing

Collect system state (running services, open ports, kernel parameters, file permissions) and ask Claude to audit it against a specific compliance framework. This turns hours of manual checklist work into a minutes-long automated review.

Automated Hardening Tools

Ansible Hardening Playbook

Ansible codifies hardening steps into repeatable, version-controlled playbooks. Here is a playbook that applies core Linux hardening:

```
---
# hardening.yml - Linux Server Hardening Playbook
- name: Apply security hardening
  hosts: all
  become: true

  tasks:
    - name: Ensure SSH root login is disabled
      lineinfile:
        path: /etc/ssh/sshd_config
        regexp: '^#?PermitRootLogin'
        line: 'PermitRootLogin no'
      notify: restart sshd

    - name: Disable password authentication
      lineinfile:
        path: /etc/ssh/sshd_config
        regexp: '^#?PasswordAuthentication'
        line: 'PasswordAuthentication no'
      notify: restart sshd

    - name: Set kernel hardening parameters
      sysctl:
        name: "{{ item.key }}"
        value: "{{ item.value }}"
        sysctl_set: true
```

OpenSCAP and Lynis

OpenSCAP is a NIST-certified compliance scanner that evaluates systems against SCAP (Security Content Automation Protocol) profiles. **Lynis** is a lightweight, agentless auditing tool ideal for quick hardening assessments:

```
# Lynis - run a full system audit
sudo lynis audit system

# Lynis - audit and save report to file
sudo lynis audit system --report-file /tmp/lynis-report.dat

# Lynis - show only warnings and suggestions
sudo lynis show warnings
sudo lynis show suggestions

# OpenSCAP - evaluate against a CIS profile
oscap xccdf eval \
  --profile xccdf_org.ssgproject.content_profile_cis \
  --results /tmp/oscap-results.xml \
  --report /tmp/oscap-report.html \
  /usr/share/xml/scap/ssg/content/ssg-ubuntu2204-ds.xml
```

Tip: Pipe Lynis output into Claude for prioritized remediation advice. Combine `lynis show suggestions` output with your environment context and ask for a ranked remediation plan with specific commands.

Container Hardening

Containers introduce their own hardening challenges. A poorly built container image can negate host-level hardening entirely. Follow these Dockerfile best practices:

```
# Use a minimal base image - not ubuntu:latest
FROM python:3.12-slim AS builder

# Create a non-root user
RUN groupadd -r appuser && useradd -r -g appuser -d /app -s /sbin/nologin appuser

# Install dependencies in a build stage
WORKDIR /app
```

```

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Production stage - minimal image
FROM python:3.12-slim

# Copy the non-root user from builder
RUN groupadd -r appuser && useradd -r -g appuser -d /app -s /sbin/nologin appuser

# Copy only what is needed
COPY --from=builder /usr/local/lib/python3.12/site-packages /usr/local/lib/python3.12/site-packages
COPY --chown=appuser:appuser . /app

WORKDIR /app
USER appuser

# Read-only filesystem where possible
# Run with: docker run --read-only --tmpfs /tmp myimage

```

Key container hardening rules: never run as root inside the container, use multi-stage builds to minimize image size, avoid installing unnecessary packages, scan images for vulnerabilities with tools like `trivy` or `grype`, and mount the filesystem as read-only with explicit tmpfs mounts for directories that need write access.

Network Hardening

Network-level hardening creates the outermost defensive layer:

- **Segmentation:** Divide your network into zones (DMZ, internal, management, database) and control traffic between them with firewalls. A compromised web server in the DMZ should not have direct access to the database tier.
- **Firewall Rules:** Follow a default-deny policy. Only allow traffic that is explicitly required. Regularly audit rules and remove stale entries. Use host-based firewalls (`iptables` / `nftables`, Windows Firewall) in addition to network firewalls.
- **TLS Configuration:** Enforce TLS 1.2+ for all services. Disable weak cipher suites (RC4, DES, 3DES). Use HSTS headers for web applications. Automate certificate rotation with tools like cert-manager or Let's Encrypt.
- **DNS Security:** Deploy DNSSEC where possible. Use DNS sinkholing to block known malicious domains. Monitor DNS query logs for indicators of data exfiltration via DNS tunneling.

Key Takeaways

Hardening is not a one-time event — it is a continuous process. Systems drift from their baselines as patches are applied, configurations are changed, and new services are deployed. Automate hardening, audit regularly, and use AI to accelerate both.

- Apply defense in depth, least privilege, and attack surface reduction as guiding principles.
- Harden SSH, kernel parameters, file permissions, and disable unnecessary services on Linux.
- Use Group Policy, Defender ASR rules, and audit policies on Windows.
- Anchor your hardening to CIS Benchmarks for a defensible, auditable baseline.
- Leverage Claude to review configurations, generate hardening scripts, and audit compliance.
- Automate with Ansible, OpenSCAP, and Lynis for repeatable, drift-resistant hardening.
- Harden containers at the image level: minimal base, non-root user, read-only filesystem.
- Segment networks, enforce default-deny firewall rules, and require TLS 1.2+.

Self-check: Can you list five hardening changes to a default `sshd_config`? Can you explain what a CIS Level 1 vs Level 2 profile means? Can you describe how to use Claude to audit a system configuration against a compliance baseline? Can you write an Ansible task that disables a list of unnecessary services?

Challenge Resources

Review these configurations and identify all security weaknesses. Use AI to help generate hardened versions.

Download Files

Synthetic configurations with embedded weaknesses. All files are plain text (.txt) and safe to open.

↓ [sshd-config-weak.txt](#) ~2 KB

↓ [linux-audit.txt](#) ~9 KB

↓ [dockerfile-insecure.txt](#) ~1 KB

↓ [windows-gpo-audit.txt](#) ~12 KB