

Lab Guide

Four exercises: git history secret hunting, leaked repo triage, crypto code review, and TLS config audit — keep secrets secret and crypto sane.

Authorized Use Only

All code and configs are **synthetic**. Never paste real production secrets into any AI, even to audit them — assume the input is logged.

Prerequisites

- Completed the [Theory](#) reading
- Access to an AI assistant (Claude, ChatGPT, or local LLM)
- Downloaded all four files into a folder named `secrets-lab/`
- Familiarity with: git, symmetric/asymmetric crypto basics, TLS handshake

Lab Scenario

A contractor's laptop was lost. Their personal GitHub mirror of an AcmeCorp repo is now public. You have the history dump, the top-level repo contents, an old crypto utility written in-house, and the public TLS configs from AcmeCorp's reverse proxies. Your job: find every leaked secret, fix the crypto, harden the TLS.

Download Lab Files

Save all 4 files into a folder named `secrets-lab/` before starting the exercises.

↓ [crypto-review.py.txt](#) ~6 KB

↓ [git-history-secrets.txt](#) ~5 KB

↓ [leaked-repo.txt](#) ~5 KB

↓ [tls-config-audit.txt](#) ~10 KB

Exercise 1 — Git History Secret Hunt

Open [git-history-secrets.txt](#). Secrets often hide in old commits even after being "removed".

Task 1.1 — Find every credential

Below is a truncated git log with diffs. Find every potential secret:

1. API keys (AWS AKIA, Stripe sk_, Google AIza, Slack xox).
2. Private keys (BEGIN RSA/EC/OPENSSH PRIVATE KEY).
3. Passwords in config, env, YAML, JSON.
4. JWT / OAuth tokens.
5. Database URIs with credentials.
6. Internal hostnames / IPs that leak infra.
7. .env files checked in by mistake.

For each:

- Commit hash, file, line
- Secret type
- Rotation urgency (minutes / hours / days)
- Command to rotate (AWS IAM CLI, vault rotate, etc.)

Then: list the commits that need to be purged from history with git-filter-repo or BFG.

[paste git-history-secrets.txt]

✓ **Self-check:** You should identify at least one secret that **MUST** be rotated within the hour.

Exercise 2 — Leaked Repo Triage

Open `leaked-repo.txt`. The current state of the repo also has secrets and architectural leaks.

Task 2.1 — Assess the exposure

Below is the tree + selected file contents of a leaked repo.
Identify:

1. Hard-coded secrets in source.
2. Cloud architecture disclosed (bucket names, internal DNS, queue URLs, IAM role ARNs).
3. Business logic that reveals vulnerabilities (auth bypass, weak password policy, disabled MFA).
4. Third-party integrations (Slack webhooks, Twilio, Sendgrid) that could be abused.
5. CI/CD secrets or deploy keys.
6. Infrastructure as code with wide permissions.

Output:

- A disclosure assessment (what the attacker learns)
- A mitigation plan (rotate, re-architect, monitor)
- A customer-notification decision tree (do we need to notify?)

[paste leaked-repo.txt]

✓ Self-check: Decide YES / NO on customer notification with one clear sentence explaining the reasoning.

Exercise 3 — Crypto Code Review

Open `crypto-review.py.txt`. Home-grown crypto is a running joke for a reason.

Task 3.1 — Spot the bugs

Below is a Python module that handles encryption, hashing, and password storage. Review for:

1. Password hashing: MD5/SHA1 instead of bcrypt/argon2.
2. Missing salt, constant salt, or salt reuse.
3. Symmetric cipher choices: DES, ECB mode, CBC without HMAC.
4. IV/nonce reuse with stream ciphers or AES-GCM.
5. PRNG: random.random() vs. secrets / os.urandom.
6. RSA with no padding or PKCS1v15 in new code.
7. Timing side channels in comparisons.
8. Key storage: hard-coded, env, keystore?

For each finding:

- Line number
- Bug category
- Corrected code block

Output the rewritten module at the end.

[paste crypto-review.py.txt]

✓ Self-check: Your rewritten code should use argon2 / AES-GCM / secrets and nothing else for the primitives.

Exercise 4 — TLS Config Audit

Open `tls-config-audit.txt`. TLS misconfig is the single easiest compliance finding.

Task 4.1 — Score against Mozilla intermediate

Below are TLS configs for several services (nginx, haproxy, postgres). Audit each for:

1. Protocols: any TLSv1.0/1.1 enabled? SSLv3?
2. Cipher suites: any RC4, 3DES, NULL, EXPORT, CBC?
3. Certificate: self-signed? Expired? SHA-1 signed? Short key?
4. HSTS: set? max-age long enough? includeSubDomains?
5. OCSP stapling, session tickets, session cache.
6. Forward secrecy: DHE/ECDHE only?
7. HTTP -> HTTPS redirect?
8. Protocols for internal services (should still be TLS 1.2+).

Output:

- Findings per service
- The corrected config snippets (paste-ready)
- A single "Mozilla intermediate" baseline the ops team can apply

[paste tls-config-audit.txt]

✓ **Self-check:** You should find at least one service missing HSTS or allowing a weak cipher.

Final Exercise — Secret-handling Policy

Task 5.1 — Policy document

Combine the findings into a one-page secret-handling policy:

1. Where secrets live (vault path scheme)
2. Rotation frequency by type
3. Pre-commit hooks / CI scanning
4. Incident playbook for a leak
5. Crypto-primitive allowlist for new code
6. TLS baseline for all services

Format as markdown.

✓ **Self-check:** The policy should be short enough to print and keep on every engineer's desk.

What's Next

Lesson 07 moves into **vuln management and patching** — SBOMs, dependency audits, and patch prioritization.