

Vulnerability Assessment & Patch Management

From discovery to remediation — building a systematic, AI-augmented approach to finding and fixing security weaknesses before attackers exploit them.

Vulnerability Scanning

CVSS

Patch Management

AI Triage

Dependency Scanning

Container Security

⚠ Educational Use Only

Vulnerability scanning should only be performed on systems you own or have explicit authorization to test. Unauthorized scanning may be detected and reported.

Why Vulnerability Management Matters

Organizations face an ever-growing attack surface. The average enterprise has thousands of known vulnerabilities at any given time, but limited resources to fix them.

The Scale Problem

26,000+ CVEs published in 2023 alone. No team can patch everything immediately.

The Prioritization Problem

Not all vulnerabilities are equal. A Critical-rated CVE on an air-gapped system may be lower risk than a Medium on a public-facing server.

Where AI Helps

AI can process scan results at scale, correlate environmental context, predict exploitability, and recommend prioritized remediation — turning data overload into actionable plans.

The Vulnerability Management Lifecycle

A continuous, five-phase cycle that never truly ends — each iteration strengthens your security posture.

1. **Discover** — Identify assets and scan for known vulnerabilities across infrastructure, applications, and dependencies
2. **Assess** — Evaluate each finding: confirm true positives, understand severity, and determine affected scope
3. **Prioritize** — Rank findings by business risk, exploitability, asset criticality, and environmental context
4. **Remediate** — Apply patches, configuration changes, mitigating controls, or accept residual risk with documentation
5. **Verify** — Confirm the fix is in place, rescan to validate, and update tracking records

Notes: Emphasize this is a cycle, not a one-time project. Most mature organizations run continuous scanning with weekly or monthly remediation sprints.

Phase 1: Discover

You cannot protect what you do not know about. Discovery combines asset inventory with vulnerability scanning.

Asset Discovery

- **Network scanning** — Nmap, Masscan to find live hosts and open ports
- **Cloud inventory** — AWS Config, Azure Resource Graph, GCP Asset Inventory
- **CMDB integration** — ServiceNow, Lansweeper for enterprise asset tracking
- **Container registries** — Docker Hub, ECR, GCR — what images are deployed?

Vulnerability Scanning

- **Authenticated scans** — Agent-based or credentialed, sees installed software versions
- **Unauthenticated scans** — Network-based, sees externally visible weaknesses

- **Application scans** — DAST tools like OWASP ZAP, Burp Suite for web app vulns
-

Scanning Tools: Nessus, OpenVAS, Qualys

Tool	Type	Strengths	Best For
Nessus	Commercial	Huge plugin library, low false positives, compliance checks	Enterprise infrastructure scanning
OpenVAS	Open Source	Free, community-maintained, GVM framework	Budget-conscious teams, learning
Qualys	Cloud SaaS	Agentless cloud scanning, asset management, continuous monitoring	Large/distributed environments

AI Integration Point

All three tools export results in structured formats (CSV, JSON, XML). Feed these directly to an LLM to summarize findings, group by remediation action, and draft executive reports.

Interpreting Nessus Scan Results

A typical Nessus export contains structured data per finding. Here is an abbreviated CSV example:

```
Plugin ID,CVE,CVSS v3,Risk,Host,Protocol,Port,Name,Synopsis
11219,CVE-2023-44487,7.5,High,10.0.1.15,tcp,443,HTTP/2 Rapid Reset DoS,Remote HTTP/2 server vulnerable
20007,CVE-2023-38545,9.8,Critical,10.0.1.22,tcp,22,curl SOCKS5 Heap Overflow,libcurl SOCKS5 proxy heap
42873,,2.1,Low,10.0.1.15,tcp,80,SSL/TLS Certificate Info,SSL certificate uses SHA-1 signature
57582,CVE-2024-3094,10.0,Critical,10.0.2.5,tcp,22,XZ Utils Backdoor (CVE-2024-3094),Backdoored xz-utils
19506,,0.0,Info,10.0.1.15,tcp,0,Nessus Scan Information,Scan metadata and configuration
```

AI Prompt: Summarize Nessus Results

"Analyze this Nessus scan CSV. Group findings by host, highlight Critical/High items first, identify which vulnerabilities have known exploits in the wild, and recommend remediation priority."

CVE & CVSS Scoring

What is a CVE?

Common Vulnerabilities and Exposures — a standardized identifier (e.g., `CVE-2024-3094`) for publicly known security flaws. Maintained by MITRE and NVD.

CVSS v3.1 Score Breakdown

Score Range	Severity	Typical SLA
9.0 – 10.0	Critical	24–72 hours
7.0 – 8.9	High	7–14 days
4.0 – 6.9	Medium	30–60 days
0.1 – 3.9	Low	90 days / next cycle
0.0	Informational	Track / no action

Notes: CVSS Base Score alone is insufficient for prioritization. Environmental and Temporal metrics matter — this is exactly where AI adds value.

Reading a CVSS Vector String

Every CVSS score is derived from a vector string that encodes attack characteristics:

```
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H (Score: 9.8 Critical)
```

```
AV:N = Attack Vector: Network (remotely exploitable)
AC:L = Attack Complexity: Low (no special conditions needed)
PR:N = Privileges Required: None (unauthenticated)
UI:N = User Interaction: None (no clicks needed)
S:U = Scope: Unchanged
C:H = Confidentiality Impact: High
I:H = Integrity Impact: High
A:H = Availability Impact: High
```

AI Prompt: Explain a CVE

"Explain CVE-2024-3094 (XZ Utils backdoor) for a non-technical audience. Include: what it does, how it gets exploited, affected versions, and recommended fix. Parse the CVSS vector CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H and explain each component."

Phase 2: Assess

Raw scan results contain noise. Assessment separates real risk from false positives and informational clutter.

Key Assessment Activities

- **Validate findings** — Confirm the vulnerability actually exists (not a scanner misidentification)
- **Check exploitability** — Does a public exploit exist? Is it in exploit-db, Metasploit, or CISA KEV?
- **Determine reachability** — Can an attacker actually reach the vulnerable service?
- **Assess data exposure** — What data/systems are at risk if exploited?
- **Identify compensating controls** — WAF rules, network segmentation, EDR monitoring

AI-Assisted Assessment

Prompt an LLM: *"For each of these 47 High/Critical findings, check if a public exploit exists in ExploitDB or CISA KEV. Flag any that are actively exploited in the wild."*

Phase 3: Prioritize — Risk-Based with AI

CVSS score alone is a poor prioritization strategy. Risk-based vulnerability management (RBVM) adds context.

Factors Beyond CVSS

- **Asset criticality** — A vuln on a payment server > the same vuln on a dev sandbox
- **Network exposure** — Internet-facing vs. internal-only vs. air-gapped
- **Exploit availability** — Known exploit in the wild = immediate escalation
- **Compensating controls** — IPS signature, WAF rule, or firewall blocks the attack path
- **Business impact** — Revenue-generating service vs. archival system
- **Regulatory requirements** — PCI DSS, HIPAA, SOC 2 mandate specific timelines

AI Contextualization Prompt

"Given this scan report and the following asset inventory (with criticality tags and network zones), re-rank vulnerabilities by effective risk. Consider: CISA KEV status, EPSS score, asset

criticality, and network exposure."

EPSS: Exploit Prediction Scoring System

EPSS is a data-driven model that estimates the probability a vulnerability will be exploited in the wild within the next 30 days.

CVSS vs. EPSS

CVSS measures theoretical severity.

EPSS measures likelihood of real-world exploitation.

A CVSS 9.8 with EPSS 0.02 may be lower priority than a CVSS 6.5 with EPSS 0.87.

Using EPSS with AI

Feed EPSS percentiles into your AI-assisted triage workflow. Combine with CISA KEV (Known Exploited Vulnerabilities catalog) for strongest prioritization signal.

```
# Query EPSS API for a CVE
import requests

cve_id = "CVE-2024-3094"
resp = requests.get(f"https://api.first.org/data/v1/epss?cve={cve_id}")
data = resp.json()["data"][0]
```

```
print(f"{cve_id}: EPSS={data['epss']}, Percentile={data['percentile']}")  
# CVE-2024-3094: EPSS=0.93, Percentile=0.99
```

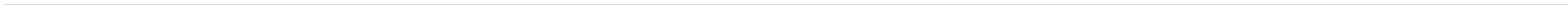
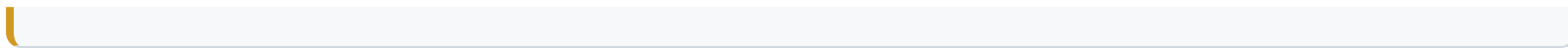
Dependency Scanning: Your Supply Chain Attack Surface

Modern applications inherit hundreds of transitive dependencies. Each is a potential vulnerability vector.

Tool	Ecosystem	Type	Integration
npm audit	Node.js	Built-in	CLI, CI/CD
pip-audit	Python	Open source	CLI, CI/CD
Snyk	Multi-language	Commercial	CLI, IDE, CI/CD, GitHub
Dependabot	Multi-language	GitHub native	Automated PRs

The Transitive Dependency Trap

Your `package.json` lists 20 packages. After install, `node_modules/` contains 1,200+. Each transitive dependency is an attack surface you did not explicitly choose.



npm audit: Realistic Output

```
$ npm audit

# npm audit report

semver <7.5.2
Severity: moderate
semver vulnerable to ReDoS - https://github.com/advisories/GHSA-c2qf-rxjj-qqgw
fix available via `npm audit fix`
node_modules/semver

xml2js <0.5.0
Severity: moderate
xml2js is vulnerable to prototype pollution - GHSA-776f-qx25-q3cc
fix available via `npm audit fix --force`
node_modules/xml2js
```

AI Triage Prompt

"Analyze this npm audit report. For each vulnerability: explain the real-world risk, whether this is exploitable in a typical Express.js web app, and rank them by urgency. Flag any that require breaking changes."



pip-audit: Python Dependency Scanning

```
$ pip-audit
```

```
Found 4 known vulnerabilities in 3 packages
```

Name	Version	ID	Fix Versions
requests	2.28.0	PYSEC-2023-74	2.31.0
flask	2.2.0	CVE-2023-30861	2.3.2
cryptography	39.0.1	CVE-2023-38325	41.0.2
cryptography	39.0.1	CVE-2023-49083	41.0.6

Key observations for AI triage:

- **requests 2.28.0** — Information disclosure via proxy headers; moderate severity
- **flask 2.2.0** — Session cookie vulnerability; depends on cookie config
- **cryptography** — Two CVEs, both fixed in 41.x; potential breaking API changes in upgrade

AI Patch Impact Prompt

"I need to upgrade cryptography from 39.0.1 to 41.0.6. List the breaking changes between these versions. Check if my code (attached) uses any deprecated APIs."

Snyk & Dependabot: Automated Remediation

Snyk

- Scans code, dependencies, containers, and IaC
- Provides **fix PRs** with tested upgrades
- Priority scoring with reachability analysis
- IDE plugins for shift-left scanning

```
# Scan a project
snyk test

# Monitor continuously
snyk monitor

# Scan a container image
snyk container test myapp:latest
```

Dependabot

- Native GitHub integration
- Automatic PRs for version updates
- Security alerts with severity levels
- Configurable update schedules

```
# .github/dependabot.yml
version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
    open-pull-requests-limit: 10
```

Container Image Scanning: Trivy & Gype

Container images bundle OS packages, language libraries, and your code — all need scanning.

Trivy

Comprehensive, fast, open-source scanner by Aqua Security. Scans images, filesystems, git repos, and Kubernetes.

```
# Scan an image
trivy image python:3.11-slim

# Scan and filter severity
trivy image --severity HIGH,CRITICAL \
  myapp:latest

# Output as JSON for AI processing
trivy image -f json -o results.json \
  myapp:latest
```

Grype

Open-source scanner by Anchore. Fast, SBOM-native, integrates with Syft for software bill of materials.

```
# Scan an image
grype python:3.11-slim

# Generate SBOM first, then scan
syft myapp:latest -o json > sbom.json
grype sbom:sbom.json

# Filter by severity
grype myapp:latest --only-fixed \
  --fail-on high
```

Trivy Scan: Realistic Output

```
$ trivy image myapp:latest

myapp:latest (debian 12.4)
=====
Total: 142 (UNKNOWN: 2, LOW: 68, MEDIUM: 49, HIGH: 19, CRITICAL: 4)

+-----+-----+-----+-----+-----+
| LIBRARY | VULNERABILITY | SEVERITY | INSTALLED VERSION | FIXED VERSION |
+-----+-----+-----+-----+-----+
| libcurl4 | CVE-2024-2398 | HIGH | 7.88.1-10+deb12u4 | 7.88.1-10+deb12u5 |
| openssl | CVE-2024-0727 | MEDIUM | 3.0.11-1~deb12u2 | 3.0.13-1~deb12u1 |
| zlib1g | CVE-2023-45853 | CRITICAL | 1:1.2.13.dfsg-1 | |
| libc6 | CVE-2024-2961 | HIGH | 2.36-9+deb12u3 | 2.36-9+deb12u7 |
| python3.11 | CVE-2024-0450 | MEDIUM | 3.11.2-6 | 3.11.2-6+deb12u2 |
+-----+-----+-----+-----+-----+
```

AI Analysis Prompt

"Analyze this Trivy scan of our production web app container. Separate OS-level and application-level vulnerabilities. Which CRITICAL/HIGH findings have fixes available? Draft a Dockerfile patch to remediate the fixable items."



AI-Assisted Patch Impact Analysis

The hardest question in patching: **"Will this patch break something?"**

What AI Can Analyze

- **Changelog parsing** — Summarize breaking changes between current and target versions
- **API surface comparison** — Identify deprecated/removed functions your code uses
- **Dependency conflict detection** — Flag incompatible version constraints across packages
- **Test coverage assessment** — Determine if critical paths have test coverage for regression detection
- **Historical incident correlation** — "Last time we upgraded X, service Y broke because of Z"

```
prompt = ""
Analyze the upgrade path from Django 4.1 to Django 5.0 for our project.

Our codebase uses:
- django.utils.timezone.utc (removed in 5.0)
- DEFAULT_FILE_STORAGE setting (replaced with STORAGES)
- url() patterns (removed, must use path())
```

List each breaking change, the files affected, and a migration plan.

"""

Patch Impact: AI Response Example

Here is what an AI-assisted analysis might return for the Django upgrade:

AI Analysis Summary

Risk Level: MEDIUM — 3 breaking changes found, all have documented migration paths.

Breaking Changes Detected:

=====

1. `django.utils.timezone.utc` → `datetime.timezone.utc`
Files affected: `utils/dates.py` (line 12), `api/views.py` (line 87)
Migration: Find-and-replace import. No logic change needed.
Risk: LOW
 2. `DEFAULT_FILE_STORAGE` → `STORAGES` dict
Files affected: `settings/production.py` (line 45)
Migration: Restructure setting to new `STORAGES` format.
Risk: LOW (config-only change)
 3. `url()` → `path()/re_path()`
Files affected: `urls.py` (14 patterns), `api/urls.py` (8 patterns)
-

Phase 4: Remediate

Remediation is not always "just patch it." There are multiple strategies depending on context and risk.

Remediation Options

Strategy	When to Use	Example
Patch/Update	Fix available, low regression risk	Apply security update to OpenSSL
Mitigate	No patch yet, or patch has high risk	WAF rule to block exploit payload
Compensating control	Cannot patch, need alternative protection	Network segmentation, IPS signature
Accept risk	Low impact, high patch cost, documented	Air-gapped system, scheduled EOL
Remove/Replace	Component is end-of-life or unnecessary	Decommission legacy service

Notes: Risk acceptance must be documented and signed off by an appropriate authority. "We forgot" is not risk acceptance.

Patch Management Strategies

A structured approach to deploying patches safely across environments.

The Patch Pipeline

1. **Staging/Test** — Apply patch to non-production environment first
2. **Automated testing** — Run regression suite, integration tests, smoke tests
3. **Canary deployment** — Roll to a small subset of production (5–10%)
4. **Monitor** — Watch error rates, latency, and functionality for 1–4 hours
5. **Full rollout** — Expand to all production systems
6. **Verification scan** — Rescan to confirm vulnerability is resolved

Rollback Planning

- **Pre-patch snapshots** — VM snapshots, database backups, container image tags
- **Blue-green deployment** — Keep previous version running; switch back if needed
- **Feature flags** — Disable new code paths without full rollback

- **Documented rollback procedure** — Step-by-step, tested before patch day
-

Emergency Patching: When the Clock is Ticking

Zero-day or actively exploited vulnerabilities require an accelerated process.

Emergency Patch Criteria

- CVE is in CISA KEV (Known Exploited Vulnerabilities)
- Active exploitation confirmed by threat intelligence
- CVSS 9.0+ AND remotely exploitable AND no authentication required
- Affects internet-facing systems with sensitive data

Accelerated Timeline

1. **Hour 0–1:** Assess exposure — which systems are affected?
2. **Hour 1–4:** Apply mitigation (WAF rules, network blocks) while testing patch
3. **Hour 4–8:** Abbreviated test cycle — smoke tests on staging

4. **Hour 8–24:** Production rollout with enhanced monitoring
5. **Day 2–3:** Full regression testing, verify fix across all systems

Notes: Emergency patches bypass the normal change window. This is why having a pre-approved emergency change process is essential.

SLA-Based Remediation Timelines

Define clear remediation SLAs based on severity and asset tier. Track compliance metrics.

Severity	Tier 1 (Critical Assets)	Tier 2 (Standard)	Tier 3 (Low-Risk)
Critical (9.0+)	24 hours	72 hours	7 days
High (7.0–8.9)	7 days	14 days	30 days
Medium (4.0–6.9)	30 days	60 days	90 days
Low (0.1–3.9)	90 days	Next cycle	Next cycle

Tracking & Metrics

- **MTTR** (Mean Time to Remediate) — by severity level
- **SLA compliance rate** — percentage of vulns fixed within SLA
- **Vulnerability debt** — total open vulns past SLA deadline
- **Risk reduction over time** — trending total risk score down

AI for Tracking

*"Generate a weekly vulnerability status report from our Jira tickets tagged 'vuln-remediation'.
Show SLA compliance by severity, overdue items, and top 5 risk contributors."*

Phase 5: Verify

Remediation without verification is hope-based security. Always confirm the fix.

Verification Methods

- **Rescan** — Run the same scanner that found the vulnerability; confirm it no longer appears
- **Targeted validation** — Attempt the specific exploit against the patched system
- **Version check** — Verify the package/OS version matches the patched version
- **Configuration audit** — For misconfigurations, verify the setting is correct
- **Automated pipeline check** — CI/CD gates that fail builds with known CVEs

```
# Verify a specific CVE is resolved
trivy image --severity CRITICAL myapp:latest | grep CVE-2024-28219
# (no output = vulnerability resolved)

# Check installed package version
dpkg -l | grep openssl
# ii  openssl  3.0.13-1~deb12u1  (patched version confirmed)
```

Notes: Automate verification wherever possible. Integrate scanning into CI/CD so patched images are validated before deployment.

AI-Powered Triage Pipeline

Combining all the concepts into an end-to-end AI-assisted vulnerability management workflow.

```
import anthropic, json

client = anthropic.Anthropic()

# 1. Load scan results
with open("trivy_results.json") as f:
    scan_data = json.load(f)

with open("asset_inventory.json") as f:
    assets = json.load(f)

# 2. AI-assisted triage
response = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=4096,
```

Anatomy of a CVE Entry

Understanding what a CVE record contains helps you make faster triage decisions.

```
{
  "id": "CVE-2024-3094",
  "description": "Malicious code was discovered in the upstream tarballs of xz-utils, starting with ve",
  "published": "2024-03-29T17:15:00",
  "cvss_v3": {
    "baseScore": 10.0,
    "baseSeverity": "CRITICAL",
    "vectorString": "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H"
  },
  "affected_versions": ["5.6.0", "5.6.1"],
  "references": [
    "https://www.openwall.com/lists/oss-security/2024/03/29/4",
    "https://tukaani.org/xz-backdoor/"
  ],
  "cisa_key": true.
}
```

Key Triage Signals

CVSS 10.0 + CISA KEV + EPSS 0.93 = Drop everything, patch NOW. This is a supply chain backdoor in a core library used by SSH.

Building a Remediation Plan

A good remediation plan is actionable, time-bound, and assigns clear ownership.

```
## Remediation Plan — Weekly Scan 2024-W14

### IMMEDIATE (24h) — Critical, Exploited in Wild
| # | CVE | Host | Action | Owner | SLA |
|---|-----|-----|-----|-----|-----|
| 1 | CVE-2024-3094 | 10.0.2.5 | Downgrade xz to 5.4.x | Ops Team | 24 hrs |
| 2 | CVE-2024-28219 | containers | Rebuild with pillow>=10.3.0 | DevOps | 24 hrs |

### HIGH PRIORITY (7 days)
| # | CVE | Host | Action | Owner | SLA |
|---|-----|-----|-----|-----|-----|
| 3 | CVE-2024-2398 | 10.0.1.15 | Update libcurl4 to deb12u5 | SysAdmin | 7 days |
| 4 | CVE-2024-2961 | 10.0.1.15 | Update libc6 to deb12u7 | SysAdmin | 7 days |
| 5 | CVE-2024-26130 | containers | Upgrade cryptography to 42.0.4 | DevOps | 7 days |
```

Shift Left: Scanning in CI/CD

Catch vulnerabilities before they reach production by integrating scanning into your pipeline.

```
# .github/workflows/security-scan.yml
name: Security Scan
on: [push, pull_request]

jobs:
  dependency-scan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Run pip-audit
        run: |
          pip install pip-audit
          pip-audit -r requirements.txt --fail-on-vuln
```

Gate, Don't Just Alert

Set `exit-code: 1` or `--fail-on-vuln` to block merges with critical vulnerabilities. Alerts without enforcement get ignored.

Common Vulnerability Management Pitfalls

Anti-Patterns

- **Scan and forget** — Running scans without acting on results
- **CVSS-only prioritization** — Ignoring context makes everything urgent
- **Alert fatigue** — Too many findings, no triage process
- **Patching without testing** — Causes outages, erodes trust in the process
- **No rollback plan** — When a patch breaks production

Best Practices

- **Continuous scanning** — Weekly minimum, daily for critical assets
- **Risk-based prioritization** — CVSS + EPSS + asset context
- **AI-assisted triage** — Scale human analysis with AI summaries
- **Patch windows** — Scheduled, tested, with rollback procedures
- **Metrics and reporting** — Track MTTR, SLA compliance, vuln debt

- **Shadow IT blind spots** — Cannot patch what you do not know about

- **Asset inventory** — Know what you have before scanning
-

Key Takeaways

- **Vulnerability management is a lifecycle** — discover, assess, prioritize, remediate, verify — repeat continuously
- **CVSS alone is not enough** — combine with EPSS, CISA KEV, and asset context for effective prioritization
- **AI accelerates every phase** — scan interpretation, triage, impact analysis, remediation planning, and reporting
- **Scan everything** — infrastructure, dependencies, containers, and IaC — shift left into CI/CD
- **Patch strategically** — test first, have rollback plans, use canary deployments for critical changes
- **Track and measure** — SLAs, MTTR, and vulnerability debt are your operational health metrics
- **Emergency processes matter** — pre-approved change procedures for zero-days save critical hours

Notes: Remind students that the goal is risk reduction, not zero vulnerabilities. Perfect is the enemy of good — focus on the highest-impact items first.

CTF Challenge: Vulnerability Triage & Remediation Plan

Scenario

Your organization just completed its weekly vulnerability scan. You have been handed the results and must produce an actionable remediation plan before the morning stand-up.

Your Mission

1. **Triage the scan** — Review the provided Nessus CSV (50+ findings) and Trivy JSON (container scan). Identify false positives and duplicates.
2. **Prioritize findings** — Use CVSS, EPSS data, CISA KEV status, and the provided asset inventory (with tier classifications) to rank findings by effective risk.
3. **Create a remediation plan** — Assign each finding to a remediation track (immediate / high / medium / accepted risk) with owner, action, and SLA deadline.

4. **Draft an AI prompt** — Write a prompt that would automate this triage process for next week's scan, including the context an AI would need.
5. **Bonus:** Identify one finding where the CVSS score is misleading (too high or too low given context) and explain why.

Success Criteria

Correct prioritization of Critical/High items, reasonable SLA assignments, at least one compensating control identified, and a reusable AI prompt for ongoing triage.
