



# Executive Security Report

Automated Security Audit

**acme/web-platform**

Branch: main · Date: 2026-02-10



# Security Overview

42

TOTAL FINDINGS

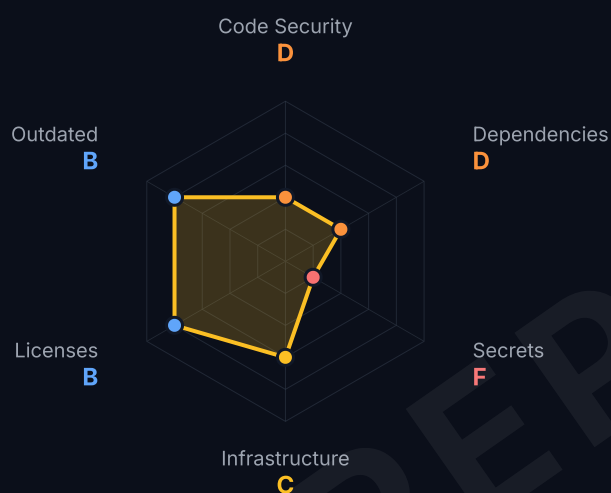
5

CRITICAL

13

HIGH

## SCORE BY CATEGORY



## FINDINGS BY SEVERITY



# Security Health Summary

The acme/web-platform project presents significant security risks that require immediate attention. Our automated analysis identified 5 critical and 13 high-severity vulnerabilities across the codebase, including exploitable SQL injection and cross-site scripting vectors in production API endpoints. The most alarming finding is a combination of injectable user input handling and exposed secrets — including a live Stripe secret key committed to the repository — that could enable an attacker to exfiltrate sensitive customer data, execute arbitrary database queries, and compromise payment processing. The application's dependency chain includes 6 packages with known CVEs, two of which allow remote code execution. While the infrastructure configuration and license compliance are in better shape, the code-level and secret-management gaps represent immediate risk to business operations and customer trust. We strongly recommend prioritizing the critical findings before the next deployment.

## Top Critical Risks

### SQL Injection & Data Exfiltration

**Business Impact:** An attacker can extract the entire user database — including hashed passwords, emails, and personal data — through the unparameterized search endpoint. Combined with the exposed database password in `docker-compose.yml`, this could lead to full database compromise, GDPR breach notification obligations, and potential regulatory fines.

**Recommendation:** Immediately parameterize all SQL queries using an ORM or query builder. Rotate database credentials and restrict direct database access to internal networks only.

### Secret Key Exposure & Payment Fraud

**Business Impact:** A live Stripe secret key (`sk_live_`) is committed in the repository history. Any developer, contractor, or attacker with repository access can process unauthorized charges, issue fraudulent refunds, or access customer payment data. The AWS secret key exposure additionally risks unauthorized cloud resource usage and data access.

**Recommendation:** Immediately rotate the Stripe key in the Stripe dashboard, revoke the AWS credentials in the IAM console, remove all secrets from the codebase, and implement a secrets management solution (environment variables or a vault). Audit Stripe logs for unauthorized transactions.

### Remote Code Execution via Dependencies

**Business Impact:** `jsonwebtoken@8.5.1` (CVE-2022-23529) allows remote code execution through crafted JWK sets, and `lodash@4.17.19` (CVE-2021-23337) enables command injection via the `template()` function. Both packages are directly imported in production code. An attacker exploiting either vulnerability gains full server access.

**Recommendation:** Update `jsonwebtoken` to `>=9.0.0` and `lodash` to `>=4.17.21`. Implement automated dependency scanning in CI/CD to catch future vulnerabilities. Run `npm audit` before every deployment.

# Priority Recommendations

**1** Parameterize all database queries and adopt a query builder or ORM to eliminate SQL injection risk across the entire codebase.

Effort: 2-3 days    Impact: Eliminates all SQL injection vectors (3 findings)

**1** Rotate and revoke all exposed secrets: Stripe API key, AWS credentials, database password, and RSA private key. Implement environment-only secret injection.

Effort: 2-4 hours    Impact: Prevents unauthorized access to payment, cloud, and database services

**1** Update all vulnerable dependencies to patched versions: jsonwebtoken  $\geq 9.0.0$ , lodash  $\geq 4.17.21$ , express  $\geq 4.19.2$ , axios  $\geq 1.6.0$ , pg  $\geq 8.11.0$ .

Effort: 1 day    Impact: Patches 6 known CVEs including 2 remote code execution

**2** Implement input validation with a schema validation library (e.g., Zod) and output encoding on all API endpoints to eliminate XSS, SSRF, and prototype pollution vectors.

Effort: 3-5 days    Impact: Eliminates injection and data integrity risks across all endpoints

**3** Harden Docker configuration: add non-root USER directive, define HEALTHCHECK, restrict exposed ports to internal networks, and upgrade base image from node:16 to node:20.

Effort: 0.5 day    Impact: Reduces container escape risk and patches 12 OS-level CVEs

## Remediation Estimate

**Estimated 2-3 weeks for a single developer to address all critical and high findings. We recommend starting with secret rotation (2-4 hours) and dependency updates (1 day) as they provide the highest risk reduction with minimal effort. SQL injection fixes (2-3 days) should follow immediately. Input validation hardening can be scheduled for the following sprint.**

# OWASP Top 10 Compliance

● A01	Broken Access Control	● A02	Cryptographic Failures
● A03	Injection	● A04	Insecure Design
● A05	Security Misconfiguration	● A06	Vulnerable Components
● A07	Auth Failures	● A08	Data Integrity Failures
● A09	Logging & Monitoring	● A10	SSRF

SAMPLE REPORT