



Technical Security Report

Automated Security Audit — Detailed Analysis

acme/web-platform

Branch: main · Date: 2026-02-10



Security Overview

42

TOTAL FINDINGS

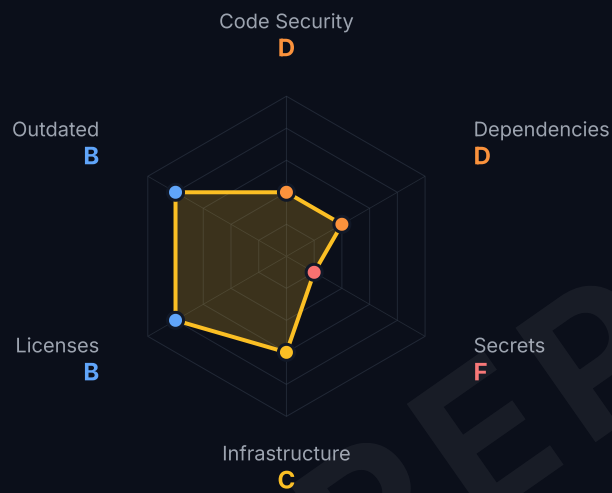
5

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

Detailed Findings

● CRITICAL SQL Injection in search endpoint

src/routes/api/search.ts:47

User-supplied query parameter is concatenated directly into a SQL string without parameterization or sanitization. An attacker can inject arbitrary SQL to extract, modify, or delete data from any table in the database.

Business Impact: Full database compromise. An attacker can exfiltrate all user records, payment data, and admin credentials in a single crafted request.

RECOMMENDED FIX

```
// VULNERABLE:
const results = await db.raw(
  `SELECT * FROM products WHERE name LIKE '%${req.query.q}%'`
);

// FIXED – use parameterized query:
const results = await db('products')
  .where('name', 'like', `%${req.query.q}%`);
```

<https://cwe.mitre.org/data/definitions/89.html>

https://owasp.org/Top10/A03_2021-Injection/

● CRITICAL Reflected XSS via URL parameter

src/routes/api/preview.ts:23

The preview endpoint renders a URL parameter directly into the HTML response without encoding. An attacker can craft a URL that executes arbitrary JavaScript in the victim's browser when clicked.

Business Impact: Session hijacking, credential theft, and phishing attacks. An attacker can steal authentication tokens and impersonate any user who clicks a crafted link.

RECOMMENDED FIX

```
// VULNERABLE:
res.send(`<h1>Preview: ${req.query.title}</h1>`);

// FIXED – escape HTML entities:
import { escapeHtml } from '../utils/sanitize';
res.send(`<h1>Preview: ${escapeHtml(req.query.title)}</h1>`);
```

<https://cwe.mitre.org/data/definitions/79.html>

https://owasp.org/Top10/A03_2021-Injection/

● CRITICAL eval() with user-controlled input

src/utils/template-engine.ts:89

The template engine uses eval() to process user-supplied template strings. This allows an attacker to execute arbitrary JavaScript on the server by injecting code through template variables.

Business Impact: Full server compromise. An attacker can execute system commands, read files, establish reverse shells, and pivot to internal infrastructure.

RECOMMENDED FIX

```
// VULNERABLE:
function renderTemplate(template: string, data: object) {
  return eval(`${template}`);
}

// FIXED – use a safe template library:
import Handlebars from 'handlebars';
function renderTemplate(template: string, data: object) {
  return Handlebars.compile(template)(data);
}
```

<https://cwe.mitre.org/data/definitions/95.html>

● HIGH Prototype pollution via recursive merge

src/utils/deep-merge.ts:15

The custom deep-merge utility does not check for __proto__, constructor, or prototype properties. An attacker can inject properties into Object.prototype, affecting all objects in the application.

Business Impact: Denial of service, authentication bypass, or remote code execution depending on downstream property access patterns.

RECOMMENDED FIX

```
// VULNERABLE:
function deepMerge(target: any, source: any) {
  for (const key in source) {
    target[key] = typeof source[key] === 'object'
      ? deepMerge(target[key] || {}, source[key])
      : source[key];
  }
  return target;
}

// FIXED – block dangerous keys:
const BLOCKED = new Set(['__proto__', 'constructor', 'prototype']);
function deepMerge(target: any, source: any) {
  for (const key in source) {
    if (BLOCKED.has(key)) continue;
    target[key] = typeof source[key] === 'object'
      ? deepMerge(target[key] || {}, source[key])
      : source[key];
  }
  return target;
}
```

<https://cwe.mitre.org/data/definitions/1321.html>

● HIGH Hardcoded JWT secret in source code

src/config/auth.ts:12

The JWT signing secret is hardcoded as a string literal in the authentication configuration file. Anyone with access to the source code can forge valid JWT tokens for any user.

Business Impact: Complete authentication bypass. An attacker can create admin-level JWT tokens and access any account without credentials.

RECOMMENDED FIX

```
// VULNERABLE:
const JWT_SECRET = 'super-secret-jwt-key-2024';

// FIXED - use environment variable:
const JWT_SECRET = process.env.JWT_SECRET;
if (!JWT_SECRET) throw new Error('JWT_SECRET must be set');
```

<https://cwe.mitre.org/data/definitions/798.html>

● HIGH Open redirect without URL validation

src/routes/api/oauth/callback.ts:34

The OAuth callback handler redirects to a user-supplied URL without validating the destination. An attacker can craft a login URL that redirects to a phishing site after authentication.

Business Impact: Phishing attacks leveraging trusted domain. Users are redirected to malicious sites after authenticating, potentially stealing credentials or OAuth tokens.

RECOMMENDED FIX

```
// VULNERABLE:
res.redirect(req.query.redirect_uri);

// FIXED - validate against allowlist:
const ALLOWED_ORIGINS = ['https://acme.com', 'https://app.acme.com'];
const url = new URL(req.query.redirect_uri);
if (!ALLOWED_ORIGINS.includes(url.origin)) {
  return res.status(400).json({ error: 'Invalid redirect' });
}
res.redirect(req.query.redirect_uri);
```

<https://cwe.mitre.org/data/definitions/601.html>

https://owasp.org/Top10/A01_2021-Broken_Access_Control/

● HIGH Path traversal in file download endpoint

src/routes/api/files/download.ts:28

The file download handler uses user input to construct file paths without sanitization. An attacker can use ../ sequences to read arbitrary files from the server filesystem.

Business Impact: Exposure of sensitive server files including /etc/passwd, environment variables, private keys, and application source code.

RECOMMENDED FIX

```
// VULNERABLE:
const filePath = path.join(UPLOADS_DIR, req.params.filename);
res.sendFile(filePath);

// FIXED – resolve and verify path stays within allowed directory:
const filePath = path.resolve(UPLOADS_DIR, req.params.filename);
if (!filePath.startsWith(path.resolve(UPLOADS_DIR))) {
  return res.status(403).json({ error: 'Access denied' });
}
res.sendFile(filePath);
```

<https://cwe.mitre.org/data/definitions/22.html>

● HIGH ReDoS vulnerability in email validation regex

src/utils/validators.ts:67

The email validation regex contains a pattern vulnerable to Regular Expression Denial of Service (ReDoS). A crafted input string causes exponential backtracking, consuming CPU and blocking the event loop.

Business Impact: Denial of service. An attacker can send a single crafted request that causes the server to hang for minutes, effectively taking the application offline.

RECOMMENDED FIX

```
// VULNERABLE:
const EMAIL_REGEX = /^( [a-zA-Z0-9_\-.]+)@([a-zA-Z0-9_\-.]+\.( [a-zA-Z]{2,5})$)/;

// FIXED – use a non-backtracking approach:
import { z } from 'zod';
const emailSchema = z.string().email();
```

<https://cwe.mitre.org/data/definitions/1333.html>

● MEDIUM Missing CSRF protection on state-changing endpoint

src/routes/api/settings.ts:15

The settings update endpoint accepts POST requests without CSRF token validation. An attacker can craft a malicious page that submits requests on behalf of authenticated users.

Business Impact: Account takeover via settings modification. An attacker can change user email addresses or connected integrations.

RECOMMENDED FIX

```
// Add CSRF middleware:
import csrf from 'csrf';
router.use(csrf({ cookie: true }));
```

<https://cwe.mitre.org/data/definitions/352.html>

● MEDIUM Insecure random number generator for tokens

src/utils/tokens.ts:8

Math.random() is used to generate password reset and email verification tokens. Math.random() is not cryptographically secure and produces predictable output.

Business Impact: Token prediction enables account takeover. An attacker can predict reset tokens and gain unauthorized access to user accounts.

RECOMMENDED FIX

```
// VULNERABLE:
const token = Math.random().toString(36).slice(2);

// FIXED — use crypto.randomBytes:
import { randomBytes } from 'node:crypto';
const token = randomBytes(32).toString('hex');
```

<https://cwe.mitre.org/data/definitions/330.html>

● MEDIUM Missing rate limiting on authentication endpoints

src/routes/api/auth/login.ts:12

The login and registration endpoints have no rate limiting configured. An attacker can perform brute-force password attacks or credential stuffing at unlimited speed.

Business Impact: Credential compromise through brute force. Accounts with weak passwords are vulnerable to automated attacks.

RECOMMENDED FIX

```
// Add rate limiting middleware:
import rateLimit from 'express-rate-limit';
const authLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 5,
  message: 'Too many attempts'
});
router.post('/login', authLimiter, loginHandler);
```

<https://cwe.mitre.org/data/definitions/307.html>

● MEDIUM Verbose error messages exposing internal details

src/middleware/error-handler.ts:22

The error handler sends full stack traces and internal error details in API responses. This leaks implementation details including file paths, library versions, and database schema information.

Business Impact: Information disclosure aids targeted attacks. Exposed stack traces reveal internal architecture to potential attackers.

RECOMMENDED FIX

```
// VULNERABLE:
res.status(500).json({
  error: err.message,
  stack: err.stack,
  query: err.query
});

// FIXED – generic error in production:
res.status(500).json({
  error: process.env.NODE_ENV === 'production'
    ? 'Internal server error'
    : err.message
});
```

<https://cwe.mitre.org/data/definitions/209.html>

● MEDIUM Session cookie without Secure flag

src/config/session.ts:18

The session cookie is configured without the Secure flag, allowing it to be transmitted over unencrypted HTTP connections. An attacker on the same network can intercept the cookie.

Business Impact: Session hijacking on non-HTTPS connections. Users on public Wi-Fi are particularly vulnerable to cookie theft.

RECOMMENDED FIX

```
// VULNERABLE:
app.use(session({
  cookie: { httpOnly: true }
}));

// FIXED – add Secure and SameSite flags:
app.use(session({
  cookie: { httpOnly: true, secure: true, sameSite: 'strict' }
}));
```

<https://cwe.mitre.org/data/definitions/614.html>

● MEDIUM Missing Content-Security-Policy header

src/middleware/headers.ts:5

The application does not set a Content-Security-Policy header. This makes XSS exploitation easier as browsers do not restrict script sources.

Business Impact: Amplifies XSS impact. Without CSP, injected scripts can load external resources, exfiltrate data to any domain, and execute cryptocurrency miners.

RECOMMENDED FIX

```
// Add CSP header:
app.use((req, res, next) => {
  res.setHeader('Content-Security-Policy',
    "default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline'"
  );
  next();
});
```

<https://cwe.mitre.org/data/definitions/1021.html>

● MEDIUM Unrestricted file upload type

src/routes/api/upload.ts:31

The file upload endpoint accepts any file type without validation. An attacker can upload executable files, web shells, or malicious scripts to the server.

Business Impact: Remote code execution via uploaded web shells. An attacker can gain persistent server access through a malicious file upload.

RECOMMENDED FIX

```
// VULNERABLE:
upload.single('file');

// FIXED - validate file type:
const ALLOWED_TYPES = ['image/jpeg', 'image/png', 'application/pdf'];
if (!ALLOWED_TYPES.includes(req.file.mimetype)) {
  return res.status(400).json({ error: 'Invalid file type' });
}
```

<https://cwe.mitre.org/data/definitions/434.html>

● MEDIUM Weak password hashing with low bcrypt cost

src/services/user-service.ts:45

Passwords are hashed using bcrypt with a cost factor of 4 (default minimum). This makes brute-force cracking trivially fast — a modern GPU can test billions of hashes per second at this cost level.

Business Impact: Password database cracking. If the database is compromised, weak hashing means all passwords can be cracked within hours.

RECOMMENDED FIX

```
// VULNERABLE:
const hash = await bcrypt.hash(password, 4);

// FIXED — use cost factor of 12 (or switch to argon2id):
const hash = await bcrypt.hash(password, 12);
// Or better: use argon2id
import argon2 from 'argon2';
const hash = await argon2.hash(password);
```

<https://cwe.mitre.org/data/definitions/916.html>

● CRITICAL lodash@4.17.19 — Command Injection via template()

package.json

lodash versions before 4.17.21 are vulnerable to command injection through the template() function when used with user-controlled input. The vulnerability allows execution of arbitrary shell commands on the server.

Business Impact: Remote code execution. An attacker can execute arbitrary system commands on the server, potentially gaining full control of the infrastructure.

RECOMMENDED FIX

```
Update lodash to >=4.17.21:
npm install lodash@latest
```

<https://nvd.nist.gov/vuln/detail/CVE-2021-23337>

<https://github.com/lodash/lodash/pull/5085>

● CRITICAL jsonwebtoken@8.5.1 — Insecure Key Retrieval (RCE)

package.json

jsonwebtoken versions before 9.0.0 do not properly validate the secretOrPublicKey parameter. An attacker can supply a crafted JWK that triggers code execution during key retrieval.

Business Impact: Remote code execution through crafted JWT tokens. An attacker who can supply a JWT with a manipulated header can execute arbitrary code on the server.

RECOMMENDED FIX

```
Update jsonwebtoken to >=9.0.0:
npm install jsonwebtoken@latest
```

<https://nvd.nist.gov/vuln/detail/CVE-2022-23529>

<https://github.com/auth0/node-jwebtoken/releases/tag/v9.0.0>

● HIGH **express@4.17.1 — Open Redirect in res.redirect()**

package.json

Express versions before 4.19.2 have an open redirect vulnerability in the res.redirect() function that does not properly validate URLs with backslashes, allowing attackers to redirect users to malicious sites.

Business Impact: Phishing attacks using trusted URLs. Attackers can craft links that appear to originate from your domain but redirect to credential-harvesting sites.

RECOMMENDED FIX

Update express to >=4.19.2:
npm install express@latest

<https://nvd.nist.gov/vuln/detail/CVE-2024-29041>

● HIGH **axios@0.21.1 — SSRF via Request Interceptor Bypass**

package.json

axios versions before 1.6.0 are vulnerable to Server-Side Request Forgery (SSRF). An attacker can bypass request interceptors and make the server send requests to internal services or cloud metadata endpoints.

Business Impact: Access to internal services and cloud metadata. An attacker can retrieve AWS/GCP credentials from metadata endpoints or scan internal infrastructure.

RECOMMENDED FIX

Update axios to >=1.6.0:
npm install axios@latest

<https://nvd.nist.gov/vuln/detail/CVE-2023-45857>

● HIGH **pg@8.7.1 — SQL Injection via Connection Parameters**

package.json

pg (node-postgres) versions before 8.11.0 are vulnerable to SQL injection through specially crafted connection parameters. An attacker who can control connection string components can inject arbitrary SQL.

Business Impact: Database compromise through connection-level injection. Even parameterized queries become unsafe if the connection itself is compromised.

RECOMMENDED FIX

Update pg to >=8.11.0:
npm install pg@latest

<https://nvd.nist.gov/vuln/detail/CVE-2024-21511>

● HIGH node:16-alpine — 12 CVEs including 2 critical (OpenSSL)

Dockerfile

The base Docker image node:16-alpine contains 12 known vulnerabilities in system packages, including 2 critical OpenSSL flaws that affect TLS certificate validation and cryptographic operations. Node.js 16 is past end-of-life and no longer receives security patches.

Business Impact: TLS security bypass and potential man-in-the-middle attacks. The expired runtime also means no future patches for Node.js-level vulnerabilities.

RECOMMENDED FIX

Update the base image to node:20-alpine in the Dockerfile:
FROM node:20-alpine

<https://endoflife.date/nodejs>

● HIGH AWS Secret Access Key exposed in .env.example

.env.example:12

A real AWS Secret Access Key (AKIA...) is present in .env.example, which is tracked by git. This file is intended as a template and is visible to anyone who clones the repository.

Business Impact: Unauthorized AWS resource access. An attacker can use these credentials to access S3 buckets, spin up EC2 instances for crypto mining, or pivot to other AWS services.

RECOMMENDED FIX

Replace real values in .env.example with placeholders:
AWS_SECRET_ACCESS_KEY=your-secret-key-here

Then revoke the exposed key in IAM and generate a new one.

https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html

● HIGH Stripe live secret key committed in source

src/config/payments.ts:8

A Stripe live secret key (sk_live_...) is hardcoded in the payments configuration file. This key grants full access to the Stripe account: create charges, refunds, and access customer payment data.

Business Impact: Financial fraud and payment data exposure. An attacker can process unauthorized charges, issue refunds to attacker-controlled accounts, and access stored customer card details.

RECOMMENDED FIX

Move to environment variable:
const STRIPE_KEY = process.env.STRIPE_SECRET_KEY;

Rotate the key immediately in the Stripe dashboard.

<https://stripe.com/docs/keys>

● HIGH RSA private key committed to repository

certs/server.pem:1

A PEM-encoded RSA private key file is committed to the repository. Private keys should never be stored in version control as they grant cryptographic identity.

Business Impact: TLS impersonation and data decryption. An attacker with the private key can impersonate the server, perform man-in-the-middle attacks, and decrypt intercepted traffic.

RECOMMENDED FIX

Remove the private key from the repository:

```
git rm certs/server.pem
```

Add *.pem to .gitignore. Generate a new key pair and store securely.

https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/04-Testing_for_Weak_Encryption

● HIGH Database password hardcoded in docker-compose.yml

docker-compose.yml:24

The PostgreSQL password is hardcoded as a plain string in docker-compose.yml, which is committed to the repository. This password is used in production.

Business Impact: Direct database access. Anyone with repository access can connect to the production database if it is network-accessible.

RECOMMENDED FIX

Use environment variables or Docker secrets:

environment:

```
POSTGRES_PASSWORD: ${DB_PASSWORD}
```

Store the actual password in a .env file excluded from git.

<https://docs.docker.com/compose/environment-variables/>

● MEDIUM Dockerfile runs as root user

Dockerfile:1

The Dockerfile does not include a USER directive, causing the application to run as root inside the container. If the application is compromised, the attacker has root privileges within the container.

Business Impact: Escalated container compromise. Root access facilitates container escape and lateral movement to the host system.

RECOMMENDED FIX

Add a non-root user:

```
RUN addgroup -S appuser && adduser -S appuser -G appuser
```

```
USER appuser
```

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#user

● MEDIUM PostgreSQL port 5432 exposed to all interfaces

docker-compose.yml:14

The docker-compose.yml file maps PostgreSQL port 5432 to the host on all interfaces (0.0.0.0:5432). This makes the database accessible from any network, not just localhost.

Business Impact: Direct database access from the internet. Combined with the exposed password, an attacker can connect to the database remotely.

RECOMMENDED FIX

```
Bind to localhost only:
ports:
  - "127.0.0.1:5432:5432"
```

<https://docs.docker.com/compose/networking/>

● MEDIUM No HEALTHCHECK defined in Dockerfile

Dockerfile:22

The Dockerfile does not define a HEALTHCHECK instruction. Without health checks, the orchestrator cannot detect when the application becomes unresponsive and restart it.

Business Impact: Extended downtime during failures. The application may be running but unresponsive with no automatic recovery.

RECOMMENDED FIX

```
Add a health check:
HEALTHCHECK --interval=30s --timeout=5s --retries=3 \
  CMD wget -qO- http://localhost:3000/health || exit 1
```

<https://docs.docker.com/engine/reference/builder/#healthcheck>

● LOW GPL-3.0 dependency in proprietary project

package-lock.json

The dependency node-forge@1.3.1 is licensed under GPL-3.0, which requires derivative works to also be distributed under GPL-3.0. This is incompatible with a proprietary/closed-source project.

Business Impact: Legal compliance risk. Distributing a proprietary application that includes GPL-3.0 code may require open-sourcing the entire project or face legal action.

RECOMMENDED FIX

```
Replace node-forge with an MIT/ISC-licensed alternative:
npm uninstall node-forge
npm install @peculiar/webcrypto # MIT licensed
```

<https://www.gnu.org/licenses/gpl-3.0.en.html>

● INFO **React 17 in use — React 19 available**

The project uses React 17. React 19 offers improved performance, automatic batching, and new hooks. Consider upgrading.

Business Impact: No immediate security impact. Missing performance improvements and new features.

RECOMMENDED FIX

```
npm install react@latest react-dom@latest
```

● INFO **TypeScript 4.9 — TypeScript 5.x available**

TypeScript 4.9 is in use. Version 5.x includes improved type checking, decorators support, and faster compilation.

Business Impact: No security impact. Improved developer experience available.

RECOMMENDED FIX

```
npm install typescript@latest
```

● INFO **No .nvmrc or .node-version file**

No Node.js version pinning file found. Different developers may run different Node.js versions, causing inconsistencies.

Business Impact: Potential build inconsistencies across environments.

RECOMMENDED FIX

```
echo "20" > .nvmrc
```

● INFO **No lockfile integrity check in CI**

The CI pipeline does not use `--frozen-lockfile` or equivalent, allowing dependency resolution differences between environments.

Business Impact: Supply chain risk from inconsistent dependency resolution.

RECOMMENDED FIX

Use `npm ci` instead of `npm install` in CI pipelines.

● INFO **console.log statements in production code**

23 `console.log` statements found in production source files. These may leak sensitive information in production logs.

Business Impact: Potential information disclosure via server logs.

RECOMMENDED FIX

Replace `console.log` with a structured logger (e.g., `pino`, `winston`).

● INFO **No CORS configuration detected**

No explicit CORS configuration found. The API may accept requests from any origin by default.

Business Impact: Cross-origin data access if sensitive endpoints are exposed.

RECOMMENDED FIX

Configure CORS with explicit allowed origins.

● INFO **Missing X-Content-Type-Options header**

The X-Content-Type-Options: nosniff header is not set. Browsers may MIME-sniff responses, potentially treating non-executable content as scripts.

Business Impact: Minor XSS amplification risk.

RECOMMENDED FIX

Add helmet middleware: `app.use(helmet())`

● INFO **No Referrer-Policy header**

The Referrer-Policy header is not configured. Browser may send full URL in the Referer header to external sites.

Business Impact: URL-based information leakage.

RECOMMENDED FIX

Set Referrer-Policy: `strict-origin-when-cross-origin`

● INFO **webpack-dev-server dependency detected**

webpack-dev-server is listed in production dependencies instead of devDependencies.

Business Impact: Larger production bundle and potential dev-only code in production.

RECOMMENDED FIX

Move to devDependencies in package.json.

● INFO **No automated security scanning in CI/CD**

No security scanning tools (SAST, SCA, secret detection) are configured in the CI/CD pipeline.

Business Impact: Vulnerabilities may be introduced without detection.

RECOMMENDED FIX

Add Shieldify scan to your CI pipeline for automated security scanning.

● INFO **Potentially unused dependencies detected**

4 packages in dependencies appear unused in the codebase: moment, bluebird, request, underscore.

Business Impact: Increased attack surface from unnecessary packages.

RECOMMENDED FIX

Remove unused dependencies: `npm uninstall moment bluebird request underscore`

● INFO **Database migrations not version-controlled**

No database migration files found in the repository. Schema changes may be applied manually.

Business Impact: Deployment inconsistencies and difficult rollbacks.

RECOMMENDED FIX

Adopt a migration tool like Prisma Migrate or knex migrations.

Vulnerable Dependency Tree

PACKAGE	VERSION	VULNERABILITIES	USED BY
lodash	4.17.19	CVE-2021-23337 (COMMAND INJECTION)	express-validator@6.14.0, webpack@5.75.0
jsonwebtoken	8.5.1	CVE-2022-23529 (RCE)	passport-jwt@4.0.1
express	4.17.1	CVE-2024-29041 (OPEN REDIRECT)	
axios	0.21.1	CVE-2023-45857 (SSRF)	stripe@11.0.0
pg	8.7.1	CVE-2024-21511 (SQL INJECTION)	knex@2.4.0
node-forge	1.3.1	GPL-3.0 LICENSE (INCOMPATIBLE)	selfsigned@2.1.1, webpack-dev-server@4.11.0

Scanner Execution Details

SCANNER	CATEGORY	FINDINGS	DURATION	STATUS
Semgrep	SAST	16	45.2s	OK
Trivy (SCA)	Dependency Audit	6	12.8s	OK
Trivy (IaC)	Infrastructure	3	3.4s	OK
Gitleaks	Secrets	4	8.9s	OK
license-checker	Licenses	1	2.1s	OK
npm audit	SCA (npm)	4	5.6s	OK

SAMPLE REPORT