CodeGuard

Enterprise Code Review Report

6/1/2026, 7:07:58 AM

payments-service (sample)

AI engine (Claude · claude-sonnet-4-6) · 4 files analyzed · Est. cost ~$0.16

D

OVERALL GRADE · 58/100

3	7	2	1	0
CRITICAL	HIGH	MEDIUM	LOW	INFO

1. Executive Verdict

The payments-service codebase presents a dangerous combination of critical security flaws, outdated vulnerable dependencies, and structural quality issues that make it unfit for production in its current state. Two critical vulnerabilities in src/payment.js — use of Math.random() for security token generation (CWE-338) and live eval() execution of presumably dynamic input (CWE-95) — alone warrant a production block. The TLS certificate verification being explicitly disabled in src/config.js:3 (CWE-295) means every outbound HTTPS call silently accepts forged certificates, fully exposing the service to man-in-the-middle attacks on payment traffic. Debug mode is hardcoded on (config.js:5) and log level is set to 'trace' unconditionally (config.js:6), guaranteeing that sensitive financial data will appear in logs. All four declared dependencies carry known CVEs and are multiple major or minor versions behind current. No test files were analyzed, suggesting absent or negligible test coverage. The only partial bright spot is that a crypto module import exists — though it is unused — indicating some developer awareness of secure primitives that was not followed through.

STRONGEST ASPECTS

+ The `crypto` module is imported in src/payment.js:1, showing intent to use a cryptographically secure primitive — a correct instinct that just needs to be acted on by replacing the Math.random() token at line 5.
+ The project uses a package.json (JSON:1) with explicit dependency declarations, giving a clear, machine-readable upgrade path: each of the four outdated packages can be bumped with a single npm update command once CVEs are validated.
+ The configuration concerns are isolated to a single file (src/config.js), meaning a targeted refactor to environment-variable-driven config (e.g., dotenv) would remediate three distinct findings (lines 3, 5, 6) in one focused change.

WEAKEST ASPECTS

- src/payment.js:5 and :15 contain two critical vulnerabilities (weak PRNG for security tokens and unrestricted eval()) in close proximity, indicating a pattern of unsafe coding practice rather than an isolated mistake.
- src/config.js:3, :5, and :6 collectively disable TLS verification, enable debug mode, and set trace-level logging with no environment gating — three production-breaking settings hardcoded together in the same file.
- package.json lists four dependencies (lodash@4.17.10, axios@0.21.1, express@4.17.1, jsonwebtoken@8.5.1)

all carrying documented CVEs, meaning the attack surface extends beyond the first-party code into the supply chain.

HIGHEST-ROI FIX

Remove eval() at src/payment.js:15 immediately (replace with a safe, explicit data-parsing alternative such as JSON.parse or a whitelist-driven dispatch map) — this eliminates the highest-severity, most directly exploitable vector (RCE) with an estimated 1–2 hours of developer effort touching only src/payment.js. Simultaneously replace Math.random() on line 5 with crypto.randomBytes() (the already-imported module) in the same session.

2. Scorecard

DIMENSION	GRADE	JUSTIFICATION
Correctness	D	Math.random() at src/payment.js:5 produces predictable tokens; eval() at :15 is functionally incorrect for safe data handling.
Efficiency & performance	C	No algorithmic inefficiencies identified in analyzed files, but dead import (src/payment.js:1) and lack of environment gating add unnecessary runtime overhead.
Architecture & boundaries	C	Configuration is hardcoded in src/config.js with no environment abstraction; DOM APIs appear in a Node.js module at src/payment.js:10.
Error handling & resilience	D	No evidence of error handling around eval() at src/payment.js:15; TLS errors are suppressed at src/config.js:3.
Security posture	F	Two critical CVEs (CWE-338 at payment.js:5, CWE-95 at payment.js:15), XSS at :10, TLS bypass at config.js:3 — four exploitable vulnerabilities across two files.
Concurrency & state	C	No concurrency primitives identified in analyzed files; cannot confirm thread-safety of token generation at src/payment.js:5.
API / contract design	C	express@4.17.1 in package.json carries CVE-601; no API input validation or sanitization visible in analyzed files.
Test coverage & quality	F	Zero test files were among the 4 files analyzed; no test framework declared in package.json findings.
Readability & idiom	C	Dead import at src/payment.js:1 and mixed environment concerns in src/config.js:1-6 reduce clarity; no structural issues beyond these.
Documentation & comments	D	No inline documentation, JSDoc, or README content was identified in any of the 4 analyzed files.
Dependency hygiene	F	All 4 declared dependencies in package.json carry known CVEs: lodash@4.17.10 (CWE-1321), axios@0.21.1 (CWE-918), express@4.17.1 (CWE-601), jsonwebtoken@8.5.1 (CWE-347).
Build & tooling	C	package.json present for dependency management, but no CI pipeline, linting, or security scanning configuration was identified in the analyzed files.

3. Dimension Assessments

D

Correctness

Two critical logic errors undermine the correctness of the service's core operations. At src/payment.js:5, Math.random() generates tokens that are predictable and not suitable for security-sensitive use. At src/payment.js:15, eval() is used where structured parsing should be, meaning the function's behavior is undefined for any input containing executable syntax. The unused crypto import at line 1 confirms the correct solution was considered but not implemented.

C

Efficiency & performance

The four analyzed files show no evidence of O(n²) algorithms or obvious memory leaks. However, the unused crypto import at src/payment.js:1 adds a module load with no benefit. Trace-level logging at src/config.js:6 will generate high log volume in any environment, increasing I/O cost and storage consumption. These are low-impact individually but signal a lack of performance discipline.

C

Architecture & boundaries

src/config.js:1 has no environment-awareness — all settings are static constants rather than environment-variable reads, meaning separate config files or code changes are required per deployment target. src/payment.js:10 references browser DOM APIs (document/innerHTML) inside what appears to be a Node.js module, suggesting either a confused module boundary or copy-paste from a frontend context. These are structural issues that will compound maintenance cost as the service grows.

D

Error handling & resilience

eval() at src/payment.js:15 has no visible try/catch, meaning any syntax or runtime error in evaluated input will propagate uncaught. More critically, disabling TLS verification at src/config.js:3 silently swallows certificate errors that should halt execution — this is anti-pattern error suppression with direct security consequences. No error handling patterns were identifiable in the 4 analyzed files beyond these negative examples.

F

Security posture

This dimension alone disqualifies the service from production. src/payment.js:5 uses Math.random() for security token generation (CWE-338), making tokens guessable. src/payment.js:15 calls eval() on input (CWE-95), enabling remote code execution. src/payment.js:10 assigns unsanitized input to innerHTML (CWE-79), enabling XSS. src/config.js:3 sets rejectUnauthorized: false or equivalent (CWE-295), disabling TLS certificate validation and enabling MITM on all outbound payment calls. These are not theoretical risks — each has a direct, well-documented exploitation path.

C

Concurrency & state

The four files analyzed contain no explicit concurrency code (no worker threads, no async lock patterns, no shared mutable state identified). However, the token generation at src/payment.js:5 using Math.random() is not safe under concurrent request load due to state predictability. No race conditions were directly observed, but the absence of any concurrency patterns in a payments service handling parallel transactions is itself a gap that warrants investigation beyond the analyzed files.

C

API / contract design

The service declares Express as a dependency (package.json) but the analyzed files show no input validation, schema enforcement, or request sanitization layer. The innerHTML assignment at src/payment.js:10 is direct evidence that user-supplied data flows to output without sanitization. The outdated express@4.17.1 (CWE-601, open-redirect vulnerability) further weakens the API surface. A payments API handling financial data should enforce strict input contracts at every endpoint.

F

Test coverage & quality

No test files (unit, integration, or end-to-end) appear in the analyzed file set. No test runner or assertion library was identified in the package.json dependency findings. For a payments service, the absence of tests means none of the critical paths — token generation, payment processing, external HTTP calls — have any automated regression protection. This is not inferred; it is a direct observation from the analyzed file manifest.

C

Readability & idiom

The unused crypto import at src/payment.js:1 creates confusion about intent — a reader cannot tell if secure randomness was intended and abandoned, or is planned but incomplete. src/config.js conflates TLS settings, debug flags, and log levels without grouping or comments, making it harder to audit at a glance. These are localized issues; the files are otherwise small enough to read in full, limiting the readability damage.

D

Documentation & comments

None of the four analyzed files contain function-level comments, JSDoc annotations, or inline explanations. For a payments service — a domain where intent and security assumptions must be explicitly stated — this means there is no documented contract for any function, no explanation of why TLS verification is disabled (src/config.js:3), and no guidance on the expected input format for the eval() call (src/payment.js:15). The absence of documentation amplifies every other risk in this audit.

F

Dependency hygiene

Every single declared dependency is outdated and carries a documented vulnerability. lodash@4.17.10 is vulnerable to prototype pollution (CWE-1321). axios@0.21.1 is vulnerable to SSRF (CWE-918). express@4.17.1 carries an open-redirect CVE (CWE-601). jsonwebtoken@8.5.1 has a signature validation bypass (CWE-347) — particularly severe for a payments service relying on JWTs for auth. Current safe versions exist for all four packages and should be adopted immediately.

C

Build & tooling

A package.json exists, which provides baseline dependency management. However, no build tooling configuration (Webpack, ESBuild, etc.), no linter configuration (.eslintrc), no CI pipeline definition, and no dependency audit step (e.g., npm audit in CI) were present in the analyzed file set. The absence of automated linting and security scanning in the build pipeline allowed all 13 identified findings to reach the audit stage uncaught.

4. In Plain Language

This payments service has several serious security problems that could allow attackers to steal payment data, run their own code on the server, or intercept encrypted communications. Additionally, all four software libraries it depends on are outdated versions with known security holes. The system is also configured to write detailed debug logs that may expose sensitive customer or financial information. None of this should be deployed to a live environment until these issues are fixed.

5. Detailed Findings

1. Cryptographically Weak PRNG Used for Security Token

CRITICAL · SECURITY · CWE-338

src/payment.js:5

Line 5: `Math.random()` is a pseudorandom number generator (PRNG) that is not cryptographically secure. Its output is deterministic and predictable given enough observations. Using it to generate a security token (e.g., session token, CSRF token, payment nonce) allows an attacker to predict or brute-force valid tokens. The `crypto` module is already imported at line 1 but is completely unused.

```
return Math.random().toString(36).slice(2);
```

Fix: Replace with `crypto.randomBytes(32).toString('hex')` (already imported at line 1). This produces 256 bits of cryptographically secure randomness, which is suitable for security-sensitive tokens. Remove the unused `Math.random()` call entirely.

2. Arbitrary Code Execution via `eval()`

CRITICAL · SECURITY · CWE-95

src/payment.js:15

Line 15: `eval(rule)` executes whatever string is passed as `rule` as live JavaScript. If `rule` is derived from any external source (user input, API response, URL parameter, database value), an attacker can run arbitrary code in the application's context — exfiltrating data, modifying payment amounts, or achieving full process compromise. Even if the caller today passes only controlled strings, this construct is one refactor away from a critical injection.

```
return eval(rule);
```

Fix: Remove `eval` entirely. Represent business rules as structured data (e.g., a JSON config or a plain object mapping rule names to functions) and dispatch to them explicitly. If dynamic expression evaluation is genuinely required, use a sandboxed parser such as `expr-eval` or `jsep` that does not have access to the JavaScript runtime.

3. TLS certificate verification explicitly disabled

CRITICAL · SECURITY · CWE-295

src/config.js:3

Line 3 sets `rejectUnauthorized: false` in the exported HTTPS options. This disables all TLS certificate chain and hostname validation, meaning any certificate — including self-signed, expired, or attacker-controlled ones — will be accepted. Any outbound HTTPS connection made using this config is fully vulnerable to man-in-the-middle (MITM) attacks; an attacker on the network path can intercept and modify traffic silently.

```
https: { rejectUnauthorized: false },
```

Fix: Remove `rejectUnauthorized: false` entirely; the default value is `true`. If a private/internal CA is in use, supply the CA certificate via the `ca` option (e.g., `ca: fs.readFileSync('internal-ca.pem')`) rather than bypassing verification. Never ship `rejectUnauthorized: false` in production or committed configuration.

4. Reflected XSS via Unsanitized `innerHTML` Assignment

HIGH · SECURITY · CWE-79

src/payment.js:10

Line 10: The `amount` value is concatenated directly into an HTML string and assigned to `element.innerHTML`. If `amount` is not a validated, trusted number (e.g., it originates from a URL parameter, API response, or user input), an attacker can inject arbitrary HTML/JavaScript — for example, `amount = '<img src=x onerror=stealCard()>'`. In a payment context this is especially dangerous as it can be used to overlay fake forms or exfiltrate card data.

```
document.getElementById("total").innerHTML = "Total: " + amount;
```

Fix: Use `element.textContent` instead of `innerHTML` when inserting plain text: `document.getElementById('total').textContent = 'Total: ' + amount;`. This treats the value as text, not markup, preventing any HTML injection. Additionally, validate that `amount` is a finite number before rendering.

5. Debug mode enabled unconditionally

HIGH · SECURITY · CWE-215

src/config.js:5

Line 5 hardcodes `debug: true` with no environment guard. Depending on how consuming code uses this flag, debug mode can activate additional code paths, expose internal state, disable security-relevant checks, or emit stack traces to clients — all of which expand the attack surface in production.

```
debug: true,
```

Fix: Gate debug mode on the runtime environment: `debug: process.env.NODE_ENV !== 'production'`. Prefer `false` as the safe default and opt-in only for local development.

6. Log level set to 'trace', risking sensitive data exposure in logs

HIGH · COMPLIANCE · CWE-532

src/config.js:6

Line 6 sets `logLevel: 'trace'` unconditionally. Trace-level logging typically captures full request/response bodies, headers (which may contain Authorization tokens, cookies, or API keys), query parameters, and internal state. Persisting or transmitting this data in production logs violates the principle of minimum necessary data retention and is frequently non-compliant with GDPR, PCI-DSS, and SOC 2 controls requiring protection of sensitive/PII data.

```
logLevel: "trace",
```

Fix: Set the default log level to `'warn'` or `'error'` for production. Use an environment-driven override: `logLevel: process.env.LOG_LEVEL || 'warn'`. Ensure trace/debug log levels are explicitly prohibited in production deployment runbooks and CI gating.

7. Outdated dependency: lodash@4.17.10

HIGH · DEPENDENCY · CWE-1321

package.json

lodash 4.17.10 is below the patched 4.17.21. Prototype pollution (CVE-2021-23337).

```
"lodash": "4.17.10"
```

Fix: Upgrade lodash to >= 4.17.21 and re-run your audit.

8. Outdated dependency: axios@0.21.1

HIGH · DEPENDENCY · CWE-918

package.json

axios 0.21.1 is below the patched 1.6.0. SSRF / credential leak via redirects.

```
"axios": "0.21.1"
```

Fix: Upgrade axios to >= 1.6.0 and re-run your audit.

9. Outdated dependency: express@4.17.1

HIGH · DEPENDENCY · CWE-601

package.json

express 4.17.1 is below the patched 4.19.2. Open redirect in older releases.

```
"express": "4.17.1"
```

Fix: Upgrade express to >= 4.19.2 and re-run your audit.

10. Outdated dependency: jsonwebtoken@8.5.1

HIGH · DEPENDENCY · CWE-347

package.json

jsonwebtoken 8.5.1 is below the patched 9.0.0. Algorithm confusion / weak verification.

```
"jsonwebtoken": "8.5.1"
```

Fix: Upgrade jsonwebtoken to >= 9.0.0 and re-run your audit.

11. Imported `crypto` Module Is Never Used

MEDIUM · QUALITY

src/payment.js:1

Line 1: The `crypto` module is required but no function in the file calls any of its APIs. This is dead code, which creates confusion about intent and — critically in this file — obscures the fact that `Math.random()` is being used instead of the secure alternative that `crypto` provides.

```
const crypto = require("crypto");
```

Fix: Either remove the import if it is truly not needed, or (strongly preferred) wire it into `newToken()` as described in the insecure-randomness finding. Dead imports in security-sensitive modules mislead reviewers into assuming cryptographic functions are in use.

12. Configuration file has no environment-awareness

MEDIUM · QUALITY

src/config.js:1

All three settings (lines 3, 5, 6) are static literals with no branching on `NODE_ENV` or any other runtime signal. A single config object that is equally applied to development, staging, and production makes it structurally impossible to enforce different security postures per environment without modifying the file itself — increasing the risk that a developer shortcut (such as those on lines 3, 5, and 6) reaches production.

```
module.exports = {
```

Fix: Adopt an environment-layered config pattern (e.g., `node-config`, `dotenv` with per-environment files, or explicit `process.env.NODE_ENV` branching). Production values should be the hardcoded safe defaults; insecure development overrides should require an explicit, local-only opt-in that cannot be committed.

13. Browser DOM API (`document`) Used in a Node.js Module

LOW · QUALITY

src/payment.js:10

Line 10: `document.getElementById` is a browser DOM API. This file also uses `module.exports` (line 18), which is a Node.js CommonJS construct. These two environments are mutually exclusive at runtime. Calling `render()` in a Node.js environment will throw `ReferenceError: document is not defined`. The module makes no environment guard or abstraction.

```
document.getElementById("total").innerHTML = "Total: " + amount;
```

Fix: Separate browser-side rendering code from server-side/shared logic into distinct modules. If this file is intended to run only in the browser, remove `module.exports` and use ES module syntax (`export`). If it must be isomorphic, inject the DOM reference as a parameter or dependency so it can be mocked/replaced per environment.

6. Remediation Prompt

Paste this into an AI coding agent to drive the fixes:

```
You are a security-focused software engineer. Fix the following verified issues in the
payments-service codebase. For each fix, apply the minimal, targeted change described. Do not
refactor unrelated code.

---

ISSUE 1 — CRITICAL | src/payment.js:15 | CWE-95: Arbitrary Code Execution via eval()
Remove the eval() call entirely. Replace it with JSON.parse() if the input is a JSON string,
or with an explicit whitelist-driven dispatch map if it selects a function by name. eval()
must not exist anywhere in this file after your change.

ISSUE 2 — CRITICAL | src/payment.js:5 | CWE-338: Weak PRNG for Security Token
Replace Math.random() with crypto.randomBytes(32).toString('hex'). The `crypto` module is
already imported at line 1 — use it. Remove the Math.random() call entirely.

ISSUE 3 — CRITICAL | src/config.js:3 | CWE-295: TLS Certificate Verification Disabled
Remove or change the setting that disables TLS certificate verification (e.g.,
rejectUnauthorized: false or NODE_TLS_REJECT_UNAUTHORIZED = '0'). TLS verification must be
enabled. If this was a workaround for a self-signed cert in dev, document it with an
environment variable guard: only allow it when NODE_ENV === 'development' and log a loud
warning.

ISSUE 4 — HIGH | src/payment.js:10 | CWE-79: Reflected XSS via innerHTML
Replace the innerHTML assignment with textContent if rendering plain text, or use a
sanitization library (e.g., DOMPurify) before assigning to innerHTML. Also add a note
explaining why DOM APIs are present in this file — if this is a Node.js module, DOM
manipulation must be removed entirely and moved to the appropriate frontend layer.

ISSUE 5 — HIGH | src/config.js:5 | CWE-215: Debug Mode Hardcoded On
Gate the debug flag on the NODE_ENV environment variable: debug should only be true when
process.env.NODE_ENV === 'development'. Use: const debug = process.env.NODE_ENV ===
'development';

ISSUE 6 — HIGH | src/config.js:6 | CWE-532: Trace Log Level Risks Sensitive Data Exposure
Change the log level to 'warn' for production. Use: const logLevel = process.env.LOG_LEVEL ||
(process.env.NODE_ENV === 'development' ? 'trace' : 'warn');

ISSUE 7 — MEDIUM | src/config.js:1 | No Environment Awareness
Refactor src/config.js to read all settings from environment variables using process.env,
with safe defaults. Use a library such as dotenv for local development. Remove all hardcoded
values for TLS settings, debug flags, and log levels.

ISSUE 8 — MEDIUM | src/payment.js:1 | Unused `crypto` Import
After fixing Issue 2, the crypto import will now be used. If for any reason it is still
unused after all fixes, remove it.

ISSUE 9 — LOW | src/payment.js:10 | Browser DOM API in Node.js Module
Confirm whether src/payment.js runs in Node.js or in a browser. If Node.js: remove all
document.* and innerHTML references and move any UI rendering to a frontend-only file. If
browser: ensure the file is not being used server-side.

ISSUE 10 — HIGH | package.json | Outdated Dependencies with CVEs
Run the following upgrades and validate the test suite (once tests exist) after each:
  - lodash: upgrade to ^4.17.21 (prototype pollution fix, CWE-1321)
  - axios: upgrade to ^1.6.0 or later (SSRF fix, CWE-918)
  - express: upgrade to ^4.21.0 or later (open-redirect fix, CWE-601)
  - jsonwebtoken: upgrade to ^9.0.0 (signature bypass fix, CWE-347)
After upgrading, run `npm audit` and confirm zero high/critical advisories remain.

---
After all changes, run `npm audit` and confirm no critical or high vulnerabilities remain.
Add `npm audit --audit-level=high` as a required step in the CI pipeline so these cannot
```

regress silently.

7. Appendix — Scope & Method

Project	payments-service (sample)
Generated	6/1/2026, 7:07:58 AM
Engine	AI (Claude · claude-sonnet-4-6)
Estimated cost	~$0.16 (6,235 in / 9,645 out tokens)
Files analyzed	4 (0 skipped)
Source size	1 KB
Languages	Python (1), JavaScript (2), JSON (1)
Total findings	13 — 3 critical, 7 high, 2 medium, 1 low, 0 info

Method: CodeGuard reviews uploaded source read-only; code is never executed. Findings are graded across 12 dimensions and mapped to CWE where applicable. AI assessments are evidence-led and cite file:line; cost is an estimate based on token usage. Benchmarks and model behaviour evolve over time.