

## 1. Cover Page

### juice-shop — Security Assessment

CLASSIFICATION: PUBLIC

This report contains findings from an automated security assessment of juice-shop. This copy is classified PUBLIC and may be shared and redistributed without restriction.

- **Project:** juice-shop
- **Prepared by:** Vollos Lens
- **Run timestamp:** 2026-06-12T13:39:23Z
- **Spec version:** v3.37
- **Classification:** PUBLIC

### 1a. Table of Contents

- 1. Cover Page
- 1a. Table of Contents
- 2. Document Control
- 3. Executive Summary
- 3c. Top 3 Risks — Detail
- 4. Scope & Limitations
- 5. Methodology
- 5a. Severity Legend
- 6. Assessment Timeline
- 7. Risk Overview
- 8. Key Findings
- 9. Priority Matrix
- 10. Attack Chains
- 11. Detailed Findings
- 12. Compliance Mapping
- 13. Action Plan (30/60/90-Day)
- 13a. AI Fix Prompts
- 14. Attestation & Re-test Statement
- 14a. Engagement Metadata & Sign-off
- 15. Appendix A — Tool Coverage & Step Scores
- 16. Appendix B — SBOM Summary
- 17. Appendix C — Glossary
- 18. Appendix D — Raw Artifacts Index
- 18a. Appendix E — Scope NOT Covered
- 18b. Appendix F — Scan Provenance

## 2. Document Control

Field	Value
Run Timestamp	2026-06-12T13:39:23Z
Spec Version	v3.37
Mode	standard
Classification	PUBLIC
Auditor	Automated Assessment by Vollos Lens
Reviewer	—

### Revision History

*No prior revisions.*

## Distribution

*No distribution list registered.*

## 3. Executive Summary

**Scope of this automated assessment.** Your application was scanned across our automated checks. A few items have specific scope notes below — none is a failed scan. The Security Score is computed only from the steps that produced a graded result.

**Coverage limit:** the live, running application was not fully tested in this assessment. See §4 Scope & Limitations for exactly what was and was not covered.

**Risk Level:** CRITICAL ●

**Security Score:** 11.8/100 ● Critical

### What This Means

The assessment identified critical security issues that could allow attackers to compromise the system or access sensitive data. Immediate remediation is recommended before production deployment.

Of the 10 high-severity findings, **9 are in production code**; the remaining 1 are located in test or fixture files (listed separately in the detailed findings, and typically not production risk).

### Top 3 Risks

#	Risk	Business impact
1	Server Takeover — routes/b2bOrder.ts:23	arbitrary code execution
2	Code-Injection Attack — routes/captcha.ts:22	arbitrary code execution from evaluated input
3	Stolen Credentials — lib/insecurity.ts:23	credential compromise: the embedded secret grants direct access

*These top risks were selected automatically by severity from the findings below. Full detail for each is in §11.*

### Recommended Actions

*Priority key. **P0**: do immediately · **P1**: do this week · **P2**: plan this sprint · **P3**: backlog.*

1. **P0 (Immediate):** Fix all P0 findings before the next deployment.
2. **P1 (This week):** Remediate P1 findings within the week.
3. **P2 (This sprint):** Plan P2 findings into the current sprint.
4. **P3 (Backlog):** Schedule P3 work into the next quarterly roadmap.

### Industry Benchmark (for budgeting context)

To help you size remediation effort against a neutral reference point: the IBM Cost of a Data Breach Report 2025 (<https://www.ibm.com/reports/data-breach>) puts the global average cost of a data breach at USD 4.44 million. We include this only as an industry benchmark to support prioritization and budgeting decisions — it is **not** a prediction or estimate of any cost associated with the specific findings in this report, and no breach is implied.

### 3c. Top 3 Risks — Detail

**Risk #1: Server Takeover — routes/b2bOrder.ts:23**

**Business impact:**

arbitrary code execution

#### Recommended action:

Remove the eval path entirely: parse `orderLinesData` as JSON against a strict field allowlist (cid, items, quantities) and reject anything else, never evaluate request data as code. Delete the `vm.runInContext/notevil` usage at `b2bOrder.ts:21-23`. If an expression must be computed, use a non-Turing-complete, schema-validated parser, not `vm` or `notevil`.

**Estimated effort:** a few hours

**Why this is #1:** Critical severity — an attacker exploiting this can directly compromise the application or its data. It sits in production code that ships to your users. So it belongs at the top of your remediation order.

#### Risk #2: Code-Injection Attack — `routes/captcha.ts:22`

##### Business impact:

arbitrary code execution from evaluated input

##### Recommended action:

Replace `eval` / `new Function(...)` / dynamic `require` on user-controlled data with a safe alternative: `JSON.parse` for data, a vetted expression parser for formulas, or an explicit dispatch map for command strings. Never pass request input to a code-execution sink.

**Estimated effort:** a few hours

**Why this is #2:** Critical severity — an attacker exploiting this can directly compromise the application or its data. It sits in production code that ships to your users. So it belongs at the top of your remediation order.

#### Risk #3: Stolen Credentials — `lib/insecurity.ts:23`

##### Business impact:

credential compromise: the embedded secret grants direct access

##### Recommended action:

Remove the hardcoded value from the source code and replace it with a runtime lookup (environment variable or a secrets manager such as AWS Secrets Manager / HashiCorp Vault). Do NOT add this source file to `.gitignore`. It is application code. If the credential was ever a real/live secret, rotate it at the provider AND purge the leaked value from git history (`git filter-repo --replace-text`), but keep the file in the repo.

**Estimated effort:** a few hours

**Why this is #3:** Critical severity — an attacker exploiting this can directly compromise the application or its data. An exposed credential grants direct access if it is live. It sits in production code that ships to your users. And the fix is fast (a small, low-risk change), so it is the highest-return item to clear first.


## 4. Scope & Limitations

### In Scope

- Package manager: `npm`
- Languages: JavaScript
- Frameworks: `express`
- Project type: General

### Out of Scope & Limitations

#### Live-application (dynamic) testing coverage

-  **Live-app (dynamic) testing:** NOT performed — we could test only your source code, not your live, running application. Our sandbox could not start a disposable copy of the app, and

no running URL was provided. What this means: problems that only appear when the app is actually running were not checked. To close this gap, give us a way to launch the app — either a Dockerfile/compose file in the repo, or a running URL — and re-run the scan.

In short: the live, running application was NOT actively tested in this assessment. The dynamic (live-app) scan did not run (see §5 and the coverage note above). This report reflects static, dependency, secret and configuration analysis only; pages behind login and JavaScript-rendered routes were not exercised.

Secret scanning (Step 4) covered the **working tree only**. Every reported secret resolves to a file currently on disk. Secrets that were committed and then removed from the working tree but still survive in git history or the reflog are OUT OF SCOPE of this assessment; a repository with that history needs a separate dedicated history sweep.

- **Awaiting one file from you — Step 2 (Dependency CVE Audit):** this check cross-references your dependencies against known CVEs, which requires a committed `package-lock.json`. Send us the lockfile and we'll re-run **this one step, free of charge**, and re-issue the report with it added. **Every other result stays exactly as delivered — only this dependency check is appended.** (*Meanwhile, we statically checked your dependency version-pinning — see §11.*)
- **Step 7 (Dynamic / Live-App Test) — not run in this assessment:** the live attack scan did not run against a running instance of your app (see the live-app coverage box above for why). We report Step 7 **un-graded rather than assign a falsely-clean score** — this is not a clean bill of health for the live app, only an un-tested one. A follow-up dynamic pass against a runnable copy of the app closes this gap.
- **Human-expert tier — Step 8 (Manual Testing) & Step 10 (Backup & IR):** these require interviewing your team and hands-on testing, so they sit in our **human-expert engagement tier**, outside this automated assessment by design — not a gap in this report.
- Manual review, threat modeling, and human pentest testing are outside the automated scope.
- This assessment is a point-in-time snapshot — posture may change.

## 5. Methodology

The Vollos Lens framework executes 10 ordered steps covering static analysis, dependency review, secret scanning, configuration audit, and adversarial AI review. Each step produces a structured JSON artifact that this report consolidates.

Step	Title	Status
Step 2	Dependency Audit	skipped
Step 3	Supply Chain	ok
Step 4	Working-Tree Secrets	ok
Step 5	Static Analysis	ok
Step 6	OWASP Top 10	ok
Step 7	Dynamic Analysis	static only (no live-app run)
Step 8	Manual Testing	N/A by design (human-expert tier — see §4)
Step 9	Infrastructure	ok
Step 10	Backup & IR	N/A by design (human-expert tier — see §4)
Step 11	Adversarial AI Review	ok

### How findings are prioritised

Each finding gets a priority from **P0 (fix now)** to **P3 (fix when convenient)**, based on two things: how dangerous it is, and how quick it is to fix.

- **Every Critical finding is P0** — a critical issue is the top priority no matter how long the fix takes.
- **A quick fix to a serious issue jumps the queue** — e.g. a 5-minute config change on a High-severity issue is also P0.
- **Minor issues that need a large rewrite sink to P3.**

Your findings, each with its priority, estimated effort and exact location, are listed in **§9 Priority Matrix**.

## 5a. Severity Legend

Severity	CVSS 3.1 score	Indicator	Fix timeline
Critical	9.0 – 10.0	dark red icon	within 7 days
High	7.0 – 8.9	orange icon	within 30 days
Medium	4.0 – 6.9	amber icon	within 90 days
Low	0.1 – 3.9	green icon	when convenient

CVSS = Common Vulnerability Scoring System v3.1, the industry standard published by FIRST (<https://www.first.org/cvss/v3.1/specification-document>). You can verify any score in this report using the official CVSS Calculator at <https://www.first.org/cvss/calculator/3.1>.

Severity buckets above match the National Vulnerability Database (NVD) qualitative severity rating scale.

### How to read the CVSS column in the detailed findings (§11).

- A plain score (e.g. 9.1) is computed from a CVSS 3.1 vector we derived for the finding’s vulnerability class.
- **<score> (estimated)** — e.g. 10.0 (estimated) — the score is estimated from the finding’s rule/CWE severity rather than a tool-supplied vector; the number is indicative, not authoritative.
- **<Severity> (no vector)** — e.g. High (no vector) — no CVSS 3.1 vector could be derived for this finding, so only the qualitative severity band (per the table above) is shown. It is not missing data; the score simply cannot be expressed as a single CVSS number.

## 6. Assessment Timeline

*Timing not recorded.*

## 7. Risk Overview

Security Score: 11.8 / 100 ● Critical

```
-----
Critical    7  #
High       10  ## (9 prod / 1 test)
Medium     45  ##### (42 prod / 3 test)
Low        205 ##### (180 prod / 25 test)
```

(prod / test) = production code vs test/fixture files; test/fixture findings are listed in full in Detailed

**78 production-actionable findings (the items to fix), out of 267 total.** The rest cluster into a few root fixes (124 are dependency-pinning advisories that collapse into a single “commit a lockfile and pin versions” action; 25 are in test or demo files (typically not a production risk); 40 are informational — third-party libraries, non-security random (display/captcha), or verified-safe ORM-parameterized queries (not production-actionable)). Every finding is still listed below with a remediation brief.

**Many findings are dependency hygiene, not code bugs.** A further 124 findings flag dependencies declared with a floating ^/~ version range and no committed lockfile — a single hygiene fix (commit a lockfile and pin versions), not 124 separate vulnerabilities. They are grouped into one remediation brief.

### Severity × Effort Heatmap

Severity ↓ / Effort →	Config	1-liner	Small	Moderate	Refactor
<b>Critical</b>	1	—	6	—	—
<b>High</b>	2	—	8	—	—
<b>Medium</b>	3	—	42	—	—
<b>Low</b>	37	124	44	—	—

## 8. Key Findings

### F01 — Server Takeover

**Severity:** Critical · **CVSS:** 10.0 · **CVSS:** 3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H · estimated · **Effort:** Small fix (1–2 hrs).

**Evidence:** AI analysis — verify before acting.

**Issue:** Traced user input to an eval sink the static scan missed (no b2bOrder.ts entry in static\_analysis\_result.json). /b2b/v2/orders (server.ts:647, behind the /b2b/v2 isAuthorized prefix at server.ts:423) reads `const orderLinesData = body.orderLinesData` (b2bOrder.ts:19) and passes it into `vm.runInContext('safeEval(orderLinesData)', sandbox, { timeout: 2000 })` (line 23). Both layers are bypassable security boundaries: Node's `vm` module is explicitly NOT a sandbox, and `notevil`'s `safeEval` (imported line 9) has known prototype-chain escapes.

**Impact:** arbitrary code execution.

**Remediation:** Remove the eval path entirely: parse `orderLinesData` as JSON against a strict field allowlist (cid, items, quantities) and reject anything else, never evaluate request data as code. Delete the `vm.runInContext/notevil` usage at b2bOrder.ts:21-23. If an expression must be computed, use a non-Turing-complete, schema-validated parser, not `vm` or `notevil`.

### F02 — Code-Injection Attack

**Severity:** Critical · **CVSS:** 10.0 · **CVSS:** 3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H · estimated · **Effort:** Small fix (1–2 hrs).

**Evidence:** Tool-confirmed.

**Issue:** The application was found calling the `eval` function OR `Function()` constructor OR `setTimeout()` OR `setInterval()` methods. If the

variables or strings or functions passed to these methods contains user-supplied input, an adversary could attempt to execute arbitrary

JavaScript

code. This could lead to a full system compromise in Node applications or Cross-site Scripting

(XSS) in web applications.

To remediate this issue, remove all calls to above methods and consider alternative methods for executing

the necessary business logic.

**Impact:** arbitrary code execution from evaluated input.

**Remediation:** Replace `eval` / `new Function(...)` / dynamic `require` on user-controlled data with a safe alternative: `JSON.parse` for data, a vetted expression parser for formulas, or an explicit dispatch map for command strings. Never pass request input to a code-execution sink.

### F03 — Code-Injection Attack

**Severity:** Critical · **CVSS:** 10.0 · **CVSS:** 3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H · estimated · **Effort:** Small fix (1–2 hrs).

**Evidence:** Tool-confirmed.

**Issue:** The application was found calling the `eval` function OR `Function()` constructor OR `setTimeout()` OR `setInterval()` methods. If the

variables or strings or functions passed to these methods contains user-supplied input, an adversary could attempt to execute arbitrary

JavaScript

code. This could lead to a full system compromise in Node applications or Cross-site Scripting (XSS) in web applications.

To remediate this issue, remove all calls to above methods and consider alternative methods for executing

the necessary business logic.

**Impact:** arbitrary code execution from evaluated input.

**Remediation:** Replace `eval` / `new Function(...)` / dynamic `require` on user-controlled data with a safe alternative: `JSON.parse` for data, a vetted expression parser for formulas, or an explicit dispatch map for command strings. Never pass request input to a code-execution sink.

#### F04 — Stolen Credentials

**Severity:** Critical · **CVSS:** Critical severity (estimate; no CVSS vector) · **Effort:** Config change (~5 min).

**Evidence:** Tool-confirmed.

**Issue:** Identified a Private Key, which may compromise cryptographic security and sensitive data encryption.

**Impact:** credential compromise: the embedded secret grants direct access.

**Remediation:** Remove the hardcoded value from the source code and replace it with a runtime lookup (environment variable or a secrets manager such as AWS Secrets Manager / HashiCorp Vault). Do NOT add this source file to `.gitignore`. It is application code. If the credential was ever a real/live secret, rotate it at the provider AND purge the leaked value from git history (`git filter-repo --replace-text`), but keep the file in the repo.

#### F05 — Internal Network Abuse

**Severity:** Critical · **CVSS:** 9.3 · **CVSS:** 3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:L/A:N · estimated · **Effort:** Small fix (1–2 hrs).

**Evidence:** Tool-confirmed.

**Issue:** This application allows user-controlled URLs to be passed directly to HTTP client libraries. This can result in Server-Side Request Forgery (SSRF). SSRF refers to an attack where the attacker can abuse functionality on the server to force it to make requests to other internal systems within your infrastructure that are not directly exposed to the internet. This allows the attacker to access internal resources they do not have direct access to.

**Impact:** internal-network pivot and cloud-metadata credential theft.

**Remediation:** Validate outbound URLs against an allowlist of permitted hosts. Block link-local and private IP ranges (169.254/16, 10/8, 172.16/12, 192.168/16). Disable HTTP redirects to untrusted destinations.

## 9. Priority Matrix

This is your fix-first running order. Each row is a plain summary; the **full detail for every ID, evidence, exact file/line, and the fix, is in §11 Detailed Findings**.

### P0 — Do immediately

ID	What it is	Severity	Effort
F01	Server Takeover	Critical	Small fix (1–2 hrs)
F02	Code-Injection Attack	Critical	Small fix (1–2 hrs)
F03	Code-Injection Attack	Critical	Small fix (1–2 hrs)
F04	Stolen Credentials	Critical	Config change (~5 min)
F05	Internal Network Abuse	Critical	Small fix (1–2 hrs)
F06	Database Attack	Critical	Small fix (1–2 hrs)

ID	What it is	Severity	Effort
F07	Database Attack	Critical	Small fix (1–2 hrs)
F11	Stolen Credentials	High	Config change (~5 min)
F12	Stolen Credentials	High	Config change (~5 min)

## P1 — Do this week

ID	What it is	Severity	Effort
F08	Code-Injection Attack	High	Small fix (1–2 hrs)
F09	Code-Injection Attack	High	Small fix (1–2 hrs)
F10	Code-Injection Attack	High	Small fix (1–2 hrs)
F13	Unauthorized Data Access	High	Small fix (1–2 hrs)
F14	Account Takeover Risk	High	Small fix (1–2 hrs)
F15	Unauthorized Data Access	High	Small fix (1–2 hrs)
F16	Open Redirect	High	Small fix (1–2 hrs)
F17	Business-Logic Abuse	High	Small fix (1–2 hrs)
F24	Container Hardening Gap	Medium	Config change (~5 min)
F25	Container Hardening Gap	Medium	Config change (~5 min)
F26	Cross-Origin Misconfig	Medium	Config change (~5 min)

## P2 — Plan sprint

79 items to plan — full detail in §11 (IDs: F18, F19, F20, F21, F22, F23, F27, F28, F29, F30, F31, F32, F33, F34, F35, F36, F37, F38, F39, F40, F41, F42, F43, F44, F45, F46, F47, F48, F49, F50, F51, F52, F53, F54, F55, F56, F57, F58, F59, F60, F61, F62, F63, F64, F65, F66, F67, F68, F69, F70, F71, F72, F73, F74, F75, F76, F77, F78, F79, F80, F81, F82, F83, F84, F85, F86, F87, F88, F89, F90, F91, F92, F93, F94, F95, F96, F97, F98, F99).

## P3 — Backlog

168 items to backlog — full detail in §11 (IDs: F224, F225, F226, F227, F228, F229, F230, F231, F232, F233, F234, F235, F236, F237, F238, F239, F240, F241, F242, F243, F244, F245, F246, F247, F248, F249, F250, F251, F252, F253, F254, F255, F256, F257, F258, F259, F260, F261, F262, F263, F264, F265, F266, F267, F100, F101, F102, F103, F104, F105, F106, F107, F108, F109, F110, F111, F112, F113, F114, F115, F116, F117, F118, F119, F120, F121, F122, F123, F124, F125, F126, F127, F128, F129, F130, F131, F132, F133, F134, F135, F136, F137, F138, F139, F140, F141, F142, F143, F144, F145, F146, F147, F148, F149, F150, F151, F152, F153, F154, F155, F156, F157, F158, F159, F160, F161, F162, F163, F164, F165, F166, F167, F168, F169, F170, F171, F172, F173, F174, F175, F176, F177, F178, F179, F180, F181, F182, F183, F184, F185, F186, F187, F188, F189, F190, F191, F192, F193, F194, F195, F196, F197, F198, F199, F200, F201, F202, F203, F204, F205, F206, F207, F208, F209, F210, F211, F212, F213, F214, F215, F216, F217, F218, F219, F220, F221, F222, F223).

## 10. Attack Chains

### Chain 1 — B2B order endpoint evaluates client-supplied data in a vm sandbox (RCE)

Severity: Critical · Confidence: high

### Chain 2 — Any authenticated user can read any basket by id (BOLA read)

Severity: High · Confidence: high

### Chain 3 — Basket item quantity check validates only the upper bound (negative quantity accepted)

Severity: Medium · Confidence: high



#### Chain 4 — Password change does not require the current password

Severity: High · Confidence: high

#### Chain 5 — Coupon can be applied/overwritten on any basket id (BOLA write)

Severity: Medium · Confidence: high

#### Chain 6 — Checkout/place-order works on any basket id and destroys its items (BOLA write)

Severity: High · Confidence: high

#### Chain 7 — Wallet balance is credited by a client-supplied amount with no verified payment

Severity: High · Confidence: high

## 11. Detailed Findings

### Step 2 — Dependency Audit

No findings for Step 2 — skipped.

### Step 3 — Supply Chain

ID	Severity	CVSS	CWE	File:Line	Rule
F27	Medium	Medium (no vector)	CWE-1104	.github/workflows/ci.yml:202	SC-UNPINNED-ACTION-GHA
F28	Medium	Medium (no vector)	CWE-1104	.github/workflows/codeql-analysis.yml:23	SC-UNPINNED-ACTION-GHA
F29	Medium	Medium (no vector)	CWE-1104	.github/workflows/codeql-analysis.yml:34	SC-UNPINNED-ACTION-GHA
F30	Medium	Medium (no vector)	CWE-1104	.github/workflows/codeql-analysis.yml:36	SC-UNPINNED-ACTION-GHA
F36	Medium	Medium (no vector)	CWE-494	package.json	SC-MISSING-LOCKFILE-NPM

+ 124 lower-priority dependency advisories collapsed for brevity — full detail for all 124 is in the machine-readable export (*security\_report.json*) delivered with this report.

ID	Package	Advisory	CVSS
F100	@ai-sdk/openai-compatible	SC-UNPINNED-RANGE-NPM	Low (no vector)
F101	@cyclonedx/cyclonedx-npm	SC-UNPINNED-RANGE-NPM	Low (no vector)
F102	@eslint/js	SC-UNPINNED-RANGE-NPM	Low (no vector)

ID	Package	Advisory	CVSS
F103	@fontsource/roboto	SC-UNPINNED-RANGE-NPM	Low (no vector)
F104	@istanbuljs/nyc-config-typescript	SC-UNPINNED-RANGE-NPM	Low (no vector)
F105	@types/chai	SC-UNPINNED-RANGE-NPM	Low (no vector)
F106	@types/clarinet	SC-UNPINNED-RANGE-NPM	Low (no vector)
F107	@types/compression	SC-UNPINNED-RANGE-NPM	Low (no vector)
...and 116 more			

#### Step 4 — Working-Tree Secrets

ID	Severity	CVSS	CWE	File:Line	Rule
F04	Critical	Critical (no vector)	CWE-798	lib/insecurity.ts:23	private-key
F12	High	High (no vector)	CWE-798	routes/login.ts:65	generic-api-key
F65	Low	Low (no vector)	CWE-798	frontend/src/app/app.guard.spec.ts:46	jwt
F66	Low	Low (no vector)	CWE-798	frontend/src/app/last-login-ip/last-login-ip.component.spec.ts:72	jwt
F80	Low	Low (no vector)	CWE-798	test/cypress/e2e/forgedJwt.spec.ts:7	jwt
F81	Low	Low (no vector)	CWE-798	test/cypress/e2e/forgedJwt.spec.ts:22	jwt
F82	Low	Low (no vector)	CWE-798	test/server/currentUserSpec.ts:35	jwt
F83	Low	Low (no vector)	CWE-798	test/server/currentUserSpec.ts:36	jwt
F84	Low	Low (no vector)	CWE-798	test/server/verifySpec.ts:265	jwt

+ 15 **generic-api-key** findings (same rule, low/medium) collapsed for brevity — full detail for all 15 is in the machine-readable export (*security\_report.json*) delivered with this report.

ID	Severity	File:Line
F63	Low	data/static/users.yml:88
F64	Low	data/static/users.yml:151
F67	Low	frontend/src/app/oauth/oauth.component.spec.ts:91
F68	Low	frontend/src/app/oauth/oauth.component.spec.ts:98
F69	Low	test/api/2fa.test.ts:42
F70	Low	test/api/2fa.test.ts:66
F71	Low	test/api/2fa.test.ts:104
F72	Low	test/api/2fa.test.ts:144
...and 7 more		

## Step 5 — Static Analysis

ID	Severity	CVSS	CWE	File:Line	Rule
F02	Critical	10.0 (estimated)	CWE-95	routes/ captcha.ts:22	eval-with-expression
F03	Critical	10.0 (estimated)	CWE-95	routes/ userProfile. ts:61	eval-with-expression
F05	Critical	9.3 (estimated)	CWE-918	routes/ profileImageUrlUpload. ts:24	node-ssrf
F06	Critical	9.1 (estimated)	CWE-943	routes/ showProductReviews. ts:31	node-nosqli- js-injection
F07	Critical	9.1 (estimated)	CWE-943	routes/ trackOrder. ts:15	node-nosqli- js-injection
F08	High	8.1 (estimated)	CWE-502	routes/ vulnCodeFixes. ts:81	yaml- deserialize
F09	High	8.1 (estimated)	CWE-502	rsn/rsnUtil. ts:135	yaml- deserialize
F10	High	8.1 (estimated)	CWE-502	server.ts:139	yaml- deserialize
F11	High	High (no vector)	CWE-798	lib/ insecurity. ts:56	hardcoded- jwt-secret
F16	High	High (no vector)	CWE-601	routes/ redirect.ts: 19	express-open- redirect
F18	Medium	6.1 (estimated)	CWE-79	routes/chat. ts:216	express-xss
F19	Medium	6.1 (estimated)	CWE-79	routes/chat. ts:226	express-xss
F20	Medium	6.1 (estimated)	CWE-79	routes/chat. ts:239	express-xss
F21	Medium	6.1 (estimated)	CWE-79	routes/chat. ts:253	express-xss
F22	Medium	6.1 (estimated)	CWE-79	routes/ dataExport. ts:108	express-xss

ID	Severity	CVSS	CWE	File:Line	Rule
F23	Medium	6.1 (estimated)	CWE-79	routes/ userProfile. ts:98	express-xss
F26	Medium	Medium (no vector)	CWE-346	server.ts:182	express-cors
F31	Medium	Medium (no vector)	CWE-208	frontend/src/ app/change- password/ change- password. component.ts: 150	possible- timing- attacks
F32	Medium	Medium (no vector)	CWE-185	lib/ codingChallenges ts:76	non-literal- regex
F33	Medium	Medium (no vector)	CWE-185	lib/ codingChallenges ts:78	non-literal- regex
F34	Medium	Medium (no vector)	CWE-338	lib/ insecurity. ts:55	node- insecure- random- generator
F37	Medium	Medium (no vector)	CWE-606	routes/ basket.ts:26	layer7- object-dos
F39	Medium	Medium (no vector)	CWE-208	routes/ changePassword. ts:28	possible- timing- attacks
F41	Medium	Medium (no vector)	CWE-606	routes/ currentUser. ts:23	layer7- object-dos
F42	Medium	Medium (no vector)	CWE-23	routes/ dataErasure. ts:50	express-lfr
F43	Medium	Medium (no vector)	CWE-606	routes/ dataExport. ts:72	layer7- object-dos
F47	Medium	Medium (no vector)	CWE-606	routes/ likeProductReviews. ts:47	layer7- object-dos
F49	Medium	Medium (no vector)	CWE-606	routes/ payment.ts:22	layer7- object-dos

+ 22 *non-literal-fs-filename* findings (same rule, low/medium) collapsed for brevity — full detail for all 22 is in the machine-readable export (*security\_report.json*) delivered with this report.

ID	Severity	File:Line
F35	Medium	lib/utils.ts:120
F44	Medium	routes/fileUpload.ts:33
F45	Medium	routes/fileUpload.ts:38
F46	Medium	routes/fileUpload.ts:45
F48	Medium	routes/order.ts:45
F50	Medium	routes/profileImageFileUpload.ts:43
F51	Medium	routes/profileImageUrlUpload.ts:29

ID	Severity	File:Line
F52	Medium	routes/videoHandler.ts:21
...and 14 more		

+ 20 informational finding(s) — ORM-parameterized query (verified safe — not injectable) — not production-actionable; listed for completeness. Full detail is in the machine-readable export (security\_report.json).

ID	Severity	File:Line	Rule
F229	Low	data/static/codefixes/chatbotPromptInjectionChallenge_2_correct.ts:8	node-nosqli-injection
F244	Low	routes/address.ts:18	node-nosqli-injection
F245	Low	routes/basket.ts:19	node-nosqli-injection
F246	Low	routes/basketItems.ts:68	node-nosqli-injection
F252	Low	routes/captcha.ts:37	node-nosqli-injection
F253	Low	routes/dataErasure.ts:31	node-nosqli-injection
F254	Low	routes/dataErasure.ts:34	node-nosqli-injection
F255	Low	routes/delivery.ts:34	node-nosqli-injection
...and 12 more			

+ 12 informational finding(s) — non-security random (display/captcha) — not production-actionable; listed for completeness. Full detail is in the machine-readable export (security\_report.json).

ID	Severity	File:Line	Rule
F225	Low	data/datacreator.ts:304	node-insecure-random-generator
F226	Low	data/datacreator.ts:322	node-insecure-random-generator
F227	Low	data/datacreator.ts:380	node-insecure-random-generator
F228	Low	data/datacreator.ts:754	node-insecure-random-generator
F230	Low	frontend/src/app/Services/conversation-storage.service.ts:17	node-insecure-random-generator
F231	Low	frontend/src/app/chatbot/chat-welcome-screen/chat-welcome-screen.component.ts:71	node-insecure-random-generator
F232	Low	frontend/src/app/coding-challenge-page/components/coding-challenge-fix-it/coding-challenge-fix-it.component.ts:119	node-insecure-random-generator
F247	Low	routes/captcha.ts:14	node-insecure-random-generator
...and 4 more			

+ 8 informational finding(s) — third-party library (not your code) — not production-actionable; listed for completeness. Full detail is in the machine-readable export (security\_report.json).

ID	Severity	File:Line	Rule
F233	Low	frontend/src/assets/private/three.js:6359	node-insecure-random-generator
F234	Low	frontend/src/assets/private/three.js:6426	node-insecure-random-generator
F235	Low	frontend/src/assets/private/three.js:6434	node-insecure-random-generator
F236	Low	frontend/src/assets/private/three.js:6442	node-insecure-random-generator
F237	Low	frontend/src/assets/private/three.js:6450	node-insecure-random-generator
F238	Low	frontend/src/assets/private/three.js:15504	node-insecure-random-generator
F239	Low	frontend/src/assets/private/three.js:15534	node-insecure-random-generator
F240	Low	frontend/src/assets/private/three.js:15567	node-insecure-random-generator

## Step 6 — OWASP Top 10

Note: these 90 finding(s) are also detected by another scan step (most overlap Step 5 Static Analysis) and are counted once there to avoid double-counting. They are listed here under their OWASP Top 10 category for reference; this step's score in §15 reflects that de-duplication, so a low score here is not a clean result.

ID	Severity	CVSS	CWE	File:Line	Rule
—	Medium	Medium (no vector)	CWE-23	routes/dataErasure.ts:50	express-lfr
—	Medium	Medium (no vector)	CWE-208	frontend/src/app/change-password/change-password.component.ts:150	node-timing-attack
—	Medium	Medium (no vector)	CWE-338	lib/insecurity.ts:55	node-insecure-random-generator
—	High	High (no vector)	CWE-798	lib/insecurity.ts:56	hardcoded-jwt-secret
—	Medium	Medium (no vector)	CWE-208	routes/changePassword.ts:28	node-timing-attack
—	Medium	Medium (no vector)	CWE-185	lib/codingChallenges.ts:76	non-literal-codingChallengesregex
—	Medium	Medium (no vector)	CWE-185	lib/codingChallenges.ts:78	non-literal-codingChallengesregex

ID	Severity	CVSS	CWE	File:Line	Rule
—	Critical	10.0 (estimated)	CWE-95	routes/ captcha.ts:22	eval-with-expression
—	Medium	6.1 (estimated)	CWE-79	routes/chat. ts:216	express-xss
—	Medium	6.1 (estimated)	CWE-79	routes/chat. ts:226	express-xss
—	Medium	6.1 (estimated)	CWE-79	routes/chat. ts:239	express-xss
—	Medium	6.1 (estimated)	CWE-79	routes/chat. ts:253	express-xss
—	Medium	6.1 (estimated)	CWE-79	routes/ dataExport. ts:108	express-xss
—	High	High (no vector)	CWE-601	routes/ redirect.ts: 19	express-open-redirect
—	Critical	9.1 (estimated)	CWE-943	routes/ showProductReviews. ts:31	node-nosqli-js-injection
—	Critical	9.1 (estimated)	CWE-943	routes/ trackOrder. ts:15	node-nosqli-js-injection
—	Critical	10.0 (estimated)	CWE-95	routes/ userProfile. ts:61	eval-with-expression
—	Medium	6.1 (estimated)	CWE-79	routes/ userProfile. ts:98	express-xss
—	Medium	Medium (no vector)	CWE-606	routes/ basket.ts:26	layer7-object-dos
—	Medium	Medium (no vector)	CWE-606	routes/ currentUser. ts:23	layer7-object-dos
—	Medium	Medium (no vector)	CWE-606	routes/ dataExport. ts:72	layer7-object-dos
—	Medium	Medium (no vector)	CWE-606	routes/ likeProductReviews. ts:47	layer7-object-dos
—	Medium	Medium (no vector)	CWE-606	routes/ payment.ts:22	layer7-object-dos
—	Medium	Medium (no vector)	CWE-346	server.ts:182	express-cors
—	High	8.1 (estimated)	CWE-502	routes/ vulnCodeFixes. ts:81	yaml-deserialize
—	High	8.1 (estimated)	CWE-502	rsn/rsnUtil. ts:135	yaml-deserialize
—	High	8.1 (estimated)	CWE-502	server.ts:139	yaml-deserialize
—	Critical	9.3 (estimated)	CWE-918	routes/ profileImageUrlUpload. ts:24	node-ssrf

+ 22 *non-literal-fs-filename* findings (same rule, low/medium) collapsed for brevity — full detail for all 22 is in the machine-readable export (*security\_report.json*) delivered with this report.

ID	Severity	File:Line
—	Low	Gruntfile.js:75
—	Low	lib/codingChallenges.ts:22
—	Low	lib/codingChallenges.ts:23
—	Low	lib/codingChallenges.ts:29
—	Medium	lib/utils.ts:120
—	Medium	routes/fileUpload.ts:33
—	Medium	routes/fileUpload.ts:38
—	Medium	routes/fileUpload.ts:45
...and 14 more		

+ 20 informational finding(s) — ORM-parameterized query (verified safe — not injectable) — not production-actionable; listed for completeness. Full detail is in the machine-readable export (*security\_report.json*).

ID	Severity	File:Line	Rule
—	Low	data/static/codefixes/chatbotPromptInjectionChallenge_2_correct.ts:8	node-nosqli-injection
—	Low	routes/address.ts:18	node-nosqli-injection
—	Low	routes/basket.ts:19	node-nosqli-injection
—	Low	routes/basketItems.ts:68	node-nosqli-injection
—	Low	routes/captcha.ts:37	node-nosqli-injection
—	Low	routes/dataErasure.ts:31	node-nosqli-injection
—	Low	routes/dataErasure.ts:34	node-nosqli-injection
—	Low	routes/delivery.ts:34	node-nosqli-injection
...and 12 more			

+ 12 informational finding(s) — non-security random (display/captcha) — not production-actionable; listed for completeness. Full detail is in the machine-readable export (*security\_report.json*).

ID	Severity	File:Line	Rule
—	Low	data/datacreator.ts:304	node-insecure-random-generator
—	Low	data/datacreator.ts:322	node-insecure-random-generator
—	Low	data/datacreator.ts:380	node-insecure-random-generator
—	Low	data/datacreator.ts:754	node-insecure-random-generator
—	Low	frontend/src/app/Services/conversation-storage.service.ts:17	node-insecure-random-generator
—	Low	frontend/src/app/chatbot/chat-welcome-screen/chat-welcome-screen.component.ts:71	node-insecure-random-generator



ID	Severity	File:Line	Rule
—	Low	frontend/src/app/coding-challenge-page/components/coding-challenge-fix-it/coding-challenge-fix-it.component.ts:119	node-insecure-random-generator
—	Low	routes/captcha.ts:14	node-insecure-random-generator
...and 4 more			

+ 8 informational finding(s) — third-party library (not your code) — not production-actionable; listed for completeness. Full detail is in the machine-readable export (security\_report.json).

ID	Severity	File:Line	Rule
—	Low	frontend/src/assets/private/three.js:6359	node-insecure-random-generator
—	Low	frontend/src/assets/private/three.js:6426	node-insecure-random-generator
—	Low	frontend/src/assets/private/three.js:6434	node-insecure-random-generator
—	Low	frontend/src/assets/private/three.js:6442	node-insecure-random-generator
—	Low	frontend/src/assets/private/three.js:6450	node-insecure-random-generator
—	Low	frontend/src/assets/private/three.js:15504	node-insecure-random-generator
—	Low	frontend/src/assets/private/three.js:15534	node-insecure-random-generator
—	Low	frontend/src/assets/private/three.js:15567	node-insecure-random-generator

## Step 7 — Dynamic Analysis

### Live-app (dynamic) test summary

- ⚠ **Live-app (dynamic) testing:** NOT performed — we could test only your source code, not your live, running application. Our sandbox could not start a disposable copy of the app, and no running URL was provided. What this means: problems that only appear when the app is actually running were not checked. To close this gap, give us a way to launch the app — either a Dockerfile/compose file in the repo, or a running URL — and re-run the scan.

No findings for Step 7 — static only (no live-app run).

## Step 8 — Manual Testing

No findings for Step 8 — N/A by design (human-expert tier — see §4).

## Step 9 — Infrastructure

ID	Severity	CVSS	CWE	File:Line	Rule
F24	Medium	Medium (no vector)	—	Dockerfile:22	DS-0001

ID	Severity	CVSS	CWE	File:Line	Rule
F25	Medium	Medium (no vector)	—	Dockerfile:22	DL3006
F85	Low	Low (no vector)	—	test/smoke/Dockerfile	DS-0002
F86	Low	Low (no vector)	—	test/smoke/Dockerfile:3	DS-0025
F87	Low	Low (no vector)	—	test/smoke/Dockerfile:1	DS-0001
F88	Low	Low (no vector)	—	Dockerfile	DS-0026
F89	Low	Low (no vector)	—	Dockerfile:5	DL3059
F90	Low	Low (no vector)	—	Dockerfile:6	DL3059
F91	Low	Low (no vector)	—	Dockerfile:7	DL3059
F92	Low	Low (no vector)	—	Dockerfile:8	DL3059
F93	Low	Low (no vector)	—	Dockerfile:9	DL3059
F94	Low	Low (no vector)	—	Dockerfile:10	DL3059
F95	Low	Low (no vector)	—	Dockerfile:11	DL3059
F96	Low	Low (no vector)	—	Dockerfile:12	DL3059
F97	Low	Low (no vector)	—	Dockerfile:13	DL3059
F98	Low	Low (no vector)	—	Dockerfile:20	DL3059
F99	Low	Low (no vector)	—	test/smoke/Dockerfile	DS-0026

## Step 10 — Backup & IR

No findings for Step 10 — N/A by design (human-expert tier — see §4).

## Step 11 — Adversarial AI Review

ID	Severity	CVSS	CWE	File:Line	Rule
F01	Critical	10.0 (estimated)	CWE-94	routes/b2bOrder.ts:23	RCE/b2b-order-vm-eval-user-input
F13	High	High (no vector)	CWE-639	routes/basket.ts:19	BOLA/basket-read-any-id
F14	High	High (no vector)	CWE-620	routes/changePassword.ts:39	AUTHZ/change-password-without-current
F15	High	High (no vector)	CWE-639	routes/order.ts:35	BOLA/basket-checkout-any-id
F17	High	High (no vector)	CWE-840	routes/wallet.ts:27	LOGIC/wallet-topup-without-charge
F38	Medium	Medium (no vector)	CWE-840	routes/basketItems.ts:92	LOGIC/basket-quantity-no-lower-bound
F40	Medium	Medium (no vector)	CWE-639	routes/coupon.ts:18	BOLA/coupon-apply-any-id

## Critical & High Findings: In Plain Language

Every Critical and High finding is explained in plain language below: what it is, why it matters to your business, where we found it, and how to fix it. Start with the Top 3 Risks (see the Top 3 Risks section earlier in this report); the full list follows here, worst-first. Each finding carries its fix-by target. Medium and Low findings are listed in the index tables above; their full description and remediation ship in the machine-readable export (security\_report.json) delivered with this report.

### F01: Server Takeover (Critical) Severity: 10.0 (Critical) · Fix within 7 days

**What this means for your business:** Code at `routes/b2bOrder.ts:23` turns input it received into code that then runs. An attacker who controls that input can run their own code on your server.

**Evidence:** `routes/b2bOrder.ts:23`, AI analysis — verify before acting

*Technical detail: Traced user input to an eval sink the static scan missed (no `b2bOrder.ts` entry in `static_analysis_result.json`). `/b2b/v2/orders` (`server.ts:647`, behind the `/b2b/v2` `isAuthorized` prefix at `server.ts:423`) reads `const orderLinesData = body.orderLinesData` (`b2bOrder.ts:19`) and passes it into `vm.runInContext('safeEval(orderLinesData)', sandbox, { timeout: 2000 })` (line 23). Both layers are bypassable security boundaries: Node's `vm` module is explicitly NOT a sandbox, and `notevil`'s `safeEval` (imported line 9) has known prototype-chain escapes.*

**Recommended fix:** Remove the eval path entirely: parse `orderLinesData` as JSON against a strict field allowlist (cid, items, quantities) and reject anything else, never evaluate request data as code. Delete the `vm.runInContext/notevil` usage at `b2bOrder.ts:21-23`. If an expression must be computed, use a non-Turing-complete, schema-validated parser, not `vm` or `notevil`.

### F02: Code-Injection Attack (Critical) Severity: 10.0 (Critical) · Fix within 7 days

**What this means for your business:** Code at `routes/captcha.ts:22` evaluates input it received as live code. An attacker who controls that input can run arbitrary code inside your application.

**Evidence:** `routes/captcha.ts:22`, Tool-confirmed

Technical detail: The application was found calling the `eval` function OR `Function()` constructor OR `setTimeout()` OR `setInterval()` methods. If the

variables or strings or functions passed to these methods contains user-supplied input, an adversary could attempt to execute arbitrary

JavaScript

code. This could lead to a full system compromise in Node applications or Cross-site Scripting (XSS) in web applications.

To remediate this issue, remove all calls to above methods and consider alternative methods for executing

the necessary business logic.

**Recommended fix:** Replace `eval` / `new Function(...)` / dynamic `require` on user-controlled data with a safe alternative: `JSON.parse` for data, a vetted expression parser for formulas, or an explicit dispatch map for command strings. Never pass request input to a code-execution sink.

### F03: Code-Injection Attack (Critical) Severity: 10.0 (Critical) · Fix within 7 days

**What this means for your business:** Code at `routes/userProfile.ts:61` evaluates input it received as live code. An attacker who controls that input can run arbitrary code inside your application.

**Evidence:** `routes/userProfile.ts:61`, Tool-confirmed

Technical detail: The application was found calling the `eval` function OR `Function()` constructor OR `setTimeout()` OR `setInterval()` methods. If the

variables or strings or functions passed to these methods contains user-supplied input, an adversary could attempt to execute arbitrary

JavaScript

code. This could lead to a full system compromise in Node applications or Cross-site Scripting (XSS) in web applications.

To remediate this issue, remove all calls to above methods and consider alternative methods for executing

the necessary business logic.

**Recommended fix:** Replace `eval` / `new Function(...)` / dynamic `require` on user-controlled data with a safe alternative: `JSON.parse` for data, a vetted expression parser for formulas, or an explicit dispatch map for command strings. Never pass request input to a code-execution sink.

#### F04: Stolen Credentials (Critical) Severity: 9.5 (Critical) · Fix within 7 days

**What this means for your business:** A secret (such as an API key, password, or token) is written directly into your source code at `lib/insecurity.ts:23`. Anyone who can read this code, such as a contractor, a leaked repository, or a former employee, gets a working key to the service it unlocks and can use it as if they were you.

**Evidence:** `lib/insecurity.ts:23`, Tool-confirmed

*Technical detail: Identified a Private Key, which may compromise cryptographic security and sensitive data encryption.*

**Recommended fix:** Remove the hardcoded value from the source code and replace it with a runtime lookup (environment variable or a secrets manager such as AWS Secrets Manager / HashiCorp Vault). Do NOT add this source file to `.gitignore`. It is application code. If the credential was ever a real/live secret, rotate it at the provider AND purge the leaked value from git history (`git filter-repo --replace-text`), but keep the file in the repo.

#### F05: Internal Network Abuse (Critical) Severity: 9.3 (Critical) · Fix within 7 days

**What this means for your business:** Code at `routes/profileImageUrlUpload.ts:24` fetches a URL supplied in the request. An attacker can point it at your internal network or cloud metadata service and reach systems that were never meant to be public.

**Evidence:** `routes/profileImageUrlUpload.ts:24`, Tool-confirmed

*Technical detail: This application allows user-controlled URLs to be passed directly to HTTP client libraries. This can result in Server-Side Request Forgery (SSRF). SSRF refers to an attack where the attacker can abuse functionality on the server to force it to make requests to other internal systems within your infrastructure that are not directly exposed to the internet. This allows the attacker to access internal resources they do not have direct access to.*

**Recommended fix:** Validate outbound URLs against an allowlist of permitted hosts. Block link-local and private IP ranges (169.254/16, 10/8, 172.16/12, 192.168/16). Disable HTTP redirects to untrusted destinations.

#### F06: Database Attack (Critical) Severity: 9.1 (Critical) · Fix within 7 days

**What this means for your business:** A database query at `routes/showProductReviews.ts:31` is built by gluing user input straight into the query text. An attacker can type database commands into an ordinary input field and read, change, or delete anything in your database, including other customers' records and login credentials.

**Evidence:** `routes/showProductReviews.ts:31`, Tool-confirmed

*Technical detail: Untrusted user input in MongoDB \$where operator can result in NoSQL JavaScript Injection.*

**Recommended fix:** Do not build a NoSQL query by merging untrusted request data. Reject query-operator keys in user input (`$where`, `$ne`, `$gt`, `$regex`, `$function`, ...), cast each expected value to its scalar type, and validate the request body against a strict schema before it reaches the driver. Use the driver's typed query API rather than passing a raw user-supplied object as a filter.

### F07: Database Attack (Critical) Severity: 9.1 (Critical) · Fix within 7 days

**What this means for your business:** A database query at `routes/trackOrder.ts:15` is built by gluing user input straight into the query text. An attacker can type database commands into an ordinary input field and read, change, or delete anything in your database, including other customers' records and login credentials.

**Evidence:** `routes/trackOrder.ts:15`, Tool-confirmed

*Technical detail: Untrusted user input in MongoDB \$where operator can result in NoSQL JavaScript Injection.*

**Recommended fix:** Do not build a NoSQL query by merging untrusted request data. Reject query-operator keys in user input (`$where`, `$ne`, `$gt`, `$regex`, `$function`, ...), cast each expected value to its scalar type, and validate the request body against a strict schema before it reaches the driver. Use the driver's typed query API rather than passing a raw user-supplied object as a filter.

### F08: Code-Injection Attack (High, in test/fixture file) Severity: 8.1 (High) · Fix within 30 days

**What this means for your business:** Code at `routes/vulnCodeFixes.ts:81` turns input it received into live code that then runs. An attacker who controls that input can run their own code on your server, which can lead to a full takeover.

*Found in a test/fixture file. Confirm this value isn't reused in production code or config. A secret or token that only exists in test fixtures is usually lower real-world risk than the same issue in production. Its severity above is unchanged, so verify before dismissing.*

**Evidence:** `routes/vulnCodeFixes.ts:81`, Tool-confirmed

*Technical detail: User controlled data in 'yaml.load()' function can result in Remote Code Injection.*

**Recommended fix:** Avoid deserialising untrusted input. If unavoidable, use schema-validating libraries (`zod`, `pydantic`) and an explicit allowlist of types. Never `Object.assign` user-controlled keys onto sensitive objects.

### F09: Code-Injection Attack (High) Severity: 8.1 (High) · Fix within 30 days

**What this means for your business:** Code at `rsn/rsnUtil.ts:135` turns input it received into live code that then runs. An attacker who controls that input can run their own code on your server, which can lead to a full takeover.

**Evidence:** `rsn/rsnUtil.ts:135`, Tool-confirmed

*Technical detail: User controlled data in 'yaml.load()' function can result in Remote Code Injection.*

**Recommended fix:** Avoid deserialising untrusted input. If unavoidable, use schema-validating libraries (`zod`, `pydantic`) and an explicit allowlist of types. Never `Object.assign` user-controlled keys onto sensitive objects.

### F10: Code-Injection Attack (High) Severity: 8.1 (High) · Fix within 30 days

**What this means for your business:** Code at `server.ts:139` turns input it received into live code that then runs. An attacker who controls that input can run their own code on your server, which can lead to a full takeover.

**Evidence:** `server.ts:139`, Tool-confirmed

*Technical detail: User controlled data in 'yaml.load()' function can result in Remote Code Injection.*

**Recommended fix:** Avoid deserialising untrusted input. If unavoidable, use schema-validating libraries (`zod`, `pydantic`) and an explicit allowlist of types. Never `Object.assign` user-controlled keys onto sensitive objects.

### F11: Stolen Credentials (High) Severity: 7.5 (High) · Fix within 30 days

**What this means for your business:** A secret (such as an API key, password, or token) is written directly into your source code at `lib/insecurity.ts:56`. Anyone who can read this code, such as a contractor, a leaked repository, or a former employee, gets a working key to the service it unlocks and can use it as if they were you.

**Evidence:** `lib/insecurity.ts:56`, Tool-confirmed

\_\_Technical detail: Hardcoded JWT secret or private key was found. Hardcoding secrets like JWT signing keys poses a significant security risk. If the source code ends up in a public repository or is compromised, the secret is exposed. Attackers could then use the secret to generate forged tokens and access the system. Store it properly in an environment variable.

Here are some recommended safe ways to access JWT secrets: - Use environment variables to store the secret and access it in code instead of hardcoding. This keeps it out of source control.\_\_

**Recommended fix:** Remove the hardcoded value from the source code and replace it with a runtime lookup (environment variable or a secrets manager such as AWS Secrets Manager / HashiCorp Vault). Do NOT add this source file to `.gitignore`. It is application code. If the credential was ever a real/live secret, rotate it at the provider AND purge the leaked value from git history (`git filter-repo --replace-text`), but keep the file in the repo.

## F12: Stolen Credentials (High) Severity: 7.5 (High) · Fix within 30 days

**What this means for your business:** A secret (such as an API key, password, or token) is written directly into your source code at `routes/login.ts:65`. Anyone who can read this code, such as a contractor, a leaked repository, or a former employee, gets a working key to the service it unlocks and can use it as if they were you.

**Evidence:** `routes/login.ts:65`, Tool-confirmed

*Technical detail: Detected a Generic API Key, potentially exposing access to various services and sensitive operations.*

**Recommended fix:** Remove the hardcoded value from the source code and replace it with a runtime lookup (environment variable or a secrets manager such as AWS Secrets Manager / HashiCorp Vault). Do NOT add this source file to `.gitignore`. It is application code. If the credential was ever a real/live secret, rotate it at the provider AND purge the leaked value from git history (`git filter-repo --replace-text`), but keep the file in the repo.

## F13: Unauthorized Data Access (High) Severity: 7.5 (High) · Fix within 30 days

**What this means for your business:** An endpoint at `routes/basket.ts:19` decides which record to return or change from an id in the request, but does not check that the record actually belongs to the user (or tenant) making the call. A logged-in user can change that id to someone else's value and read or modify another customer's data, crossing the wall between tenants that is supposed to keep each account's data private.

**Evidence:** `routes/basket.ts:19`, AI analysis — verify before acting

*Technical detail: `/rest/basket/:id` is mounted with `security.isAuthenticated()` (`server.ts:398`), which is `expressJwt({«REDACTED: Generic Secret (unquoted) (20 chars)» (lib/insecurity.ts:54)`, it only validates that the JWT is well-formed and signed, never that the basket belongs to the caller. `retrieveBasket` then loads `BasketModel.findOne({ where: { id } })` straight from `req.params.id(basket.ts:18-19)` with no owner/tenant filter. The session user is touched only inside `thechallengeUtils.solveIf(...)` scoring call at line 22, that is telemetry, NOT an authorization guard.*

**Recommended fix:** In `retrieveBasket`, derive the caller's basket id from the session (`security.authenticatedUsers.from(req).bid`) and 403 when it differs from `req.params.id`; or scope the query `BasketModel.findOne({ where: { id, UserId: <session user id> } })`. Apply the same ownership check across the sibling `:id` basket routes below.

## F14: Account Takeover Risk (High) Severity: 7.5 (High) · Fix within 30 days

**What this means for your business:** Code at `routes/changePassword.ts:39` lets someone set a new account password without first proving they know the current one. Anyone who already holds a logged-in session, or who tricks the user's browser into sending the request, can change the password and lock the real owner out, taking over the account.

**Evidence:** `routes/changePassword.ts:39`, AI analysis — verify before acting

*Technical detail: The handler authenticates the caller via the session token (`lib/insecurity.ts authenticatedUsers`, `changePassword.ts:33-37` rejects an unknown token), but the current-password check at line 39 is written *if**



*(currentPassword && security.hash(currentPassword) !== loggedInUser.data.password), it ONLY runs when currentPassword is truthy. Omit the current query parameter entirely and the check is skipped: line 51 calls user.update({ «REDACTED: Generic Secret (unquoted) (29 chars)» }) with no proof the caller knows the existing password. Any party holding a valid session token (e.g.*

**Recommended fix:** Make current mandatory: reject with 401 when it is empty/absent (before the update), and compare it constant-time against loggedInUser.data.password. Change if (currentPassword && ...) at line 39 so a missing current password is a hard failure, not a bypass. Move the endpoint from GET to POST so the secret is not placed in the query string / logs.

#### F15: Unauthorized Data Access (High) Severity: 7.5 (High) · Fix within 30 days

**What this means for your business:** An endpoint at routes/order.ts:35 decides which record to return or change from an id in the request, but does not check that the record actually belongs to the user (or tenant) making the call. A logged-in user can change that id to someone else's value and read or modify another customer's data, crossing the wall between tenants that is supposed to keep each account's data private.

**Evidence:** routes/order.ts:35, AI analysis — verify before acting

*Technical detail: Write-side twin of the basket-read BOLA. /rest/basket/:id/checkout is covered only by the security.isAuthenticated() prefix (server.ts:398), valid token, no ownership. placeOrder reads const id = req.params.id (order.ts:34) and BasketModel.findOne({ where: { id } }) (line 35) with no owner check, then on success runs BasketItemModel.destroy({ where: { BasketId: id } }) (line 50). An authenticated attacker checks out a victim's basket and wipes its items, a destructive, state-changing cross-user action. Found by sweeping every :id basket route, not just the read.*

**Recommended fix:** Before loading the basket in placeOrder, confirm req.params.id equals the session user's basket id (security.authenticatedUsers.from(req).bid) or scope the findOne by the session UserId; return 403 on mismatch. This single ownership check also closes the basket-read and coupon BOLAs.

#### F16: Open Redirect (High) Severity: 7.5 (High) · Fix within 30 days

**What this means for your business:** A significant security weakness was found at routes/redirect.ts:19. Left unfixed, an attacker could abuse it to access or disrupt data and functions in this area. The technical detail below explains exactly what was found.

**Evidence:** routes/redirect.ts:19, Tool-confirmed

*Technical detail: Passing untrusted user input in redirect() can result in an open redirect vulnerability. This could be abused by malicious actors to trick users into being redirected to websites under their control to capture authentication information.*

To prevent open redirect vulnerabilities:

- Always validate and sanitize user inputs, especially URL parameters or query strings that may influence the flow of the application.
- Use allowlists (lists of permitted URLs) to validate redirect targets against known, trusted URLs before performing the redirect.

**Recommended fix:** Do not build redirects from user-controlled input. Redirect only to a server-side allowlist of paths/hosts, or use a fixed relative path. If an external redirect is required, validate the target against an explicit allowlist before issuing the 3xx.

#### F17: Business-Logic Abuse (High) Severity: 7.5 (High) · Fix within 30 days

**What this means for your business:** A business rule at routes/wallet.ts:27 is enforced loosely enough that the normal flow can be bent. An attacker can push the feature into a state you did not intend, for example a price, quantity, discount, or ownership it should reject, and gain something they are not entitled to.

**Evidence:** routes/wallet.ts:27, AI analysis — verify before acting

*Technical detail: Verified guard first: the cross-user IDOR is NOT exploitable here, /rest/wallet/balance PUT runs security.appendUserId() (server.ts:626) which OVERWRITES req.body.UserId with the session user id*

(*lib/insecurity.ts:177-178*), so a caller cannot top up someone else's wallet. The real flaw is the amount: *addWalletBalance* only checks that a card with *req.body.paymentId* exists for the user (*wallet.ts:24*), then runs *WalletModel.increment* (*{ balance: req.body.balance }*) (line 27).

**Recommended fix:** Do not treat *req.body.balance* as authorized credit. Charge the selected card for that exact amount through the payment flow and only *increment* by the amount the processor confirms as captured. Reject non-positive and out-of-range values server-side before the increment, and make card existence a precondition of the charge, not a substitute for it.

## Reproduce Locally (Critical / High)

How to confirm each Critical/High finding locally. Run in a non-production checkout; these are verification steps, not weaponised exploits.

### F01: Server Takeover

1. Open *routes/b2bOrder.ts:23* and confirm untrusted data reaches a dynamic-eval / deserialisation sink.
2. Reproduce locally by tracing the data flow with a benign marker value in a non-production checkout.

### F02: Code-Injection Attack

1. Open *routes/captcha.ts:22* and confirm untrusted data reaches a dynamic-eval / deserialisation sink.
2. Reproduce locally by tracing the data flow with a benign marker value in a non-production checkout.

### F03: Code-Injection Attack

1. Open *routes/userProfile.ts:61* and confirm untrusted data reaches a dynamic-eval / deserialisation sink.
2. Reproduce locally by tracing the data flow with a benign marker value in a non-production checkout.

### F04: Stolen Credentials

1. Open *lib/insecurity.ts:23* and confirm the flagged value is a live credential, not a placeholder or test fixture.
2. Re-run your secret scanner against the repository to reproduce; if it is real, treat it as exposed and rotate it immediately.

### F05: Internal Network Abuse

1. Open *routes/profileImageUrlUpload.ts:24* and confirm a user-controlled URL/host is fetched server-side without an allow-list.
2. Locally, point the input at a harmless internal endpoint you control and confirm the server makes the outbound request.

### F06: Database Attack

1. Open *routes/showProductReviews.ts:31* and confirm user-controlled input reaches the SQL query without parameterisation.
2. In a NON-production checkout, send a benign test input (e.g. a value with a single quote) to the endpoint and observe whether the query breaks — do NOT run against production data.

### F07: Database Attack

1. Open *routes/trackOrder.ts:15* and confirm user-controlled input reaches the SQL query without parameterisation.
2. In a NON-production checkout, send a benign test input (e.g. a value with a single quote) to the endpoint and observe whether the query breaks — do NOT run against production data.

### F08: Code-Injection Attack

1. Open *routes/vulnCodeFixes.ts:81* and confirm untrusted data reaches a dynamic-eval / deserialisation sink.
2. Reproduce locally by tracing the data flow with a benign marker value in a non-production checkout.

### F09: Code-Injection Attack

1. Open *rsn/rsnUtil.ts:135* and confirm untrusted data reaches a dynamic-eval / deserialisation sink.



2. Reproduce locally by tracing the data flow with a benign marker value in a non-production checkout.

#### F10: Code-Injection Attack

1. Open `server.ts:139` and confirm untrusted data reaches a dynamic-eval / deserialisation sink.
2. Reproduce locally by tracing the data flow with a benign marker value in a non-production checkout.

#### F11: Stolen Credentials

1. Open `lib/insecurity.ts:56` and confirm the flagged value is a live credential, not a placeholder or test fixture.
2. Re-run your secret scanner against the repository to reproduce; if it is real, treat it as exposed and rotate it immediately.

#### F12: Stolen Credentials

1. Open `routes/login.ts:65` and confirm the flagged value is a live credential, not a placeholder or test fixture.
2. Re-run your secret scanner against the repository to reproduce; if it is real, treat it as exposed and rotate it immediately.

#### F13: Unauthorized Data Access

1. Open `routes/basket.ts:19` and confirm the object reference is trusted from the request without an ownership / authorization check.
2. Locally, request another test user's object id with your own session and confirm access is wrongly granted.

#### F14: Account Takeover Risk

1. Open `routes/changePassword.ts:39` and confirm the route/action runs without enforcing the required authentication / authorization.
2. Locally, call the endpoint without (or with a low-privilege) session and confirm it still succeeds.

#### F15: Unauthorized Data Access

1. Open `routes/order.ts:35` and confirm the object reference is trusted from the request without an ownership / authorization check.
2. Locally, request another test user's object id with your own session and confirm access is wrongly granted.

#### F16: Open Redirect

1. Open `routes/redirect.ts:19` and confirm a user-controlled value drives a redirect target without an allow-list.
2. Locally, supply a benign external URL and confirm the app redirects to it.

#### F17: Business-Logic Abuse

1. Open `routes/wallet.ts:27` and confirm the flagged code/configuration is present as reported.
2. Re-run the tool that produced this finding against a local checkout to reproduce it.

## 12. Compliance Mapping

Findings are grouped by the compliance controls they map to. Each distinct mapping is listed once, naming the findings that share it (full per-finding detail is in §11).

Findings	PCI DSS v4	SOC2 CC	ISO 27001:2022	OWASP 2021	CIS Benchmark
25 findings (F04, F11, F12, F63, F64, F65, F66, F67, F68, F69, ... +15 more (F04–F84, full list in §11 / JSON))	PCI DSS v4.0 §8.3.1, §8.6.1, §8.6.2	SOC 2 CC6.1, CC6.2	ISO/IEC 27001:2022 A.5.16, A.8.24	—	—

Findings	PCI DSS v4	SOC2 CC	ISO 27001:2022	OWASP 2021	CIS Benchmark
1 finding (F05)	PCI DSS v4.0 §6.3.1, §6.3.2	SOC 2 CC6.6	ISO/IEC 27001:2022 A.8.20, A.8.25	—	—
3 findings (F08, F09, F10)	PCI DSS v4.0 §6.3.1, §6.3.3	SOC 2 CC6.8	ISO/IEC 27001:2022 A.8.25, A.8.28	—	—
1 finding (F16)	PCI DSS v4.0 §6.3.1	SOC 2 CC6.1	—	—	—
6 findings (F18, F19, F20, F21, F22, F23)	PCI DSS v4.0 §6.3.1, §6.3.2	SOC 2 CC6.1, CC6.8	ISO/IEC 27001:2022 A.8.25, A.8.28	—	—
128 findings (F27, F28, F29, F30, F100, F101, F102, F103, F104, F105, ... +118 more (F27–F223, full list in §11 / JSON))	PCI DSS v4.0 §6.3.2, §6.3.3	SOC 2 CC6.8	ISO/IEC 27001:2022 A.8.25, A.8.8	—	—
22 findings (F35, F44, F45, F46, F48, F50, F51, F52, F53, F54, ... +12 more (F35–F243, full list in §11 / JSON))	PCI DSS v4.0 §6.3.1, §6.3.2	SOC 2 CC6.1, CC6.6	ISO/IEC 27001:2022 A.8.3, A.8.25	—	—
1 finding (F36)	PCI DSS v4.0 §6.3.1, §6.3.2, §6.3.3	SOC 2 CC6.8	ISO/IEC 27001:2022 A.8.25, A.8.28	—	—

**80 findings have no specific framework mapping** (IDs: F01, F02, F03, F06, F07, F13, F14, F15, F17, F24, ... +70 more (F01–F267, full list in §11 / JSON)). These are still real issues — see §11 for each.

### 13. Action Plan (30/60/90-Day)

Findings are rolled up by risk type within each 30/60/90-day window (count + typical fix effort). Every finding is included; the full per-item list with file locations is in the Technical report. Timelines align with the Severity Legend (§5a).

**30-day — Critical (within 7 days) + High (within 30 days)**

Risk type	Count	Typical fix effort
Code-Injection Attack	5	Small fix (1–2 hrs)
Stolen Credentials	3	Config change (~5 min)
Database Attack	2	Small fix (1–2 hrs)
Unauthorized Data Access	2	Small fix (1–2 hrs)
Account Takeover Risk	1	Small fix (1–2 hrs)
Business-Logic Abuse	1	Small fix (1–2 hrs)
Internal Network Abuse	1	Small fix (1–2 hrs)
Open Redirect	1	Small fix (1–2 hrs)
Server Takeover	1	Small fix (1–2 hrs)

**60-day — Medium — fix within 60 days**

Risk type	Count	Typical fix effort
File-System Escape	18	Small fix (1–2 hrs)
Security Weakness	8	Small fix (1–2 hrs)
Browser Hijack	6	Small fix (1–2 hrs)
Unpinned CI/CD Dependency	4	Small fix (1–2 hrs)
Container Hardening Gap	2	Config change (~5 min)
Timing Side-Channel	2	Small fix (1–2 hrs)
Business-Logic Abuse	1	Small fix (1–2 hrs)
Cross-Origin Misconfig	1	Config change (~5 min)
Missing Dependency Lockfile	1	Small fix (1–2 hrs)
Predictable Randomness	1	Small fix (1–2 hrs)
Unauthorized Data Access	1	Small fix (1–2 hrs)

**90-day — Low — fix within 90 days / when convenient**

Risk type	Count	Typical fix effort
Dependency Pinning / Lockfile Gap	124	One-liner fix (~15 min)
Stolen Credentials	22	Config change (~5 min)
Database Attack	20	Small fix (1–2 hrs)
Predictable Randomness	20	Small fix (1–2 hrs)
Container Hardening Gap	15	Config change (~5 min)
File-System Escape	4	Small fix (1–2 hrs)

**13a. AI Fix Prompts**

A pasteable prompt for every finding, stored in the **fix-prompts/** directory alongside this report (findings that share one root fix — such as dependency-pinning — are grouped into a single brief that lists every affected item). Open the file for a finding, copy its contents, and paste into your AI coding tool (Cursor, Copilot, Claude Code) — each prompt is self-contained (location, issue, remediation, references) so no extra context is required.

Honesty label per artifact: a **fix\_prompt\_grounded: true** header means the prompt cites a verified file:line from your repo; **false** means the generator used the deterministic CWE-template fallback (still actionable, but less precise).

F-ID	Severity	Prompt file	Grounded
F100	Low	<b>fix-prompts/4L001-F100.md</b>	no (fallback)
F63	Low	<b>fix-prompts/4L002-F63.md</b>	no (fallback)
F89	Low	<b>fix-prompts/4L003-F89.md</b>	no (fallback)
F225	Low	<b>fix-prompts/4L004-F225.md</b>	no (fallback)
F01	Critical	<b>fix-prompts/1C001-F01.md</b>	yes (candidate)
F02	Critical	<b>fix-prompts/1C002-F02.md</b>	yes (candidate)
F03	Critical	<b>fix-prompts/1C003-F03.md</b>	yes (candidate)
F04	Critical	<b>fix-prompts/1C004-F04.md</b>	yes (candidate)
F05	Critical	<b>fix-prompts/1C005-F05.md</b>	yes (candidate)
F06	Critical	<b>fix-prompts/1C006-F06.md</b>	yes (candidate)
F07	Critical	<b>fix-prompts/1C007-F07.md</b>	yes (candidate)
F08	High	<b>fix-prompts/2H001-F08.md</b>	yes (candidate)
F09	High	<b>fix-prompts/2H002-F09.md</b>	yes (candidate)
F10	High	<b>fix-prompts/2H003-F10.md</b>	yes (candidate)
F11	High	<b>fix-prompts/2H004-F11.md</b>	yes (candidate)
F12	High	<b>fix-prompts/2H005-F12.md</b>	yes (candidate)

F-ID	Severity	Prompt file	Grounded
F13	High	fix-prompts/2H006-F13.md	yes (candidate)
F14	High	fix-prompts/2H007-F14.md	yes (candidate)
F15	High	fix-prompts/2H008-F15.md	yes (candidate)
F16	High	fix-prompts/2H009-F16.md	yes (candidate)
F17	High	fix-prompts/2H010-F17.md	yes (candidate)
F18	Medium	fix-prompts/3M001-F18.md	yes (candidate)
F19	Medium	fix-prompts/3M002-F19.md	yes (candidate)
F20	Medium	fix-prompts/3M003-F20.md	yes (candidate)
F21	Medium	fix-prompts/3M004-F21.md	yes (candidate)
F22	Medium	fix-prompts/3M005-F22.md	yes (candidate)
F23	Medium	fix-prompts/3M006-F23.md	yes (candidate)
F24	Medium	fix-prompts/3M007-F24.md	yes (candidate)
F25	Medium	fix-prompts/3M008-F25.md	yes (candidate)
F26	Medium	fix-prompts/3M009-F26.md	yes (candidate)
F27	Medium	fix-prompts/3M010-F27.md	yes (candidate)
F28	Medium	fix-prompts/3M011-F28.md	yes (candidate)
F29	Medium	fix-prompts/3M012-F29.md	yes (candidate)
F30	Medium	fix-prompts/3M013-F30.md	yes (candidate)
F31	Medium	fix-prompts/3M014-F31.md	yes (candidate)
F32	Medium	fix-prompts/3M015-F32.md	yes (candidate)
F33	Medium	fix-prompts/3M016-F33.md	yes (candidate)
F34	Medium	fix-prompts/3M017-F34.md	yes (candidate)
F35	Medium	fix-prompts/3M018-F35.md	yes (candidate)
F36	Medium	fix-prompts/3M019-F36.md	no (fallback)
F37	Medium	fix-prompts/3M020-F37.md	yes (candidate)
F38	Medium	fix-prompts/3M021-F38.md	yes (candidate)
F39	Medium	fix-prompts/3M022-F39.md	yes (candidate)
F40	Medium	fix-prompts/3M023-F40.md	yes (candidate)
F41	Medium	fix-prompts/3M024-F41.md	yes (candidate)
F42	Medium	fix-prompts/3M025-F42.md	yes (candidate)
F43	Medium	fix-prompts/3M026-F43.md	yes (candidate)
F44	Medium	fix-prompts/3M027-F44.md	yes (candidate)
F45	Medium	fix-prompts/3M028-F45.md	yes (candidate)
F46	Medium	fix-prompts/3M029-F46.md	yes (candidate)
F47	Medium	fix-prompts/3M030-F47.md	yes (candidate)
F48	Medium	fix-prompts/3M031-F48.md	yes (candidate)
F49	Medium	fix-prompts/3M032-F49.md	yes (candidate)
F50	Medium	fix-prompts/3M033-F50.md	yes (candidate)
F51	Medium	fix-prompts/3M034-F51.md	yes (candidate)
F52	Medium	fix-prompts/3M035-F52.md	yes (candidate)
F53	Medium	fix-prompts/3M036-F53.md	yes (candidate)
F54	Medium	fix-prompts/3M037-F54.md	yes (candidate)
F55	Medium	fix-prompts/3M038-F55.md	yes (candidate)
F56	Medium	fix-prompts/3M039-F56.md	yes (candidate)
F57	Medium	fix-prompts/3M040-F57.md	yes (candidate)
F58	Medium	fix-prompts/3M041-F58.md	yes (candidate)
F59	Medium	fix-prompts/3M042-F59.md	yes (candidate)
F60	Medium	fix-prompts/3M043-F60.md	yes (candidate)
F61	Medium	fix-prompts/3M044-F61.md	yes (candidate)
F62	Medium	fix-prompts/3M045-F62.md	yes (candidate)
F85	Low	fix-prompts/4L005-F85.md	no (fallback)
F86	Low	fix-prompts/4L006-F86.md	yes (candidate)
F87	Low	fix-prompts/4L007-F87.md	yes (candidate)
F88	Low	fix-prompts/4L008-F88.md	no (fallback)
F99	Low	fix-prompts/4L009-F99.md	no (fallback)

F-ID	Severity	Prompt file	Grounded
F224	Low	fix-prompts/4L010-F224.md	yes (candidate)
F241	Low	fix-prompts/4L011-F241.md	yes (candidate)
F242	Low	fix-prompts/4L012-F242.md	yes (candidate)
F243	Low	fix-prompts/4L013-F243.md	yes (candidate)

## 14. Attestation & Re-test Statement

This automated assessment is provided “AS IS” without warranty. It is NOT a legal compliance audit. See EULA.md and NOTICE shipped with vollos-lens for full terms, limitation of liability, and third-party attributions. Security findings depend on user-installed scanner versions and ruleset quality (BYO mode — Wave 1d). Skill does not guarantee coverage. See EULA §5.1 (Disclaimer 3).

Field	Value
Assessed By	Automated Assessment by Vollos Lens
Date	2026-06-12T13:39:23Z
Valid Until	2026-09-10
Recommended Re-test	Within 90 days or after major release

DISCLAIMER: This assessment is a point-in-time snapshot as of 2026-06-12T13:39:23Z. No warranty expressed or implied. Liability limited per engagement SOW. Security posture may change; re-assessment required for ongoing assurance.

Methodology Attribution: This report uses methodology derived from OWASP Top 10, OWASP API Security Top 10 2023, OWASP LLM Top 10 2025, and OWASP Mobile Top 10 2024 — © OWASP Foundation, licensed CC-BY-SA-4.0 (<https://creativecommons.org/licenses/by-sa/4.0/>). The OWASP Foundation does not endorse this report. Tool licenses and full third-party attributions: see the NOTICE file shipped with vollos-lens.

### 14a. Engagement Metadata & Sign-off

#### Engagement Metadata

Field	Value
Customer	juice-shop
Report ID	—
Report version	—
Scan started (UTC)	—
Scan ended (UTC)	—
Scan duration	—
Repository commit	6244c59a47ba
Branch	master
Lines of code scanned	227410
Dependencies scanned	—
Files scanned	1257
Tools invoked	secret scanner, configuration scanner, dependency scanner, static code analysis

#### Sign-off

- **Security Reviewer:** Automated Assessment by Vollos Lens
- **Title:** Automated Security Assessment, Vollos Lens
- **Contact:** —
- **Date of sign-off:** 2026-06-12

Signature:

-----  
Automated Assessment by Vollos Lens (printed)

This report is provided per the Engagement Terms agreed between the customer and the Security Reviewer. It is subject to the EULA shipping with the Vollos Lens skill, in particular §3 (Limitation of Liability), §4 (Not a Compliance Audit), §6 (Coverage Limitations), and §8 (Supply Chain Risk).

## 15. Appendix A — Tool Coverage & Step Scores

### Tool Inventory

Tool	Version	Status
secret scanner	8.21.2	ok
configuration scanner	2.12.0	conditional-skip
dependency scanner	2.3.8, 0.70.0	ok
static code analysis	1.163.0	ok

*Status is each scanner's availability recorded at the start of the scan — **ok**: a required scanner, installed and meeting the minimum version; **conditional-skip**: an optional scanner that is installed. It reflects tool provisioning, not per-finding execution.*

### Step Scores

Step	Title	Score	Weight
Step 2	Dependency Audit	—	0.08
Step 3	Supply Chain	0	0.10
Step 4	Working-Tree Secrets	25	0.14
Step 5	Static Analysis	0	0.08
Step 6	OWASP Top 10	0	0.18
Step 7	Dynamic Analysis	—	0.08
Step 8	Manual Testing	—	0.04
Step 9	Infrastructure	47	0.13
Step 10	Backup & IR	—	0.02
Step 11	Adversarial AI Review	0	0.15

## 16. Appendix B — SBOM Summary

*SBOM not generated — pass `--sbom` flag.*

## 17. Appendix C — Glossary

- **API key** — A secret string an application uses to authenticate to an external service. Leaks of API keys allow attackers to impersonate your application.
- **Authentication** — Verifying who a user claims to be (usually via password, token, or biometric).
- **Authorization** — Deciding what an authenticated user is allowed to do.
- **CVE** — Common Vulnerabilities and Exposures — a public identifier for a known security issue (e.g. CVE-2024-12345). Tracked at [nvd.nist.gov](https://nvd.nist.gov).
- **CVSS** — Common Vulnerability Scoring System — a 0.0 to 10.0 severity score for a vulnerability.
- **CWE** — Common Weakness Enumeration — a category of weakness (e.g. CWE-79 = Cross-Site Scripting).
- **Dependency** — A third-party library or package your code uses.
- **Endpoint** — A specific URL path that accepts requests.
- **Exploit** — Working code or technique that turns a vulnerability into actual impact.

- **OWASP** — Open Worldwide Application Security Project — a non-profit publishing security guidance and the Top 10 most common web application risks.
- **Patch** — A code change that fixes a security issue.
- **Repository** — The full source code history of a project, typically stored in git.
- **SARIF** — Static Analysis Results Interchange Format — a JSON schema that GitHub, GitLab, and many tools accept for security findings.
- **Token** — A short-lived credential that proves identity or permission (e.g. JWT, OAuth access token).
- **Vulnerability** — A weakness in software that could be exploited to cause unauthorized impact.
- **SBOM** — Software Bill of Materials — list of project dependencies.
- **P0/P1/P2/P3** — Priority tier — P0 is immediate action.

## 18. Appendix D — Raw Artifacts Index

File	Size	SHA256 (short)
<code>security_report.json</code>	415415	109328ff
<code>security_report.sarif</code>	334875	04b4d952

## 18a. Appendix E — Scope NOT Covered

The following are EXPLICITLY OUTSIDE the scope of this engagement and were NOT assessed in this report:

- Social engineering and phishing simulation
- Physical security (premises, hardware, device theft)
- Third-party SaaS internals (Stripe, AWS managed services, Auth0, etc.)
- Manual penetration testing (network exploitation, lateral movement)
- Compliance certification (SOC 2, ISO 27001, PCI-DSS, HIPAA, GDPR)
- Mobile application native code (iOS/Android binaries; only web layers scanned where present)
- Continuous monitoring (this is a point-in-time scan only)
- Personnel security (background checks, access reviews)
- Disaster recovery and business continuity testing

If any of the above areas are relevant to your business, please engage a qualified specialist for each. We are happy to provide referrals on request.

## 18b. Appendix F — Scan Provenance

Vulnerability data as of 2026-06-12.

Field	Value
Run timestamp (UTC)	2026-06-12T04:43:50Z
Target git commit	6244c59a47ba4436cb00e9ad9c565bb2056582ec

### Scanner Coverage (versions pinned)

Scanner category	Version	Status
secret scanner	8.21.2	ok
configuration scanner	2.12.0	conditional-skip
dependency scanner	2.3.8	ok
static code analysis	1.163.0	ok
dependency scanner	0.70.0	ok

## Vulnerability DB Snapshots

Scanner category	DB Snapshot	Refreshed	Note
dependency scanner	online-live (queried 2026-06-12)	yes	online mode = live osv.dev advisories (fresh) but NOT snapshot-pinned; re-run pins to scanner version, not a DB id
dependency scanner	2026-06-12T01:16:29.442164271Z	yes	DB refreshed from network

The mechanical layer is reproducible given the pinned scanner versions + the DB snapshot above; the AI-exploratory layer is same-coverage, near-same-content. A re-run can be pinned to the snapshot date listed here.

---

© 2026 Vollos Lens. This document contains information proprietary to juice-shop. Reproduction without written consent is prohibited. Report prepared 2026-06-12.