Security Audit Report

Prepared by: CYBRING
Prepared for: Ferra DLMM Backend

Version: 1.0 (public version)
Date: 29th Sep 2025
Commit/Hash: a71e7c69ab90214a3290b024bf88ebad5e718d9d
Auditor: Anh Bui, Duc Cuong Nguyen

Contents

1 Introduction

This document presents the results of the initial security assessment of the `Ferra`'s DLMM API service. This service provides structured access to on-chain liquidity and market data. Built with TypeScript; a Node.js HTTP service that ingests and processes relevant contract activity, storing normalized records in a database.

1.1 Objective

This security audit report was initially provided by `CYBRING` on 9th September 2025. This audit was conducted to assess the robustness, reliability, and security of the code base. The goal was to identify vulnerabilities, ensure compliance with best security practices, and provide mitigation measures.

After the initial audit report, a reassessment was conducted on 29th Sep 2025 to verify the status of the reported issues. All issues were acknowledged and will be prioritized for future improvement.

1.2 Disclaimer

This security audit is not produced to replace any other type of assessment and does not aim to guarantee the discovery of all security issues within the scope of the assessment. Further, economic modeling, business-logic desirability, and market comparisons were explicitly out of scope.

While this audit was carried out in good faith and technical proficiency, it's crucial to understand that no single audit can guarantee the absolute security of a backend service. To minimize risks, `CYBRING` recommends a multi-faceted approach involving multiple independent assessments. Please note that this report is not intended to provide financial advice.

This is the public version; some operational details and PoCs were generalized or removed.

2 Scope of Audit

The audit was conducted on commit a71e7c69ab90214a3290b024bf88ebad5e718d9d from git repository `https://github.com/Ferra-Labs/ferra-dlmm-api/`. Further details regarding The `Ferra` audit scope are provided below:

- **Codebase details:**
  - Language: TypeScript
  - Runtime: Node.js
  - Initial audit's commit hash: 39daae5af72b33088c70fc78c9006de3fbaf5828
  - Reassessment audit's commit hash: ca71524a241ffefa3780741c16ea549b14ab605f

3 Audit Summary

CYBRING assessed Ferra's DLMM backend for security, reliability, and operational hardening. The initial review found 8 issues (2 High / 3 Medium / 2 Low; no Critical). The Ferra team addressed all High and Medium items and partially updated dependency risks. We recommend keeping dependency upgrades on a regular cadence and continuing to strengthen abuse-prevention telemetry. No vulnerabilities remain that we consider high risk for typical production deployments at the time of this publication The assessment report is summarized in Table 1. Details of findings can be found in Section 5.

Severity	Count
High	2
Medium	3
Low	2

Table 1: Initial assessment summary

Summary of reassessment can be found in Table 2.

Issue	Severity	Status	Remediation Evidence
5.1	High	Fixed	ca71524a
5.2	High	Fixed	ca71524a
5.3	Medium	Fixed	ca71524a
5.4	Medium	Fixed	ca71524a
5.5	Medium	Fixed	ca71524a
5.6	Low	Fixed	ca71524a
5.7	Low	Partially fixed	ca71524a

Table 2: Final assessment summary

4 Methodology

4.1 Audit Items

CYBRING conducts the following procedures to enhance the security posture of Ferra backend services:

- **Pre-auditing:** Understand business workflows and trust boundaries, review architecture (API gateway, services, DB, message queues, cache, files/storage), enumerate external integrations, and collect artifacts (OpenAPI/Swagger, env/config, deployment manifests, CI/CD pipeline overview).
- **Auditing:** Examine the service from multiple perspectives:
  - **Manual code review:** Inspect request handlers and middleware for input validation, authentication/authorization, multi-tenant scoping, unsafe patterns (e.g., unsanitized dynamic queries, SSRF, command/OS calls, path traversal),

error handling, logging/redaction, and privacy. Verify schema-first validation is enforced on every route and response.

- **Static analysis & supply-chain review:** Run linters and type checks (strict TS config), secret scanning, and SCA (dependency vulnerability and license checks). Review `package.json` scripts, lockfile hygiene, dependency pinning, and transitive risk (e.g., postinstall scripts).
- **Dynamic testing (DAST) & API fuzzing:** Exercise endpoints against the OpenAPI contract with negative tests and fuzzing (boundary sizes, invalid types, malformed JSON, stateful sequences). Probe for authz bypass, IDOR, business-logic flaws, CORS/CSRF misconfig, rate-limit gaps, regex DoS (ReDoS), large-payload DoS, and event-loop blocking.
- **Configuration & deployment review:** Evaluate security plugins (e.g., rate-limit, cors, CSRF protections when using cookies), HTTP security headers, TLS/HSTS at the edge, cookie attributes, environment configuration, secrets management, Docker/K8s hardening (user, fs perms, resource limits), and CI/CD guardrails.
- **Authentication & authorization testing:** Verify token/session lifecycle (rotation, revocation, device binding), claim validation (iss/aud/exp/nbf), privilege boundaries, tenant isolation, and reference-monitor enforcement on *every* sensitive action.
- **Resilience & availability checks:** Assess input size/time complexity limits, file upload controls, compression/zip-bomb defenses, outbound egress controls, and graceful error handling; sample load to identify hot paths that can block the event loop.

- **First deliverable and consulting:** Provide an initial report, affected endpoints, risk ratings, and prioritized remediation guidance. Offer implementation Q&A.
- **Reassessment:** Verify fixes, re-test edge cases, and check for regressions or newly introduced issues.
- **Final deliverable:** Deliver a comprehensive report with final statuses, risk summaries, and hardening checklist.

Details of audit items can be found in Appendix A

4.2 Risk Rating

The OWASP Risk Rating Methodology is applied to backend issues using:

- **Likelihood:** how easily the flaw can be discovered/exploited (preconditions, attack surface, required auth/role, exploit reliability).
- **Impact:** business impact on confidentiality, integrity, availability, financial loss, privacy, and compliance.

Impact		Likelihood		
		Low	*Medium*	*High*
High		Medium	High	Critical
Medium		Low	Medium	High
Low		Informational	Low	Medium

Table 3: Overall Risk Severity

4.3 Audit Categories

- **Common vulnerabilities:** Assessed against OWASP Top 10 (Web) and OWASP API Security Top 10 (2023).
- **Advanced vulnerabilities:** Targeted exploitation of business logic, multi-tenant isolation, and chained flaws (e.g., SSRF → metadata access, authn bypass → IDOR).
- **Security best practices:** Coding standards, TypeScript strictness, schema-first validation, secure plugin configuration, deployment hardening, observability, and incident readiness.

5 Detailed Findings

5.1 Pair filter parameterization causes zero matches for both exact and wildcard queries

- **Location**: Pair service
- **Description:** Filter values are wrapped in PostgreSQL dollar-quote delimiters before being passed as bound parameters, e.g. code pushes $$$value$$ into *queryParams*. Parameter binding already handles quoting. Wrapping values in $$...$$ turns the parameter into a literal containing dollar signs; wildcards like % are treated as literal characters, not pattern markers. Additionally, token_x_type uses = (case-sensitive), while token_y_type uses ILIKE (case-insensitive). This inconsistency can hide matches if clients don't match exact case for coin_x.
- **Risk:** High (Impact: High, Likelihood: Medium)
- **CWE:** CWE-20: Improper Input Validation
- **Mitigation:** Stop wrapping parameter values in $$...$$; pass raw values to the parameter array. Apply pattern matching only in SQL, not in the parameter value: For exact: token_x_type = $1. For wildcard: token_y_type ILIKE $1
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.2 Transaction retry/bail bug due to missing await causes non-retryable errors to be retried

- **Location**: DB wrapper

- **Description:** If *singleTxFn* returns a Promise and isn't awaited, errors won't be caught by that try/catch. The bail path will not execute for async errors you intended to handle specially.
- **Risk:** High (Impact: High, Medium)
- **CWE:** CWE-754: Improper Check or Handling of Exceptional Conditions
- **Mitigation**: Change return *singleTxFn(bail)* to return *await singleTxFn(bail)*.
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.3 Missing important security headers

- **Location**: Server's config
- **Description:** There are several missing items with the application's HTTP security headers: *[redacted]*
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)
- **CWE**: CWE-1173: Improper Use of Validation Framework
- **Mitigation**: To improve the security of your application's HTTP headers, the following actions should applied: *[redacted]*
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.4 Hardcoded credentials

- **Location**: Core library
- **Description:** API key is hardcoded.
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)
- **CWE:** CWE-798: Use of Hard-coded Credentials
- **Mitigation:** To fix this issue and prevent it from happening again, follow these steps:
  - Remove the key from your codebase and its Git history.
  - Immediately rotate the key and load it from a secret manager.
  - Implement a pre-receive secret scanning to block commits containing secrets.
  - Add a CI/CD secret scanning step to your pipeline.
  - Block the commitment of `.env` files and similar sensitive configuration files.
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.5 Using the same key for JWT and Encryption

- **Location**: environment variables
- **Description:** *[redacted-1]* defaults to *[redacted-2]*.
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)
- **CWE:** CWE-320: Key Management Errors
- **Mitigation:** Require separate strong keys
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.6 Abuse-prevention controls insufficient (rate/ban/telemetry)

- **Location**: Server's config
- **Description:** Abuse controls omit onExceeded/ban callbacks for detection and temporary blocking.
- **Risk:** Low (Impact: Medium, Likelihood: Low)
- **CWE:** CWE-770: Allocation of Resources Without Limits or Throttling
- **Mitigation**: Use onExceeded/ban to log and optionally block abusive clients; groupId lets you share a bucket across related endpoints
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.7 Outdated Third-Party Dependencies

- **Location**: Dependency list
- **Description:** An analysis of the project's dependencies identified several third-party libraries that are either outdated or contain known security vulnerabilities. Using these vulnerable components can expose the application to various risks.

  The following dependencies were identified as outdated and should be updated to their specified minimum versions to resolve known vulnerabilities: *[redacted]*
- **Risk:** Low (Impact: Medium, Likelihood: Low)
- **CWE:** CWE-1104: Use of Unmaintained Third Party Components
- **Mitigation:** To mitigate this finding, all identified dependencies should be updated to their specified secure versions. Regularly updating dependencies is a critical practice for maintaining the security and stability of the application.
- **Status [29/09/2025]:** `Ferra` team has partially fixed this issue.

A Appendix

Area	Representative Checks (TypeScript)
Input & Data Validation	JSON schema on all routes (AJV/TypeBox/Zod); type coercion off where unsafe; max body size; strict content types; canonicalization; defense against prototype pollution and mass assignment.
Authentication	Passwordless/OIDC/JWT/session design; token storage (cookies vs. headers); rotation/revocation; session fixation; cookie flags (HttpOnly, Secure, SameSite); MFA hooks where applicable.
Authorization & Tenancy	RBAC/ABAC enforcement on every handler; resource-scoped checks; IDOR; insecure direct joins; cross-tenant data leakage; reference monitor centralization.
Crypto & Secrets	Key management, JWKS pinning/rotation, alg selection; env secret handling (no secrets in repo/logs); KMS/secret store usage; TLS config.
Transport & Headers	HSTS, CSP, X-Frame-Options/frame-ancestors, X-Content-Type-Options, cache policy; security-header middleware (Helmet-style) configuration; redirect safety.
CORS & CSRF	Origin/host validation; allowed methods/headers; credentialed requests; CSRF defenses; preflight handling.
Business Logic	Workflow abuse, replay, re-entrancy-like sequences across endpoints, order-of-operations, race conditions, money/-points/limits enforcement.
Storage & Query Safety	SQL/NoSQL injection; Prisma/TypeORM query patterns; transaction boundaries; isolation levels; migration safety; pagination/limit caps.
SSRF & Egress	URL fetchers (DNS rebinding, IP allow-list bypass, localhost/metadata access), redirect chaInspins, file://, gopher://, and %2F tricks.
File Handling	MIME sniffing, extension checks, upload size/number limits, image processing safety, decompression bombs, antivirus hooks (if applicable).
DoS	Performance Rate limits (middleware or gateway); per-IP/account quotas; slowloris protection; timeouts; ReDoS; event-loop blocking hotspots.
Error Handling & Logging	No stack traces to clients; safe error codes; PII/secret redaction; correlation IDs; security/audit logs with tamper resistance.
Dependency & Build	SCA results; lockfile hygiene; banned/postinstall scripts; reproducible builds (`npm ci`); TS compiler strictness; `ts-node` not in prod.
Config & Deploy	Docker/K8s least privilege (non-root, RO FS); health/readiness; env var validation; feature flags; production toggles; debug endpoints disabled.
Observability & IR	Metrics/alerts for auth failures, 4xx/5xx spikes, rate-limit hits; trace sampling; runbooks and incident hooks.
Privacy & Compliance	Data minimization; retention/TTL; access logging for sensitive records; export/delete workflows; consent and notice.
Documentation & API Contract	OpenAPI accuracy; response schemas; deprecation policy; versioning; idempotency keys for mutating endpoints.

Table 4: Backend (TypeScript) audit items assessed by `CYBRING`.