Security Audit Report

Prepared by: CYBRING
Prepared for: Ferra SDKs

Version: 1.0 (public version)
Date: 29th Sep 2025
Commit/Hash: b3d0422cff1b430077ed541eb32fc537acb26fb2
Auditor: Anh Bui, Duc Cuong Nguyen

Contents

1 Introduction

This document presents the results of the initial security assessment of the `Ferra` SDKs. It is a modular smart-routing quote engine that models on-chain liquidity as a directed graph, enumerates bounded multi-hop swap paths, simulates execution to rank routes, and (optionally) splits order flow across top candidates for efficiency.

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 (e.g., target price ceilings relative to other launch platforms) 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 SDKs. 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 5fcd2e3395ea0b6fad9d3c6fa377e966c09234be from git repository `https://github.com/Ferra-Labs/ferra-sdks/`. Further details regarding The `Ferra` audit scope are provided below:

- **Codebase details:**
  - Language: TypeScript
  - Initial audit's commit hash: 5fcd2e3395ea0b6fad9d3c6fa377e966c09234be
  - Reassessment audit's commit hash: b3d0422cff1b430077ed541eb32fc537acb26fb2

3 Audit Summary

We initially assessed the `Ferra` SDKs at commit 5fcd2e33 (29th Sep 2025). We reported 17 issues (1 High, 13 Medium, 3 Low). The High (DLMM price-impact denominator)

is fixed in the reassessment commit (b3d0422c). Most Medium items address schema validation and routing/math edge cases; many are fixed and the rest are acknowledged with mitigation plans. We recommend integrators upgrade to $\geq$ b3d0422c, enable per-hop and total minOut, centralize Sui address validation, and adopt strict schema-first checks across all untrusted data.

The assessment report is summarized in Table 1. Details of findings can be found in Section 5.

Severity	Count
High	1
Medium	13
Low	3

Table 1: Initial assessment summary

Summary of reassessment can be found in Table 2.

Issue	Severity	Status	Remediation Evidence
5.1	Medium	Acknowledged	
5.2	Medium	Acknowledged	
5.3	Medium	Fixed	b3d0422c
5.4	Medium	Fixed	b3d0422c
5.5	Low	Fixed	b3d0422c
5.6	Low	Fixed	b3d0422c
5.7	Medium	Fixed	b3d0422c
5.8	Medium	Fixed	b3d0422c
5.9	Medium	Fixed	b3d0422c
5.10	Medium	Fixed	b3d0422c
5.11	Medium	Acknowledged	
5.12	Medium	Fixed	b3d0422c
5.13	Medium	Fixed	b3d0422c
5.14	High	Fixed	b3d0422c
5.15	Medium	Fixed	b3d0422c
5.16	Medium	Acknowledged	
5.17	Low	Acknowledged	

Table 2: Final assessment summary

4 Methodology

4.1 Audit Items

`CYBRING` conducts the following procedures to evaluate the security and correctness of the `Ferra` SDKs and client libraries:

- **Pre-auditing:** Define SDK scope and threat model, enumerate the public API surface, external services consumed (RPCs, quoters/oracles, indexers), and critical invariants (pricing, routing, fee, slippage, numeric ranges). Collect artifacts:

TypeDoc/API reference, TypeScript config, build/bundle scripts (e.g., tsup/rollup), `package.json` and lockfiles, test suites, sample integrations.

- **Auditing:** Examine the SDK from multiple perspectives:
  - **Manual code review:** Inspect exported entry points and internal modules for parameter validation and type-safety; verify schema checks on all untrusted inputs (decoded network responses, user-supplied options). Flag unsafe patterns (prototype-pollution via deep merges, unsafe regex, dynamic `eval`/`Function`, URL parsing pitfalls, path/URL join issues), error handling consistency, and side-effect boundaries (no network/I/O at import time).
  - **Numeric & algorithmic correctness:** Review arithmetic domains (`bigint` vs `number`), rounding/precision rules, overflow/underflow guards, division-by-zero checks, NaN/Infinity handling, time/decimal unit conversions, and AMM/price-impact/fee/slippage formulae. Validate route scoring/determinism and boundary conditions (empty routes, single-hop, extreme liquidity).
  - **Static analysis & supply chain:** Run linters and strict type checks, secret scanning, and SCA (vuln & license). Assess `package.json` scripts, lockfile hygiene/pinning, transitive risks (e.g., `postinstall`), and disallow dangerous APIs. Verify publish-time integrity (provenance, 2FA) where applicable.
  - **Dynamic testing & fuzzing:** Exercise public APIs with property-based tests (boundary sizes, invalid types, random sequences), corpus-based fuzzing for parsers, and golden test vectors. Where applicable, cross-validate outputs against reference implementations or simulated networks (dev/test nets) to ensure economic/math correctness. Verify idempotence, cancellation, and retry behavior.
  - **Packaging & distribution review:** Confirm correct `exports` map, ESM/CJS bundles, type declarations (`.d.ts`), source maps, and tree-shakability (`sideEffects`). Ensure no dynamic code loading by default, no environment-specific assumptions, and safe browser compatibility (no Node-only APIs unless gated).
  - **Wallet/crypto handling (if applicable):** Check that the SDK does not persist or exfiltrate secrets, enforces address/chain-id formats, uses safe randomness where required, and crafts transactions with fail-closed defaults (e.g., `minAmountOut`, slippage limits, correct nonces).
- **First deliverable and consulting:** Provide an initial report with affected modules/functions, version(s), 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 an SDK hardening checklist.

Details of audit items can be found in Appendix A

4.2 Risk Rating

The OWASP Risk Rating Methodology is applied to **SDK/library issues** using:

- **Likelihood:** how easily a flaw can be triggered or exploited through the public API.
- **Impact:** effect on correctness and safety of consuming applications, including economic loss (mispriced trades, incorrect fees/slippage), integrity of crafted transactions, privacy leakage (if telemetry/PII is handled), availability of the caller (hangs, unbounded CPU/memory), and supply-chain blast radius.

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

Table 3: Overall Risk Severity

4.3 Audit Categories

- **Common library vulnerabilities:** Input/schema validation gaps, prototype pollution, insecure parsing/merging, ReDoS, cryptographic misuse or weak randomness, unsafe number handling (`number` vs `bigint`), and exception/edge-case handling that leads to unsafe defaults.
- **Advanced (domain-specific) vulnerabilities:** Algorithmic/economic correctness in routing, fee and price-impact calculations, slippage/min-out enforcement (per hop and aggregate), determinism across runtimes, trust in external data sources (quoters/oracles/RPCs) and authenticity checks.
- **Security best practices:** TypeScript strictness, schema-first validation for all untrusted data, typed error taxonomy and fail-closed behaviors, versioning & deprecation policy, supply-chain and publish integrity, packaging/bundle hygiene, and minimal/no telemetry by default (explicit opt-in if needed).

5 Detailed Findings

5.1 Aggregator: Unsanitized External Route Data

- **Location**: *[redacted]*
- **Description:** The *[redacted]* function performs unsafe type assertion on external API response data without validation.
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)
- **CWE**: CWE-345 (Insufficient Verification of Data Authenticity)

- **Mitigation**: Replace Type Assertion with Validation:
  - Implement runtime validation for all route data fields
  - Validate pool addresses using isValidSuiAddress() before use
  - Check that numerical values are valid (not NaN, within reasonable ranges)
  - Use a validation library (e.g., Zod) to define strict TradingRoute schema
- **Status [29/09/2025]:** `Ferra` team has acknowledged this issue. Validation is done at the backend.

5.2 Aggregator: Slippage protection incomplete

- **Location**: *[redacted]*
- **Description**: Only *[redacted]* path uses `minAmountOut`; *[redacted]* path ignores `minAmountOut`. Multi-hop steps before final have no per-hop minimums—large MEV/slippage risk.
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)
- **CWE**: CWE-840: Business Logic Errors
- **Mitigation**: Enforce per-hop and total minAmountOut across multi-hop routes; propagate constraints from Aggregator to path executors; fail-closed on any hop miss.
- **Status [29/09/2025]:** `Ferra` team has acknowledged this issue.

5.3 Aggregator: Weak address validation

- **Location**: *[redacted]*
- **Description:** Accepts non-hex and malformed addresses; name suggests it returns invalid but is used as "isValid." In clmm modules it's used for simulation account and sender checks.
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)
- **CWE:** CWE-20 (Improper Input Validation)
- **Mitigation**: Replace custom checks with Sui SDK validation (isValidSuiAddress). Enforce hex length and prefix constraints; reject mixed-case unless checksum is validated. Treat invalid addresses as hard errors with typed error codes. Centralize validation across modules (Aggregator + DLMM).
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.4 Aggregator: Wrong descending comparator

- **Location**: *[redacted]*
- **Description:** Incorrect/unstable ordering when sorting balances descending; could affect coin selection heuristics.
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)
- **CWE:** CWE-697 (Incorrect Comparison)
- **Mitigation**: For descending sorting, *sortByBalanceDes:* `(a > b) ? -1 : (a < b) ? 0 : 1` statement should be changed to `(a > b) ? -1 : (a < b) ? 1 : 0`.
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.5 Aggregator: Silent error on RPC calls

- **Location**: RPC Utils
- **Description:** Upstream callers can't distinguish real failures; undefined may propagate to transaction assembly causing runtime errors.
- **Risk:** Low (Impact: Low, Likelihood: Low)
- **CWE**: CWE-703 (Improper Check or Handling of Exceptional Conditions)
- **Mitigation**: Do not return undefined on failures. Log with correlation IDs; upstream must branch on success/failure. Add retries with backoff only for idempotent reads.
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.6 Aggregator: Gas estimation doesn't check for devInspect errors

- **Location**: RPC Utils
- **Description:** Uses `devResult.effects` without verifying `devResult.error`. If `devInspect` fails, accessing effects may throw or miscompute gas; leads to incorrect fee logic in callers.
- **Risk:** Low (Impact: Low, Likelihood: Low)
- **CWE**: CWE-252 (Unchecked Return Value)
- **Mitigation**: Guard `devResult` before use. Add tests for failed `devInspect` and ensure callers surface a user-readable reason.
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.7 CLMM: Inconsistent Basis Point Systems Across Modules

- **Location**: Multiple locations
- **Description:** The codebase evolved with different modules adopting different basis point conventions without establishing a unified standard.
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)
- **CWE**: CWE-840 (Business Logic Errors)
- **Mitigation**: Document the expected basis point system for each module's API. Create utility functions to convert between basis point systems. Long-term: Standardize on a single basis point system across all modules.
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.8 CLMM: Broken overflow "cap" + wrong constant leads to silent under/over-payment

- **Location**: Reward module
- **Description:** growthDelta is clamped with a hard-coded constant 340282366920938463463374594873840 4 (2^{128} − 1/10e5) and, when exceeded, the code sets the delta to 1. This arbitrary cap neither guarantees safety for all liquidity values nor correctness; in edge cases the downstream checkMulShiftRight(liquidity, growthDelta, 64, 128) can still overflow or require a revert.
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)
- **CWE**: CWE-190 (Integer Overflow or Wraparound)
- **Mitigation**: Replace the magic cap with a correct, derived bound: compute $U128\_MAX = 1 << 128 - 1$ and use a per-position cap $growthDeltaCap = floor(((U128\_MAX) << 64)/max(1, liquidity))$ then $growthDelta = min(growthDelta, growthDeltaCap)$
- **Status [29/09/2025]:** `Ferra` team has fixed this issue. The function is removed.

5.9 CLMM: Wrong pool selection when byAmountIn is false

- **Location**: Swap module
- **Description:** When trading by desired output (`byAmountIn=false`), code compares `amount_out` and picks the minimum; correct behavior is to minimize `amount_in` among non-exceed candidates that deliver the requested output. This picks suboptimal/expensive pools; can cause needless value loss.
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)
- **CWE**: CWE-840 (Business Logic Errors)

- **Mitigation**: For `byAmountIn=false`, choose the candidate with the smallest `amount_in` (subject to !is_exceed).
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.10 CLMM: All-candidates-exceeded path returns a bogus result

- **Location**: Swap module
- **Description:** If all events have `is_exceed=true`, `optimalPoolIndex` stays 0 but you still dereference index 0 and return a "result".
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)
- **CWE**: CWE-703 (Improper Handling of Exceptional Conditions)
- **Mitigation**: After the loop, if no candidate was selected, return null (or throw a typed error).
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.11 CLMM: Event - pool mapping relies on implicit order

- **Location**: *[redacted]*
- **Description:** *[redacted]* assumed to match *[redacted]*. Assumes dev-inspect events are returned in the exact call order.
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)
- **CWE**: CWE-758 (Reliance on Undefined/Unspecified Behavior)
- **Mitigation**: *[redacted]*.
- **Status [29/09/2025]:** `Ferra` team has acknowledged this issue.

5.12 CLMM: Unsanitized External API Data

- **Location**: Router module
- **Description:** No schema validation (e.g., poolInfo.address, fee). Directly inserts unvalidated fee into a Map key multiplied by 100; accepts arbitrary strings for coin addresses & pool IDs. Malicious/malfunctioning API could inject bogus pools/fees leading to mis-routing, DoS (graph bloat), or constructing transactions referencing attacker-chosen objects.
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)
- **CWE**: CWE-345 (Insufficient Verification of Data Authenticity)
- **Mitigation**: Add strict schema validation (numeric fee within known set; address regex; filter out duplicates; cap counts).
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.13 CLMM: Inconsistent Price Impact Calculation

- **Location**: Swap module
- **Description:** The formula $priceImpactPercentage = (|initial - final|/initial) * 100$ uses final marginal price, not the volume-weighted execution price. The change in spot (initial→final) measures how far the pool moved, not what the user actually received on average. If the trade crosses multiple ticks, the final spot may be far from the average fill; the real slippage could be smaller (or occasionally larger if curvature is nonlinear). Mixed interpretation: It conflates "pool price movement" with "price impact on execution," which are related but not identical. Does not account for direction (a2b vs b2a) relative to mid-price or expected execution price (because absolute is used for result); may misrepresent positive vs negative slippage; small division by zero risk if initialPrice = 0 (extreme edge).
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)
- **CWE**: CWE-840 (Business Logic Errors)
- **Mitigation**:
  - Return signed impact for direction; positive means you got worse than spot; negative means price improved;
  - UI can display both signed and unsigned. Use mid-price benchmark; handle zero guard.
- **Status [29/09/2025]:** `Ferra` team has fixed this issue using execution price for impact calculation.

5.14 DLMM: Price-impact math uses bin id as denominator

- **Location**: Swap module
- **Description:** Dividing a price delta by bin id (an index) is inaccurate. This results in wrong impact readouts and users may choose bad slippage; UX/auto-routing may misbehave.
- **Risk:** High (Impact: High, Likelihood: High)
- **CWE**: CWE-840 (Business Logic Errors)
- **Mitigation**: Replace denominator=bin index with denominator=current price; test across bin transitions and edge bins.
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.15 DLMM: Address Validation Logic Flaw

- **Location**: Utils
- **Description:** Name suggests it returns invalid but is used as "isValid."
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)

- **CWE**: CWE-20 (Improper Input Validation)
- **Mitigation**: Replace with proper validation using Sui SDK (isValidSuiAddress). Centralize validation across modules (Aggregator + DLMM).
- **Status [29/09/2025]:** `Ferra` team has fixed this issue.

5.16 DLMM: Unvalidated External API Data Consumption

- **Location**: *[redacted]*
- **Description:** Malicious data injection through compromised API responses.
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)
- **CWE**: CWE-345 (Insufficient Verification of Data Authenticity)
- **Mitigation**: *[redacted]*
- **Status [29/09/2025]:** `Ferra` team has acknowledged this issue.

5.17 Outdated Third-Party Dependencies

- **Location**: *[redacted]*
- **Description:** An analysis of the project's dependencies identified several third-party libraries that are either outdated.
- **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 acknowledged this issue.

A Appendix

Area	**Key checks (TypeScript SDK)**
Types & Schemas	Validate all external inputs (RPC/config/JSON); strict Zod/TypeBox; no unsafe coercion.
Math & Units	BigInt/BN; fixed-point rounding explicit; bps = 10^{-4}; zero/overflow guards.
AMM & Routing	`exactIn`/`exactOut`; per-hop & total `minOut`; consistent fee basis; price-impact = VWAP vs spot; deterministic tie-break.
Events & IDs	Explicit pool/correlation IDs; order-independent mapping; early reject invalid addresses.
RPC & Errors	Timeouts; bounded retries; cancellation; idempotency; typed errors/`Result`; never return `undefined`.
Determinism & Performance	Pure functions; input caps; bounded complexity; hot-path micro-benchmarks.
Packaging & Deps	Lockfile hygiene; SCA; no `postinstall`; correct `exports`/`types`; tree-shakable.
Docs & Tests	Public API documented; semver; unit + property/fuzz tests; CI across Node/Browser.

Table 4: SDK/library audit checklist assessed by `CYBRING`.