Defense By Thesis

Security Audit Report

Midl Execution Layer

Midl Runes Indexation

Initial Report // August 15, 2025
Final Report // September 17, 2025

Team Members

Ahmad Jawid Jamiulahmadi // Senior Security Auditor
Mukesh Jaiswal // Senior Security Auditor

Table of Contents

About Thesis Defense

Defense is the security auditing arm of Thesis, Inc., the venture studio behind tBTC, Fold, Mezo, Acre, Taho, Etcher, and Embody. At Defense, we fight for the integrity and empowerment of the individual by strengthening the security of emerging technologies to promote a decentralized future and user freedom. Defense is the leading Bitcoin applied cryptography and security auditing firm. Our team of security auditors have carried out hundreds of security audits for decentralized systems across a number of ecosystems including Bitcoin, Ethereum + EVMs, Stacks, Cosmos SDK, NEAR and more. We offer our services within a variety of technologies including smart contracts, bridges, cryptography, node implementations, wallets and browser extensions, and dApps.

Defense will employ the Defense Audit Approach and Audit Process to the in scope service. In the event that certain processes and methodologies are not applicable to the in scope services, we will indicate as such in individual audit or design review SOWs. In addition, Thesis Defense provides clear guidance on successful Security Audit Preparation.

Section 1.0 Scope

Technical Scope

- **Repository:** https://github.com/maestro-org/maestro-symphony
- **Audit Commit:** `6de7682102250ae236a42dd10e7d602d9c71fa78`
- **Verification Commit:** `569c900728863c9368a99f7af6bfce9f5dc78e8f`

Section 2.0 Executive Summary

Schedule

This security audit was conducted from July 28, 2025 to August 13, 2025 by 2 senior security auditors for a total of 5 person-weeks.

Overview

The Midl Runes Indexer works to maintain an accurate, queryable view of Rune activity on Bitcoin while coping with mempool churn and chain reorganizations. It ingests data from a Bitcoin Core node over two distinct interfaces, applies indexing logic inside a tightly controlled per-task execution unit, and persists results to RocksDB with precise rollback support. The design separates temporary mempool state from confirmed chain state, ensuring that downstream consumers can adopt clear finality policies without accidental leakage between views.

Data Sources & Trust Model

The indexer learns about the chain from a Bitcoin Core node in two ways. Over the P2P wire (a custom `peer_session`), it receives headers and blocks and gets woken up on new-block announcements. Over JSON-RPC (a `peer_rpc` client), it queries chain tips and requests a coherent mempool block template. This split gives high-throughput ingest on the data plane and a stable control/metadata plane for summaries that the wire protocol doesn't expose cleanly. The system treats the node as the authoritative view for practical purposes but assumes reorgs and adversarial conditions can occur, so it delays finalization behind a confirmation policy and makes rollbacks cheap.

Worker/stages and Scheduling

A staging worker runs an event loop: refresh the best known tip, decide what work to do next (confirmed block(s) or a mempool snapshot), and execute that work as an `IndexingTask`. The worker maintains a sliding window of recent chain points so it can quickly pivot during reorgs. New-block notifications from P2P act as low-latency triggers; the JSON-RPC tip is used to anchor the target point and to pull a consistent mempool template when the worker chooses to refresh the mempool view.

IndexingTask: The Core Execution Unit

Every block or mempool snapshot is processed inside an `IndexingTask`. When a task starts, it captures a consistent read view of the database (a snapshot at the last commit timestamp). All reads during the task first consult the task's `write_buffer` —an in-memory overlay that holds the task's pending mutations—and fall back to the snapshot only if the key has not been touched. This provides strict read-your-writes semantics.

The `write_buffer` is also the performance and correctness hinge: repeated writes to the same key are coalesced to the final state, and on the first modification of any key the task records that key's original value into an `originals` map. Those originals are later serialized into the `rollback_buffer` so the entire task can be undone exactly. When the task finishes, all buffered mutations are flushed as one atomic RocksDB write batch stamped with a new commit timestamp. Either the whole task lands or none of it does. This single-shot commit keeps interleavings clean and makes recovery logic straightforward.

Storage Layout & Rollback Buffer

State lives in RocksDB across typed tables with explicit key encodings and per-table prefixes (preventing cross-table collisions and enabling ordered iteration by design). Rollback information is written to a dedicated table keyed primarily by block height (with additional components to disambiguate entries). Because height is the first sort key and is encoded in an order-preserving way, the indexer can prune and traverse rollback records efficiently using range operations.

Reorgs and Precise Rollback

When the chain reorgs—or when a mempool snapshot must be discarded—the system replays previous values recorded for each modified key, restoring the database to its exact pre-task state. This is deterministic and bounded: the worker computes the rollback target point and scans rollback entries in height order back to that point, then proceeds along the new best chain. Because the `write_buffer` captured originals on first touch and the commit was atomic, there are no half-applied states to reconcile.

Garbage Collection Policy

To prevent unbounded growth, the rollback table for confirmed history is pruned after successful application of a block: any rollback records whose heights are strictly older than `current_height − max_rollback` are deleted via a height-bounded range delete. The end of the range is exclusive, so entries at exactly `current_height − max_rollback` remain. Saturating arithmetic on the bound avoids underflow at very low heights.

Mempool Handling (strict separation from confirmed state)

Mempool is treated as a temporary, discardable overlay. A mempool snapshot is applied via an `IndexingTask` just like a block, producing normal mutations and recording originals—but it is not finalized as chain history. When a new mempool snapshot is requested or a confirmed block arrives, the prior snapshot is rolled back immediately and its rollback entries are removed as part of that event-driven cleanup. The height-window GC discussed above applies only to confirmed history; mempool never relies on that path, which prevents temporary state from lingering.

Runes Parsing and Validation

During indexing, transactions are scanned for Runestone payloads using the external ordinals library. Valid runestones (etch/mint/transfer) drive indexing effects, while malformed or out-of-bounds payloads are classified as cenotaphs and recorded for provenance only—they do not execute edicts or change balances. This division of responsibility—protocol decoding/classification in the library, state mutation/rollback/orchestration in the indexer—keeps the indexer focused on storage and recovery while relying on a battle-tested parser for the on-chain encoding.

Integrity Properties and Operational Stance

The design provides atomicity (task-wide all-or-nothing commits), snapshot-consistent reads with read-your-writes, and exact rollback at task granularity (both for confirmed and mempool work). Confirmed and mempool views are strictly separated, which makes downstream finality policies simple to enforce. Operationally, the system benefits from multiple data sources (P2P for throughput, RPC for coherence), and it is resilient to ordinary reorgs thanks to the rollback buffer and GC window.

Threat Model

This threat model outlines the areas investigated during the security assessment of the Bitcoin Runes Indexer. The objective was to evaluate—including but not limited to—the correct handling of reorgs, mempool blocks, and Runes, while verifying secure-by-design principles and correct implementation.

For this security audit we assumed that the Bitcoin Core node that the Runes Indexer is receiving input data from is trusted. By design, the indexer can only receive inputs from the trusted node, significantly reducing the attack surface.

Testing and Security Analysis areas included:

P2P Ingestion (peer_session)

- Header links by previous hash, header ↔ block hash agreement, and how block requests are planned
- Locator construction/cursor window behavior and recovery from off-branch locators
- Handling of duplicate deliveries

RPC Usage (peer_rpc)

- Use of getchaintips/getblocktemplate for tip selection and mempool snapshots

IndexingTask (Per-Task Execution Unit)

- Snapshot isolation for reads; read-your-writes via the task's write buffer
- First-touch capture of original values to enable precise rollback
- Atomic single-batch commit semantics

Write Buffer Semantics

- Merging repeated updates and delete behavior within a task
- In-memory-first reads, snapshot fallback, and prevention of inconsistent intermediate views
- Interaction boundaries between multiple indexers within the same task

Storage Model (RocksDB)

- Table prefixing and order-preserving key encodings to prevent cross-table collisions
- Safety of multi_get result pairing
- Read options/snapshots and consistency guarantees across apply/rollback cycles

Rollback Buffer & Garbage Collection

- Composite key with height-first ordering; correctness of range deletes (end-exclusive)
- Sizing of max_rollback relative to confirmation policy
- Isolation of mempool rollback keys from confirmed GC paths to avoid premature deletion

Reorg Handling

- Deterministic rollback using recorded previous values to a specified point
- Replay on the chosen branch; behavior when reorg depth approaches or exceeds GC horizon

Mempool Handling

- Snapshot lifecycle: apply → discard/rollback on update or confirmed block arrival
- Event-driven cleanup of temporary writes and removal of associated rollback entries

Runes Parsing & Validation

- Delegation to the external ordinals parser (Runestone/edicts); rejection paths (e.g., cenotaphs)
- Malformed input handling and fuzz testing of encoding/decoding logic to prevent crashes and parsing anomalies
- Edge conditions around edict bounds, pointer/output indices, and transaction structure

Performance & Scalability Testing

- High-volume UTXO query performance under stress conditions with large UTXO sets (~26,000 entries)
- Parallel access performance during concurrent indexing and API queries
- Large runes balance handling and indexer capacity
- Header truncation vs cursor window interactions (e.g., truncation < window causing avoidable DB fallbacks)

Data Integrity & Consistency

- On-chain consistency verification by cross-checking balances and state data against blockchain data
- API parameter validation and appropriate error reporting for invalid inputs

Project Documentation

The project documentation provided for this review only included instructions for running the implementation and for interacting with the endpoints. The code has sufficient code comments.

Section 3.0
Key Findings Table

Issues	Severity	Status
ISSUE #1 Memory Leak Drains Memory (Known Issue)	Critical	Fixed
ISSUE #2 Single-Node Trust Risks Wrong-Chain Indexing	Medium	Acknowledged
ISSUE #3 Incorrect Data Returned for an Invalid Rune	Low	Acknowledged
ISSUE #4 Incorrect Example Route Defined for `rune_balance_at_utxo` Endpoint in OpenAPI Specification	None	Acknowledged
ISSUE #5 Header Truncation Below Cursor Window Causes Unnecessary DB Fallback & Lower Throughput	None	Acknowledged

Severity definitions can be found in Appendix A

Section 4.0
Findings

We describe the security issues identified during the security audit, along with their potential impact. We also note areas for improvement and optimizations in accordance with best practices. This includes recommendations to mitigate or remediate the issues we identify, in addition to their status before and after the fix verification.

ISSUE#1

Memory Leak Drains Memory (Known Issue)

Critical Fixed

Description

Rune and Bitcoin indexers can experience progressive memory consumption that could leads to out-of-memory kills, with processes consuming significant levels of RAM. In memory leak issues, allocated memory is not properly released after use, leading to progressive memory consumption that is not reclaimed by the garbage collector or the memory management system. Over time, this results in degraded application performance and potential system instability. The issue affects multiple indexer implementations and remains an active development challenge, with no definitive fixes available for the underlying memory management problems in Bitcoin meta-protocol indexing.

The Symphony/Midl team observed that the Indexer uses more than 20 GB of RAM, which would result in a critical issue for the Midl node that the Indexer is running on.

Impact

Memory lead would lead to progressive slowdown in indexing performance as available memory decreases. This would ultimately lead to the indexer crashing. In addition to the indexer being impacted, the resource exhaustion could also affect the functionality of the node that the indexer is running on.

Verification Status

The memory leak issue is addressed by switching to `jemalloc`, tightening RocksDB's memory limits, and improving cleanup (direct I/O with flush/compact), while also avoiding wasteful caching of ephemeral or unspendable UTXOs — together keeping RAM stable and releasing it reliably back to the OS.

During the verification, our team ran the Midl Runes Indexer and confirmed that the use of RAM is within expected levels during and after the sync to the tip of the BTC blockchain.`

ISSUE#2

Single-Node Trust Risks Wrong-Chain Indexing

Medium Acknowledged

Description

The indexer relies on a single Bitcoin Core node for headers/blocks and mempool data. If that node is compromised, or eclipsed, the indexer can ingest an incorrect chain view or a filtered mempool.

Impact

Incorrect Runes state, deep rollbacks during realignment, and dependent systems acting on incorrect data.

Recommendation

We recommend adopting a multi-node quorum: maintain `>= 3` independent nodes and require M-of-N agreement on tip (height, hash, chainwork) and a short window of `getblockhash(height)` behind tip before advancing. On disagreement, halt indexing and alert; resume only when quorum is restored.

ISSUE#3

Incorrect Data Returned for an Invalid Rune

Low Acknowledged

Location

src/serve/utils.rs#L57-L62

Description

The `RuneIdentifier::parse()` function is responsible for parsing both the Rune ID and Rune name. However, during the parsing of the Rune name, it removes all spacers without enforcing the rule that spacers are only valid when placed between two letters. This leads to incorrect parsing behavior where invalid Rune names are mistakenly treated as valid.

Impact

This issue affects the `rune_info_batch` endpoint. When an invalid Rune name is provided, the parser strips all spacers, potentially transforming the input into a valid Rune format. As a result, instead of returning null for an invalid Rune, the endpoint may return data for an unintended valid Rune from the indexer.

Example-

```
curl -X POST "http://localhost:8080/runes/info" \
    -H "Content-Type: application/json" \
    -d '["", "BESTINSLOT••XYZ"]' | jq .
```

In this case, the input `BESTINSLOT••XYZ` is invalid, but due to the removal of spacers, it resolves to a valid rune.

Recommendation

We recommend enforcing proper validation rules during parsing—specifically, ensuring that spacers are only allowed between two letters. This will prevent invalid rune names from being incorrectly interpreted as valid.

ISSUE#4

Incorrect Example Route Defined for `rune_balance_at_utxo` Endpoint in OpenAPI Specification

None Acknowledged

Location

src/serve/routes/runes/rune_balance_at_utxo.rs

Description

In the OpenAPI, the route for the `rune_balance_at_utxo` endpoint is mentioned as `path = /{rune}/utxos/{utxo}/balance` instead of `path=/runes/{rune}/utxos/{utxo}/balance`.

Due to this discrepancy, testing the endpoint as documented results in no data being returned from the indexer.

Impact

None

Recommendation

We recommend updating the OpenAPI specification to reflect the correct route.

ISSUE#5

Header Truncation Below Cursor Window Causes Unnecessary DB Fallback & Lower Throughput

None Acknowledged

Location

sync/stages/pull/stage.rs#L343 src/sync/stages/pull/stage.rs#L398 src/sync/stages/pull/stage.rs#L430-L461

Description

In the pull stage, in the `fetch_chain_actions` function, the worker truncates the received headers to 10 before building the recent-cursor window (target 50) and before the processing loop. As a result, the cursor builder always sees `<=10` in-memory headers and unnecessarily falls back to RocksDB to top up to 50, even when the peer actually returned `>= 50` headers. The processing loop then also caps each batch to 10 blocks purely due to this self-truncation.

Impact

- Unnecessary DB reads every cycle to fill the cursor.
- Lower throughput (more scheduler cycles to catch up).

Recommendation

We recommend setting the header fetch truncation to `>=` CURSOR_WINDOW (e.g., 50) so the cursor is built from the full in-memory header batch—triggering DB fallback only when the peer returns fewer than CURSOR_WINDOW and improving block-processing throughput.

Section 5.0
Appendix

Severity Rating Definitions

At Thesis Defense, we utilize the Immunefi Vulnerability Severity Classification System - v2.3.

Severity	Definition
Critical	• Execute arbitrary system commands • Retrieve sensitive data/files from a running server, such as: ∘ /etc/shadow ∘ database passwords ∘ blockchain keys (this does not include non-sensitive environment variables, open source code, or usernames) • Taking down the application/website • Taking down the NFT URI • Taking state-modifying authenticated actions (with or without blockchain state interaction) on behalf of other users without any interaction by that user, such as: • Changing registration information • Commenting • Voting • Making trades • Withdrawals, etc. • Changing the NFT metadata • Subdomain takeover with already-connected wallet interaction • Direct theft of user funds Malicious interactions with an already-connected wallet, such as: • Modifying transaction arguments or parameters • Substituting contract addresses • Submitting malicious transactions • Direct theft of user NFTs • Injection of malicious HTML or XSS through NFT metadata
High	• Injecting/modifying the static content on the target application without JavaScript (persistent), such as: ∘ HTML injection without JavaScript ∘ Replacing existing text with arbitrary text ∘ Arbitrary file uploads, etc. • Changing sensitive details of other users (including modifying browser local storage) without already-connected wallet interaction and with up to one click of user interaction, such as: ∘ Email or password of the victim, etc. ∘ Improperly disclosing confidential user information, such as: ▪ Email address ▪ Phone number ▪ Physical address, etc. ▪ PSubdomain takeover without already-connected wallet interaction

Severity	Definition
= Medium	• Changing non-sensitive details of other users (including modifying browser local storage) without already-connected wallet interaction and with up to one click of user interaction, such as: ◦ Changing the first/last name of user ◦ Enabling/disabling notifications ◦ Injecting/modifying the static content on the target application without JavaScript (reflected), such as: ▪ Reflected HTML injection ▪ Loading external site data ▪ Redirecting users to malicious websites (open redirect)
✓ Low	• Changing details of other users (including modifying browser local storage) without already-connected wallet interaction and with significant user interaction, such as: ◦ Iframing leading to modifying the backend/browser state (must demonstrate impact with PoC) ◦ Taking over broken or expired outgoing links, such as: ▪ Social media handles, etc ▪ Temporarily disabling user to access target site, such as: ▪ Locking up the victim from login ▪ Cookie bombing, etc.
» None	• We make note of issues of no severity that reflect best practice recommendations or opportunities for optimization, including, but not limited to, gas optimization, the divergence from standard coding practices, code readability issues, the incorrect use of dependencies, insufficient test coverage, or the absence of documentation or code comments.

Section 6.0
Appendix B

Thesis Defense Disclaimer

Thesis Defense conducts its security audits and other services provided based on agreed-upon and specific scopes of work (SOWs) with our Customers. The analysis provided in our reports is based solely on the information available and the state of the systems at the time of review. While Thesis Defense strives to provide thorough and accurate analysis, our reports do not constitute a guarantee of the project’s security and should not be interpreted as assurances of error-free or risk-free project operations. It is imperative to acknowledge that all technological evaluations are inherently subject to risks and uncertainties due to the emergent nature of cryptographic technologies.

Our reports are not intended to be utilized as financial, investment, legal, tax, or regulatory advice, nor should they be perceived as an endorsement of any particular technology or project. No third party should rely on these reports for the purpose of making investment decisions or consider them as a guarantee of project security.

Links to external websites and references to third-party information within our reports are provided solely for the user’s convenience. Thesis Defense does not control, endorse, or assume responsibility for the content or privacy practices of any linked external sites. Users should exercise caution and independently verify any information obtained from third-party sources.

The contents of our reports, including methodologies, data analysis, and conclusions, are the proprietary intellectual property of Thesis Defense and are provided exclusively for the specified use of our Customers. Unauthorized disclosure, reproduction, or distribution of this material is strictly prohibited unless explicitly authorized by Thesis Defense. Thesis Defense does not assume any obligation to update the information contained within our reports post-publication, nor do we owe a duty to any third party by virtue of making these analyses available.