SOVA LABS SECURITY AUDIT

zokyo

July 10th 2025 | v. 1.0

Technical Summary

Our team audited the Sova blockchain implementation, focusing primarily on the `sova-evm` and `payload` crates, which govern Bitcoin finality enforcement, custom pre-compile logic, and the EVM execution lifecycle. The audit emphasized the mechanisms through which Bitcoin metadata is ingested, validated, and used to influence storage writes and transaction broadcasts within the EVM.

We examined the Sova node's interaction with the Bitcoin network via Sentinel and its fallback behavior under RPC failures. The BroadcastTransaction and setBitcoinBlockData precompiles were analyzed for correctness, determinism, and replay protection. We confirmed that Bitcoin-related state changes are intercepted and conditionally committed based on slot lock status as reported by Sentinel.

The inspector logic was deeply reviewed to ensure it correctly enforces slot-level locking and reverts on violation. We identified critical flow checkpoints that are conditionally set, potential misinterpretations of `Status::Unknown`, and missing guardrails that could lead to user-facing issues under network stress or Sentinel unavailability. Lock scoping was found to be global rather than user-specific, raising DoS and fairness concerns. The payload-building and simulation phases were reviewed for determinism and correctness in multi-validator environments.

While the `UBTC_CONTRACT_CODE` was technically out of scope, we partially referenced it to assess how applications invoke Bitcoin pre-compiles. Byte-code analysis suggests basic checks are present, but formal replay prevention remains unclear in the absence of full source code.

Overall, no critical exploits were found, but several architectural and operational risks were identified, along with opportunities to improve observability, robustness, and developer ergonomics.

In total, fourteen findings were raised: three high, five medium, three low, and three informational. Our analysis confirms that while core components operate securely, critical mechanisms related to cross-chain consistency, lock enforcement, and transaction broadcast handling present edge cases that require additional robustness and safeguards.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code was taken from the Sova Labs repository:
Repo: https://github.com/SovaNetwork/sova-reth
Initial commit - 7c8e9c7180af1c10b4a52207b3c69053db7d2d9c
PR with last fixes - https://github.com/SovaNetwork/sova-reth/pull/111

Codebase under the scope: Files in:

- crates/payload
- crates/sova-evm

Throughout the review process, care was taken to ensure that the codebase:

- Ensuring protocol-level logic aligns with intended execution semantics.
- Verifying deterministic behavior across transaction lifecycle and EVM state transitions.
- Identifying potential edge cases, misconfigurations, or unsafe design patterns.
- Confirming adherence to Rust best practices, memory safety, and modular isolation.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Sova Labs codebase. To do so, the code was reviewed line by line by our developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review.

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contract/s by industry leaders.
03	Thorough manual review of the codebase line by line.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

Each issue is also assigned a **severity level**, which reflects the potential impact on the **Sova Network's protocol logic, execution integrity, or infrastructure safety**. Severity is determined based on the likelihood and consequences of misuse, misbehavior, or failure at the protocol level:

Critical

The issue affects the codebase in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the codebase to compile or operate in a significant way.

Medium

The issue affects the ability of the codebase to operate in a way that doesn't significantly hinder its behavior.

Low

The issue has minimal impact on the codebase's ability to operate.

Informational

The issue has no impact on the codebase's ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Incomplete Handling of `Status::Unknown` in Lock Enforcement	High	Resolved
2	Global Lock Scoping Introduces Revert and DoS Risks	High	Partially Resolved
3	Inconsistent Bitcoin Finality Signaling	High	Resolved
4	No Consensus on Bitcoin Block Data	Medium	Invalid
5	Incomplete Replay Protection for Bitcoin Broadcasts	Medium	Resolved
6	Non-Deterministic Checkpoint Initialization in initialize_interp	Medium	Resolved
7	Storage Locks May Be Skipped on Early Exit	Medium	Resolved
8	Slice Boundary Violation in TXID Extraction	Medium	Resolved
9	Limited Observability of Lock Decisions	Low	Resolved
10	Code Duplication in Inspector Logic	Low	Acknowledged
11	Gas Accounting Overflow Risk	Low	Resolved
12	Lock Expiration Mechanism Not Explicit	Informational	Resolved
13	Typo in a warning message	Informational	Resolved
14	Typo in a crate alias	Informational	Resolved

Incomplete Handling of `Status::Unknown` in Lock Enforcement

When the Sentinel returns `Status::Unknown`, the system triggers a revert without restoring the checkpoint or clearing internal caches. This abrupt behavior risks blocking legitimate user operations caused by transient Sentinel issues, like RPC downtime or latency.

Recommendation:

Introduce a retry mechanism before reverting on `Status::Unknown`. Differentiate treatment of unknown status depending on the operation type. For Bitcoin-related actions like `BroadcastTransaction`, maintain strict reverts; for generic storage updates, log a warning and proceed to avoid false reverts.

Fixed: https://github.com/SovaNetwork/sova-reth/pull/104 & https://github.com/SovaNetwork/sova-reth/pull/111

Zokyo team's comment: we have additional comment here, if performance permits, consider a retry mechanism with exponential backoff before reverting on `Timeout` or `ServiceUnavailable` to tolerate brief Sentinel/RPC disruptions.

Client's response: As of right now we will not add that feature, but would like to keep it in mind for the future.

Global Lock Scoping Introduces Revert and DoS Risks

Locks are applied globally across all users and transactions. This creates risk of legitimate users being reverted due to unrelated prior access to the same slots by others. Attackers could abuse this to trigger deliberate reverts by preemptively accessing shared storage.

Recommendation:

Scope locks to user, session, or transaction context. Include sender identity or originator data when acquiring locks to prevent cross-user conflict and reduce attack surface for DoS.

Fixed: https://github.com/SovaNetwork/sova-reth/pull/100

Zokyo team's comment: this is a valuable mitigation, and we consider it sufficient for this audit cycle. In future versions, you may consider scoping locks per address or operation, rather than solely filtering on the contract level. This would enable finer-grained lock management, especially in multi-actor environments.

Client's response: Agreed future focus will be on finer-grain lock management by all node operators. This will also help us enable cool multi-signer implementations for the network signing mechanics

Inconsistent Bitcoin Finality Signaling

When the Sova node fails to obtain block metadata from the Bitcoin client, the fallback logic injects a default-constructed `SovaL1BlockInfo` struct with all-zero values. This data is then embedded in an EVM transaction via the `setBitcoinBlockData` precompile, which may be interpreted by downstream contracts as valid Bitcoin finality data. This creates the illusion that a valid Bitcoin block was observed, potentially causing downstream contracts to incorrectly assume finality.

Example:
If the Bitcoin node returns an error or times out, the injected block metadata has `_blockHeight = 0` and `_blockHash = 0x000...`,
yet still appears in the EVM state as if valid.

Recommendation:
Suppress the generation of `setBitcoinBlockData` transactions entirely if data from the Bitcoin client is unavailable or malformed. Log the failure, but do not commit state based on synthetic values. Only insert cross-chain metadata when successfully fetched and verified.

Fixed: https://github.com/SovaNetwork/sova-reth/pull/109

No Consensus on Bitcoin Block Data

Every Sova validator talks to its own Bitcoin node to fetch the latest block height and hash, which is then pushed into the EVM state through a pre-compile call. But there's no system in place to make sure all validators agree on what Bitcoin block data to use. This means different validators could end up using slightly different Bitcoin info when building a block - depending on network delays, RPC issues, or even misconfigured nodes.

Example:
If two validators build blocks using different Bitcoin data, it could lead to subtle differences in what they think the chain state should be. That kind of inconsistency weakens trust in the protocol and can cause forks or failures in critical smart contracts that rely on that data.

Recommendation:
Instead of each validator acting alone, the Bitcoin block data should be treated as part of the block's consensus. For example:
- Have the block producer sign the Bitcoin data as part of the payload.
- Let other validators double-check this data before accepting the block.
- Consider fetching Bitcoin data at a fixed time window tied to the chain's progression to ensure consistency.

Client's comment: The auditors' concern assumes a multi-validator model, but Sova uses Optimism's single sequencer architecture where only the sequencer fetches Bitcoin data during block production. The Bitcoin data is embedded as a system transaction in the canonical block content, not fetched independently by validators. When validators replay blocks, they execute the exact same Bitcoin data transaction that the sequencer included, ensuring all validators reach identical state deterministically. This eliminates the consistency problem the auditors identified since validators don't make independent Bitcoin RPC calls during block validation. The OP Stack's dispute resolution mechanism provides consensus safety - if a sequencer includes incorrect Bitcoin data, validators can challenge the block via fault proofs with economic slashing incentives. This design achieves stronger consensus properties than traditional multi-validator agreement because Bitcoin data becomes cryptographically committed canonical block content rather than externally-fetched data. The sentinel locks and state updates are applied consistently across all validators because they process identical transaction data during deterministic replay.

Incomplete Replay Protection for Bitcoin Broadcasts

Sova includes a pre-compile that lets the uBTC contract send raw Bitcoin transactions. While only the uBTC contract is allowed to use this function, there's nothing stopping it from sending the same transaction data multiple times.
Even if the Bitcoin network rejects repeated broadcasts, Sova still treats each call as a successful action.

Why this matters:
- Smart contracts might react to each call thinking a new Bitcoin transaction was sent.
- Logs could be emitted multiple times, confusing off-chain systems like indexers or bots.
- There's no way to tell from inside Sova whether the same request was already handled.
- It adds noise and weakens confidence that contract actions reflect real-world outcomes.

Example:
A contract sends a valid broadcast call, and everything looks fine. Later, the same call is sent again with the same data. Sova processes it normally, even though the Bitcoin node ignores it. To a smart contract or block explorer, it looks like a second broadcast succeeded.

Recommendation:
Add replay protection in the uBTC contract:
- Track past broadcast requests and reject duplicates.
- Use nonces or timestamps to make each request unique.
- Prevent contracts from resubmitting identical data.

This will help ensure that every broadcast processed by Sova actually reflects a real Bitcoin transaction attempt.

Client's comment: This is handled in the network's native BTC wrapper predeploy contract `usedTxids` mapping. https://github.com/SovaNetwork/contracts/blob/98d9ac1308ee8efd1b428988f0e02d027c22a5d0/src/SovaBTC.sol#L35

Non-Deterministic Checkpoint Initialization in initialize_interp

The `initialize_interp` function resets the checkpoint only if it is currently `None`. This creates a non-deterministic state during nested or repeated executions and can lead to cases where rollback becomes impossible, triggering warnings like “No checkpoint available for reversion”.

Recommendation:
Always reset the checkpoint and clear the internal cache upon entering `initialize_interp` to ensure clean transaction boundaries and consistent rollback behavior.

Fixed: https://github.com/SovaNetwork/sova-reth/commit/3388e7a530689e5217de7db464469a3b51575e78

Storage Locks May Be Skipped on Early Exit

If execution halts before reaching the `BroadcastTransaction` precompile, any storage slots accessed beforehand will neither be locked nor released. This could result in incorrect assumptions about lock state and potential state inconsistencies across users.

Recommendation:
Ensure that accessed slot data is handled consistently even if a transaction exits prematurely. A mechanism should be introduced to either register partial locks or discard changes in a traceable manner.

Client's comment: This behavior is intentional and correct by design. The storage lock mechanism is specifically engineered to protect Bitcoin finality, not to track general storage access patterns. Storage accesses are only locked when they lead to actual Bitcoin broadcast operations because if no Bitcoin transaction is created, no Bitcoin finality protection is needed. The EVM's native revert mechanism already handles storage state consistency for failed transactions. This design ensures the lock system is precisely scoped to protect cross-chain Bitcoin operations rather than imposing unnecessary restrictions on general EVM storage access.

Fixed: https://github.com/SovaNetwork/sova-reth/pull/107

MEDIUM-5 | RESOLVED

Slice Boundary Violation in TXID Extraction

In `handle_cache_btc_data`, the system extracts the first *32* bytes from `outcome.result.output[..32]`` without checking the length. This can panic at runtime if output is too short, potentially breaking block production or validator consensus.

Recommendation:
Validate `outcome.result.output.len()` before slicing, and return a structured error if the length is insufficient. Avoid panics in execution-critical code.

Fixed: https://github.com/SovaNetwork/sova-reth/pull/108

LOW-1 | RESOLVED

Limited Observability of Lock Decisions

Current implementation lacks visibility into per-slot locking decisions, making it difficult to trace lock events or debug unintended reverts.

Recommendation:
Add structured logging for each slot decision, including transaction context and timestamps. Consider exposing Sentinel lock status via an EVM view call to support contract-level and off-chain diagnostics.

Fixed: https://github.com/SovaNetwork/sova-reth/pull/104

Code Duplication in Inspector Logic

Location:

- sova-evm/src/inspector/mod.rs lines 236-252 and 428-444
- sova-evm/src/execute.rs lines 220-247 and 354-381
- sova-evm/src/execute.rs lines 251-281 and 385-415
- sova-evm/src/execute.rs lines 321-337 and 458-474

Several logic blocks across inspector components, particularly in lock checking, sentinel status handling, and outcome reversion, are duplicated rather than encapsulated in reusable helper functions. This pattern reduces maintainability and risks introducing inconsistent behavior when one instance is modified and others are not.

Recommendation:

Refactor repeated logic into well-defined internal methods. This includes sentinel status checks, revert handling, and cache manipulation. Centralizing this logic will improve auditability and reduce potential future errors during feature changes or debugging.

Client's comment:

We are going to double check all of this code for discrepancies in the redundant code. It is a big change for use to modify this in the current state of the codebase. There are upstream changes to reth (v1.5.0) which will require us to completely rebuild our execution flows and where/how the slot revert application phase happens as well as the slot locking phases.

LOW-3 | RESOLVED

Gas Accounting Overflow Risk

The custom gas calculation logic for Bitcoin pre-compiles uses native `u64` arithmetic without overflow checks. While EVM `input` sizes are capped under current protocol rules, this could lead to silent arithmetic overflow if reused in different contexts or under relaxed gas limits.

Example:
Gas is computed as `base_cost + (input_len * cost_per_byte)`, which may overflow if `input_len` is large (e.g. u64::MAX / 2).

Recommendation:
Use `checked_add` and `checked_mul`, or saturating arithmetic to ensure resilience.

Fixed: https://github.com/SovaNetwork/sova-reth/pull/106

INFORMATIONAL-1 | RESOLVED

Lock Expiration Mechanism Not Explicites

Slot locks in the Sentinel do expire after a set number of Bitcoin blocks, but this behavior is not clearly documented and may be misunderstood by integrators.

Recommendation:
Document the TTL policy clearly and consider exposing the expiration time or block count in Sentinel responses or EVM-accessible metadata for full transparency.

Fixed: https://github.com/SovaNetwork/sova-reth/pull/106

INFORMATIONAL-2 | RESOLVED

Typo in a warning message

File: `crates/payload/src/builder.rs` @217
Message: "No sequencer txs received"

Recommendation:
Fix to: "No sequencer txs received"

Fixed: https://github.com/SovaNetwork/sova-reth/pull/9

INFORMATIONAL-3 | RESOLVED

Typo in a crate alias

File: `crates/sova-evm/src/precompiles/mod.rs` @13
Alias: ReqwestClient

Recommendation:
Rename to `RequestClient` for clarity.

Fixed: https://github.com/SovaNetwork/sova-reth/pull/106

We are grateful for the opportunity to work with the Sova Labs team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Sova Labs team put in place a bug bounty program to encourage further analysis of the codebase by third parties.

zokyo