Security Audit Report

Volta Soroban Smart Contract

v1.0

November 19, 2025

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Stellar Development Foundation to perform a security audit of Volta Soroban.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/VoltaHQ/smart-contract-soroban
Commit	`0d7f14c6bc3eb341fc623a2cd851e63a3f20f119`
Scope	All contracts were in scope.
Fixes verified at commit	`042692c483dccffef7f49f1962cd4f126af15304` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes, such as additional features, have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The Volta Soroban contract implements threshold-governed multisig control with rule-based call authorization.

Owners can propose configuration changes, grant or revoke per-user rules, execute calls, or upgrade the contract.

Any user with rules may execute cross-contract calls if all rules in their set match the target function, contract address, and argument values via typed comparators; non-owners without matching rules are rejected.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium-High	The client provided detailed documentation and diagrams.
Test coverage	Medium-High	Test coverage reported by `cargo tarpaulin` is `75%`

Summary of Findings

No	Description	Severity	Status
1	Premature proposal rejection due to incorrect threshold comparison logic	Major	Resolved
2	Lack of pre-validation for non-config proposals allows creating invalid or no-op proposals	Minor	Resolved
3	Proposal TTL mechanism may lead to unintended data loss for long-duration proposals	Minor	Acknowledged
4	TTL threshold calculation causes underflow	Minor	Resolved
5	Improper handling of nested `Result` in `execute_invoke` leads to false positive success states	Informational	Acknowledged
6	Multiple errors are declared but never used	Informational	Resolved
7	Inefficient owner verification using linear search increases gas cost	Informational	Resolved
8	Potentially confusing comparison semantics in rule evaluation require clear documentation	Informational	Acknowledged
9	Configuration change invalidates all pending proposals without explicit rejection	Informational	Acknowledged
10	Incorrect error variant returned for unauthorized caller during proposal interaction	Informational	Resolved

Detailed Findings

1. Premature proposal rejection due to incorrect threshold comparison logic

Severity: Major

In `contracts/volta/src/types.rs:149-156`, the logic determining a proposal's status contains two design flaws that compromise the integrity of the governance process.

- **Premature rejection**

  Proposals are rejected as soon as the number of rejected votes meets the configured threshold, without accounting for remaining uncast votes. This allows premature rejection even when sufficient remaining voters could still reach the approval threshold.

  For instance, with a threshold of 2 and five total voters, if two voters reject early, the proposal is immediately marked as rejected, even though the three remaining voters could still approve it. This behavior prevents full voter participation and may lead to proposals being rejected that would otherwise have passed.

- **Misinterpretation of abstention votes**

  Second, the logic incorrectly treats abstain votes as rejections, triggering proposal failure when the abstain count reaches the threshold. This misrepresents voter intent, as abstentions typically signify neutrality or disengagement rather than opposition.

Recommendation

We recommend modifying the proposal evaluation logic to reject only when it becomes mathematically impossible to achieve the approval threshold, given the remaining votes.

Atain votes should be treated as neutral and excluded from rejection conditions.

Status: Resolved

2. Lack of pre-validation for non-config proposals allows creating invalid or no-op proposals

Severity: Minor

The `propose` function in `contracts/volta/src/lib.rs:43-71` handles multiple proposal types but only performs comprehensive validation for configuration proposals.

While `Config` proposals undergo thorough checks, including validation and change detection, other proposal types, like user rules modifications, rule revocations, and contract upgrades are only validated at the type system level without verifying their semantic correctness or practical effect.

This validation gap permits the creation of proposals that are either guaranteed to fail during execution or constitute no-ops, such as proposing empty user rule sets, revoking rules from users with no existing rules, or upgrading to identical contract code.

Although these proposals will eventually fail during execution, they unnecessarily consume storage and clutter the proposal ecosystem, potentially confusing users and wasting computational resources during proposal evaluation.

Recommendation

We recommend implementing comprehensive pre-validation for all proposal types, similar to the existing config validation, including verifying that user rule proposals contain non-empty and semantically valid rulesets, ensuring revocation proposals target users with existing rules, and checking that upgrade proposals reference distinct and available contract code.

Status: Resolved

3. Proposal TTL mechanism may lead to unintended data loss for long-duration proposals

Severity: Minor

In `contracts/volta/src/types.rs:184-191`, a fixed one-week TTL (Time To Live) is set for proposals using temporary storage.

While this approach efficiently manages storage by automatically cleaning up inactive proposals, it creates a potential issue where proposals with voting periods longer than one week could be prematurely deleted if no voting activity occurs within any consecutive week-long period. The TTL extension only triggers when there's storage activity (like voting), meaning proposals that remain inactive for a full week would be silently removed regardless of their actual deadline or remaining voting time.

Recommendation

We recommend aligning the TTL duration with the actual proposal deadline or implementing a mechanism that automatically extends TTL based on the remaining voting period rather than relying solely on activity-based extensions.

Status: Acknowledged

4. TTL threshold calculation causes underflow

Severity: Minor

In `contracts/volta/src/types.rs:309-319`, the threshold for extending time-to-live (TTL) is computed by subtracting fixed constants from `max_ttl`.

On the Soroban platform, `max_ttl` is determined from the ledger header and is typically limited to approximately six months on public networks. The current logic subtracts a full year (`YEAR_OF_LEDGERS`) from this value, which causes an underflow on `u32` and results in a runtime panic due to overflow checks.

Since `extend_ttl` only triggers when the current TTL is below this threshold, no extension occurs, allowing data, such as user rules or configurations, to expire silently. This behavior can disrupt contract operations, cause unexpected state loss, or crash function calls that rely on persistent entries.

Recommendation

We recommend dynamically computing the buffer based on the current maximum TTL.

Status: Resolved

5. Improper handling of nested `Result` in `execute_invoke` leads to false positive success states

Severity: Informational

In `contracts/volta/src/lib.rs:227-246`, the `execute_invoke` function calls `try_invoke_contract::<Val, InvokeError>`, which returns a nested `Result` type.

The outer `Err` indicates a failure to invoke the callee, while the inner `Err` denotes a failure to convert the returned value into the expected type `T` (via `TryFromVal`).

However, the `execute_invoke` function incorrectly handles this structure by only checking the outer `Err`. When the inner branch (`Ok(Err(...))`) occurs, the function silently returns `Ok(())`, thereby suppressing conversion failures.

This results in false-positive success states, where a contract call appears successful despite internal conversion errors. Such behavior can break logical invariants, mask runtime bugs, and cause proposals or contract executions to appear complete when they are not.

Recommendation

We recommend explicitly handling the `Ok(Err(...))` case by mapping it to a domain-specific error.

Status: Acknowledged

The client acknowledges the issue, stating that it is an intentional design choice. The potential conversion would only occur when falling back to a generic `Val`, however that does not imply that the invocation of the other contract failed since the outer `Err` handler covers that case. Given that the vault is a generic multisig vault with user-defined rules, the type of the returned value is not required to be known here. Representing a success in this manner is therefore considered the correct handling and aligns with the intended semantics of the proposal, regardless of the specific return type.

6. Multiple errors are declared but never used

Severity: Informational

The contract implements a `ContractError` enum with supported error messages in `contracts/volta/src/types.rs:500`.

However, only some of them are currently used in the business logic.

The `AlreadyInitialized`, `InvalidConfig`, `NotPreApproved`, `ConversionError`, and `InvokeNotAllowed` are included in an enum, but are never called by any of the functions. They are redundant.

Recommendation

We recommend removing the redundant errors or reconsidering their usage as part of the business logic.

Status: Resolved

7. Inefficient owner verification using linear search increases gas cost

Severity: Informational

In `contracts/volta/src/types.rs:57-59`, the contract verifies ownership by scanning a vector of owners sequentially. This approach performs a linear search (`O(n)` complexity) for every ownership check, iterating through each entry until a match is found.

While this method is acceptable for small owner sets, it becomes increasingly inefficient as the number of owners grows, leading to unnecessary computation and higher gas consumption on each verification. Given that the configuration structure is queried frequently, the cumulative overhead can have a noticeable impact on performance and execution costs, especially in governance workflows where multiple ownership validations occur per transaction.

Recommendation

We recommend replacing the vector-based storage with a set data structure that provides `O(1)` lookup complexity, such as using Soroban's map or set types.

Status: Resolved

8. Potentially confusing comparison semantics in rule evaluation require clear documentation

Severity: Informational

The `compare_numbers` function in `contracts/volta/src/types.rs:419-431` implements comparison logic where the parameter ordering could lead to different interpretations of rule semantics. The current implementation compares `other_value OP rule_value`, which means for a rule configuration like `{field: arg0, comparator: Gt, value: 100}`, it evaluates as "`arg0 > 100`" (transaction argument must be greater than 100).

While this appears to be the intended behavior for access control rules, the parameter naming and comparison direction could be misinterpreted during code maintenance or rule definition. The ambiguity lies in whether rules should be read as *"field OPERATOR value"* or *"value OPERATOR field"* in natural language terms.

Recommendation

We recommend adding clear documentation to the rule system that explicitly defines the semantic interpretation of comparison rules. The documentation should specify that rules follow the pattern *"field OPERATOR value"* (e.g., "`arg0 > 100`" means the argument must be greater than 100).

Status: Acknowledged

9. Configuration change invalidates all pending proposals without explicit rejection

Severity: Informational

In `contracts/volta/src/lib.rs:152-175`, the `execute_config` function intentionally invalidates all proposals created prior to a configuration change.

When executed, it records the “last config change” as the next proposal ID. Subsequent proposal loads use a guard condition that rejects any proposal whose ID is less than this marker.

This mechanism ensures that outdated proposals cannot execute under obsolete owners or threshold parameters, effectively enforcing state consistency.

However, it also renders all pre-change proposals inaccessible, resulting in a confusing user experience. Functions like `vote` and `revoke_proposal` return generic errors, and no event

is emitted to clarify the invalidation. As a result, users may interpret the behavior as a system malfunction rather than an intentional invalidation.

Recommendation

We recommend explicitly handling invalidated proposals by transitioning them to a `Rejected` state when detected, emitting a corresponding event to communicate the reason.

Status: Acknowledged

10. Incorrect error variant returned for unauthorized caller during proposal interaction

Severity: Informational

In `contracts/volta/src/lib.rs:161-163`, the ownership validation incorrectly returns the error `ContractError::ConfigChangeSinceProposal` when the caller is not part of the current owner list. This error message is misleading, as the failure is caused by an unauthorized caller rather than a configuration change.

Using the wrong error variant obscures the true reason for the failure, weakening access control clarity and potentially confusing both developers and users.

Recommendation

We recommend replacing the returned error with a dedicated authorization error.

Status: Resolved