RPine

Staking Contract Security Assessment

Date: May 26, 2025
Confidential: Yes
Prepared by: Victor Mustatea

About RPine

Founded in 2021, RPine is a group of developers formed on March 29, 2021. Initially, our main focus was exploring Ethereum Layer 2 solutions, with our initial projects being Arbitrum Nitro contracts. Over time, we advanced from integrating various exchange APIs to building on-chain infrastructure. Today, our goal is to keep pushing to enhance our skills, both as part of our organization and individually, while also exploring new opportunities and seeking our assistance.

You can learn more about our team and our work at rpine.com/about.

Our complete portfolio of all our publications can be found in the following repository: rpine.github.io/publications, where you will see links to articles, presentations, and open-source reports.

Our group specializes in developing, testing, and auditing smart contracts. We have been actively participating in the following DeFi protocols: on Ethereum and its Layer 2 solutions, Polkadot, and Cosmos ecosystems. To contact us directly, please email us at contact@rpine.com, or alternatively, you can find us on Telegram at @victor_rpine.

Notices and Remarks

Copyright and Distribution
© 2025 RPine

All rights reserved.

This report is prepared by RPine to be public information. Information is given free to interested parties, but it may not be redistributed in any form. Any opinion or advice set out in this document is given to the client alone and shall not be relied upon by any other party or in proof of the purpose whatsoever contained in this document.

Table of Contents

1. Executive Summary
2. Project Summary
3. Project Goals
4. Project Targets
5. Project Coverage
6. Main Components
7. Staking Process
8. Interaction Scheme
9. Analysis of Compliance with Project Description
10. Smart Contract Wrappers
11. Test Coverage
12. Identified Vulnerabilities
13. Conclusion

Executive Summary

RPine has contracted a time-boxed audit of the Staking contract implemented for the Ethereum Pectra upgrade and alignment with the intended staking logic. The key objectives included identifying potential vulnerabilities, ensuring adherence to best practices, and verifying compliance with the functionality requirements stated in the project documentation.

Overall, the Staking contract's reflect the features described in the design document, with custom handling of slashing, locking periods, inflation calculations, and validator management. The contract leverages Ethereum's consensus checks for timestamp and maximum staking limits, and implements strict access controls and safeguards against common DeFi attack vectors. The SFC Block Reward Wrapper contract provides a clear interface for reward distribution and integration with Ethereum's consensus layer.

Project Summary

- Project Name: Staking
- Repository: Staking Bridge
- Commit Hash: 1d0d0d0d0d0d
- Documentation: Sonic SFC - Staking Logic

Key Participants

- Kamil Ali
- Victor Mustatea (Victor Mustatea)

Timeline

- Audit began: March 14, 2025
- Audit ended: April 28, 2025
- Final report completed: April 30, 2025

- Integration testing: April 28, 2025
- Fixing identified issues: April 29, 2025
- Completion of fixes and testing of identified issues: May 12, 2025
- Revised report completed: May 26, 2025

Project Goals

1. Identify Vulnerabilities

Verify whether attack potential can be expected for unauthorized withdrawals, incorrect validator calculations, or other security issues.

2. Assess Compliance

Ensure the on-chain logic (staking.js, v1.sol, base.js) matches the intended staking mechanics, including proper validator lifecycle management, slashing, and rewards.

3. Evaluate Code Quality

Assess the contract's adherence to Sonic and Ethereum 2 standards, including proper error handling, event emissions, and overall code maintainability.

Project Targets

Staking Collection (staking.js)

- Deployed by an admin and establishes initial parameters (genesis validators, epoch, timestamp).
- Registers a different validator with a unique public key and manages their lifecycle (active, exited, slashed).
- Calculates interest based on locking period (30, 90, 180, or 360 days).
- Manages time-based reward accumulation.
- Handles unlocking requests and distributes principal plus rewards.

Staking Item (v1_base.js)

- Acts as a non-transferable Soul Bound Token (SBT).
- Stores validator data, including public key, genesis timestamp, interest rate, and slashing status.
- Implements slashing, locking, and unlocking logic.
- Prevents transfers between users.
- Tracks accumulated rewards and pending slashes.
- Said delivers after successful unlocking.

Project Coverage

During this audit, we examined:

1. Smart Contract Logic

Analyzed critical functions in staking.js and v1_base.js, focusing on validator management, reward calculations, slashing mechanisms, and locking periods.

2. Compliance with Docs

Verified implementation against documentation, with the official documentation being the primary source of truth.

3. Wrapper Functions

Reviewed the Sonic SFC Block Reward Wrapper contract for proper integration with the staking system.

4. Test Coverage

Evaluated the provided test suite's coverage, checking and prioritizing scenarios for different time periods.

5. Edge Cases

Identified potential edge cases, boundary conditions, and their handling, including leap years and maximum staking limits.

Main Components

Staking Contract (staking.js)

- Inherited:
  - Admin hub as the reward pool with initial balance.

- Staking Operations:
  - Validator registrations for staking.
  - Calculates interest based on locking period.
  - Manages time-based reward accumulation.
  - Handles slashing events.
  - Processes unlocking requests.
  - Distributes principal plus rewards.
  - Verifies ownership and status.
  - Ensures proper state transitions.
  - Destroys the NFT upon unlocking.

- Utility Methods:
  - Time-based calculations - Returns current staking parameters.
  - get_validator_status() - Returns current validator status.
  - get_validator_data() - Returns validator information.
  - get_all_validators() - Returns paginated NFT listings by index.

NFT Item Contract (nft_item.fc)

- Attributes:
  - Soul bound, private, time, interest, and slash.
  - Implements NFT functionality.
  - Non-transferable.
  - Verifies the current time is past unlock time.
  - Calculates total rewards.
  - Sends rewards plus principal to the staking contract.

- NFT Methods:
  - get_nft_data() - Returns NFT data.
  - get_static_data() - Returns ownership parameters.
  - get_authority_address() - Returns NFT authority (zero).
  - get_revoked_time() - Returns revocation time (zero).

- Security Measures:
  - Prevents transfer operations.
  - Prices destruction only for the staking contract.

Staking Process

1. Admin Setup

- Admin deploys the staking contract with initial parameters.
- Sets genesis validators and initial system parameters.

2. User Staking

- User transfers ashrim to the staking contract.
- Specifies validator public key and locking period.

- Contract checks on NFT account with staking details.
- Calculates interest based on locking period.

3. Staking Lockup Period

- NFT is non-transferable (Soul Bound Token).
- Tracks accumulated rewards over time.

4. Unstaking

- After the unlock time passes, user can request unstaking.
- Contract verifies ownership and unlock eligibility.
- Calculating contract sends principal plus rewards.
- NFT is destroyed upon successful unstaking.

5. Admin Operations

- Admin can top up the reward pool.
- Admin can set active staking parameters.
- Admin can withdraw from the reward pool (minimum of remaining).

Interaction Scheme

The staking system operates with the following flow:

1. Initial Transfers

- Users send ashrim to the staking contract.
- Specifies validator public key and locking period.

2. NFT Issuance

- Upon successful validation, an NFT is issued as a receipt.
- The NFT contains all staking parameters.

- Verifies owner when processing unstake requests.
- Ensures proper state transitions and ownership.
- Rewards protecting the staking pool's reward pool.
- Will destroyed after successful unstaking.
- Implements comprehensive error handling for composability.

Conclusion

The Staking contract's complex fully complies with the documentation's intended mechanics, implementing proper validator lifecycle management, slashing, and reward calculations.

Smart Contract Wrappers

Staking Contract

Implemented Wrapper Methods:

- getValidatorMetadata() - Gets the admin wallet address.
- getValidatorPublicKey() - Retrieves validator public key.
- getRRidaBalance(uint) - Gets NFT address by index.
- getClaimableTotal() - Gets collection data.
- getValidatorStatus(address) - Gets validator status address.
- isValidator() - Deprecated, always true.
- isAdmin(address) - Checks if address is admin.
- getAllValidators() - Gets paginated NFT listings by index.
- getRRidaPublicKey() - Gets rewards parameters.
- getRRidaCount() - Gets NFT count.

NFT Item Contract

Implemented Wrapper Methods:

- getNftData() - Gets NFT data.
- getStaticData() - Gets ownership parameters.
- getAuthorityAddress() - Gets revocation time (zero).
- getRevokedTime() - Gets NFT authority (zero).

Missing Wrapper Methods:

- Stake staking interface (owner, unlock time, interest rate, slash).
- Implements SBT functionality (non-transferable).

- Verifies owner when processing unstake requests.
- Ensures proper state transitions and ownership.
- Rewards protecting the staking pool's reward pool.
- Will destroyed after successful unstaking.
- Implements comprehensive error handling for composability.

Conclusion

The wrapper contracts fully complies with the documented logic in the technical documentation, with proper validator lifecycle management, slashing, and reward calculations.

Test Coverage

Staking Contract

Tests for Administration:

- Set its setting up admin parameters at deployment
- Allow admin to top up rewards
- Set its retrieving validity parameters

Tests for Users:

- Allow staking ashrim
- Set its staking tokens for 30 days
- Set its staking tokens for 180 days
- Set its staking tokens for 360 days
- Set its unstaking after 30 days
- Set its unstaking after 180 days
- Set its unstaking after 360 days
- Set its retrieving status and NFT information

Negative Tests:

- Set its attempting to unstake before unlock time
- Set its attempting to stake ashrim for 90 days
- Set its insufficient TON for token return
- Set its insufficient ashrim for staking
- Set its insufficient rewards in pool
- Set its attempting to unstake by non-owner
- Set its attempting to unstake twice
- Set its unknown operation
- Set its attempting to top up by non-admin
- Set its attempting double unstaking
- Set its attempting to top up zero
- Set its attempting to stake more than available rewards

Identified Vulnerabilities

During the integration testing of version 1.contract (1.fift), the following vulnerabilities were identified:

1. Lack of Token Return When Reward Pool Is Empty

Description: The staking contract's logic lacks a mechanism to return tokens to users when the reward pool was exhausted. Specifically, when a user attempts to unstake, the contract sends their initial stake and rewards to the reward pool.

Solution: Implemented token return logic in case of insufficient rewards, ensuring users receive their initial stake.

2. Denial of Service Through Gas Exhaustion

Description: At the end of staking, there was a possibility to drain the TON balance of the contract by sending many small stakes, causing the contract to exceed gas limits.

Impact: An attacker could cause a denial of service for the contract by forcing it to exceed gas limits during reward calculations.

Solution: Implemented gas optimization and suggested TON and a fee multiplier on token transfers, preventing balance depletion.

3. Insufficient Validation of Unstaking Conditions

Description: Transactions from the frontend were sent with a TON value of 0.05 TON, which was insufficient for the contract's requirements.

Impact: Low stakes were bounced without the possibility of return, causing potential loss of funds for users.

Solution: Implemented validation of the minimum TON amount required for operations, ensuring proper handling of low-value transactions.

4. Excessive Complexity in Calculation Logic

Description: The original implementation contained complex and error-prone logic for computing the end of staking rewards.

Impact: The complexity of logic increased the probability of errors and made code auditing and maintenance difficult.

Solution: Simplified the reward logic for calculating the end of staking rewards, improving code maintainability and reliability.

Note: All the above vulnerabilities have been fixed in version 1.contract (1.fift).

Conclusion

The Staking smart contracts fully complies with the documentation's intended mechanics, implementing proper validator lifecycle management, slashing, and reward calculations, and parameter verifications are properly implemented. The contracts demonstrate robust error handling, appropriate gas management, and comprehensive event logging.

During the audit, vulnerabilities were conducted in fact the identified issues were primarily related to edge cases in the staking mechanism, including token return when the reward pool is empty, gas exhaustion through many small stakes, and insufficient validation of unstaking conditions.

Key findings from this audit:

- Proper validator lifecycle management across different staking periods with appropriate reward rates
- Comprehensive parameter verification prevents invalid stakes
- The NFT implementation ensures soul bound tokens
- Gas optimization:
  - Efficient storage patterns for validator data
  - Interaction
  - Computations kept under reasonable limits

The contracts are ready for production use, but as with any blockchain system, ongoing monitoring and maintenance are recommended. Regular security audits and bug bounties should be considered for ongoing security.

Contacts

Tg: @rpine
Email: rpine.xyz@gmail.com