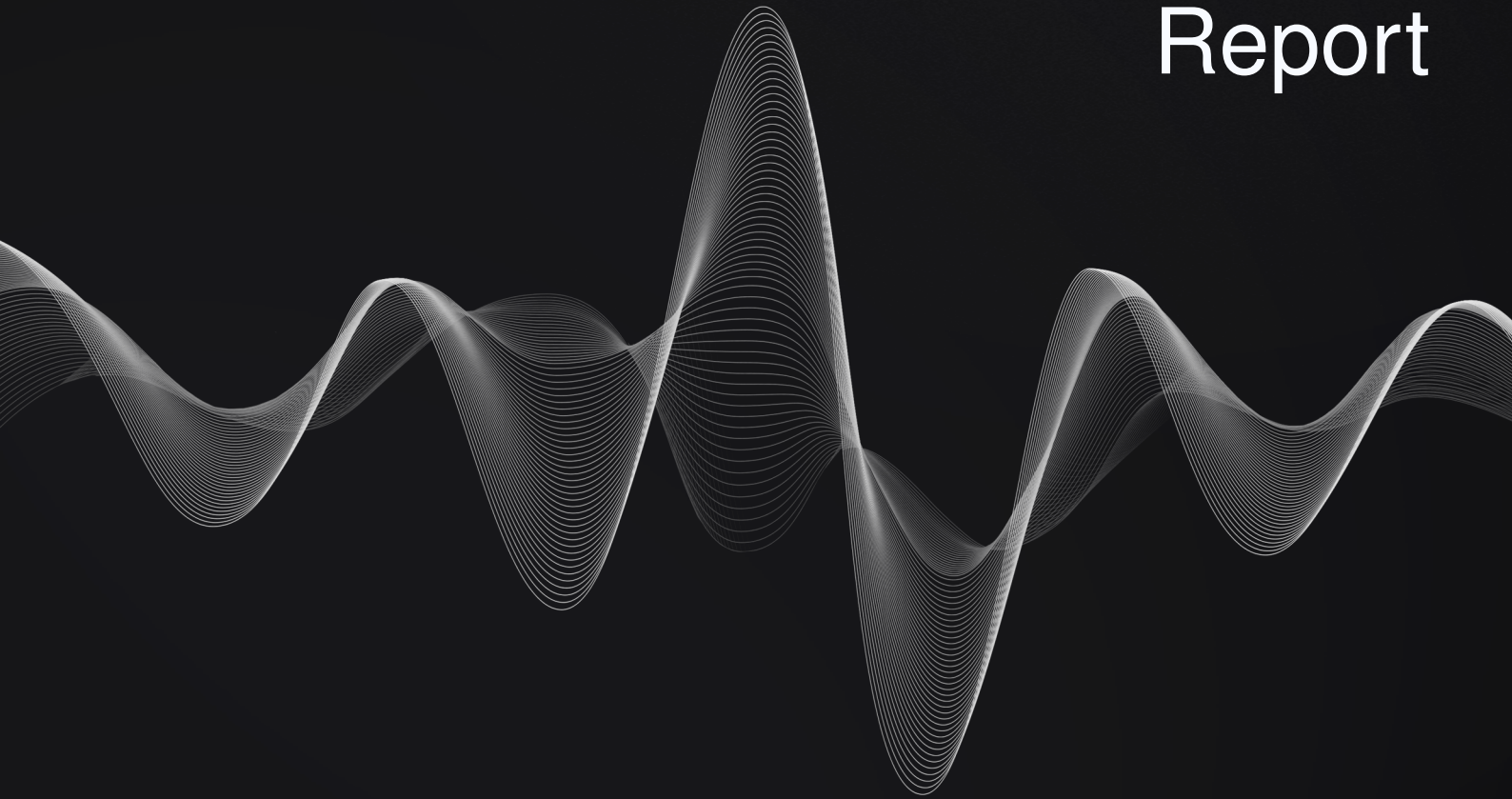




Rhea Finance

Liquid Staking Protocol Smart Contract Audit Report






Document Control

PUBLIC

FINAL(v2.1)

Audit_Report_RHEA-LSP_FINAL_21

Sep 1, 2025		v0.1	João Simões: Initial draft
Sep 4, 2025		v0.2	João Simões: Added findings
Sep 8, 2025		v1.0	Charles Dray: Approved
Oct 7, 2025		v1.1	João Simões: Reviewed findings
Oct 9, 2025		v2.0	Charles Dray: Finalized
Oct 9, 2025		v2.1	Charles Dray: Published

Points of Contact	Marco Sun	Rhea Finance	marco@ref.finance
	Charles Dray	Resonance	charles@resonance.security
Testing Team	João Simões	Resonance	joao@resonance.security
	Ilan Abitbol	Resonance	ilan@resonance.security
	Luis Arroyo	Resonance	luis.arroyo@resonance.security

Copyright and Disclaimer

© 2025 Resonance Security, Inc. All rights reserved.

The information in this report is considered confidential and proprietary by Resonance and is licensed to the recipient solely under the terms of the project statement of work. Reproduction or distribution, in whole or in part, is strictly prohibited without the express written permission of Resonance.

All activities performed by Resonance in connection with this project were carried out in accordance with the project statement of work and agreed-upon project plan. It's important to note that security assessments are time-limited and may depend on information provided by the client, its affiliates, or partners. As such, the findings documented in this report should not be considered a comprehensive list of all security issues, flaws, or defects in the target system or codebase.

Furthermore, it is hereby assumed that all of the risks in electing not to remedy the security issues identified henceforth are sole responsibility of the respective client. The acknowledgement and understanding of the risks which may arise due to failure to remedy the described security issues, waives and releases any claims against Resonance, now known or hereafter known, on account of damage or financial loss.

Contents

1 Document Control	2
Copyright and Disclaimer	2
2 Executive Summary	4
System Overview	4
Repository Coverage and Quality.....	4
3 Target	6
4 Methodology	7
Severity Rating.....	8
Repository Coverage and Quality Rating.....	9
5 Findings	10
Storage Staking Mechanism Ignores accounts Collection.....	11
Owner Transfer Does Not Revoke Super Admin Privileges	12
Delayed Withdrawal Of Non-staked Amounts	13
Unused Functions.....	14
Missing Usage Of NEAR SDK Integer Types For Input And Output	15
Owner Use Of Contract Funds On set_beneficiary()	16
A Proof of Concepts	17

Executive Summary

Rhea Finance contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between August 25th, 2025 and September 8th, 2025. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 3 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 10 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide Rhea Finance with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.



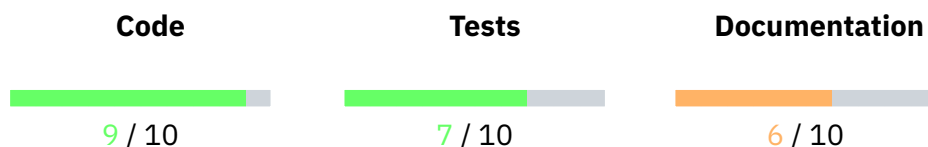
System Overview

Rhea Finance is a liquid staking solution for NEAR, designed to let users stake \$NEAR and receive \$rNEAR in return directly in their own wallets. By holding \$rNEAR, users not only earn staking rewards from but also gain opportunities to leverage across DeFi ecosystems on their native chains. Unstaking can be done instantly with a fee through Rhea DEX, or traditionally with a four-epoch waiting period.

Behind the scenes, Rhea Finance's protocol maintains a weighted validator pool, balancing stakes across validators based on performance, stability, reputation, and decentralization goals. Operator bots automate staking and syncing of rewards, ensuring that \$rNEAR's value relative to \$NEAR grows consistently as validator rewards are added.



Repository Coverage and Quality



Resonance's testing team has assessed the Code, Tests, and Documentation coverage and quality of the system and achieved the following results:

- The code follows development best practices and makes use of known patterns, standard libraries, and language guides. It is easily readable and uses the latest stable version of relevant components. Overall, **code quality is excellent**.

- Unit and integration tests are included. The tests cover both technical and functional requirements. Code coverage is undetermined. Overall, **tests coverage and quality is good.**
- The documentation includes the specification of the system, some technical details for the code, some relevant explanations of workflows and interactions. Overall, **documentation coverage and quality is average.**

Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: [ref-finance/rnear-contract/contracts](#)
- Hash: 341cd8c970895e1b6bcb433699b20253922d621d

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- Financial related attacks

Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

Source Code Review - Rust NEAR

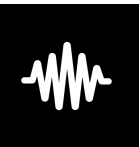
During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Rust NEAR audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Race conditions caused by asynchronous cross-contract calls
- Frontrunning attacks
- Storage staking
- Potentially problematic storage layout patterns
- Manual state rollbacks in callbacks
- Access control issues

- Denial of service
- Inaccurate business logic implementations
- Unoptimized Gas usage
- Arithmetic issues
- Client code interfacing



Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

		Likelihood		
		Very Likely	Likely	Unlikely
Impact	Strong	Critical	High	Medium
	Medium	High	Medium	Low
	Weak	Medium	Low	Info



Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.

Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

- **"Quick Win"** Requires little work for a high impact on risk reduction.
-|.. **"Standard Fix"** Requires an average amount of work to fully reduce the risk.
- ...||| **"Heavy Project"** Requires extensive work for a low impact on risk reduction.

RES-01	Storage Staking Mechanism Ignores accounts Collection	Acknowledged
RES-02	Owner Transfer Does Not Revoke Super Admin Privileges	Resolved
RES-03	Delayed Withdrawal Of Non-staked Amounts	...	Acknowledged
RES-04	Unused Functions	Resolved
RES-05	Missing Usage Of NEAR SDK Integer Types For Input And Output	Acknowledged
RES-06	Owner Use Of Contract Funds On set_beneficiary()	Acknowledged



Storage Staking Mechanism Ignores accounts Collection

Medium RES-RHEA-LSP01

Architecture

Acknowledged

Code Section

- [lst/src/storage.rs](#)

Description

The storage management solution implemented by the protocol only accounts for the protocol's fungible token storage requirements, however, the `IterableMap` `accounts` present in the contract's state is also controlled and used by users of the protocol. Storage staking requirements for this collection should also be enforced for all users, to prevent the smart contract account from paying for storage used and managed by its users.

Recommendation

It is recommended to include the storage usage requirements of the `accounts` collection in the implemented storage management mechanism. This is already measured in the variable `account_storage_usage`, which should be added as a requirement for storage staking.

Status

The issue was acknowledged by Rhea Finance's team. The development team stated "It's designed to be like this. The goal is to provide the same storage experience as a regular FT in the NEAR network. The running party of this protocol knows and accepts the fact that they need cover some extra storage cost."



Owner Transfer Does Not Revoke Super Admin Privileges

Medium RES-RHEA-LSP02

Business Logic

Resolved

Code Section

- [lst/src/owner.rs#L6-L16](#)

Description

The protocol implements multiple roles using `near-plugins` to be used across the platform for different users. There is also the implementation of the `owner` of the protocol, which, in turn, is initialized to maintain the role of super admin.

When the owner of the protocol is changed through the use of the function `set_owner()`, the previous owner maintains the role of super admin within the smart contract. This allows the previous owner to assign themselves new roles and potentially take over important functionalities, or ultimately take control of the entire protocol.

Recommendation

It is recommended to ensure that super admin, as well as other already assigned roles, be unassigned from the owner when it is being changed.

Status

The issue has been fixed in [2d0dd20f320af4f41f51de7d64a31dded8a3cde7](#).



Delayed Withdrawal Of Non-staked Amounts

Medium RES-RHEA-LSP03

Business Logic

Acknowledged

Code Section

- [lst/src/account.rs#L82-L89](#)
- [lst/src/account.rs#L107](#)

Description

The protocol implements staking and unstaking mechanisms for user deposits. It is possible for a user that just deposited funds into the protocol, to immediately withdraw them provided that they have not been staked in the meantime. However, when a user stakes the deposit, they need to first unstake and then wait 4 epochs before being able to withdraw the funds.

An edge case exists in the following scenario:

1. User deposits 100 NEAR at epoch 1.
2. User stakes 100 NEAR at epoch 2.
3. User unstakes 100 NEAR at epoch 10.
4. User deposits 20 NEAR at epoch 11.
5. User attempts to withdraw 20 NEAR at epoch 12, but is denied.
6. User can only withdraw the 20 NEAR (or even the full 120 NEAR) at epoch 15.

This scenario exists because a single variable containing the epoch for the next possible withdrawal is used, `account.unstaked_available_epoch_height`.

Recommendation

It is recommended to either maintain a collection that assigns a withdrawal timeline for each deposited amount, or ensure all deposited amounts have a delayed withdrawal, whether they have been previously staked or not.

Status

The issue was acknowledged by Rhea Finance's team. The development team stated "It's to keep the same behavior as a regular staking pool."



Unused Functions

Info

RES-RHEA-LSP04

Code Quality

Resolved

Code Section

- [lst/src/lib.rs#L314-L319](#)
- [lst/src/utils.rs#L73-L83](#)

Description

The following functions and modifiers were found to be unused within the system:

- `per_account_storage_cost()`
- `RewardFeeFraction::assert_valid()`
- `RewardFeeFraction::multiply()`

Unused functions increase the complexity and readability of the smart contract's code and their inclusion should be discouraged whenever possible.

Recommendation

It is recommended to remove unused functionalities from production-ready code.

Status

The issue has been fixed in 2d0dd20f320af4f41f51de7d64a31dded8a3cde7.



Missing Usage Of NEAR SDK Integer Types For Input And Output

Info

RES-RHEA-LSP05

Code Quality

Acknowledged

Code Section

- [lst/src/stake_pool_itf.rs#L77](#)
- [lst/src/stake_pool_itf.rs#L82](#)

Description

The functions `get_number_of_accounts()` and `get_accounts()` make use of input parameters of type `u64`. This type is longer than 52 bits and, as such, is not serializable by JSON, therefore making it impossible to retrieve a properly decoded value when calling this function through an external integrated interface.

Recommendation

It is recommended to make use of NEAR SDK capitalized types in favor of the `i64-i128/u64-u128` native types when dealing with input and output parameters:

- `u64` - `U64`
- `u128` - `U128`
- `i64` - `I64`
- `i128` - `I128`

Status

The issue was acknowledged by Rhea Finance's team. The development team stated "It's to keep the same behavior as a regular staking pool."



Owner Use Of Contract Funds On `set_beneficiary()`

Info

RES-RHEA-LSP06

Business Logic

Acknowledged

Code Section

- [lst/src/owner.rs#L20-L45](#)

Description

The smart contract allows for a owner to be specified during initialization and to execute administrative functions that deal with funds, e.g. the function `set_beneficiary()`. Due to this fact, and the fact that an owner can be set to a different `AccountId` than the one of the deployer, a malicious or compromised owner may execute functions while the payment of the storage is made from the smart contract's account, i.e. the deployer.

It should be noted that the severity of this finding has been decreased due to the fact that the function `set_beneficiary()` can only be used to increase the collection for up to 10 entries as per the constant `MAX_BENEFICIARIES = 10`.

Recommendation

It is recommended to either ensure the owner of the smart contract is also the deployer during initialization, or the owner is enforced to also use implement storage staking mechanisms to pay for the additional storage they use, much like any other user.

Status

The issue was acknowledged by Rhea Finance's team.

Proof of Concepts

No Proof-of-Concept was deemed relevant to describe findings in this engagement.