SECURITY AUDIT REPORT

Nira

DATE

23 April 2025

PREPARED BY

0xTeam.

WEB3 AUDITS

info@0xTeam.Space

0xTeamSpace

0xTeamSpace

Contents

Revision History & Version Control

Version	Date	Description
1.0	09 April 2025	Initial Audit Report
2.0	14 April 2025	Second Audit Report
3.0	23 April 2025	Final Audit Report

0xTeam conducted a comprehensive Security Audit on the Nira to ensure the overall code quality, security, and correctness. The review focused on ensuring that the code functions as intended, identifying potential vulnerabilities, and safeguarding the integrity of Nira's operations against possible attacks.

Report Structure

The report is divided into two primary sections:

1. **Executive Summary** : Provides a high-level overview of the audit findings.
2. **Technical Analysis** : Offers a detailed examination of the Smart contracts code.

Note :
The analysis is static and exclusively focused on the Smart contracts code. The information provided in this report should be utilised to understand the security, quality, and expected behaviour of the code.

1.0 Disclaimer

This is a summary of our audit findings based on our analysis, following industry best practices as of the date of this report.However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. The audit focuses on Smart contracts coding practices and any issues found in the code, as detailed in this report. For a complete understanding of our analysis, you should read the full report. We have made every effort to conduct a thorough analysis, but it's important to note that you should not rely solely on this report and cannot make claims against us based on its contents. We strongly advise you to perform your own independent checks before making any decisions. Please read the disclaimer below for more information.

DISCLAIMER: By reading this report, you agree to the terms outlined in this disclaimer. If you do not agree, please stop reading immediately and delete any copies you have. This report is for informational purposes only and does not constitute investment advice. You should not rely on the report or its content, and 0xTeam and its affiliates (including all associated companies, employees, and representatives) are not responsible for any reliance on this report.The report is provided "as is" without any guarantees. 0xTeam excludes all warranties, conditions, or terms, including those implied by law, regarding quality, fitness for a purpose, and use of reasonable care. Except where prohibited by law, 0xTeam is not liable for any type of loss or damage, including direct, indirect, special, or consequential damages, arising from the use or inability to use this report. The findings are solely based on the Smart contracts code provided to us.

2.0 Executive Summary

2.1 Overview

0xTeam has meticulously audited the Nira Smart contracts project from 14 March 2025 to 27 March 2025. The primary objective of this audit was to assess the security, functionality, and reliability of the Nira's before their deployment on the blockchain. The audit focused on identifying potential vulnerabilities, evaluating the contract's adherence to best practices, and providing recommendations to mitigate any identified risks. The comprehensive analysis conducted during this period ensures that the Nira is robust and secure, offering a reliable environment for its users.

2.2 Scope

The scope of this audit involved a thorough analysis of the Nira Smart contracts, focusing on evaluating its quality, rigorously assessing its security, and carefully verifying the correctness of the code to ensure it functions as intended without any vulnerabilities.

Files in Examination:

Language	Rust
In-Scope	• src/rewards.rs • src/staking.rs • src/state.rs • src/admin.rs • src/initialize.rs
Fixed Review Commit Hash	b0237416b01174940e2744710fcd01b6db5a48ab

OUT-OF-SCOPE: External Smart contracts code, other imported code.

2.3 Audit Summary

Name	Verified	Audited	Vulnerabilities
Nira	Yes	Yes	Refer Section 5.0

2.4 Summary of Findings

ID	Title	Severity	Fixed
[H-01]	Reward Inflation Due to Missing Cap on Emissions	HIGH	✔
[M-01]	Missing Reentrancy Guard on Stake/Unstake Functions	MEDIUM	✔
[M-02]	Incorrect Reward Calculation on Early Unstake	MEDIUM	✔
[L-01]	Inefficient Storage Layout for Farmer Accounts	LOW	✖

[L-02]	Admin Privileges Are Not Time-Locked	LOW	✔
[I-01]	Redundant Initialization Check	INFO	✖

2.5 Vulnerability Summary

High	Medium	Low	Informational
1	2	2	1

High Medium Low Informational

2.6 Recommendation Summary

Severity

Issues

	High	Medium	Low	Informational
Open	1	2	2	1
Resolved	1	2	1	
Acknowledged			1	1
Partially Resolved				

- **Open:** Unresolved security vulnerabilities requiring resolution.
- **Resolved:** Previously identified vulnerabilities that have been fixed.
- **Acknowledged:** Identified vulnerabilities noted but not yet resolved.
- **Partially Resolved:** Risks mitigated but not fully resolved.

3.0 Checked Vulnerabilities

We examined Smart contracts for widely recognized and specific vulnerabilities. Below are some of the common vulnerabilities considered.

Category	Check Items
Source Code Review	➔ Reentrancy Vulnerabilities ➔ Ownership Control ➔ Time-Based Dependencies ➔ Gas Usage in Loops ➔ Transaction Sequence Dependencies ➔ Style Guide Compliance ➔ EIP Standard Compliance ➔ External Call Verification ➔ Mathematical Checks ➔ Type Safety ➔ Visibility Settings ➔ Deployment Accuracy ➔ Repository Consistency
Functional Testing	➔ Business Logic Validation ➔ Feature Verification ➔ Access Control and Authorization ➔ Escrow Security ➔ Token Supply Management ➔ Asset Protection ➔ User Balance Integrity ➔ Data Reliability ➔ Emergency Shutdown Mechanism

4.0 Techniques , Methods & Tools Used

The following techniques, methods, and tools were used to review all the smart contracts

- **Structural Analysis:**
  This involves examining the overall design and module architecture of the Rust smart contracts. We ensure that the contracts are logically organized, modular, scalable, and adhere to Rust's resource-oriented programming principles. Special focus is given to critical aspects such as ownership models, resource management, access control, and upgradeability patterns. This step is crucial for identifying structural weaknesses that could lead to vulnerabilities or future maintenance challenges.

- **Static Analysis:**
  Static analysis is performed using automated tools to scan the Rust codebase for common security vulnerabilities without executing the code. Issues such as improper resource handling, access control flaws, integer overflows/underflows (despite Rust's safe arithmetic), and unexpected aborts are checked. Static analysis helps detect potential problems at an early stage, allowing for prompt remediation.

- **Code Review / Manual Analysis:**
  An in-depth manual review of the Rust smart contracts is conducted to ensure the logic correctly implements the intended protocol behavior as described in the project documentation. During this phase, we verify the security properties, business logic correctness, error handling, and integration of on-chain assets. Manual review also helps in validating findings from static analysis and uncovering any hidden risks not detectable by automated tools.

- **Dynamic Analysis:**
  Dynamic analysis involves deploying and interacting with the Rust smart contracts in controlled test environments such as localnet or devnet. Various scenarios are simulated to observe contract behavior under different conditions. This step includes running comprehensive test cases, transaction simulations, property-based tests, and gas profiling to ensure that the contracts are efficient, resilient, and secure during real-world operations.

- **Tools and Platforms Used for Audit:**
  Cargo , Aderyn, Rust Analyzer ,Rustile , Rust CLI Tests,Fuzz & Custom Scripts .
  .

Note: The following values for "Severity" mean:

- High: Direct and severe impact on the funds or the main functionality of the protocol.
- Medium: Indirect impact on the funds or the protocol's functionality.
- Low: Minimal impact on the funds or the protocol's main functionality.
- Informational: Suggestions related to good coding practices and gas efficiency.

5.0 Technical Analysis

HIGH

Issue#1
[H-1] Reward Inflation Due to Missing Cap on Emissions

Severity
HIGH

Location
`src/rewards.rs` - function `distribute_rewards()`

Issue Description
The farming smart contract does not set an upper limit on how many rewards can be distributed over time. Without a cap, if the reward rate is accidentally or maliciously set too high, users can mint an excessive amount of tokens, drastically inflating the token supply. This can lead to massive devaluation of the token, exploitation of liquidity pools, and system-wide instability.

Recommendation
Introduce a maximum reward cap per pool. Ensure that total minted rewards do not exceed the expected maximum allocation even if farming parameters are manipulated.

```
if total_rewards_minted + pending_rewards > max_total_rewards {
    return Err(ErrorCode::RewardCapExceeded.into());
}
```

Status
Resolved

Medium

Issue#2
[M-1] Missing Reentrancy Guard on Stake/Unstake Functions

Severity
Medium

Location
`src/staking.rs - functions stake(), unstake()`

Issue Description
The `stake()` and `unstake()` functions allow external calls during critical operations. Without reentrancy protection, an attacker could manipulate the staking balance mid-transaction, causing inconsistencies, double counting, or double rewards. Reentrancy attacks are a major known threat (e.g., DAO Hack on Ethereum).

Recommendation
Implement a reentrancy guard mechanism by locking function entry and unlocking upon function exit.

Status
Resolved

Issue#3

[M-2] Incorrect Reward Calculation on Early Unstake

Severity

Medium

Location

`src/rewards.rs - function calculate_pending_rewards()`

Issue Description

Currently, the reward claim logic does not properly handle users who exit before the reward vesting period is complete. Users can claim rewards proportional to full staking time even though they staked only for a partial duration, allowing over-claiming.

Recommendation

Calculate pending rewards based on the effective staked time, not based on total expected yield.

```
let time_staked = current_time - user_stake_start_time;
let effective_reward = (time_staked as u128 * reward_rate) /
full_vesting_duration;
```

Status

Resolved

Low

Issue#4
[L-1] Inefficient Storage Layout for Farmer Accounts

Severity
Low

Location
`src/state.rs - struct FarmerAccount`

Issue Description
The `FarmerAccount` structure is not optimized for storage layout. Mixed types cause extra padding (e.g., mixing u8, u64, and u128), which unnecessarily increases storage costs on blockchains like Solana or Substrate.

Recommendation
Group fields with similar sizes together to reduce memory padding.

```
pub struct FarmerAccount {
    pub staked_amount: u64,
    pub last_claimed: u64,
    pub pending_rewards: u128,
}
```

Status
Acknowledged

Issue#5
[L-2] Admin Privileges Are Not Time-Locked

Severity
Low

Location
`src/admin.rs - functions update_reward_rate(), change_pool_weight()`

Issue Description
Currently, the contract admin can instantly change important parameters like reward rates, pool weights, or emission speeds. Instant changes can create unfair situations where users are affected without any warning.

Recommendation
Add a delay mechanism for critical admin changes.

```
pub struct PendingAdminChange {
    pub new_admin: Pubkey,
    pub effective_after: u64, // block time
}

if clock.unix_timestamp < pending_change.effective_after {
    return Err(ErrorCode::ChangeNotEffectiveYet.into());
}
```

Status
Resolved

Informational

Issue#6
[I-1] Redundant Initialization Check

Severity
INFO

Location
`src/initialize.rs - function initialize_contract()`

Issue Description

The `initialize()` function unnecessarily checks `msg.sender` or `env::caller()` even though it's guaranteed by blockchain runtime (e.g., contract creator at instantiation).

Recommendation
Simplify initialization code to remove redundant validations, reducing gas and improving readability.

Status
Acknowledged

6.0 Auditing Approach and Methodologies Applied

The smart contract was audited using a comprehensive approach to ensure the highest level of security and reliability. Careful attention was given to the following key areas to ensure the overall quality of the code:

- **Code quality and structure**: We conducted a detailed review of the codebase to identify any potential issues related to code structure, readability, and maintainability. This included analysing the overall architecture of the smart contract and reviewing the code to ensure it follows best practices and coding standards.
- **Security vulnerabilities**: Our team used manual techniques to identify any potential security vulnerabilities that could be exploited by attackers. This involved a thorough analysis of the code to identify any potential weaknesses, such as buffer overflows, injection vulnerabilities, signatures, and deprecated functions.
- **Documentation and comments**: Our team reviewed the code documentation and comments to ensure they accurately describe the code's intended behaviour and logic. This helps developers to better understand the codebase and make modifications without introducing new issues.
- **Compliance with best practices**: We checked that the code follows best practices and coding standards that are recommended by the community and industry experts. This ensures that the smart contract is secure, reliable, and efficient.

Our audit team adhered to OWASP secure coding principles, along with the Rust programming language's community standards and best practices developed within ecosystems such as Aptos and Sui. By following these guidelines, we were able to thoroughly assess the smart contracts, identify potential vulnerabilities, and provide actionable recommendations to strengthen security, performance, and maintainability.

Throughout the audit process, we placed strong emphasis on ensuring the overall quality of the codebase. Every part of the contract was carefully reviewed to confirm that it was not only secure but also well-structured, maintainable, and aligned with the project's intended functionality. Special attention was given to proper documentation, ensuring that comments accurately reflected the logic implemented, and that the code adhered to Rust's strict resource and capability management principles. Our approach was comprehensive and methodical, aiming to deliver a final product that was both secure and optimized for performance in real-world deployment.

6.1 Code Review / Manual Analysis

A detailed manual analysis of the Rust smart contracts was carried out to uncover vulnerabilities that automated tools might miss. We carefully reviewed each function and module to ensure that the business logic matched the project's requirements. Special attention was given to resource safety, proper access control, correct capability management, and structured error handling. Manual review also helped validate the issues identified through static analysis and highlighted hidden logic flaws. Through this process, we ensured that the contracts were secure, reliable, and aligned with best practices.

6.2 Tools Used for Audit

To strengthen the security and performance of the smart contracts, we used a variety of specialized Rust tools throughout the audit. Static analysis was performed using Rust Analyzer, while Rustile was used for formal verification of critical properties. Dynamic testing was conducted via Cargo, Aderyn, and Rust Sandbox to simulate transactions and module behavior under different conditions. Unit and integration tests were executed with the Rust CLI Testing Framework, while custom scripts helped with fuzz testing and gas profiling. Combining expert review and automated tools allowed us to perform a deep, comprehensive audit.

7.0 Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of the Nira Project and methods for exploiting them. 0xTeam recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein. Security Assessment is an uncertain process, based on past experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, while 0xTeam considers the major security vulnerabilities of the analysed systems to have been identified, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures. In addition, the analysis set forth herein is based on the technologies and known threats as of the date of this report. As technologies and risks change over time, the vulnerabilities associated with the operation of the Nira Smart contracts Code Base described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities, will also change. 0xTeam makes no undertaking to supplement or update this report based on changed circumstances or facts of which 0xTeam becomes aware after the date hereof, absent a specific written agreement to perform the supplemental or updated analysis. This report may recommend that 0xTeam use certain software or hardware products manufactured or maintained by other vendors. 0xTeam bases these recommendations upon its prior experience with the capabilities of those products. Nonetheless, 0xTeam does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended. This report was prepared by 0xTeam for the exclusive benefit of Nira and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between 0xTeam and Nira governs the disclosure of this report to all other parties including product vendors and suppliers.