BAIL security

Mira Network
MultiTierStaking

FINAL REPORT

June ‘2025

Disclaimer:

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

The content of this assessment is not an investment. The information provided in this report is for general informational purposes only and is not intended as investment, legal, financial, regulatory, or tax advice. The report is based on a limited review of the materials and documentation provided at the time of the audit, and the audit results may not be complete or identify all possible vulnerabilities or issues. The audit is provided on an "as-is," "where-is," and "as-available" basis, and the use of blockchain technology is subject to unknown risks and flaws.

The audit does not constitute an endorsement of any particular project or team, and we make no warranties, expressed or implied, regarding the accuracy, reliability, completeness, or availability of the report, its content, or any associated services or products. We disclaim all warranties, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We assume no responsibility for any product or service advertised or offered by a third party through the report, any open-source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, and the related services and products. We will not be liable for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract.

The contract owner is responsible for making their own decisions based on the audit report and should seek additional professional advice if needed. The audit firm or individual assumes no liability for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract. The contract owner agrees to indemnify and hold harmless the audit firm or individual from any and all claims, damages, expenses, or liabilities arising from the use or reliance on the audit report or the smart contract.

By engaging in a smart contract audit, the contract owner acknowledges and agrees to the terms of this disclaimer.

1. Project Details

Important:
Please ensure that the deployed contract matches the source-code of the last commit hash.

Project	Mira Network - MultiTierStaking
Website	mira.network
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/Aroha-Labs/mira-staking-v0/blob/879b7a790743d88193ac1425ae8d994a9aa0a125/src/MultitierStaking.sol
Resolution 1	https://github.com/Aroha-Labs/mira-staking-v0/blob/fb7314219342549fdb3f27da1940157a02815a7d/src/MultitierStaking.sol

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)	Failed resolution
High	1	1			
Medium					
Low	1			1	
Informational	6			6	
Governance					
Total	8	1		7	

2.1 Detection Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users
Governance	Governance privileges which can directly result in a loss of funds or other potential undesired behavior

3. Detection

MultiTierStaking

The MultiTierStaking contract is a simple staking contract which allows the contract owner to create pools with a corresponding stakeToken and rewardToken. A pool has the following properties:

> totalStake: The total amount of stakeTokens deposited by users
> startBlock: The period after which new deposits are not allowed
> endBlock: The period after which users can withdraw their stake and claim rewards
> rewardMultiplier: The multiplier which determines reward amount based on deposited stakingToken amount
> rewardsClaimed: The amount of rewards which have been claimed

Users can deposit before the startBlock has reached and then withdraw their stake and claim their rewards once the endBlock has reached.

Appendix: rewardMultiplier

The rewardMultiplier determines how much rewards users will receive based on the deposited amount using the following formula:

> (depositAmount * (rewardMultiplier - 10000)) / 10000

Therefore, if rewardMultiplier is 2000, the reward amount after endBlock will be equal to the deposited amount of stakeToken.

It is expected that the owner manually transfers in reward tokens to honor claims and withdrawals.

Core Invariants:

INV 1: Users can only deposit before startBlock

INV 2: Users can only withdraw at/after endBlock

INV 3: First created pool must have ID zero

INV 4: Users can only withdraw once

INV 5: Claimable rewards must be in relation to deposited amount

Privileged Functions

- transferOwnership
- renounceOwnership

Issue_01	Funds may remain indefinitely locked due to claimableRewards calculation
Severity	**High**
Description	Whenever a deposit happens, claimableRewards are calculated as follows: > (depositAmount * (rewardMultiplier - 10000)) / 10000 This means, with an increasing amount of deposited funds, the entitled reward amount to users is increased as well. On top of that, there is currently no deposit limit and no enforcement that the rewards are indeed entitled for this pool (aka deposited by governance). Therefore, the entitled claimable reward amount can grow indefinitely, without limitation. This can result in multiple issues, just to name a few: - Stuck funds due to large deposit amount and corresponding insufficient reward distribution which results in a revert during withdrawal - Commingling of rewards for other pool reward tokens which then leave other pools insolvent

Recommendations	Consider: a) Enforcing a deposit cap for each pool b) Pre-calculating required reward amount based on deposit cap and reward multiplier c) Require transfer of maximum rewards for each pool during pool creation d) Implementation of recovering any unconsumed rewards Point d) can either be a centralized recover function or a function which implements additional calculation to determine the excess reward amount. **In the latter scenario, additional auditing time beyond the standard resolution is required to cover such a functionality in an effort to detect unexpected edge-cases, depending on the implementation**.
Comments / Resolution	Resolved, the client implemented a deposit limit and ensures that rewards for the maximum deposit amount are pre-owned by the DAO. More specifically, the client explains the process as follows: "Upon pool creation, the total reward allocation will be secured in a DAO-controlled multisig wallet managed by our custody partner. Once the staking period closes, we will recalculate the actual rewards needed based on the final deposit amount and transfer only the required rewards from the DAO multisig to the staking pool contract. This approach ensures reward funds remain secure while allowing us to recover any unused rewards back to the DAO treasury."

Issue_02	rewardToken can be same as stakeToken
Severity	Low
Description	Currently, there is no enforcement that rewardToken is different to stakeToken. This can result in unexpected side-effects, such as draining stakeToken in case there is insufficient reward balance.
Recommendations	Consider implementing such a validation in the constructor.
Comments / Resolution	Acknowledged.

Issue_03	Lack of zero address validation
Severity	Informational
Description	The constructor determines stakeToken and rewardToken. There is currently no address(0) validation.
Recommendations	Consider validating the parameters accordingly.
Comments / Resolution	Acknowledged.

Issue_04	Lack of support for transfer-tax tokens
Severity	Informational
Description	This contract is not compatible with transfer-tax tokens. If these token types are used for any purpose within the contract, this will result in down-stream issues and inherently break the accounting.
Recommendations	Consider not using these tokens.
Comments / Resolution	Acknowledged.

Issue_05	Violation of CEI
Severity	**Informational**
Description	Throughout the contract there are one or multiple spots which violate the checks-effects-interactions pattern, to ensure a protection against invalid states, all external calls should strictly be implemented after any checks and effects (state variable changes). *IERC20(stakeToken).safeTransferFrom(msg.sender, address(this), _amount);* *// update user staked amount, and total staked amount for the pool* *user.amount += _amount;* *user.claimableRewards += (_amount * (pool.rewardMultiplier - 10000)) / 10000;* *pool.totalStake += _amount;*
Recommendations	Consider executing state changes before token transfers.
Comments / Resolution	Acknowledged.

Issue_06	Difference in token decimals requires rewardMultiplier adjustment
Severity	**Informational**
Description	The rewardMultiplier determines how much rewards are entitled based on the deposited amount. A rewardMultiplier of 110_000 results in 10x rewards for a specific deposited amount. This property can be distorted if there is a difference in decimals between the stakeToken and the rewardToken.
Recommendations	Consider keeping this in mind and setting rewardMultiplier accordingly. No code change is required.
Comments / Resolution	Acknowledged.

Issue_07	getUserStakes may revert if called by another contract
Severity	**Informational**
Description	The getUserStakes function returns all stakes from a specific address. This executes a loop over all pools which can become gas-exhaustive if called by another contract in a not view-only fashion and ultimately revert if the block gas limit is exceeded.
Recommendations	Consider keeping this limitation in mind
Comments / Resolution	Acknowledged.

Issue_08	Truncation of claimableRewards
Severity	Informational
Description	The claimableRewards calculation eventually truncates the result: > (depositAmount * (rewardMultiplier - 10000)) / 10000 This can become specifically an issue if the stakeToken has low decimals: > (1.9999e6 * (10010 - 10000)) / 10000 > 1999.9 > 1999 It can then result in a deviation between the real claimable amounts and the return value from getRequiredFunding (as getRequiredFunding aggregates all stakes and thus may prevent rounding of individual stakes).
Recommendations	We do explicitly not recommend any code-change as this issue is only informational and does not expose any harm. It should simply be acknowledged.
Comments / Resolution	Acknowledged.