Arcadia Staking Audit Report

Version 2.0

Audited by:

alexxander

bytes032

January 26, 2025

Contents

1 Introduction **2**

1.1 About Renascence 2

1.2 Disclaimer 2

1.3 Risk Classification 2

2 Executive Summary **3**

2.1 About Arcadia Staking 3

2.2 Overview 3

2.3 Issues Found 3

3 Findings Summary **4**

4 Findings **5**

1 Introduction

1.1 About Renascence

Renascence Labs was established by a team of experts including HollaDieWaldfee, MiloTruck, alexxander and bytes032.

Our founders have a distinguished history of achieving top honors in competitive audit contests, enhancing the security of leading protocols such as Reserve Protocol, Arbitrum, MaiaDAO, Chainlink, Dodo, Lens Protocol, Wenwin, PartyDAO, Lukso, Perennial Finance, Mute and Taurus.

We strive to deliver tailored solutions by thoroughly understanding each client's unique challenges and requirements. Our approach goes beyond addressing immediate security concerns; we are dedicated to fostering the enduring success and growth of our partners.

More of our work can be found here.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an 'as-is' and 'as-available' basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

1.3.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol’s core functionality
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol’s functionality
- Low - Funds are **not** at risk

1.3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

2 Executive Summary

2.1 About Arcadia Staking

The protocol consists of two primary contracts `AbstractStaker.sol` and `AAAStaker.sol`. The `AbstractStaker` contract implements logic for the distribution of multiple rewards in exchange for staking a single asset. The `AAAStaker.sol` contract builds upon `AbstractStaker` by introducing queued withdrawals. Users that wish to unstake their tokens have to initiate a withdrawal and wait out a configured withdrawal period to execute their withdrawal request. However, users are also enabled to execute their withdrawal earlier in exchange of incurring an early withdrawal penalty which is calculated based on how much time is left of the request's withdrawal period.

2.2 Overview

Project	Arcadia Staking
Repository	arcadia-staking
Commit Hash	7fd14e4784d7...
Mitigation Hash	b931eed126c3...
Date	14 January 2025 - 17 January 2025

2.3 Issues Found

Severity	**Count**
High Risk	1
Medium Risk	1
Low Risk	4
Informational	0
Total Issues	**6**

3 Findings Summary

ID	Description	Status
H-1	Removed rewards are excluded from reward state tracking	Resolved
M-1	It is possible to block all operations in AbstractStaker	Resolved
L-1	Batch withdrawals may execute wrong withdrawal requests	Resolved
L-2	Changing withdrawal period has no effect on current withdrawal requests	Resolved
L-3	No validation if reward exists in `AbstractStaker.discontinueReward()`	Resolved
L-4	The function `AbstractStaker.recoverERC20()` reverts for inactive rewards	Resolved

4 Findings

High Risk

[H-1] Removed rewards are excluded from reward state tracking

Context:

- AbstractStaker.sol#L391
- AbstractStaker.sol#L412

Description: The staking protocol should update its state every time the total supply or a user's balance changes to ensure the correct distribution of rewards to depositors. In `AbstractStaker.sol`, this is handled by the `_updateRewardState` and `_updateUserState` functions. These functions are called for each active reward whenever a user's balance changes:

```
    function _updateRewards(address from, address to) internal returns (RewardKey[]
    memory rewardKeys) {
        // Cache variables.
»      rewardKeys = activeRewards;
        uint256 totalSupply_ = totalSupply();
        uint256 balanceFrom = balanceOf(from);
        uint256 balanceTo = balanceOf(to);

        // Update state.
        uint128 rewardPerToken;
»      for (uint256 i; i < rewardKeys.length; ++i) {
            rewardPerToken = _updateRewardState(rewardKeys[i], totalSupply_);
            _updateUserState(rewardKeys[i], from, balanceFrom, rewardPerToken);
            _updateUserState(rewardKeys[i], to, balanceTo, rewardPerToken);
        }
    }
```

However, when a reward is deactivated or terminated, it is removed from the `activeRewards` list. This means the state for that reward will no longer be updated:

```
    function deactivateReward(address reward, uint256 rewardId) external {
        RewardKey rewardKey = RewardKeyLogic._getRewardKey(reward, rewardId);
        if (block.timestamp < rewardState[rewardKey].endTime + 1 weeks) revert
        DeactivationNotAllowed();

        // Remove reward from active rewards.
»      _removeActiveReward(rewardKey);
        // Set status to inactive: from 2 -> 0 or from 3 -> 1.
        // If reward is already inactive, do nothing.
        uint256 status = rewardState[rewardKey].status;
        if (status == 2) rewardState[rewardKey].status = 0;
        else if (status == 3) rewardState[rewardKey].status = 1;
    }
```

This poses a critical risk to the staking protocol. For instance, a user could claim rewards for a deactivated reward, transfer their staking shares to another address, and claim the rewards again—effectively stealing reward tokens from the contract.

Recommendation: Ideally, the state should be tracked for all rewards. However, to prevent the active rewards list from growing indefinitely, it is recommended to remove rewards from the active list only when they are terminated, as it is not possible to claim rewards for terminated rewards.

Arcadia: Fixed in PR-5.

Renascence: The issue has been resolved. `AbstractStaker.deactivateReward()` has been removed and the status of rewards has been refactored. Removing a reward from `activeRewards` can only be done through `terminateReward()` which transfers any left unclaimed rewards to the distributor.

Medium Risk

[M-1] It is possible to block all operations in AbstractStaker

Context:

- AbstractStaker.sol#L439-L441

Description: When the reward state is updated, `AbstractStaker` casts the `rewardPerToken` growth to the `uint128` type:

```
    function _updateRewardState(RewardKey rewardKey, uint256 totalSupply_) internal
    returns (uint128 rewardPerToken) {
        // Cache variables.
        uint256 endTime = Math.min(block.timestamp, rewardState[rewardKey].endTime);
        uint256 lastUpdateTime = rewardState[rewardKey].lastUpdateTime;
        rewardPerToken = rewardState[rewardKey].lastRewardPerToken;

        // Update reward state.
        if (totalSupply_ > 0 && endTime > lastUpdateTime) {
            // unchecked: Can overflow, what matters is the delta in RewardPerToken
            between two interactions.
            unchecked {
»              rewardPerToken = rewardPerToken
                    + SafeCastLib.safeCastTo128(
                        uint256(rewardState[rewardKey].rewardRate).mulDivDown(endTime
                        - lastUpdateTime, totalSupply_)
                    );
            }
```

The maximum value that can be safely cast to `uint128` using `safeCastTo128` without reverting is approximately `3.4 * 10**38`. This limit may not be sufficient in specific scenarios. For example:

1. A user stakes 1 wei.
2. The reward distributor adds `10000e18` reward tokens to the contract, emitting over 10 days. This results in a rate of `10000e36 / (86400 * 10)`.
3. After 1 day, if the reward state is updated, the calculated `rewardPerToken` growth becomes `10000e36 * 86400 / (86400 * 10 * 1) = 1e39` . This value exceeds the limit for `uint128` and causes the `safeCastTo128` function to revert.

Since the state update occurs before any staking-related operation, the system would be blocked until the problematic reward is removed. Due to the specific conditions required (e.g., single staker, dust supply), the severity of this issue is categorized as medium.

Recommendation: It is recommended to use the `uint256` type to store `rewardPerToken`.

Arcadia: Fixed in PR-9.

Renascence: The issue has been resolved as per the recommendation.

Low Risk

[L-1] Batch withdrawals may execute wrong withdrawal requests

Context:

- AAAStaker.sol#L175-L177

Description: Users can batch-execute multiple withdrawal requests in `AAAStaker`:

```
    function withdraw(uint256[] memory indices) public nonReentrant {
        uint256 total;
        uint256 total_;
        uint256 redeemable;
        uint256 redeemable_;
        uint256 penalty;
        uint256 penalty_;
        for (uint256 i; i < indices.length; ++i) {
            (total_, redeemable_, penalty_) = getWithdrawalAmounts(msg.sender,
            indices[i]);
            total += total_;
            redeemable += redeemable_;
            penalty += penalty_;

            // Remove withdrawal entry.
»          withdrawalRequests[msg.sender][indices[i]] =
                withdrawalRequests[msg.sender][withdrawalRequests[msg.sender].length
                - 1];
            withdrawalRequests[msg.sender].pop();
        }
```

However, this implementation is flawed due to the array manipulation inside the `for` loop. When a withdrawal request is removed, the last element of the array is swapped into the removed element's position. This alters the order of elements in the `withdrawalRequests` array during the loop, which can lead to:

- out-of-bounds reverts If the user intends to execute all withdrawal requests;
- an intended withdrawal request may be swapped with a previously deleted element, causing the wrong request to be executed.

Recommendation: Consider processing user specified `indices` in reverse order.

Arcadia: Fixed in PR-6.

Renascence: The issue has been resolved as per the recommendation.

[L-2] Changing withdrawal period has no effect on current withdrawal requests

Context:

- AAAStaker.sol#L241-L244

Description:

When the `AAAStaker` owner changes the withdrawal period, the update only affects stakers who submit their withdrawal requests after the change. This can lead to unfair treatment of users who submitted requests before the update.

For example, if the original withdrawal period was 100 days and is reduced to 50 days, users with existing requests would still need to wait the original 100 days to withdraw without incurring a penalty, while new requests would only require a 50-day wait.

Recommendation:

Consider storing only the initial date of the withdrawal request and calculating the penalty dynamically using the current withdrawal period.

Arcadia: Fixed in PR-8.

Renascence: The issue has been resolved by using the reward period that is most advantageous to the user.

[L-3] No validation if reward exists in `AbstractStaker.discontinueReward()`

Context:

- AbstractStaker.sol#L149-L157

Description It is possible for the owner to discontinue a non-existing reward, changing its status to `1`, which results in any newly added reward being discarded at the start.

Recommendation It is recommended to validate if the reward exists before discontinuing it.

Arcadia: Fixed in PR-7.

Renascence: The issue has been resolved. Function `AbstractStaker.discontinueReward()` now verifies that the reward exists by checking if the reward's distributor is set.

[L-4] The function `AbstractStaker.recoverERC20()` reverts for inactive rewards

Context:

- AbstractStaker.sol#L647-L657

Description: The description of `recoverERC20()` states that an asset is recoverable if it is not an active reward or the staking token. Currently, the function loops over the `rewards` state array that contains all rewards instead of the `activeRewards` state array that contains only the active rewards and will revert on trying to rescue inactive rewards.

Recommendation:

```
# AbstractStaker.recoverERC20()

-         for (uint256 i; i < rewards.length; ++i) {
-             (reward,) = rewards[i]._getRewardFromKey();
+         for (uint256 i; i < activeRewards.length; ++i) {
+             (reward,) = activeRewards[i]._getRewardFromKey();
```

Arcadia: Fixed in PR-10.

Renascence: The issue has been resolved as per the recommendation.