certora

Security Assessment

ether.fi

Ether-Fi

March 2025

Prepared for EtherFi

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
EtherFi RewardsManager	https://github.com/etherfi-protocol/smart-contracts/blob/982625edfe628441d686314322e16f026d24b8df/src	Audit start: 982625e Latest version reviewed: 4172e90	EVM

Project Overview

This document describes the specification and verification of **EtherFi's RewardsManager** using manual code review. The work was undertaken from **03/03/2025** to **07/03/2025.** The focus was on PR#236

The following contract list is included in our scope:

```
src/CumulativeMerkleRewardsDistributor.sol
src/interfaces/ICumulativeMerkleRewardsDistributor.sol
```

The team performed a manual audit of all the Solidity contracts**.** During this review, the Certora team discovered bugs in the Solidity contracts code, as listed on the following page.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	-	-	-
Low	2	2	2
Informational	5	5	5
Total	7	7	7

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

Detailed Findings

ID	Title	Severity	Status
L-01	`CLAIM_DELAY` Uses Block Numbers Instead of Time, Causing Inconsistent Delays Across Chains	Low	Fixed
L-02	`lastRewardsCalculatedToBlock` May Be In The Future	Low	Fixed

Low Severity Issues

L-01 `CLAIM_DELAY` Uses Block Numbers Instead of Time, Causing Inconsistent Delays Across Chains		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: CumulativeMerkleRewardsDistributor.sol	Status: Fixed	

Description: The `CLAIM_DELAY` constant is defined in terms of block numbers (`14400` blocks). Since different blockchains have varying block times, the actual delay before finalizing the Merkle root differs across networks. For example:

- On Ethereum (~12s per block), the delay is approximately **2 days**, which may be sufficient for the verification. Moreover the block time can change in the future and it is not constant.
- On Arbitrum (~0.25s per block), the forced delay is only **1 hour**, which may be too short to verify if `processRewards` was called with incorrect amounts.

This inconsistency allows the `REWARDS_MANAGER_ADMIN` to potentially call the `finalizeMerkleRoot()` before the verification of `processRewards`

Recommendations: Use `block.timestamp` instead of the number of blocks for time sensitive operations

Customer's response: Fixed in commit [4ffd0d7]()

Fix Review: Fix confirmed

L-02 `lastRewardsCalculatedToBlock` May Be In The Future

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: CumulativeMerkleRewardsDistributor.sol	Status: Fixed	

Description: The value provided for `lastRewardsCalculatedToBlock[token]` in `finalizeMerkleRoot()` is completely trusted. This value can be set in the future which would mislead users. Even if the function is admin restricted it still should include basic validation because the contract shouldn't accept bad state.

```
JavaScript
    function finalizeMerkleRoot(address _token, uint256 _finalizedBlock) external
whenNotPaused {
        if(!roleRegistry.hasRole(REWARDS_MANAGER_ADMIN, msg.sender)) revert IncorrectRole();
        if(!(block.number >= lastPendingMerkleUpdatedToBlock[_token] + CLAIM_DELAY)) revert
InsufficentDelay();
        bytes32 oldClaimableMerkleRoot = claimableMerkleRoots[_token];
        claimableMerkleRoots[_token] = pendingMerkleRoots[_token];
        lastRewardsCalculatedToBlock[_token] = _finalizedBlock;
        emit ClaimableMerkleRootUpdated(_token, oldClaimableMerkleRoot,
claimableMerkleRoots[_token], _finalizedBlock);
    }
```

Recommendations: Consider requiring that the provided value (`_finalizedBlock`) is at most `block.number`.

Customer's response: Fixed in commit 4ffd0d7

Fix Review: Fix confirmed

Informational Severity Issues

I-01. Use More Modern Syntax For Designating Memory-Safe Assembly

Description: Comment annotation for memory-safe assembly can be replaced with the more modern block annotation which is available in the 0.8.24 Solidity version used by the code. Check out the solidity documentation here.

Link

```
JavaScript
 /// @solidity memory-safe-assembly
 assembly {  // solhint-disable-line no-inline-assembly
```

Recommendation: Replace the comments with this:

```
JavaScript
assembly ("memory-safe") {  // solhint-disable-line no-inline-assembly
```

Customer's response: Fixed in commit 68fb3c6

Fix Review: Fix confirmed

I-02. MerkleRootUpdated Event Is Unused

Description: In `ICumulativeMerkleRewardsDistributor` there is a `MerkleRootUpdated` event that is unused. Other, more specific events are used in the places it would be intended for.

```
JavaScript
event MerkleRootUpdated(bytes32 oldMerkleRoot, bytes32 newMerkleRoot);
```

Recommendation: Remove that event

Customer's response: Fixed in commit 68fb3c6

Fix Review: Fix confirmed

I-03. Inconsistent Use of Revert Strings When Custom Errors Are More Prevalent

Description: The `CumulativeMerkleRewardsDistributor` mostly uses custom errors in explicit revert scenarios (which are generally more gas-efficient than revert strings). There are two exceptions: the check in `_requireNotPaused()` and reversion due to a failed ETH transfer in `claim()` both use a revert string instead.

JavaScript
```
function claim() external whenNotPaused override {
        //...
        if(!success) {
              revert("ETH Transfer failed");
        }
        //...
}

function _requireNotPaused() internal view virtual {
         require(!paused, "Pausable: paused");
}
```

Recommendation: Consider using custom errors in these cases for consistency and gas efficiency.

Customer's response: Fixed in commit 68fb3c6

Fix Review: Fix confirmed

I-04. No License Identifier In CumulativeMerkleRewardsDistributor

Description: The `CumulativeMerkleRewardsDistributor` lacks a SPDX License Identifier

Recommendation: Add a SPDX License Identifier at the top of the contract

Customer's response: Fixed in commit 68fb3c6

Fix Review: Fix confirmed

I-05. Unchecked Math Operations In Proof Verification

Description: Assembly math operations are not checked for overflow. The computation of the final value for the loop pointer during proof verification could be made to overflow with maliciously crafted calldata, skipping the entire loop. This doesn't appear to be exploitable, because an exploit would still require solving a constrained hash collision problem. However, adding a sanity check on the proof length would cost very little gas and prevent any possible pathological scenario (e.g. proof.length < 1000 should be sufficient for any conceivable practical case, while still ensuring no overflow can occur).

Link

```javascript
for { let end := add(ptr, mul(0x20, proof.length)) } lt(ptr, end) { ptr := add(ptr, 0x20) } {
```

Recommendation: Consider adding a check for the `merkleProof` to limit the length of it.

Customer's response: Fixed in commit 68fb3c6

Fix Review: Fix confirmed

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.