Security Assessment

ether.fi – Core Contracts Combined Audit Report

December 2025 - January 2026

Prepared for ether.fi

certora ether.fi

Table of contents

Project Summary

Project Scope

Project Name	Initial Commit Hash	Latest Commit Hash	Platform	Start Date	End Date
Liquid Referrer	80a6b62	639b4d97	EVM	05/12/2025	09/12/2025
Restaking Rewards Router	06fa88db	7df831d6	EVM	18/01/2026	19/01/2026
Cross pod approval	32ef74f	a6b8291c	EVM	19/01/2026	20/01/2026

Project Overview

This document describes the manual code review of several modules and changes to the core contracts repository.

The work was a 3 day effort undertaken between **05/12/2025** and **20/01/2026**

The team performed a manual audit of the Solidity smart contracts. During the manual audit, the Certora team discovered bugs in the Solidity smart 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	-	-	-
Informational	8	8	6
Total	8	8	6

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
Liquid Referrer			
I-01	Deposit does not support native tokens	Info	Fixed
I-02	Inconsistency in the initialization call chain	Info	Fixed
I-03	Tellers are not whitelisted	Info	Fixed
I-04	Pausable functionality not utilized	Info	Fixed
Restaking Rewards Router			
I-01	Event emissions in receive() lacks info on the sender	Info	Fixed
I-02	Contract should consider more than one reward token being accumulated	Info	Acknowledged
Cross pod approval			
I-01	Inconsistent Pubkey Hashing Method Used in Consolidation Rate-Limiting Logic	Info	Fixed
I-02	totalConsolidationGwei overestimates and can consume much more capacity than was actually used	Info	Acknowledged

Liquid Referrer

Project Overview

This report presents the findings of a manual code review for the **Liquid Referrer** audit within the **EtherFi** core contracts. The work was undertaken from **December 5th to December 9th 2025**

The following contract list is included in the scope of this audit:

- `src/helpers/LiquidRefer.sol`

The code modifications examined during this review were introduced in the following pull request - PR#326.

Informational Issues

I-01. Deposit does not support native tokens

Description: The deposit function in LiquidRefer forwards msg.value to the teller contract:

```
JavaScript
shares = teller.deposit{value: msg.value}(depositAsset, depositAmount, minimumMint);
```

However the deposit function only handles ERC20 tokens since it always relies on the TransferFrom flow. As result if someone sends msg.value, the call will always revert, due to validation in the teller

Recommendation: Remove native token forwarding, since it is not supported

Customer's response: Fixed in commit e8a38ab & 639b4d97

Fix Review: Fixed

I-02. Inconsistency in the initialization call chain

Description: Inside the initializer of LiquidRefer, the initializers for UUPS and Pausable are not invoked. Although they do not assign any variables, it is a good practise to include them

Recommendation: Add the additional initializers in the initialize function

Customer's response: Fixed in commit [e8a38ab]

Fix Review: Fixed

I-03. Tellers are not whitelisted

Description: Currently the caller can provide any teller address to be called by LiquidRefer. It is always advisable to use whitelisted addresses, instead of allowing the contracts to execute calls to arbitrary and untrusted addresses

Recommendation: Consider whitelisting the teller, which would ensure reliable and consistent behaviour. Also it will eliminate any potential attack surface

Customer's response: Fixed in commit e8a38ab

Fix Review: Fixed

I-04. Pausable functionality not utilized

Description: The functionality provided by PausableUpgradeable is not used inside the contract

Recommendation: Consider if the pausing mechanism would be used now or in the future and in case not, it can be removed from the contract

Customer's response: Fixed in commit e8a38ab

Fix Review: Fixed

Restaking Rewards Router

Project Overview

This report presents the findings of a manual code review for the **Restaking Rewards Router** audit within the **EtherFi** core contracts. The work was undertaken from **January 18th to January 19th 2026**

The following contract list is included in the scope of this audit:

- `src/RestakingRewardsRouter.sol`

The code modifications examined during this review were introduced in the following pull request - PR#353 .

Informational Issues

I-01. Event emissions in receive() lacks info on the sender

Description: The receive() handler emits the EthSent event when forwarding ETH using address(this), the LiquidityPool, and the amount, but it does not include msg.sender, which reduces observability.

Recommendation: Add the msg.sender as additional parameter in the event

Customer's response: Fixed in commit 1a10a60f

Fix Review: Fixed

I-02. Contract should consider more than one reward token being accumulated

Description: The ERC20 recovery mechanism is limited to a single hardcoded rewardTokenAddress. If this router is used as the recipient in EigenLayer's RewardsCoordinator (which can transfer more than one reward token), any additional reward tokens transferred to it will remain stuck.

Recommendation: Consider allowing recovery of arbitrary ERC20 tokens, or renaming the function and role to explicitly reflect that only a single reward token is supported.

Customer's response: Acknowledged

Fix Review: Acknowledged

Cross pod approval

Project Overview

This report presents the findings of a manual code review for the **Cross pod approval** audit within the **EtherFi** core contracts. The work was undertaken from **January 19th to January 20th 2026**

The following contract list is included in the scope of this audit:

- `src/EtherFiNodesManager.sol`
- `src/LiquidityPool.sol`

The code modifications examined during this review were introduced in the following pull request - PR#348.

Informational Issues

I-01. Inconsistent Pubkey Hashing Method Used in Consolidation Rate-Limiting Logic

Description:
In getTotalConsolidationGwei, the contract compares srcPubkey and targetPubkey using keccak256(requests[i].srcPubkey) instead of the canonical calculateValidatorPubkeyHash helper used elsewhere in the codebase. Other parts of the system consistently rely on calculateValidatorPubkeyHash to derive validator identity according to the expected SSZ-style hashing scheme.
Using a different hashing approach here does not currently introduce incorrect behavior, but it creates an inconsistency in how validator identities are reasoned about across the contract.

Recommendation:
For consistency and clarity, consider using calculateValidatorPubkeyHash for comparing source and target validators in consolidation logic, even if the comparison is only used for rate-limiting purposes.

Customer's response: Fixed in commit a6b8291c

Fix Review: Fixed

I-02. totalConsolidationGwei overestimates and can consume much more capacity than was actually used

Description: The requestConsolidation function in LiquidityPool has been updated to implement rate limiting through a getTotalConsolidationGwei internal function that loops through the requests and sums the amount of gwei that are requested to be consolidated.

The getTotalConsolidationGwei function always uses that maximum validator amount of 2048 ETH. Although meant to act as an additional security layer that ensures none of the requests is underestimated, this also creates a scenario where even 32 ETH consolidations would be consumed as if they were 2048 ETH (64x more). This might throttle the rate limiter prematurely and temporarily block further consolidation requests, although the actual amounts consumed were significantly less.

In addition consolidation requests that succeed through EigenLayer, can still fail asynchronously on the Beacon chain, which requires them to be retried. However the full 2048 ETH would still be consumed and reduce the capacity.

Recommendation: Implementing rate limiting for consolidation requests is a challenging task, that cannot be applied without any drawbacks.

For example an alternative would be to query EigenLayer on the latest snapshotted state of the validator and get the stake amounts from there. However if the validator is not snapshotted to the latest state, this would again mean over/underestimating the actual balances.

Both approaches have tradeoffs. The team should be aware of them and the imperfections related to rate-limiting on those type of request

Also make sure to properly validate the consolidation requests off-chain before submitting them to the contract to prevent wasting the bucket capacity

Customer's response: Acknowledged

Fix Review: Acknowledged

Disclaimer

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.