Security Assessment

ether.fi – Core Contracts Combined Audit Report

November 2025

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
Node Operators Self Registration	098cf09	700dc0	EVM	07/11/2025	10/11/2025
Role for Consolidating of Validators	92d8324	92d8324	EVM	07/11/2025	10/11/2025
EtherFiRestaker Hot Fix	46e1fb83	46e1fb83	EVM	21/11/2025	24/11/2025
PR Consolidation	25312df	25312df	EVM	04/12/2025	04/12/2025

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 **07/11/2025** and **04/12/2025**

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	2	2	2
Informational	2	2	-
Total	4	4	2

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
Node Operators Self Registration			
L-01	The hash created for registered validator is prone to collision	Low	Fixed
L-02	The dataRoot might differ between registration and creation phase	Low	Fixed
Role for Consolidating of Validators			
-	-	-	-
EtherFiRestaker Hot Fix			
-	-	-	-
PR Consolidation			

Node Operators Self Registration

Project Overview

This report presents the findings of a manual code review for the **Node Operators Self Registration** audit within the **EtherFi** core contracts. The work was undertaken from **November 7th to November 10th 2025**

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

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

The code modifications examined during this review were introduced initially in the following pull request - PR#301. Later they were moved into a consolidated PR with the other items in this report which is the following and final one – PR#333

Low Severity Issues

L-01 The hash created for registered validator is prone to collision		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: StakingManager.sol	Status: Fixed	

Description: The new registerBeaconValidators() function in Staking manager creates a hash of the provided parameters by using abi.encodePacked like this:

```
JavaScript
bytes32 validatorCreationDataHash = keccak256(abi.encodePacked(depositData[i].publicKey,
depositData[i].signature, depositData[i].depositDataRoot,
depositData[i].ipfsHashForEncryptedValidatorKey, bidIds[i], etherFiNode));
```

The created hash is later used to make sure the validator is created with the same parameters used upon registration.

The problem here is that abi.encodePacked() has a known issue of leading to hash collisions when being used on dynamic values (bytes, arrays, etc.). This is exactly the case here where we have the publicKey & signature fields (which are bytes) concatenated together.

Since encodePacked strips the length and metadata of dynamic values leaving, this allows the following to be possible:

```
JavaScript
bytes memory pk1 = hex"aaaa";
bytes memory sig1 = hex"bbbb";

bytes32 packedHash1 = abi.encodePacked(pk1,sig1,...)
```

```
bytes memory pk2 = hex"aa";
bytes memory sig2 = hex"aabbbb";

bytes32 packedHash2 = abi.encodePacked(pk2,sig2,...)

packedHash1 == packedHash2 (this would evaluate to true)
```

Practically this means that the caller can provide two different inputs that evaluate to the same hash, which should not be allowed in a system.

Since the function is permissioned and the likelihood of the above is quite low, the severity of this issue is low, however given the easy fix and enhanced security it would be best to fix it

Recommendation: Use abi.encode instead of encodePacked, which retains the length of original data and prevents the above vector

Customer's response: Fixed in commit 700dc0

Fix Review: Fixed

certora

ether.fi

L-02 The dataRoot might differ between registration and creation phase		
Severity: **Low**	Impact: **High**	Likelihood: **High**
Files: StakingManager.sol	Status: Fixed	

Description: In the new version of the code of StakingManager, a new function has been added called registerBeaconValidators(), which does the following check:

```
JavaScript
bytes32 computedDataRoot = generateDepositDataRoot(depositData[i].publicKey,
depositData[i].signature, withdrawalCredentials, initialDepositAmount);

if (computedDataRoot != depositData[i].depositDataRoot) revert IncorrectBeaconRoot();
```

This check has been removed from createBeaconValidators() function with the assumption that it cannot differ from the registration phase. However one of the parameters provided to generateDepositDataRoot() can change, which is the initialDepositAmount variable. It is a constant variable and in case the team upgrades the contracts to a new implementation with a new value for it, this would change the output of generateDepositDataRoot(), preventing validator creation, since the values have changed since the registration.

However since the computedDataRoot check has been removed, the creation process would complete, instead of failing.

Recommendation: Do the computedDataRoot check inside createBeaconValidators as well to ensure consistency and ensure parameter changes would be caught

Customer's response: Fixed in commit 700dc0
Fix Review: Fixed

Role for Consolidating of Validators

Project Overview

This report presents the findings of a manual code review for the **Role for Consolidating of Validators** audit within the **EtherFi** core contracts. The work was undertaken from **November 7th to November 10th 2025**

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

- `src/EtherFiNodesManager.sol`

The code modifications examined during this review were introduced initially in the following pull request - PR#302 . Later they were moved into a consolidated PR with the other items in this report which is the following and final one - PR#333

EtherFiRestaker Hot Fix

Project Overview

This report presents the findings of a manual code review for the **EtherFiRestaker Hot Fix** audit within the **EtherFi** core contracts. The work was undertaken from **November 21st to November 24th 2025**

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

- `src/EtherFiRestaker.sol`

The code modifications examined during this review were introduced initially in the following pull request - PR#321**.** Later they were moved into a consolidated PR with the other items in this report which is the following and final one - PR#333

Informational Issues

I-01. EL reported shares might be lower than the actually withdrawn

Description: Sometimes the reported shares from EL might be lower than the actually withdrawn ones because of the slashing factor being subject to change when slashableUntil >= uint32(block.number).

From EigenLayer's code comments:
"Note that if the slashableUntil block is the current block or in the future, then the slashing factors are still subject to change before the withdrawal is complete, which may result in fewer shares being withdrawn."

Recommendations: The team should be aware of the above behaviour when reading data from EL

Customer's response: Acknowledged

Fix Review: Acknowledged

I-02. Calculating pending withdrawals from eigenlayer could be optimized and reduce potential of DOS for multiple withdraws

Description: In the update to the getAmountInEigenLayerPendingForWithdrawals() function, the approach to fetching pending withdraws is as follows:

- call eigenLayerDelegationManager.getQueuedWithdrawals()
- compute the hash for each withdrawal
- call eigenLayerDelegationManager.getQueuedWithdrawal()

The first two steps are redundant and can be optimized, by calling eigenLayerDelegationManager.getQueuedWithdrawalRoots() (also used internally by getQueuedWithdrawals()) . This way the function gets all roots, loops through them and calls eigenLayerDelegationManager.getQueuedWithdrawal(), no need to re-calculate each withdrawal on every step.

Another note to the team is that if there are too many withdraws, the function might get gas intensive and cause potential problems in the future if it gets used in state modifying flows

Recommendation: Call getQueuedWithdrawalRoots() instead of getQueuedWithdrawals() and use the already calculated roots to optimize the function

Customer's response: Acknowledged

Fix Review: Acknowledged

PR Consolidation

Project Overview

This report presents the findings of a manual code review for the **PR Consolidation** audit within the **EtherFi** core contracts. The work was undertaken on **December 4th 2025**

PR#333 is the final PR that consolidates all audited items from this report into one unified PR, so that the EthFi team can merge and deploy the interdependent changes at once

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.