certora

Security Assessment

ether.fi – Prelude V3

May-June 2025

Prepared for ether.fi

Table of content

Project Summary

Project Scope

Project Name	Initial Commit Hash	Latest Commit Hash	Platform	Start Date	End Date
Prelude V3	Hash	Hash	EVM	15/04/2025	30/04/2025
Prelude V3 Extension	Hash	Hash	EVM	19/06/2025	20/06/2025

Project Overview

This document describes the manual code review of **ether.fi Prelude V3**. The work was a **8 day** effort undertaken from **15th May** to **30th May** and **19th June to 20th June**

During the manual audit, 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	1	1	1
High	-	-	-
Medium	4	4	4
Low	1	1	1
Informational	6	6	6
Total	12	11	10

Severity Matrix

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

certora

ether.fi

Detailed Findings

ID	Title	Severity	Status
C-01	Missing access control in StakingManager::_authorizeUpgrade allows arbitrary upgrades	Critical	Fixed
M-01	Incorrect empty tokens array passed to completeQueuedWithdrawal, causing revert	Medium	Fixed
M-02	Incorrect slashability check in completeQueuedWithdrawals() causes withdrawals to be skipped	Medium	Fixed
M-03	Incorrect task cleanup in _handleValidators() prevents execution of SendExitRequests tasks	Medium	Fixed
M-04	CompleteQueuedWithdrawals will revert for a large number of queued withdrawals	Medium	Fixed
L-01	Lack of EtherFiNode address validation enables arbitrary call forwarding	Low	Fixed

Prelude V3 Review

Project Overview

This report presents the findings of a manual code review for **Prelude V3** within the **EtherFi smart-contracts** repository. The work was undertaken from **15th May to 30th May**

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

- src/AssetRecovery.sol
- src/EETH.sol
- src/EtherFiAdmin.sol.sol
- src/EtherFiNode.sol
- src/EtherFiNodesManager.sol
- src/EtherFiOracle.sol
- src/LiquidityPool.sol
- src/StakingManager.sol
- src/TNFT.sol
- src/WeETH.sol
- helpers/*
- libraries/*
- archive/*

The code modifications examined during this review were implemented in the following pull request - PR#256

Critical Severity Issues

C-01 Missing access control in StakingManager::_authorizeUpgrade allows arbitrary upgrades

Severity: **Critical**	Impact: **High**	Likelihood: **High**
Files: StakingManager.sol	Status: Fixed	

Description: The StakingManager contract is a UUPS proxy. In such contracts, the `_authorizeUpgrade` function is the only protection guarding against unauthorized upgrades.

This implementation is completely open — it lacks any access control (e.g. `onlyOwner`, `onlyRole`, etc.). As a result, any caller can upgrade the contract to an arbitrary implementation using `upgradeToAndCall()`.

The OpenZeppelin UUPS documentation clearly states:

"*This function should revert when msg.sender is not authorized... Normally, this function will use an access control modifier such as onlyOwner.*"

Recommendations: Add a proper access control modifier (e.g., `onlyOwner` or `onlyProtocolUpgrader(msg.sender)`) to `_authorizeUpgrade` to restrict who can perform upgrades

Customer's response: Fixed in commit

Fix Review: Fixed

Medium Severity Issues

M-01 Incorrect empty tokens array passed to completeQueuedWithdrawal, causing revert

Severity: **Medium**	Impact: **High**	Likelihood: **High**
Files: EtherFiNode.sol	Status: Fixed	

Description: The completeQueuedWithdrawals() function in EtherFiNode passes an empty tokens array to delegationManager.completeQueuedWithdrawal(), under the assumption that token population is unnecessary for Beacon ETH withdrawals. However, the DelegationManager enforces a strict check that tokens.length == withdrawal.strategies.length. Since in order to withdraw Beacon ETH we have to pass the beaconChainETHStrategy strategy as an input this would cause a length mismatch between the tokens passed and the strategies so the call will revert.

This makes the current implementation non-functional for completing queued withdrawals, even when the conditions are otherwise valid

```javascript
// DelegationManager.sol

function _completeQueuedWithdrawal(
        Withdrawal memory withdrawal,
        IERC20[] calldata tokens,
        bool receiveAsTokens
    ) internal {
        require(tokens.length == withdrawal.strategies.length, InputArrayLengthMismatch());
        .....
    }
```

Recommendations: Populate the tokens array accordingly

Customer's response: Fixed in commit

Fix Review: Fixed

M-02 Incorrect slashability check in completeQueuedWithdrawals() causes withdrawals to be skipped

Severity: **Medium**	Impact: **High**	Likelihood: **High**
Files: EtherFiNode.sol	Status: Fixed	

Description: The EtherFiNode::completeQueuedWithdrawals() function uses the following logic to determine whether a queued withdrawal should be processed:

```
JavaScript
if (uint32(block.number) > slashableUntil) continue;
```

This condition is inverted. The intended behavior, as correctly implemented in DelegationManager::_completeQueuedWithdrawal(), is to allow withdrawals after the slashableUntil block has passed:

require(uint32(block.number) > slashableUntil, WithdrawalDelayNotElapsed());

In the current EtherFiNode implementation, once block.number > slashableUntil, the withdrawal is skipped, whereas it should instead be processed at that point.

As a result, queued withdrawals that are ready for completion are perpetually skipped.

Recommendations: Invert the check

Customer's response: Fixed in commit

Fix Review: Fixed

M-03 Incorrect task cleanup in _handleValidators() prevents execution of SendExitRequests tasks

Severity: **Medium**	Impact: **High**	Likelihood: **High**
Files: EtherFiAdmin.sol	Status: Fixed	

Description: In EtherFiAdmin::_handleValidators(), the following line was mistakenly removed:

```
JavaScript
_enqueueValidatorManagementTask(_reportHash, _report.liquidityPoolValidatorsToExit,
emptyTimestamps, TaskType.SendExitRequests);
```

and instead, the line related to TaskType.ProcessNodeExit was retained:

```
JavaScript
_enqueueValidatorManagementTask(_reportHash, _report.exitedValidators,
_report.exitedValidatorsExitTimestamps, TaskType.ProcessNodeExit);
```

This contradicts the cleanup that happened in executeValidatorManagementTask(), where the TaskType.ProcessNodeExit and TaskType.MarkBeingSlashed branches were removed—implying those tasks are deprecated.

As a result, TaskType.SendExitRequests, which is supposed to be active and functional, is no longer being enqueued. Additionally, in EtherFiOracle::generateReportHash(), the report continues to hash the exitedValidatorsExitTimestamps, which is now unused. Instead, it should hash liquidityPoolValidatorsToExit, as that is what the remaining valid task type (SendExitRequests) actually uses.

Recommendations: Re-add the call to enqueue TaskType.SendExitRequests in _handleValidators(). Remove the call to enqueue TaskType.ProcessNodeExit, as that task type is deprecated. In EtherFiOracle::generateReportHash(), update the report hashing logic to remove exitedValidatorsExitTimestamps and include liquidityPoolValidatorsToExit instead.

Customer's response: Fixed in this commit

Fix Review: Fixed

M-04 CompleteQueuedWithdrawals will revert for a large number of queued withdrawals

Severity: **Medium**	Impact: **High**	Likelihood: **High**
Files: EtherFiNode.sol	Status: Fixed	

Description: If the node queued a large number of withdrawals, then there would be no way through the regular API to free the funds, that's because completeQueuedWithdrawals always iterates over all the queued withdrawals which might cause out of gas error.

JavaScript

```
    function completeQueuedWithdrawals(bool receiveAsTokens) external {
       ...

        (IDelegationManager.Withdrawal[] memory queuedWithdrawals, ) =
delegationManager.getQueuedWithdrawals(address(this));
        for (uint256 i = 0; i < queuedWithdrawals.length; i++) {
            delegationManager.completeQueuedWithdrawal(queuedWithdrawals[i], tokens,
receiveAsTokens);
       ...
}
```

Recommendations: Add an API which allows to specify specific withdrawals to be completed.

Customer's response: Fixed in this commit

Fix Review: Fixed

Low Severity Issues

L-01 EtherFiNodesManager::createEigenPod() cannot be used outside of initial instantiation flow

Severity: **Low**	Impact: **High**	Likelihood: **High**
Files: EtherFiNodesManager.sol	Status: Fixed	

Description: The function EtherFiNodesManager::createEigenPod() is intended to allow deferred creation of an EigenPod for an already-deployed EtherFiNode. However, this function currently cannot be used because the system does not yet associate the validator ID with the corresponding EtherFiNode contract address until StakingManager::createBeaconValidators() is called.

```javascript
function createEigenPod(uint256 id) public onlyAdmin returns (address) {
        return IEtherFiNode(etherfiNodeAddress(id)).createEigenPod();
    }
```

The problem arises due to the lookup logic in etherfiNodeAddress(uint256 id), which returns address(0) for pubkey hashes that haven't been explicitly mapped yet via etherFiNodeFromPubkeyHash[bytes32(id)]. Since calling createEigenPod() relies on etherfiNodeAddress(id), it ends up calling IEtherFiNode(address(0)).createEigenPod(), which will revert.

This effectively breaks the intended second path of creating nodes where an admin could:

1. Call instantiateEtherFiNode(_createEigenPod = false) to deploy the EtherFiNode.
2. Later call createEigenPod() separately.

Because createBeaconValidators() (which links the ID to the node address) requires the node to already have an EigenPod, this flow becomes circular and cannot complete.

Recommendations: Remove or deprecate the createEigenPod() path, enforcing EigenPod creation only through instantiateEtherFiNode(_createEigenPod = true)

Customer's response: Fixed in this commit

Fix Review: Fixed

Informational Issues

I-01. Lack of EtherFiNode address validation enables arbitrary call forwarding

Description: The functions EtherFiNodesManager::forwardExternalCall(address[] calldata nodes) and forwardEigenPodCall(address[] calldata etherFiNodes) allow a trusted forwarder to relay calls to EtherFiNode contracts. However, neither function verifies whether the provided node addresses actually correspond to valid, registered EtherFiNodes—i.e., entries stored in etherFiNodeFromPubkeyHash or DEPRECATED_etherfiNodeAddress.

Because of this, a malicious or compromised forwarder could provide arbitrary addresses as nodes and successfully relay calls to contracts that are not valid EtherFiNodes. This breaks the intended security model where forwarded calls should only be allowed to known and registered node contracts.

While the caller is restricted via onlyCallForwarder, the lack of address verification still allows abuse within that role. Moreover, the allowed selectors and targets mechanism provides a superficial restriction, but if the underlying contract is not a valid EtherFiNode, forwarding functionality can still be misused.

Recommendation: Add validation in both forwardExternalCall(address[] calldata nodes) and forwardEigenPodCall(address[] calldata etherFiNodes) to ensure that all nodes[i] are valid EtherFiNodes

Customer's response: Fixed in this commit.

Fix Review: Fixed

I-02. EtherFiNodesManager can be paused, but pausing has no effect

Description: The EtherFiNodesManager contract includes pauseContract() and unPauseContract() functions gated by appropriate roles (PROTOCOL_PAUSER and PROTOCOL_UNPAUSER). These functions invoke OpenZeppelin's _pause() and _unpause() methods to toggle the contract's paused state.

However, none of the external or public functions in EtherFiNodesManager are guarded with the whenNotPaused or whenPaused modifiers. This means that toggling the pause state has no actual effect on the contract's behavior. All functions remain callable regardless of whether the contract is paused or not. The same applies to the StakingManager as well.

Recommendation: Apply the whenNotPaused modifier to functions that should be disabled during an emergency pause

Customer's response: Fixed in this commit & commit. Some methods are not yet guarded:

1. Admin only
    a. linkLegacyValidatorIds
    b. updateAllowedForwardedExternalCalls
    c. updateAllowedForwardedEigenpodCalls

Fix Review: Fixed

I-03. EtherFiNode::sweepFunds function inaccessible through standard flow

Description: The sweepFunds() function in EtherFiNode is designed to forward any leftover ETH balance held by the node to the liquidityPool. It is restricted to the onlyAdmin role, which is intended to be held by contracts like EtherFiNodesManager and StakingManager.

However, EtherFiNodesManager does not invoke this function directly anywhere in the codebase, making it inaccessible through the expected administrative path. While it's technically still possible to call sweepFunds() via the EtherFiNode.forwardExternalCall() function (by targeting the node itself as to = address(this) and encoding the sweepFunds() call), this is non-obvious and inconvenient.

Recommendation: Explicitly expose a sweepFunds call path in EtherFiNodesManager

Customer's response: Fixed in this commit

Fix Review: Fixed

I-04. Inconsistent upgrade authorization in EtherFiNodesManager

Description: The _authorizeUpgrade() function in the EtherFiNodesManager contract uses onlyOwner for access control. In contrast, all other upgradable contracts in the system enforce upgrade authorization using a centralized role (PROTOCOL_UPGRADER role) from the RoleRegistry.

Recommendation: Replace onlyOwner in _authorizeUpgrade() with the standardized role check

Customer's response: Partially fixed in this commit

Fix Review: Partially fixed

I-05. LegacyStakingManagerState Has Incorrect Length (15 Instead of 14)

Description: The StakingManager contract defines a struct named LegacyStakingManagerState containing a single member array that represents the deprecated state variables. However, the annotated field layout mapping only contains 14 distinct fields. The array legacyState is oversized by one slot

Recommendation: Update the array size to be 14 elements instead of 15

Customer's response: Fixed in this commit

Fix Review: Fixed

I-06. queueWithdrawal API can be adapted to support more functionality.

Description: Currently queueWithdrawal does not allow arrays as inputs for the API, If in the future such API would be supported that update would be breaking.

Recommendation: Change queueWithdrawal input to be an array, and inside check that the length of that array is 1

Customer's response: Fixed in this commit

Fix Review: Fixed

Prelude V3 Extension Review

Project Overview

This report presents the findings of a manual code review for **Prelude V3 Extension** within the **EtherFi smart-contracts** repository which deprecates some unused variables. The work was undertaken from **19th June to 20th June**

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

- src/EtherFiAdmin.sol.sol
- src/EtherFiNodesManager.sol
- src/StakingManager.sol

The code modifications examined during this review were implemented in the following pull request - PR#267

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.