Security Assessment

ether.fi – Core Contracts Combined Audit Report

September - October 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
EL-triggerable exits and validator consolidation	218a983	89f0e17	EVM	01/09/2025	09/09/2025
Instant withdraws of stEth	72b2dcf	037da63	EVM	25/09/2025	29/09/2025
Adapter contract for weETH withdrawals	f9f7b156	f9f7b156	EVM	17/10/2025	17/10/2025

Project Overview

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

The work was a 13 day effort undertaken between **01/09/2025** and **17/10/2025**

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	1	1	1
Medium	1	1	1
Low	8	8	3
Informational	7	7	1
Total	17	17	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
EL-triggerable exits and validator consolidation			
L-01	Incorrect rateLimiter consumption for full exits	Low	Acknowledged
L-02	Some withdrawals may not be processed, leading to overconsumption of the rateLimiter and loss of fees	Low	Acknowledged
L-03	Fee overpayment will not be returned to the caller of requestConsolidation and requestExecutionLayerTriggeredWithdrawal	Low	Acknowledged
L-04	setRemaining on bucketLimiter does not cap to capacity	Low	Fixed
L-05	Invalid consolidation requests can be made, leading to loss of fees	Low	Acknowledged
L-06	Wrong target address defined in DiscoverCurrentWhitelist script	Low	Fixed
Instant withdraws of stEth			
H-01	Redemptions with stEth will cause reward dilution and permanent locking of rewards	High	Fixed

M-01	Approvals are spend, without being used	Medium	Fixed
L-01	Low watermark threshold not validated properly	Low	Fixed
L-02	canRedeem will return false, even if redemptions are possible	Low	Acknowledged
Adapter contract for weETH withdrawals			
-	-	-	-

EL-triggerable exits and validator consolidation

Project Overview

This report presents the findings of a manual code review for the **EL-triggerable exits and validator consolidation** audit within the **EtherFi** core contracts. The work was undertaken from **September 1st to September 9th 2025**

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

- `lib/BucketLimiter.sol`
- `src/EtherFiNode.sol`
- `src/EtherFiNodesManager.sol`
- `src/EtherFiRateLimiter.sol`
- `src/StakingManager.sol`
- `src/eigenlayer-interfaces/IEigenPod.sol`
- `src/interfaces/IEtherFiNode.sol`
- `src/interfaces/IEtherFiNodesManager.sol`
- `src/interfaces/IEtherFiRateLimiter.sol`
- `src/interfaces/IStakingManager.sol`

The code modifications examined during this review were implemented in the following pull request - PR#278

Low Severity Issues

L-01 Incorrect rateLimiter consumption for full exits

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

Description: The following formula is used in getTotalEthRequested() to determine the totalGwei:

```
JavaScript
uint256 gweiAmount = requests[i].amountGwei == 0 ? FULL_EXIT_GWEI :
uint256(requests[i].amountGwei);
```

However, this incorrectly assumes that every full exit is withdrawing 2_048_000_000_000.

As a result, full withdrawals will overconsume an incorrect amount of gwei for full exits.

Recommendation: Consider supplying only the exact amounts during a withdrawal for an accurate consumption.

Customer's response: *"This is a known assumption and a non-blocker, after we finish consolidating this will be much more close to accurate. Also Eigenpod does not track individual validator balance on the execution layer so there is no way to fetch them. They only sum the last known balances when doing a checkpoint proof. "*

Fix Review: Acknowledged

certora

ether.fi

L-02 Some withdrawals may not be processed, leading to overconsumption of the rateLimiter and loss of fees		
Severity: **Low**	Impact: **High**	Likelihood: **High**
Files: EtherFiNodesManager.sol	Status: Acknowledged	

Description: In the eigenPod the following conditions are not explicitly validated, but may lead to a failed processing of the withdrawal request:

```
JavaScript
   /// Some requirements that are NOT checked by the pod:
     /// - request.pubkey MUST be a valid validator pubkey
    /// - request.pubkey MUST belong to a validator whose withdrawal credentials are this pod
    /// - If request.amount is for a partial exit, the validator MUST have 0x02 withdrawal
credentials
    /// - If request.amount is for a full exit, the validator MUST NOT have any pending
partial exits
    /// - The validator MUST be active and MUST NOT have initiated exit
```

As a result, if a withdrawal request is sent, but does not satisfy any of these conditions, it will not be processed, but still consumed from the rateLimiter and send the msg.value.

Recommendation: Consider implementing additional validation, to ensure all requests are processed.

Customer's response: *"This is also a non-blocker, since this will be our fault if it happens and the admin of the EtherFiRateLimiter could always fix it if we screwed up. Also there is no way we can know if it failed on the beacon layer from the execution layer because we cannot query that state. It is up to us to call this method correctly"*

Fix Review: Acknowledged

L-03 Fee overpayment will not be returned to the caller of requestConsolidation and requestExecutionLayerTriggeredWithdrawal

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

Description: In the eigenPod, any fee overpayment is returned to the msg.sender:

```
JavaScript
  // Refund remainder of msg.value
        if (remainder > 0) {
            Address.sendValue(payable(msg.sender), remainder);
        }
```

The eigenPod::requestWithdrawal() and eigenPod::requestConsolidation() are invoked by the EtherFiNode contract, so it is going to be the receiver of the remainder. However, in the sweepFunds() function, any balance that is held in the Node is transferred to the liquidityPool:

```
JavaScript
   function sweepFunds() external onlyEtherFiNodesManager returns (uint256 balance) {
        uint256 balance = address(this).balance;
        if (balance > 0) {
            (bool sent,) = payable(address(liquidityPool)).call{value: balance, gas:
20000}("");
            if (!sent) revert TransferFailed();
            emit FundsTransferred(address(liquidityPool), balance);
        }
        return balance;
    }
```

As a result, any fee overpayment will be sent to the liquidityPool instead of the original payer of the fee.

Recommendation: Consider returning the fee to the payer, instead of the liquidity pool.

Customer's response: *"This is known and a non-blocker. If this happens it is our fault and the sum (currently 1 gwei/request) accrued in liquidityPool and can be claimed by the admin eventually."*

Fix Review: Acknowledged

L-04 setRemaining on bucketLimiter does not cap to capacity

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

Description: The setRemaining() function in EtherFiRateLimiter:

```
JavaScript
function setRemaining(bytes32 id, uint64 remaining) external onlyAdmin {
        if (!limitExists(id)) revert UnknownLimit();

        BucketLimiter.setRemaining(limits[id], remaining);
        emit RemainingUpdated(id, remaining);
    }
```

Uses the underlying BucketLimiter library which defines setRemaining like this:

```
JavaScript
function setRemaining(Limit storage limit, uint64 remaining) internal {
        refill(limit);
        limit.remaining = remaining;
    }
```

The issue is that the remaining field is set directly to the new value, but there is no check on the capacity as in all other updating functions in the library. This allows to set remaining to a value which is beyond the max capacity, which should not be allowed

Recommendation: Cap the remaining value up to the size of capacity

```JavaScript
function setRemaining(Limit storage limit, uint64 remaining) internal {
        refill(limit);
        limit.remaining = remaining > limit.capacity ? limit.capacity : remaining ;
    }
```

Customer's response: Fixed in commit [f50b87]

Fix Review: Fixed

L-05 Invalid consolidation requests can be made, leading to loss of fees

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

Description: The following conditions are required for the correct processing of a consolidation request but are not checked by the eigenPod:

```
JavaScript
      /// Some requirements that are NOT checked by the pod:
    /// - If request.srcPubkey == request.targetPubkey, the validator MUST have 0x01
credentials
    /// - If request.srcPubkey != request.targetPubkey, the target validator MUST have 0x02
credentials
    /// - Both the source and target validators MUST be active on the beacon chain and MUST
NOT have
    ///   initiated exits
    /// - The source validator MUST NOT have pending partial withdrawal requests (via
`requestWithdrawal`)
    /// - If the source validator is slashed after requesting consolidation (but before
processing),
    ///   the consolidation will be skipped.
```

As a result, if a request does not satisfy any of these requirements, the processing of that request will be failed, but the fee will still be paid.

For example the function assumes that *"eigenlayer will revert if all validators don't belong to the same pod"* and uses the srcPubkey of the first request to get the eigenPod node. However in eigenlayer only the targetPubKey is verified against the Pod. This means that requestConsolidation() can be called with source validator keys that do not belong to the Pod

and eigenlayer will still **not** revert, making the transaction succeed. This will subsequently fail silently on the beacon chain

Recommendation: Consider adding additional validation in order to avoid sending invalid requests.

Customer's response: *"This probably might be an issue but since only Admin can call this function, this will be our fault if it happens, thus a non–blocker."*

Fix Review: Acknowledged

L-06 Wrong target address defined in DiscoverCurrentWhitelist script

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

Description: Inside the getAllTargets() function of the script the LiquidityPool address is wrong. It is set to 0x35F4f28A8d3Ff20EEd10e087e8F96Ea2641E6AA2 (which is the contract of another protocol), while it should be 0x308861A430be4cce5502d0A12724771Fc6DaF216 as pointed in the deployed addresses in the docs

Additionally consider:

- Reducing the targets array size from 25 to 8 to reduce the unnecessary loops
- Inside CleanupOldWhitelist script verifyCleanup() is never run at the end of run()

Recommendation: Address the above inconsistencies

Customer's response: *"Fixed the liquidity Pool address and targets size array issue. The `verifycleanup()` is supposed to be called explicitly for this script. There is a bigger deployment and verification script, that does the actual cleanup, deployment and upgrade which is this."*

Fix Review: Fixed in 581c602

Informational Issues

I-01.Revert on zero address when calling updateConsumer()

Description: Inside updateConsumers() function of EtherFiRateLimiter, add sanity check that prevents address(0) to be enabled as consumer

Recommendation: Add sanity check

Customer's response: Acknowledged

Fix Review: Acknowledged

Instant Withdrawals of stEth

Project Overview

This report presents the findings of a manual code review for the **Instant withdrawals of stEth** audit within the **EtherFi** core contracts. The work was undertaken from **September 25th to September 29th 2025**

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

- `src/EtherFiRedemptionManager.sol`
- `src/EtherFiRedemptionManagerTemp.sol`
- `src/EtherFiRestaker.sol`
- `src/LiquidityPool.sol`
- `src/interfaces/ILiquidityPool.sol`

The code modifications examined during this review were implemented in the following pull request - PR#294

High Severity Issues

H-01 Redemptions with stEth will cause reward dilution and permanent locking of rewards

Severity: **High**	Impact: **High**	Likelihood: **High**
Files: EtherFiRedemptionManager.sol	Status: Fixed	

Description: In both the redeemEEth() and redeemWeEth() the output token can be set to Eth or stEth. When stEth is used, the user will transfer in eEth/weEth and the restaker is going to release stEth:

```
JavaScript
IERC20(address(eEth)).safeTransferFrom(msg.sender, address(this), eEthAmount);
...
etherFiRestaker.transferStETH(receiver, eEthAmountToReceiver);
```

This approach locks eEth into the EtherFiRedemptionManager, which will still earn rewards (to the detriment of other stakers), but must **NEVER** be redeemed, as the tokens which were backing it were already transferred to the receiver.

Recommendation: Consider using an alternative approach in order to minimize discrepancies between the Eth and stEth outputs and avoid any stuck tokens. For example:
1) User transfers in the eEth amount

2) eEth is burned from the contract.

3) liquidityPool::rebase() is invoked, which will decrease the totalValueOutOfLp by the stEth amount that the user is going to receive (requires updating the rebase() function to also reduce the value of totalValueOutOfLp)

4) Invoke transferStEth

Customer's response: Fixed in commit 4151ea3d

Fix Review: Fixed

Medium Severity Issues

M-01 Approvals are spend, without being used

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

Description: In both the redeemEEthWithPermit() and redeemWeEthWithPermit() the receiver is passed in as the owner field of the permit function:

```
JavaScript
try eEth.permit(receiver, address(this), permit.value, permit.deadline, permit.v, permit.r, permit.s) {}
```

However, eEth is always transferred from msg.sender. As a result, anytime the receiver is not the msg.sender, the functions will consume the permit of the receiver, but it will try to transfer from msg.sender and revert. Or in case msg.sender has given enough approval prior to that it will consume the receiver permit, without actually utilizing it

Recommendation: Consider replacing back the receiver with msg.sender.

Customer's response: Fixed in commit caab8ae

Fix Review: Fixed

Low Severity Issues

L-01 Low watermark threshold not validated properly

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

Description: The redemption manager implements a watermark check when calling canRedeem() to make sure the balances are not below the expected minimum in order for instant redemptions to be allowed.

```
JavaScript
function canRedeem(uint256 amount, address token) public view returns (bool) {
        uint256 liquidEthAmount = getInstantLiquidityAmount(token);
        if (liquidEthAmount < lowWatermarkInETH(token)) {
            return false;
        }
        uint64 bucketUnit = _convertToBucketUnit(amount, Math.Rounding.Up);
        bool consumable = BucketLimiter.canConsume(tokenToRedemptionInfo[token].limit,
bucketUnit);
        return consumable && amount <= liquidEthAmount;
    }
```

The check is not complete and allows the balances to still be reduced below the watermark after a redemption. Consider the following example:

- Balance in redemption manager is 100 tokens and watermark is 50
- Bob wants to instantly redeem 70 tokens
- Since liquidEthAmount > lowWatermarkInETH(token) it would be allowed

- Bob withdraws 70 tokens and the balances left in the manager are 30, which is below the 50 watermark minimum

Recommendations: To proper way to check for how much can be redeemed, without affecting the minimum watermark, is to check for the difference between liquidEthAmount & lowWatermarkInETH(token) and only then allow a redeem up to that balance:

```javascript
 if (liquidEthAmount <= lowWatermarkInETH(token)) {
            return false;
  }

availableToRedeem = liquidEthAmount - lowWatermarkInETH(token)

if (amount > availableToRedeem) {return false}
```

Customer's response: Fixed in commit caab8ae

Fix Review: Fixed

L-02 canRedeem will return false, even if redemptions are possible

Severity: **Low**	Impact: **High**	Likelihood: **High**
Files: EtherFiRedemptionManager.sol	Status: Acknowledged	

Description: The canRedeem() function will perform all the validations based on the amount of eEth or weEth. However, this will include the amount from the feeShareToStakers and the eEthFeeAmountToTreasury, which do not require any available stEth tokens.

```
JavaScript
  require(canRedeem(eEthAmount, outputToken), "EtherFiRedemptionManager: Exceeded total
redeemable amount");
```

As a result, even if there are enough stEth tokens to satisfy the withdrawal, the canRedeem function will still return false.

Recommendations: Consider performing the canRedeem check only on the eEthAmountToTransfer.

Customer's response: *"The fee is very small should not make significant difference and allows us to keep the implementation the same per token and explaining the calculation is easier"*

Fix Review: Acknowledged

Informational Issues

I-01. Comments are not updated

Description: In the EtherFiRedemptionManager, NatSpec and comments do not mention anything regarding the new StEth logic.

Recommendations: Consider updating the comments.

Customer's response: Fixed in commit caab8ae

Fix Review: Fixed

I-02. Additional sanity check

Description: Upon calling _processETHRedemption() there is the following sanity check:

```
JavaScript
require(liquidityPool.withdraw(address(this), eEthAmountToReceiver) == sharesToBurn, "invalid
num shares burnt");
```

It makes sure that the amount of shares sent is relevant to the amount of assets received. The returned shares of liquidityPool.withdraw() is the result of calling sharesForWithdrawalAmount()

The same sanity check should also be added to _processStETHRedemption(), where even though there are no actual shares being burned, there should still be a check to make sure that the eEthAmountToReceiver being sent as stETH equals to the amount of shares being locked into the redemption manager.

Recommendations: Call sharesForWithdrawalAmount(eEthAmountToReceiver) and do a sanity check before transferring stETH

Customer's response: Acknowledged

Fix Review: Acknowledged

I-03 Users will earn slightly more by splitting their redemptions

Description: The _processStETHRedemption will pay a fee to the stakers, by burning a portion of the shares without transferring out any liquidity. As a result, upon each redemption, the share price will be slightly higher:

```
JavaScript
liquidityPool.burnEEthShares(feeShareToStakers);
```

As a result, users will be able to earn slightly more tokens, by splitting the redemptions.

Recommendations: This is part of the redemption design and it is important that the team is aware of this side effect

Customer's response: Acknowledged

Fix Review: Acknowledged

I-04 Attackers may delay normal withdrawals

Description:

The following flow will be used for normal withdrawals from the stEth strategy:

1) EtherFiRestaker::queueWithdrawals()

2)EtherFiRestaker::completeQueuedWithdrawals()

3) EtherFiRestaker::stEthRequestWithdrawal()

4) EtherFiRestaker::stEthClaimWithdrawals()

However, anyone will be able to disrupt this flow, by redeeming stEth between steps 2 and 3, potentially delaying the normal withdrawal flow.

Recommendations: Consider executing steps 2 and 3 in the same transaction to avoid such issues.

Customer's response: *"We will perform 2 and 3 in 1 tx if we need to perform this flow"*

Fix Review: Acknowledged

Adapter contract for weETH withdrawals

Project Overview

This report presents the findings of a manual code review for the **Adapter contract for weETH withdrawals** audit within the **EtherFi** core contracts. The work was undertaken on **October 17th 2025**

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

- `src/helpers/WeETHWithdrawAdapter.sol`

The code modifications examined during this review were implemented in the following pull request - PR#287

Informational Issues

I-01. Adapter admin role not used

Description: The WeETHWithdrawAdapter contract defined the WEETH_WITHDRAW_ADAPTER_ADMIN_ROLE as a constant variable, but it never uses it

Recommendations: Remove the redundant role

Customer's response: *"We will keep it for future usage"*

Fix Review: Acknowledged

I-02. Validate if the permit amount is sufficient

Description: The WeETHWithdrawAdapter contract defines a requestWithdrawWithPermit() function, which allows providing approval and executing a withdrawal request in one transaction

```
JavaScript
function requestWithdrawWithPermit(
        uint256 weETHAmount,
        address recipient,
        PermitInput calldata permit
    ) external whenNotPaused returns (uint256 requestId) {
....
}
```

In addition to the permit data, weETHAmount is also provided, which is the amount that should be transferred from the caller. And since the actual allowance is contained in permit.value it makes sense to do a top level check that weETHAmount is not above the allowance value and revert early

Recommendation: Add the following top level validation:

```
JavaScript
if (weETHAmount > permit.value ) revert("Insufficient permit value");
```

Customer's response: Acknowledged

Fix Review: Acknowledged

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.