Security Assessment

ether.fi – EigenLayer Slashing

March 2025

Prepared for ether.fi

Table of Contents

Project Summary

Project Scope

Project Name	Repository (link)	Commit Hashes	Platform
EtherFi smart contracts	etherfi-protocol/cash-v3	Audit start: PR 3 at 146c1831 Audit end: PR 246 at 7b2c777a	EVM

Project Overview

This document describes the manual code review of PR 218, PR 239 and PR 246 related to EigenLayer's Slashing Feature Release applied to ether.fi's v2.49 release.

The work was a 5 day-effort undertaken from 08/01/2025 to 21/03/2025.

The following contract list is included in our scope:

1. src/EtherFiNode.sol
2. src/EtherFiNodesManager.sol
3. src/helpers/EtherFiViewer.sol
4. src/EtherFiRestaker.sol
5. src/EtherFiRedemptionManager.sol
6. src/Liquifier.sol

The team performed a manual audit of all 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	4	4	1 full, 2 partial, 1 unfixed
Total	4	4	1 full, 2 partial, 1 unfixed

Severity Matrix

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

Informational Severity Issues

I-01. Delete rogue `console.log` imports

These shouldn't be deployed in production

Affected code:

- src/EtherFiNode.sol

```
# File: src/EtherFiNode.sol

EtherFiNode.sol:14: import "forge-std/console.sol";
```

- src/EtherFiNodesManager.sol

```
# File: src/EtherFiNodesManager.sol

EtherFiNodesManager.sol:18: import "forge-std/console.sol";
```

Certora's fix review: Ok, fixed.

I-02. Draft Dependencies

Draft contracts have not received adequate security auditing or are liable to change with future developments.

Affected code:

- src/EtherFiRestaker.sol

```
# File: src/EtherFiRestaker.sol

EtherFiRestaker.sol:9: import
"@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol";
```

- src/Liquifier.sol

```
# File: src/Liquifier.sol

Liquifier.sol:9: import
"@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol";
```

Certora's fix review: This remained unfixed.

I-03. Unused `error` definition

Note that there may be cases where an error superficially appears to be used, but this is only because there are multiple definitions of the error in different files. In such cases, the error definition should be moved into a separate file. The instances below are the unused definitions.

Affected code:

- src/EtherFiNode.sol

```
# File: src/EtherFiNode.sol

EtherFiNode.sol:45:     error CallFailed(bytes data);
```

- src/EtherFiNodesManager.sol

```
# File: src/EtherFiNodesManager.sol

EtherFiNodesManager.sol:105:     error NonZeroAddress();
```

- src/EtherFiRestaker.sol

```
# File: src/EtherFiRestaker.sol

EtherFiRestaker.sol:54:     error StrategyShareNotEnough();

EtherFiRestaker.sol:57:     error NotRegistered();

EtherFiRestaker.sol:58:     error WrongOutput();
```

- src/Liquifier.sol

```
# File: src/Liquifier.sol

Liquifier.sol:96:     error StrategyShareNotEnough();
```

```
Liquifier.sol:101:     error NotRegistered();

Liquifier.sol:102:     error WrongOutput();
```

Certora's fix review:

- Fixed for EtherFiNode.sol and EtherFiNodesManager.sol
- Unfixed for EtherFiRestaker.sol and Liquifier.sol

I-04. Event is never emitted

The following are defined but never emitted. They can be removed to make the code cleaner.

Affected code:

- src/EtherFiNodesManager.sol

```
# File: src/EtherFiNodesManager.sol

EtherFiNodesManager.sol:79:     event FundsWithdrawn(uint256 indexed
_validatorId, uint256 amount);

EtherFiNodesManager.sol:81:     event NodeExitRequestReverted(uint256
_validatorId);

EtherFiNodesManager.sol:83:     event NodeEvicted(uint256 _validatorId);
```

- src/Liquifier.sol

```
# File: src/Liquifier.sol

Liquifier.sol:92:     event CompletedQueuedWithdrawal(bytes32
_withdrawalRoot);

Liquifier.sol:93:     event QueuedStEthWithdrawals(uint256[] _reqIds);

Liquifier.sol:94:     event CompletedStEthQueuedWithdrawals(uint256[]
_reqIds);
```

Certora's fix review:

- Fixed for EtherFiNodesManager.sol
- Unfixed for Liquifier.sol

Appendix: Pull Request Risk Analysis

Updating EigenLayer libraries and interfaces after PR 679 is merged

While it could be thought that the libraries and interfaces are fixed, this isn't actually the case. The PR https://github.com/Layr-Labs/eigenlayer-contracts/pull/679 is still open and changing. As an example, this is the diff between `src/eigenlayer-libraries/SlashingLib.sol` and `SlashingLib.sol` in the PR:

```
diff --git a/src/eigenlayer-libraries/SlashingLib.sol
b/src/eigenlayer-libraries/SlashingLib.sol
index 56063cf..b6ce241 100644
--- a/src/eigenlayer-libraries/SlashingLib.sol
+++ b/src/eigenlayer-libraries/SlashingLib.sol
@@ -2,26 +2,28 @@
 pragma solidity ^0.8.27;

 import "@openzeppelin/contracts/utils/math/Math.sol";
-import
"@openzeppelin-upgradeable/contracts/utils/math/SafeCastUpgradeable.sol";
+import "@openzeppelin-upgrades/contracts/utils/math/SafeCastUpgradeable.sol";

-/// @dev the stakerScalingFactor and operatorMagnitude have initial default
values to 1e18 as "1"
-/// to preserve precision with uint256 math. We use `WAD` where these
variables are used
-/// and divide to represent as 1
+/// @dev All scaling factors have `1e18` as an initial/default value. This
value is represented
+/// by the constant `WAD`, which is used to preserve precision with uint256
math.
+///
+/// When applying scaling factors, they are typically multiplied/divided by
`WAD`, allowing this
+/// constant to act as a "1" in mathematical formulae.
 uint64 constant WAD = 1e18;

 /*
  * There are 2 types of shares:
-  *      1. depositShares
+  *      1. deposit shares
```

```
  *          - These can be converted to an amount of tokens given a strategy
  *              - by calling `sharesToUnderlying` on the strategy address
(they're already tokens
  *              in the case of EigenPods)
- *          - These live in the storage of EPM and SM strategies
- *      2. shares
+ *          - These live in the storage of the EigenPodManager and individual
StrategyManager strategies
+ *      2. withdrawable shares
  *          - For a staker, this is the amount of shares that they can
withdraw
- *          - For an operator, this is the sum of its staker's withdrawable
shares
- *
- * Note that `withdrawal.scaledShares` is scaled for the
beaconChainETHStrategy to divide by the beaconChainScalingFactor upon queueing
- * and multiply by the beaconChainScalingFactor upon withdrawal
+ *          - For an operator, the shares delegated to them are equal to the
sum of their stakers'
+ *            withdrawable shares
+ *
+ * Along with a slashing factor, the DepositScalingFactor is used to convert
between the two share types.
  */
 struct DepositScalingFactor {
     uint256 _scalingFactor;
@@ -29,7 +31,6 @@ struct DepositScalingFactor {

 using SlashingLib for DepositScalingFactor global;

-// TODO: validate order of operations everywhere
 library SlashingLib {
     using Math for uint256;
     using SlashingLib for uint256;
@@ -70,14 +71,10 @@ library SlashingLib {
     }

     function scaleForQueueWithdrawal(
-        uint256 sharesToWithdraw,
-        uint256 slashingFactor
+        DepositScalingFactor memory dsf,
+        uint256 depositSharesToWithdraw
     ) internal pure returns (uint256) {
-        if (slashingFactor == 0) {
```

```
-             return 0;
-         }
-
-         return sharesToWithdraw.divWad(slashingFactor);
+         return depositSharesToWithdraw.mulWad(dsf.scalingFactor());
      }

      function scaleForCompleteWithdrawal(uint256 scaledShares, uint256
slashingFactor) internal pure returns (uint256) {
```

It'll be very important to not deploy with outdated libraries and interfaces.

ether.fi's response: Interfaces and libraries have been updated on the latest branch https://github.com/etherfi-protocol/smart-contracts/pull/239

Certora's response: Confirmed.

EtherFiNode.sol

Assumptions about `pendingWithdrawalFromRestakingInGwei` and `completedWithdrawalFromRestakingInGwei`

`pendingWithdrawalFromRestakingInGwei` and `completedWithdrawalFromRestakingInGwei` are getting deprecated and won't be used in the code anymore. They're currently used as internal bookkeeping, but also for state validations.

Without them, new behaviors that were prevented are now allowed.

`processFullWithdraw` can now be called any number of times without the limitation that `completedWithdrawalFromRestakingInGwei` introduced

This is the diff:

```
@@ -213,61 +207,32 @@ contract EtherFiNode is IEtherFiNode, IERC1271 {

     function processFullWithdraw(uint256 _validatorId) external
onlyEtherFiNodeManagerContract ensureLatestVersion {
         updateNumberOfAssociatedValidators(0, 1);
-
-        if (isRestakingEnabled) {
-            // TODO: revisit for the case of slashing
-            require(completedWithdrawalFromRestakingInGwei >= 32 ether / 1
gwei, "INSUFFICIENT_BALANCE");
-            completedWithdrawalFromRestakingInGwei -= uint64(32 ether / 1
gwei);
-        }
     }
```

When `isRestakingEnabled` was enabled, each call to `processFullWithdraw` would induce a subtraction from `completedWithdrawalFromRestakingInGwei`. Now the `EtherFiNodeManagerContract` can call `processFullWithdraw` until `_numAssociatedValidators` gets to 0:

```
File: EtherFiNode.sol
184:     function updateNumberOfAssociatedValidators(uint16 _up, uint16 _down)
public onlyEtherFiNodeManagerContract ensureLatestVersion {
185:         if (_up > 0) _numAssociatedValidators += _up;
186:         if (_down > 0) _numAssociatedValidators -= _down;
187:     }
```

Additionally, the input parameter `uint256 _validatorId` isn't used at all in the function, which means it should either be removed or validated.

ether.fi's response: you are correct that with misuse this variable could get out of sync. These changes are part 1 of a move where we will rely on eigenlayer's validator bookkeeping instead of our own. I don't believe there is a risk of lost funds here as we transition to this new model. In a followup release, we will deprecate `_numAssociatedValidators` entirely
Certora's response: Acknowledged.

`_completeQueuedWithdrawals` is now called without any validation around `receiveAsTokens`

The following is removed entirely:

```
-        if (_receiveAsTokens) {
...
-            require(pendingWithdrawalFromRestakingInGwei >=
uint64(totalAmount / 1 gwei), "NO_PENDING_WITHDRAWAL");
-            pendingWithdrawalFromRestakingInGwei -= uint64(totalAmount / 1
gwei);
-            completedWithdrawalFromRestakingInGwei += uint64(totalAmount / 1
gwei);
-        } else {
...
-            require(pendingWithdrawalFromRestakingInGwei == 0,
"PENDING_WITHDRAWAL_NOT_ZERO");
         }
```

This means that `_receiveAsTokens == false` can be passed if there are pending withdrawals, and that `_receiveAsTokens == true` won't be internally limiting the amount of calls to _completeQueuedWithdrawals or checking if there are actually pending withdrawals.

The risks seem limited given the following code on EigenLayer:

```
File: DelegationManager.sol
517:     function _completeQueuedWithdrawal(
518:         Withdrawal memory withdrawal,
519:         IERC20[] calldata tokens,
520:         bool receiveAsTokens
521:     ) internal {
522:         require(tokens.length == withdrawal.strategies.length,
InputArrayLengthMismatch());
523:         require(msg.sender == withdrawal.withdrawer,
WithdrawerNotCaller());
524:         bytes32 withdrawalRoot = calculateWithdrawalRoot(withdrawal);
525:         require(pendingWithdrawals[withdrawalRoot],
WithdrawalNotQueued());
...
548:         delete pendingWithdrawals[withdrawalRoot];
```

However it also means that if the case of a previously `pendingWithdrawalFromRestakingInGwei == 0`, this call here would now revert inside the DelegationManager.

As a side note, the following check is now redundant due to DelegationManager.sol#L523:

```
File: EtherFiNode.sol
228:         for (uint256 i = 0; i < withdrawals.length; i++) {
- 229:             require(withdrawals[i].withdrawer == address(this) &&
withdrawals[i].staker == address(this), "INVALID");
+ 229:             require(withdrawals[i].staker == address(this), "INVALID");
```

ether.fi's response: This logic was originally added specifically to rescue some funds that had their withdrawals queued incorrectly and we can now return to the original logic. We're fine with the ability to still complete the queued withdrawal for both shares and tokens.

Certora's response: Acknowledged.

splitBalanceInExecutionLayer()

While seemingly complex at first, the function is in fact quite straightforward.

One optimisation would be the following:

```
File: EtherFiNode.sol
331:             for (uint256 i = 0; i < withdrawals.length; i++) {
332:                 assert (withdrawals[i].strategies.length == 1); // only
BeaconETH strategy is used
- 333:                 for (uint256 j = 0; j < shares[i].length; j++) {
- 334:                     _withdrawal_queue += shares[i][j];
+ 334:                     _withdrawal_queue += shares[i][0];
- 335:                 }
336:             }
```

This is due to the fact that we have `assert (withdrawals[i].strategies.length == 1)` and that `shares` is populated as follows in DelegationManager.sol#L924:

```
File: DelegationManager.sol
922:         for (uint256 i; i < totalQueued; ++i) {
923:             withdrawals[i] = queuedWithdrawals[withdrawalRoots[i]];
924:             shares[i] = new uint256[](withdrawals[i].strategies.length);
//@audit-info the array in shares[i]'s size will always be 1 for EtherFi
```

ether.fi's response: We opt to keep the original form and take the small gas optimization hit here for now.

Certora's response: Acknowledged.

withdrawableBalanceInExecutionLayer()

If we assume that only immediate balances can be withdrawn, and that an EtherFi node will never be the `claimer` for EtherFiRestaker, then this is probably correct.
However, I'd like to ask for a double check as if something isn't actually missing here.

```
-    /// @notice balance (wei) of this safe that could be immediately
withdrawn.
-    ///         This only differs from the balance in the safe in the case of
restaked validators
-    ///         because some funds might not be withdrawable yet due to
eigenlayer's queued withdrawal system
+    /// @notice balance (wei) of this safe that could be immediately
withdrawn
+    // This withdrawable balance amount is updated after completion of the
queued withdarawal with `receiveAsToken = True`
     function withdrawableBalanceInExecutionLayer() public view returns
(uint256) {
         uint256 safeBalance = address(this).balance;
-        uint256 claimableBalance = 0;
-        if (isRestakingEnabled) {
-            IDelayedWithdrawalRouter delayedWithdrawalRouter =
IDelayedWithdrawalRouter(IEtherFiNodesManager(etherFiNodesManager).delayedWith
drawalRouter());
-            IDelayedWithdrawalRouter.DelayedWithdrawal[] memory
claimableWithdrawals =
delayedWithdrawalRouter.getClaimableUserDelayedWithdrawals(address(this));
-            for (uint256 x = 0; x < claimableWithdrawals.length; x++) {
-                claimableBalance += claimableWithdrawals[x].amount;
-            }
-        }
-        return safeBalance + claimableBalance;
+        return safeBalance;
     }
```

As a side note, there's a typo:

- `withdarawal` vs `withdrawal`

```
-    // This withdrawable balance amount is updated after completion of the
queued withdarawal with `receiveAsToken = True`
+    // This withdrawable balance amount is updated after completion of the
queued withdrawal with `receiveAsToken = True`
```

_queueEigenpodFullWithdrawal()

The logic was quite simplified.
While we've established before that `numAssociatedValidators` now has an additional path to be desynchronized and forced to 0 or 1 by EtherFiNodeManagerContract if a path in the `EtherFiNodeManager` permits it (like, hypothetically, calling `fullWithdraw` several times), this here delegates most of the logic.

Some simplifications to note:

- `withdrawableShares` is unused and can be removed for clarity

```
- (uint256[] memory withdrawableShares, uint256[] memory depositShares) =
delegationManager.getWithdrawableShares(address(this), strategies);
+ (, uint256[] memory depositShares) =
delegationManager.getWithdrawableShares(address(this), strategies);
```

- typo `withdarwal` vs `withdrawal`

```
-        // Note that the actual withdarwal amount can change if the slashing
happens
+        // Note that the actual withdrawal amount can change if the slashing
happens
```

- Unnecessary external calls to `delegationManager.beaconChainETHStrategy()`:

```
File: EtherFiNode.sol
598:     function _queueEigenpodFullWithdrawal() private returns (bytes32[]
memory fullWithdrawalRoots) {
...
604:         // get the withdrawable amount
605:         IStrategy[] memory strategies = getStrategies();
...
620:         } else {
- 621:             uint256 eigenLayerBeaconStrategyShare =
delegationManager.beaconChainETHStrategy().underlyingToShares(32 ether);
```

```
+ 621:              uint256 eigenLayerBeaconStrategyShare =
strategies[0].underlyingToShares(32 ether);
622:              depositSharesToWithdraw = Math.min(depositShares[0],
eigenLayerBeaconStrategyShare);
623:          }
...
- 630:          strategies[0] = delegationManager.beaconChainETHStrategy();
...
639:      }
```

This is because `getStrategies()` is hardcoded as such:

```
File: EtherFiNode.sol
641:      function getStrategies() public view returns (IStrategy[] memory) {
642:          IDelegationManager delegationManager =
IEtherFiNodesManager(etherFiNodesManager).delegationManager();
643:          IStrategy[] memory strategies = new IStrategy[](1);
644:          strategies[0] = delegationManager.beaconChainETHStrategy();
645:          return strategies;
646:      }
```

and that `beaconChainETHStrategy` is a public constant on `DelegationManager`:

```
File: DelegationManagerStorage.sol
39:      IStrategy public constant beaconChainETHStrategy =
IStrategy(0xbeaC0eeEeeeeEEeEeEEEEeeEEeEeeeEeeEEBEaC0);
```

EtherFiNodesManager.sol

The changes are actually trivial here as only the use of `middlewareTimesIndexes` has been removed.

EtherFiViewer.sol

EigenPod_hasRestaked()

The change is quite big and introduces an assumption:

```
     function EigenPod_hasRestaked(uint256[] memory _validatorIds) external
view returns (bool[] memory _hasRestaked) {
         _hasRestaked = new bool[](_validatorIds.length);
         for (uint256 i = 0; i < _validatorIds.length; i++) {
-            _hasRestaked[i] = _getEigenPod(_validatorIds[i]).hasRestaked();
+            // now every validator within eigenlayer is guaranteed to have
this flag set
+            _hasRestaked[i] =
_getEtherFiNode(_validatorIds[i]).isRestakingEnabled();
         }
     }
```

Given that `hasRestaked` became `__deprecated_hasRestaked`:

```
File: EigenPodStorage.sol
16:     /// @notice DEPRECATED: previously used to track whether a pod had
activated restaking
17:     bool internal __deprecated_hasRestaked;
```

It should be correct, but we'd like to discuss how to be absolutely certain of that.

ether.fi's response: this is only used by our backend services and not by any contract code. We also now restake 100% of our validators. Since the pods can be re-used, it makes more sense to check if restaking is turned on at the etherfiNode level since it is guaranteed to be on at the eigenpod level ever since an upgrade they did last year.

Certora's response: Acknowledged.

EtherFiRestaker.sol

rewardsCoordinator

This variable is set in the constructor, and then the admin can set a claimer.
Given that proceed claims checks that either a claimer is set, or the claimer is the earner: there isn't a frontrunning attack vector here:

```
File: RewardsCoordinator.sol
314:     function _processClaim(RewardsMerkleClaim calldata claim, address
recipient) internal {
315:         DistributionRoot memory root =
_distributionRoots[claim.rootIndex];
316:         _checkClaim(claim, root);
317:         // If claimerFor earner is not set, claimer is by default the
earner. Else set to claimerFor
318:         address earner = claim.earnerLeaf.earner;
319:         address claimer = claimerFor[earner];
320:         if (claimer == address(0)) {
321:             claimer = earner;
322:         }
```

This here seems safe but the whole behavior of the claimer is external to the protocol.

Given that the `processClaim` functionality has a push pattern (`safeTransfer`) instead of a pull pattern (`safeTransferFrom`), no approval seems to be missing.

undelegate()

This is quite complex. The function was too simplified, as if the whole business logic was now held on EigenLayer's DelegationManager. And, indeed, it seems to be the case:

```
File: DelegationManager.sol
392:         uint256[] memory slashingFactors = _getSlashingFactors(staker,
operator, strategies);
393:
394:         // Queue a withdrawal for each strategy independently. This is
done for UX reasons.
395:         for (uint256 i = 0; i < strategies.length; i++) {
396:             IStrategy[] memory singleStrategy = new IStrategy[](1);
397:             uint256[] memory singleDepositShares = new uint256[](1);
398:             uint256[] memory singleSlashingFactor = new uint256[](1);
399:             singleStrategy[0] = strategies[i];
400:             singleDepositShares[0] = depositedShares[i];
401:             singleSlashingFactor[0] = slashingFactors[i];
402:
403:             // Remove shares from staker's strategies and place
strategies/shares in queue.
404:             // The operator's delegated shares are also reduced.
405:             withdrawalRoots[i] = _removeSharesAndQueueWithdrawal({
```

We're still nevertheless sceptical about the following line's removal:

```
_queueWithdrawlsByShares(address(lido), shares);
```

It also seems that, previously, `withdrawalRoots.length == 0` was enforced, but now the opposite is expected, probably simply because the logic was moved.

Still, we're not certain about the correctness of the lack of this call, even if it looks indeed redundant:

```
File: EtherFiRestaker.sol
186:     function
queueWithdrawalsWithParams(IDelegationManagerTypes.QueuedWithdrawalParams[]
memory params) public onlyAdmin returns (bytes32[] memory) {
187:         bytes32[] memory withdrawalRoots =
eigenLayerDelegationManager.queueWithdrawals(params);
```

ether.fi's response: The removal of `_queueWithdrawlsByShares` is based on the implementation of `EigenLayer.DelegationManager`'s `undelegate`. It initiates the withdrawals of all shares (https://github.com/Layr-Labs/eigenlayer-contracts/blob/slashing-magnitudes-fixes/src/contracts/core/DelegationManager.sol#L403-L404). That is, `EtherFiRestaker` does not need to handle it.

Certora's response: Acknowledged.

getTotalPooledEtherSplits()

The following changed:

```
@@ -320,21 +261,24 @@ contract EtherFiRestaker is Initializable,
UUPSUpgradeable, OwnableUpgradeable,
         TokenInfo memory info = tokenInfos[_token];
         if (info.elStrategy != IStrategy(address(0))) {
             uint256 restakedTokenAmount = getRestakedAmount(_token);
-            restaked = liquifier.quoteByFairValue(_token,
restakedTokenAmount); /// restaked & pending for withdrawals
-            unrestaking =
getEthAmountInEigenLayerPendingForWithdrawals(_token);
+            uint256 unrestakingTokenAmount =
getAmountInEigenLayerPendingForWithdrawals(_token);
+            restaked = liquifier.quoteByFairValue(_token,
restakedTokenAmount); // restaked & pending for withdrawals
+            unrestaking = liquifier.quoteByFairValue(_token,
unrestakingTokenAmount); // restaked & pending for withdrawals
         }
         holding = liquifier.quoteByFairValue(_token,
IERC20(_token).balanceOf(address(this))); /// eth value for erc20 holdings
-        pendingForWithdrawals = getEthAmountPendingForRedemption(_token);
+        pendingForWithdrawals = liquifier.quoteByFairValue(_token,
getAmountPendingForRedemption(_token));
     }
```

`getEthAmountInEigenLayerPendingForWithdrawals` was simply renamed `getAmountPendingForRedemption` without any change of functionality (trivial change). However, while the computation for `restaked` stayed the same, `unrestaking` changed, and the returned value `pendingForWithdrawals` changed too:

```
-            unrestaking =
getEthAmountInEigenLayerPendingForWithdrawals(_token);
+            uint256 unrestakingTokenAmount =
getAmountInEigenLayerPendingForWithdrawals(_token);
+            unrestaking = liquifier.quoteByFairValue(_token,
unrestakingTokenAmount); // restaked & pending for withdrawals
```

```
-        pendingForWithdrawals = getEthAmountPendingForRedemption(_token);
+        pendingForWithdrawals = liquifier.quoteByFairValue(_token,
getAmountPendingForRedemption(_token));
```

We need an explanation regarding this change.

ether.fi's response: the role of the function `getTotalPooledEtherSplits` remains identical. it needs to account ALL of its stETH holdings which can have 4 different states:

- `holding`: EtherFiRestaker is just holding it as an ERC20 balance
- `restaked`: EtherFiRestaker has deployed it to be reestaked to EigenLayer
- `unrestaking`: EtherFiRestaker has initiated the withdrawal of the restaked stETH from EigenLayer. it is in the queue
- `pendingForWithdrawals`: EtherFiRestaker has initiated the redemption of stETH for ETH from Lido

In the previous implementation:

- (`unrestaking`, `pendingForWithdrawals`) was not using `liquifier.quoteByFairValue`. now we enforce it

Certora's response: Acknowledged.

EtherFiRedemptionManager.sol

The change here is related to the use of `roleRegistry` (going from `onlyOwner` to `onlyProtocolUpgrader`).

No concerns to be raised.

Liquifier.sol

The change here is trivial (removing an event).

No concerns to be raised.

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.