Security Assessment & Formal Verification *Draft* Report

Ether-Fi

September 2024

Prepared for EtherFi

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
EtherFi smart contracts	etherfi-protocol/smart-contracts: ether.fi's smart contracts repo (github.com)	89b84628f6b090257e2d34ae1108613c9de56522	EVM

Project Overview

This document describes the specification and verification of **EtherFi smart-contracts** using the Certora Prover and manual code review findings. The work was undertaken from **29/7/2024** to **8/9/2024.**

The following contract list is included in our scope:

```
src/*
lib/BucketLimiter.sol
```

excluding:

```
iEarlyAdopterPool.sol
NFTExchange.sol
TVLOracle.sol
Treasury.sol
LoyaltyPointsMarketSafe.sol
```

The Certora Prover demonstrated that the implementation of the **Solidity** contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all the Solidity contracts**.** During the verification process and the manual audit, the Certora team discovered bugs in the Solidity contracts code, as listed on the following page.

Please note that a few more formal rules are not included in this report, as they were proven with an unreleased version of the Certora Prover. Once those rules are proven on a released version of the Certora Prover, we will add them to the next version of this document.

Protocol Overview

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	2	-	-
Medium	12		
Low	5		
Informational	11		
Total	30		

Severity Matrix

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

Detailed Findings

ID	Title	Severity	Status
H-01	{Fee income can be lost due to lack of slippage protection on swaps of fees to native tokens}	High	{Not yet fixed}
M-01	{MEV Extraction via deposit rebalancing and withdraw logic}	Medium	{Fixed}
L-01	{Vaults and Strategies can be maliciously initialized due to lack of atomic initialization}	Low	{Not yet fixed}

High Severity Issues

H-01 Unfair payout distribution / DoS in full withdrawal

Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: EtherFiNodeManager.sol	Status: {Fixed/Not Fixed}	Violated Property: {Property name/if any}

Description:

fullWithdraw() function withdraws funds to Stakers in FIFO style which can cause race conditions.

Which can cause 2 problems:

1. Staker DoS

2. Unfair payout distribution - last Staker realizes unnecessary loss

```JavaScript
function getFullWithdrawalPayouts(
    IEtherFiNodesManager.ValidatorInfo memory _info,
    IEtherFiNodesManager.RewardsSplit memory _SRsplits
) public view onlyEtherFiNodeManagerContract returns (uint256 toNodeOperator, uint256 toTnft,
uint256 toBnft, uint256 toTreasury) {
    if (version == 0 || numAssociatedValidators() == 1) {
        return calculateTVL(0, _info, _SRsplits, true);
    } else if (version == 1) {

        uint256[] memory payouts = new uint256[](4);
```

```
        // here withdrawableBalanceInExecutionLayer() will likely return > 32 eth if there
are more than 1 validator
        uint256 principal = (withdrawableBalanceInExecutionLayer() >= 32 ether) ? 32 ether :
withdrawableBalanceInExecutionLayer();
        (payouts[2], payouts[1]) = _calculatePrincipals(principal);
        (payouts[0], payouts[1], payouts[2], payouts[3]) = _applyNonExitPenalty(_info,
payouts[0], payouts[1], payouts[2], payouts[3]);

        return (payouts[0], payouts[1], payouts[2], payouts[3]);
    } else {
        require(false, "WRONG_VERSION");
    }
}
```

Exploit Scenario:

1. There are 20 validators registered for a single node.
2. All validators ask for exit and all exits are processed. Thus, node balance would be 640 eth (20 * 32 eth).
3. Each validator is then slashed 1 eth then the node balance is 620 eth.
4. Then 19 stakers call EtherfiNodeManager.fullWithdraw(). The protocol will send payout 30 eth and 2 eth for T-NFT and B-NFT holders respectively. (608 ethers are withdrawn from node and 12 ether remains inside the node)
5. Then last validator calls EtherfiNodeManager.fullWithdraw() and fails because of the revert condition require(balance >= 16 ether * numExitedValidators, "INSUFFICIENT_BALANCE");
6. Or last validator calls EtherfiNodeManager.fullWithdraw() and receives smaller payout

Recommendations: {Suggest a rough idea of how to solve the issue}

Customer's response: {Customer feedback}

Fix Review: {Comments about the fix}

H-02 Unfair slashing for finalized withdrawal requests

Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: WithdrawalRequestNFT.sol	Status: {Fixed/Not Fixed}	Violated Property: {Property name/if any}

Description: The current implementation of the withdrawal requests claim process is unfair from the perspective of the owner. The intermediate period between the finalization by the oracle and the actual claim introduces only risk and no possible gain - that is because the withdrawn value is recalculated according to the latest share value, and the minimum between the value upon finalization and the newly calculated value is chosen.

Let us break the flow into three steps:

- **Request to withdraw:** NFT is minted for owner, pool shares are transferred to the NFT contract, owner loses control of the shares (the request is irreversible). The underlying shares ETH amount is partially idle in the pool and partially in the consensus layer. One could effectively think of the withdrawal request NFT contract as the keeper of those shares, but is like any other user. The shares are still participating.

- **Finalize after oracle report:** the new consensus data is fed to the protocol, validators have exited and rewards or slashing losses are rebasing the share value. The requests are finalized, meaning the underlying shares shouldn't be participating anymore. A new value is calculated for the request, based on the minimum of the new share value and the value when the request was created. This is rather fair as to disincentivize users from leaving the protocol. Also, it's unclear whether their underlying ETH was used in that time to register new validators into the consensus layer whose future performance is unknown. At that point, the shares value should be frozen, and be miscorrelated from any pool performance.

- **Claim:** the user decides to redeem the NFT worth. Although the user's requests were finalized, his shares are still perceived as participating in the system, which contradicts

the logic of finalization. The new share value calculation is pessimistic and is taken as the lowest between the new and the old value (see code snippet below). A user has no control of his shares (couldn't be sold or transferred) and is exposed to only slashing risk, without rewards gain - because of minimum calc type.

```JavaScript
function getClaimableAmount(uint256 tokenId) public view returns (uint256) {
        require(tokenId < nextRequestId, "Request does not exist");
        require(tokenId <= lastFinalizedRequestId, "Request is not finalized");
        require(ownerOf(tokenId) != address(0), "Already Claimed");

        IWithdrawRequestNFT.WithdrawRequest memory request = _requests[tokenId];

        // send the lesser value of the originally requested amount of eEth or the current
eEth value of the shares
        uint256 amountForShares = liquidityPool.amountForShare(request.shareOfEEth);
        uint256 amountToTransfer = (request.amountOfEEth < amountForShares) ?
request.amountOfEEth : amountForShares;
        uint256 fee = uint256(request.feeGwei) * 1 gwei;

        return amountToTransfer - fee;
    }
```

Exploit Scenario: Lock the share price for all finalized requests and don’t introduce negative rebasing in the claim of the request.

Recommendations: {Suggest a rough idea of how to solve the issue}

Customer’s response: {Customer feedback}

Fix Review: {Comments about the fix}

Medium Severity Issues

M-01 DOS batchDeposits by frontrunning with cancelBid

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: {Files}	Status: {Fixed/Not Fixed}	Violated Property: {Property name/if any}

Description: There is no necessary delay for someone to cancel a bid. A bad actor could take advantage of this by creating many bids, frontrunning a call to batchDeposit and causing StakingManager.sol line 130 'require(auctionManager.numberOfActiveBids() >= _numberOfValidators, "NOT_ENOUGH_BIDS");' to revert.

It would be impossible for the validator spawner to tell which bids are safe to include in batchDeposit, so the bad actor could continue to do this indefinitely.

```
JavaScript
function batchDepositWithBidIds(uint256[] calldata _candidateBidIds, uint256
_numberOfValidators, address _tnftHolder, address _bnftHolder, bool _enableRestaking, uint256
_validatorIdToShareWithdrawalSafe)
        public whenNotPaused nonReentrant onlyLiquidityPool returns (uint256[] memory)
    {
        require(_candidateBidIds.length >= _numberOfValidators && _numberOfValidators <=
maxBatchDepositSize, "WRONG_PARAMS");
        require(auctionManager.numberOfActiveBids() >= _numberOfValidators,
"NOT_ENOUGH_BIDS");

        return _processDeposits(_candidateBidIds, _numberOfValidators, _validatorSpawner,
_tnftHolder, _bnftHolder, _enableRestaking, _validatorIdToShareWithdrawalSafe);
    }
```

Recommendations: We recommend implementing a time delay in order to cancel bids.

Customer's response: {Customer feedback}

Fix Review: {Comments about the fix}

M-02 The contract may not have sufficient ETH to enable canceling bids.

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: AuctionManager.sol	Status: Not Fixed	Violated Property: {Property name/if any}

Description: To cancel a bid, a bidder needs to call `cancelBid` or `cancelBidBatch`, where the contract sends the deposited ETH back to the bidder. However, a situation may arise where the contract lacks sufficient ETH to return to the bidder, preventing the bid from being canceled. This issue can occur after invoking `transferAccumulatedRevenue` since there are no restrictions on the amount of ETH the contract must retain.

Impact: Bidders may be unable to cancel their bids.

Recommendations: Consider implementing a mechanism to monitor the amount of ETH associated with currently active bids and restrict the contract from transferring funds below this threshold.

M-03 The contract may lose ETH upon upgrade.

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: AuctionManager.sol	Status: Not Fixed	

Description: When calling `initializeOnUpgrade`, the `accumulatedRevenue` is set to zero, even though there can be an existing ETH balance in the contract before the upgrade. This can

cause the contract to lose ETH because it is untracked as there is currently no mechanism to forward this balance within the protocol.

Impact: ETH balance can be lost in the contract.

M-04 Liquifier L1SyncPool couldn't be set to any address		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: Liquifier.sol	Status: {Fixed/Not Fixed}	

Description:
The L1SyncPool in Liquifier isn't set anywhere and couldn't be set, so it's always zero, thus any reference to it is meaningless.

Recommendations: Initialize L1SyncPool address in the constructor and add an appropriate owner-only update function for it.

Customer's response: {Customer feedback}

Fix Review: {Comments about the fix}

M-05 EtherFiRewardsRouter Treasury isn't set to any address

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: EtherFiRewardsRouter.sol	Status: {Fixed/Not Fixed}	

Description:
The contract EtherFiRewardsRouter has a linked treasury contract that is the destination of all ERC20 tokens that are ought to be recovered. Currently the treasury address is not set in the constructor, which means it would remain with a zero value forever - hence hindering the ERC20 tokens recovery mechanism.

Recommendations: Add a treasury address to the initialization of the contract.

Customer's response: {Customer feedback}

Fix Review: {Comments about the fix}

M-06 Whitelist double privilege in AuctionManager.sol

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: AuctionManager.sol	Status: {Fixed/Not Fixed}	

Description: Currently setting whitelistEnabled to true would only allow for whitelisted operator managers to bid for an auction. The team expressed this is not the desired functionality, and

whitelistEnabled should only allow whitelisted managers to be able to exercise their privileges without excluding others from bidding.

```JavaScript
function createBid(
        uint256 _bidSize,
        uint256 _bidAmountPerBid
    ) external payable whenNotPaused nonReentrant returns (uint256[] memory) {
        require(_bidSize > 0, "Bid size is too small");
        if (whitelistEnabled) {
            require(
                nodeOperatorManager.isWhitelisted(msg.sender),
                "Only whitelisted addresses"
            );
            require(
                msg.value == _bidSize * _bidAmountPerBid &&
                    _bidAmountPerBid >= whitelistBidAmount &&
                    _bidAmountPerBid <= maxBidAmount,
                "Incorrect bid value"
            );
        } else {
            if (
                nodeOperatorManager.isWhitelisted(msg.sender)
            ) {
                require(
                    msg.value == _bidSize * _bidAmountPerBid &&
                        _bidAmountPerBid >= whitelistBidAmount &&
                        _bidAmountPerBid <= maxBidAmount,
                    "Incorrect bid value"
                );
            } else {
                require(
                    msg.value == _bidSize * _bidAmountPerBid &&
                        _bidAmountPerBid >= minBidAmount &&
                        _bidAmountPerBid <= maxBidAmount,
                    "Incorrect bid value"
                );
            }
        }
```

Recommendations: We recommend changing the logic above to reflect the intended behavior, for example:

```javascript
function createBid(
      uint256 _bidSize,
      uint256 _bidAmountPerBid
  ) external payable whenNotPaused nonReentrant returns (uint256[] memory) {
      require(_bidSize > 0, "Bid size is too small");
      if (whitelistEnabled) {
          if (
              nodeOperatorManager.isWhitelisted(msg.sender)
          ) {
              require(
                  msg.value == _bidSize * _bidAmountPerBid &&
                      _bidAmountPerBid >= whitelistBidAmount &&
                      _bidAmountPerBid <= maxBidAmount,
                  "Incorrect bid value"
              );
          } else {
              require(
                  msg.value == _bidSize * _bidAmountPerBid &&
                      _bidAmountPerBid >= minBidAmount &&
                      _bidAmountPerBid <= maxBidAmount,
                  "Incorrect bid value"
              );
          }
      }
```

Customer's response: {Customer feedback}

Fix Review: {Comments about the fix}

M-07 Swapping EETH for StETH could front-run Liquifier withdrawal request		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: Liquifier.sol	Status: {Fixed/Not Fixed}	

Description: Calling *swapEEthForStEth()* by any swapper has the ability to determine the total amount of EETH / StETH shares the Liquifier holds. Any such swapping operation bears no loss (or gain) to the swapper since the shares being swapped represent the same amount of ETH. Therefore, the external swapper could in high probability, in any moment, lead to DoS of others to swap as well, and also prevent the contract from calling `stEthRequestWithdrawal()`.

Customer's response: {Customer feedback}

Fix Review: {Comments about the fix}

M-08 StETH deposit consumption rate could be reached by repeated action

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: Liquifier.sol	Status: {Fixed/Not Fixed}	

Description: By repeatedly calling *depositWithERC20()* with StETH and then *swapEEthForStEth()* for the same deposited amount, one could switch back and forth between owning StETH and EETH with no loss, since the pricing ratio of these two tokens is always 1:1.
In that way, the user bears no loss, the Liquifier is left with the same amount of EETH and StETH, but the consumption of StETH keeps increasing, with no actual assets delta being deposited to the system. So eventually, a griefer could reach the limit and block deposits from other users.

Customer's response: {Customer feedback}

Fix Review: {Comments about the fix}

M-09 Lack of etherFanEEthShares can lead to executeTasks DOS

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: Liquifier.sol	Status: {Fixed/Not Fixed}	

Description: There is no guarantee that etherFanEEthShares is non-zero. If the quantity of native ETH in MembershipManager is larger than the thresholdAmount, and etherFanEEthShares is equal zero, line 272 will revert due to a division by zero, leading to a DOS of EtherFiAdmin.executeTasks.

Recommendations: As discussed with the team, the purpose of thresholdAmount is not clear. Therefore we recommend changing the condition on MembershipManager line 270 to check whether etherFanEEthShares is zero or not.

Customer's response: {Customer feedback}

Fix Review: {Comments about the fix}

M-10 Ether Fan rewards can be forcibly decreased by griefing

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: MembershipManager.sol LiquidityPool.sol	Status: {Fixed/Not Fixed}	

Description: ethRewardsPerEEthShareAfterRebase variable value can be decreased by donating Eeth shares to MembershipManager.sol in line 272. This would lead to _distributeStakingRewardsV0 and _distributeStakingRewardsV1 awarding less shares for the rest of recipients.

Recommendations: We recommend not fetching contract balances directly, as done in line 269. Instead we advise to keep track of such variables with dedicated variables that cannot be manipulated by external actors interacting with the system without going through its accounting system.

Customer's response: {Customer feedback}

Fix Review: {Comments about the fix}

M-11 Donation of EETH share can lead to executeTasks DOS

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: MembershipManager.sol	Status: {Fixed/Not Fixed}	

Description: The difference between the variables ethRewardsPerEEthShareBeforeRebase and ethRewardsPerEEthShareAfterRebase, namely ethRewardsAmountPerEEthShare gets passed to calculateRewardsPerTierV0/V1, then it's output is passed to calculateRescaledTierRewards. If ethRewardsAmountPerEEthShare is zero, GlobalIndexLibrary.sol line 96 will suffer a division by zero and the call will revert.

A malicious actor could front run the call to executeTasks and manipulate the ethRewardsPerEEthShareAfterRebase value via donating or burning Eeth shares to MembershipManager.sol in order to always ensure that ethRewardsAmountPerEEthShare equals zero. Note that for this manipulation to work the attacker would need to ensure line 270 to always be true, which could be significantly expensive depending on the value of thresholdAmount.

This would allow them to DOS the executeTasks function indefinitely for as long as their capital lasts, preventing the finalization of requests and the enqueuing of validator management tasks that could be against their favor. This DOS could enable them to make a profit if the price mechanism creates a temporary arbitrage with other protocols integrated with EtherFi that have or have not yet priced in the coming slashes or rewards that are being delayed.

Recommendations: We recommend not fetching contract balances directly, as done in line 269. Instead we advise to keep track of such variables with dedicated variables that cannot be manipulated by external actors interacting with the system without going through it's accounting system.

Customer's response: {Customer feedback}

Fix Review: {Comments about the fix}

M-12 Whale can grief deposits through permissionless burn

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: MembershipManager.sol LiquidityPool.sol	Status: {Fixed/Not Fixed}	

Description: A large holder of EETH shares could simply burn their shares directly. This would cause a great discrepancy between totalValueOutOfLp + totalValueInLp (getTotalPooledEther) and the total number of EETH shares. The total number of EETH shares would drastically decrease, while the total pooled ether would remain the same. This imbalance would negatively impact the number of shares new users would get when depositing in the contract as evidenced by LiquidityPool.sol line 520 `(_depositAmount * eETH.totalShares()) / totalPooledEther;`.

Recommendations: We recommend making the EETH.burnShares a strictly permissioned function, which can be only called through the internal accounting system of the protocol.

Customer's response: {Customer feedback}

Fix Review: {Comments about the fix}

Low Severity Issues

L-01 Changing reward splits affect past Stakers and LPs actions.

Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: EtherFiNodesManager.sol	Status: {Fixed/Not Fixed}	Violated Property: {Property name/if any}

Description: In setStakingRewardsSplit(), EtherFi admins can change reward split factors anytime to any value. In particular, reward splits can be changed to be less attractive for Stakers and LPers. The reward split change affects past Staking and LPing actions but should not.

```javascript
function setStakingRewardsSplit(uint64 _treasury, uint64 _nodeOperator, uint64 _tnft, uint64 _bnft) public {
    if (!roleRegistry.hasRole(NODE_ADMIN_ROLE, msg.sender)) revert IncorrectRole();
    if (_treasury + _nodeOperator + _tnft + _bnft != SCALE) revert InvalidParams();

    stakingRewardsSplit.treasury = _treasury;
    stakingRewardsSplit.nodeOperator = _nodeOperator;
    stakingRewardsSplit.tnft = _tnft;
    stakingRewardsSplit.bnft = _bnft;
}
```

Recommendations: {Suggest a rough idea of how to solve the issue}

Customer's response: {Customer feedback}

Fix Review: {Comments about the fix}

L-02 Redundant code block

Severity: **Low**	Impact: **Low**	Likelihood: **High**
Files: StakingManager.sol	Status: {Fixed/Not Fixed}	Violated Property: {Property name/if any}

Description: StakingManager.sol line 415 is unreachable as msg.sender is always equal to liquidityPoolContract. The function _cancelDeposit can only be reached through the function batchCancelDeposit which has the onlyLiquidityPool modifier. Therefore, lines 414-417 are redundant. In case this code is not removed and code updates are made, where msg.sender is not always equal to liquidityPoolContract, this could lead to refunding the wrong address and consequently insolvency.

```JavaScript
bool isFullStake = (msg.sender != liquidityPoolContract);
        if (isFullStake) {
            _refundDeposit(msg.sender, stakeAmount);
        }
```

Recommendations: We recommend removing the code block.

Customer's response: {Customer feedback}

Fix Review: {Comments about the fix}

L-03 Whitelist minimum bid amount should not have to be bigger than regular bid amount

Severity: **Critical**	Impact: **High**	Likelihood: **High**
Files: AuctionManager.sol	Status: {Fixed/Not Fixed}	Violated Property: {Property name/if any}

Description: The require check in AuctionManager.sol line 372 prevents whitelistBidAmount to be less than minBidAmount, which is inconsistent with the require check made in line 346 where minBidAmount can be bigger or equal to whitelistBidAmount.

JavaScript

```
    function setMinBidPrice(uint64 _newMinBidAmount) external {
        if (!roleRegistry.hasRole(AUCTION_ADMIN_ROLE, msg.sender)) revert IncorrectRole();

        require(_newMinBidAmount < maxBidAmount, "Min bid exceeds max bid");
        require(_newMinBidAmount >= whitelistBidAmount, "Min bid less than whitelist bid
amount");
        minBidAmount = _newMinBidAmount;
    }


...


    function updateWhitelistMinBidAmount(
        uint128 _newAmount
    ) external onlyOwner {
        require(_newAmount < minBidAmount && _newAmount > 0, "Invalid Amount");
        whitelistBidAmount = _newAmount;
    }
```

Recommendations: We recommend changing _newAmount < minBidAmount to _newAmount <= minBidAmount.

Customer's response: {Customer feedback}

Fix Review: {Comments about the fix}

L-04 Pausable contracts array could have duplicates

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: Pauser.sol	Status: {Fixed/Not Fixed}	

Description: The Pauser contracts keeps an array of all pausable contracts in the system, such that the owner could pause or unpause all of them in a batch by calling either unpauseAll() or pauseAll(). However, the OZ implementation of a pausable contract reverts if the contract is already in the desired state. So trying to pause/unpause a contract twice would fail the whole transaction if there are duplicates in the pausable array. There is no such check for duplicates in the current implementation.

Pauser.sol:

```javascript
function addPausable(IPausable _pausable) external onlyRole(PAUSER_ADMIN) {
        pausables.push(_pausable);
        emit PausableAdded(address(_pausable));
    }
```

@openzeppelin/contracts/security/Pausable.sol

```javascript
/**
     * @dev Throws if the contract is paused.
     */
    function _requireNotPaused() internal view virtual {
        require(!paused(), ": paused");
    }

    /**
     * @dev Throws if the contract is not paused.
     */
    function _requirePaused() internal view virtual {
        require(paused(), ": not paused");
    }
```

Recommendations: When pushing a new contract, add a check if it already exists in the array.

Customer's response: {Customer feedback}

Fix Review: {Comments about the fix}

L-05 Liquiifer doesn't white-list special tokens

Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: Liquifier.sol	Status: {Fixed/Not Fixed}	

Description: The three special tokens in the Liquifier `lido`, `cbEth` and `wbEth` aren't white-listed by default, which could lead to reverts in some functions in the Liquifier, for example `quoteByMarketValue()`.

Recommendations: `lido`, `cbEth` and `wbEth` should be white-listed on `initialize` .

Informational Severity Issues

I-01. Inconsistent role protection

Description: In some places `require` keyword is used for role protection. In other places modifiers are used.

```JavaScript
// example with require keyword in LiquidityPool.sol
function deposit(address _user, address _referral) external payable
returns (uint256) {
    require(msg.sender == address(membershipManager), "Incorrect
Caller");
    ...
}

// example with modifier in MembershipNFT.sol
function burn(address _from, uint256 _tokenId, uint256 _amount)
external onlyMembershipManagerContract {
    _burn(_from, _tokenId, _amount);
}
```

Recommendation: Use consistent style for role protection

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

I-02. Modifier not needed for view function

Description: In `EtherFiNode.sol` the `calcualteTVL()` function is marked as view and has a modifier with role protection which is unnecessary.

```JavaScript
// example with require keyword in LiquidityPool.sol
function calculateTVL(...) public view onlyEtherFiNodeManagerContract
... {
    ...
}
```

Recommendation: Use 1 role protection

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

I-03. Doubled role protection

Description: In `EtherFiNodesManager.sol` the `updateAllowedForwardedExternalCalls()` function has 2 modifiers with role protection which is unnecessary.

```JavaScript
// example with modifier in EtherFiNodesManager.sol
function updateAllowedForwardedExternalCalls(bytes4, address, bool)
external onlyOwner {
    if (!roleRegistry.hasRole(WHITELIST_UPDATER, msg.sender)) {
        revert IncorrectRole();
    }
    ...
}
```

Recommendation: Remove unnecessary role protection

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

I-04. Inconsistent revert style

Description: In some places `require` keyword is used for reverting flow. In other places custom errors are thrown.

```JavaScript
// example with require keyword in LiquidityPool.sol
function deposit(address _user, address _referral) external payable
returns (uint256) {
    require(msg.sender == address(membershipManager), "Incorrect
Caller");
    ...
}

// example with modifier in EtherFiNodesManager.sol
function updateAllowedForwardedExternalCalls(bytes4 _selector,
address _target, bool _allowed) external onlyOwner {
    if (!roleRegistry.hasRole(WHITELIST_UPDATER, msg.sender)) {
        revert IncorrectRole();
    }
    ...
}
```

Recommendation: Use consistent revert style

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

I-05. Confusing function naming

Description: We have two functions with very similar names, batchDeposit() and deposit(). From this perspective it is easy to draw conclusions that batchDeposit() contains a loop calling deposit(), or some similar functionality. However, the deposit function does something entire differently and is not even related to the same mechanism.

Recommendation: We recommend changing the name of the deposit() function to provideLiquidity(), which provides more clarity and distinction between intended uses for both functions.

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

I-06. isFinalized modifier is underutilized

Description: WithdrawRequestNFT.sol line 85: `require(tokenId <= lastFinalizedRequestId, "Request is not finalized");` could be substituted by the **isFinalized** modifier.

Recommendation: We recommend removing WithdrawRequestNFT.sol line 85 and adding the isFinalized modifier to the getClaimableAmount function.
Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

I-07. Unused function

Description: The function moveFundsToManager in EtherFiNode.sol is unreachable and never called.

Recommendation: We recommend removing the moveFundsToManager function.

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

I-08. Redundant if block

Description: EtherFiNode.sol line 141 contains an unnecessary if block for the purpose of simply registering validators.

Recommendation: We recommend performing such a check only the first time before an EigenPod gets to be created.

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

I-09. Typos

Description: LiquidityPool.sol line 363 comments are missing the 'F' in B-NFT.
LiquidityPool.sol line 524 initializes a variable with the name balanace, instead of balance.
EtherFiNode.sol line 72 documentation contains a misspelling of EtherFiNode where it's missing the 'F'.
EtherFiAdmin.sol line 199 misspells 'new' as 'thew'.

Recommendation: We recommend fixing typos for better readability.

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

I-10. Redundant calculation - waste of gas

Description: In EtherFiNode.sol, `getFullWithdrawalPayouts`: the result of the call to `withdrawableBalanceInExecutionLayer` could be cached (no need to calculate twice) since it yields the same value at both calls.

Recommendation: Save the output value of `withdrawableBalanceInExecutionLayer`() memory and use it in the ternary operator. Alternatively, consider implementing a "min" function.

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

I-11. Redundant role check in Liquifier

Description: Check for correct role on line 245 in `stEthRequestWithdrawal()` is currently redundant as it's being checked in the internal function `stEthRequestWithdrawal(uint256)` as well.

Recommendation: Remove the check from the external function.

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

I-12. Redundant variable

Description: Initialization of variable mintedShare (MembershipManager.sol line 271) is redundant as it's never used.

Recommendation: Remove variable.

Customer's response: {Customer's response}

Fix Review: {Comments about the fix}

Formal Verification

Verification Notations

Formally Verified	The rule is verified for every state of the contract(s), under the assumptions of the scope/requirements in the rule.
Formally Verified After Fix	The rule was violated due to an issue in the code and was successfully verified after fixing the issue
Violated	A counter-example exists that violates one of the assertions of the rule.

General Assumptions and Simplifications

1. We used Solidity Compiler version 8.24 to verify the protocol.

Formal Verification Properties

Auction Manager

Module General Assumptions

- The following properties are proved for src/AuctionManager.sol contract.
- We assume that all loops iterate at most three times.
- Function upgradeToAndCall(address, bytes) was filtered out.

Module Properties

P-01. Correctness of creation of bids	
Status: Verified	

Rule Name	Status	Description	Link to rule report
integrityOfCreateBid	Verified	*This rule verifies that createBid works as intended: amount of bids, active bids, and used keys updated correctly* *The right bids were added.*	*Report*

P-02. Active bids solvency.	
Status: Violated	

Rule Name	Status	Description	Link to rule report
activeBidsSolvency	Violated	*This rule verifies that the contract holds at least the sum of all active bids amounts field amount of eth.* *M-02 The contract may not have sufficient ETH to enable canceling bids.*	*Report*

P-03. numberOfBids equals the sum of all bids.

Status: Verified

Rule Name	Status	Description	Link to rule report
numberOfBidsEqTheSumOfAllBids	Verified	*This rule verifies that numberOfBids equals the sum of all bids.*	*Report*

P-04. numberOfActiveBids equals the sum of all active bids.

Status: Verified

Rule Name	Status	Description	Link to rule report
numberOfActiveBidsCorrect	Verified	*This rule verifies that numberOfBids equals the sum of all active bids (where isActive = true).*	*Report*

P-05. Bids are immutable.

Status: Verified

Rule Name	Status	Description	Link to rule report
bidImmutability	Verified	*This rule verifies that bids are immutable after creation (except isActive)*	*Report*

EtherFi Nodes Manager

Module General Assumptions

- We assume that all loops iterate at most three times.

Module Properties

P-01. Unused withdrawal safes are not used.

Status: Verified

Rule Name	Status	Description	Link to rule report
noValidatorForUnusedNodes	Verified	*Verifies that if there is a node in unused nodes array than there is no validator that is linked to it.*	*Report*
amountOfValidatorPerEtherFiNodeEqualsNumAssociatedValidators	Verified	*Verifies that the amout of linked validators to a node by the manager equals the actual amount of associated validators by the node itself.*	*Report*

P-02. Unused withdrawal safes are not duplicated.

Status: Verified

Rule Name	Status	Description	Link to rule report
unusedWithdrawalSafesUniqueness	Verified	*Verifies that there are no duplicated etherFi nodes in the unusedWithdrawalSafes array.*	*Report*

P-03. EtherFiNode version correct.

Status: Verified	Assumptions:

Rule Name	Status	Description	Link to rule report
versionIsOneOnlyIfAssociated	Verified	*Verifies that if there are associated validators to a node then its version is one.*	*Report*

EtherFi Node

Module General Assumptions

- We assume that all loops iterate at most three times.

Module Properties

P-01. Registering a validator and then unregistering it should never revert.	
Status: Verified	Assumptions: The validator transferred to the right phase before unregistering it.

Rule Name	Status	Description	Link to rule report
registerValidatorThenUnregisteringNeverReverts	Verified	*This rule verifies that registering a validator and then unregistering it should never revert.*	*Report*

P-02. Correctness of Validator state transitions.	
Status: Verified	

Rule Name	Status	Description	Link to rule report
validatorStateTransitions	Verified	*This rule verifies that the Validator state transitions are correct according to protocol documentation.*	*Report*

P-03. Correlation between completed and pending withdrawals.

Status: Verified

Rule Name	Status	Description	Link to rule report
pendingCompliitedWithdrawalsCorrelation	Verified	*This rule verifies that whenever the pending withdrawal amount is decreased (completed) the number of completed withdrawals increases.*	Report

P-04. Associated validator IDs are unique per etherFi node.

Status: Verified

Rule Name	Status	Description	Link to rule report
validatorIdsAreUnique	Verified	*This rule verifies that the associated validator IDs are unique per etherFi node.*	Report
validatorIdNeverZero	Verified	*This rule verifies that any associated validator ID can never be zero.*	Report

Staking Manager

Module General Assumptions

- We assume that all loops iterate at most three times.
- Function upgradeToAndCall(address, bytes) was filtered out.

Module Properties

P-01. Batch depositing with bid IDs works as intended.

Status: Verified

Rule Name	Status	Description	Link to rule report
integrityOfBatchDepositWithBidIds	Verified	*This rule verifies that after a successful call for batchDepositWithBidIds, the relevant bids are inactive and have the correct staker info.*	*Report*

P-02. Batch canceling with bid IDs works as intended.

Status: Verified

Rule Name	Status	Description	Link to rule report
integrityOfBatchCancelDeposit	Verified	*This rule verifies that after a successful call for batchCancelDeposit the relevant bids are active again, have no staker, the relevant NFTs are burned, and the contract's ETH balance is increased.*	*Report*

P-03. Correctness of batch canceling with bid IDs.

Status: Verified

Rule Name	Status	Description	Link to rule report
integrityOfBatchRegisterValidators	Verified	*This rule verifies that after a successful call for batchRegisterValidators, the contract ETH balance remains the same (ETH flows to and from the contract) and both TNFT and BNFT are minted to the right owner.*	*Report*

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.