Security Assessment

Ether-Fi - Withdrawal Fee

January 2025

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	fdbf21a7	EVM

Project Overview

This document describes the manual code review of PR 207 and PR 227, following a previous audit from Certora.
The work was a 3 day-effort undertaken from **13/01/2025** to **23/01/2025.**

The following contract list is included in our scope:

1. `lib/BucketLimiter.sol`
2. `src/EtherFiRedemptionManager.sol`
3. `src/LiquidityPool.sol`
4. `src/WithdrawRequestNFT.sol`
5. `script/deploys/DeployEtherFiRestaker.s.sol`
6. `script/deploys/DeployEtherFiWithdrawalBuffer.s.sol`
7. `script/deploys/DeployPhaseTwo.s.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	2	2	2
High	-	-	-
Medium	1	1	1
Low	1	1	1
Total	4	4	4

Severity Matrix

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

Critical Severity Issues

C-01 An attacker can take ownership of EtherFiRedemptionManager		
Severity: **High**	Impact: **High**	Likelihood: **High**
Files: [EtherFiRedemptionManager.sol#L67](#) [weEth_withdrawal_v2.s.sol#L44](#)	Status: Fixed	

Description:

The call to `EtherFiRedemptionManager.initialize()` is frontrunnable:

```
File: EtherFiRedemptionManager.sol
67:     function initialize(uint16 _exitFeeSplitToTreasuryInBps, uint16
_exitFeeInBps, uint16 _lowWatermarkInBpsOfTvl, uint256 _bucketCapacity,
uint256 _bucketRefillRate) external initializer {
```

According to Foundry's documentation about Scripts (**"Watch out for frontrunning"**: https://book.getfoundry.sh/tutorials/best-practices#scripts), the transactions inside `vm.startBroadcast` and `vm.stopBroadcast` only broadcasts the transactions but these aren't packed into 1 transaction. This means that the Script needs to be "made robust against chain-state changing between the simulation and broadcast."

The current deploy Script is as such:

```
File: weEth_withdrawal_v2.s.sol
36:     function deploy_upgrade() internal {
37:         UUPSProxy etherFiRedemptionManagerProxy = new
UUPSProxy(address(new EtherFiRedemptionManager(
                addressProvider.getContractAddress("LiquidityPool"),
                addressProvider.getContractAddress("EETH"),
                addressProvider.getContractAddress("WeETH"),
                treasury,
                roleRegistry)), "");
            EtherFiRedemptionManager etherFiRedemptionManagerInstance = EtherFiRedemptionManager(payable(etherFiRedemptionManagerProxy));
            etherFiRedemptionManagerInstance.initialize(10_00, 1_00, 1_00, 5 ether, 0.001 ether); // 10% fee split to treasury, 1% exit fee, 1% low watermark

            withdrawRequestNFTInstance.upgradeTo(address(new WithdrawRequestNFT(treasury)));
            withdrawRequestNFTInstance.initializeOnUpgrade(pauser, 50_00); // 50% fee split to treasury

            liquidityPoolInstance.upgradeTo(address(new LiquidityPool()));
liquidityPoolInstance.initializeOnUpgradeWithRedemptionManager(address(etherFiRedemptionManagerInstance));
        }
```

This means that between the deployment of `etherFiRedemptionManagerInstance` and the call to `etherFiRedemptionManagerInstance.initialize()`, an attacker can frontrun the transaction to take ownership of the `etherFiRedemptionManagerInstance`.

While the call from the deployment script to `etherFiRedemptionManagerInstance.initialize()` is expected to revert, the script execution itself is not halted unless explicitly configured (e.g., by checking transaction success or using require statements in the script). If a contract function reverts during a broadcasted transaction (i.e., after vm.startBroadcast). The revert reason (if any) is recorded in the transaction logs.

This means that the line `liquidityPoolInstance.initializeOnUpgradeWithRedemptionManager(address(etherFiRedemptionManagerInstance))` will initialize a malicious `etherFiRedemptionManagerInstance` (malicious `owner` and parameters).

Also, the contract `EtherFiRedemptionManager`'s implementation can be maliciously changed which means that the attacker can take complete ownership of a privileged contract.

Users that want to be "early" will lose their funds. A price manipulation vector is also now open in LiquidityPool (`burnEEthShares()` directly callable).

Recommendation: Either only authorize the deployer to call `etherFiRedemptionManagerInstance.initialize()` or prevent the deployment script from partially failing.

ether.fi's response: Fixed in commit [fix: certora audit fixes]()

C-02 Stealing funds from users due to silently failing on permits

Severity: **High**	Impact: **High**	Likelihood: **High**
Files: EtherFiRedemptionManager.sol#L108-L111 EtherFiRedemptionManager.sol#L120-L123	Status: Fixed	

Description:

It often happens that users give infinite approval to a protocol.
In the case of a call to `transferFrom` with `from == msg.sender`, this is usually safe.
However, if the `from` address is from a user input, then an attacker can steal funds from users.
The calls to `redeemEEthWithPermit()` and `redeemWeEthWithPermit()` have a call to `permit()` which can be made to fail intentionally by an attacker, simply by providing a fake signature.
Then, the input parameters `owner` and `receiver` would be entirely controlled by the attacker.
Normal users that simply approved the contract to use `redeemEEth` or `redeemWeEth` will get their approved funds stolen by an attacker.

ether.fi's response: Fixed in commit fix: certora audit fixes

Medium Severity Issues

M-01 batchCancelDepositByAdmin can be DOSd by the target BNFT Staker

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: LiquidityPool.sol#L398 LiquidityPool.sol#L402 LiquidityPool.sol#L418 LiquidityPool.sol#L571	Status: Fixed	

Description:

Call flow `batchCancelDepositByAdmin -> _batchCancelDeposit -> _sendFund`:

```
File: LiquidityPool.sol
573:     function _sendFund(address _recipient, uint256 _amount) internal {
574:         uint256 balanace = address(this).balance;
575:         (bool sent, ) = _recipient.call{value: _amount}("");
576:         require(sent && address(this).balance == balanace - _amount,
"SendFail");
577:     }
```

The `_recipient` just needs to send back 1 wei to the `LiquidityPool` through a direct donation (`LiquidityPool.sol#L114: receive() external payable`) which would force the check L576 to revert with `SendFail`

ether.fi's response: Fixed in commit remove batch cancel deposit by admin since not required

Low Severity Issues

L-01 Use of unsafe transfer/transferFrom		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: LiquidityPool.sol WithdrawRequestNFT.sol	Status: Fixed	

Description:

Just like this is well implemented at those places:

```
src/EtherFiRedemptionManager.sol:
  150:          IERC20(address(eEth)).safeTransfer(treasury,
eEthFeeAmountToTreasury);
  253:          IERC20(address(eEth)).safeTransferFrom(user, address(this),
eEthAmount);
  265:          IERC20(address(weEth)).safeTransferFrom(user, address(this),
weEthAmount);
```

Consider doing the same here:

```
src/LiquidityPool.sol:
  218:          eETH.transferFrom(msg.sender, address(withdrawRequestNFT),
amount);
  254:          eETH.transferFrom(msg.sender, address(withdrawRequestNFT),
amount);

src/WithdrawRequestNFT.sol:
  268:          eETH.transfer(treasury, eEthAmountToTreasury);
```

ether.fi's response: Fixed in commit fix: certora audit fixes

Informational Severity Issues

I-01. Owner can renounce while system is paused

Across the protocol, the contract owner or single user with a role is not prevented from renouncing the role/ownership while the contract is paused, which would cause any user assets stored in the protocol to be locked indefinitely.

I-02. Use `Ownable2Step.transferOwnership` instead of `Ownable.transferOwnership`

Use Ownable2Step.transferOwnership which is safer. Use it as it is more secure due to 2-stage ownership transfer.

Recommended Mitigation Steps

Use Ownable2Step.sol

```
function transferOwnership(address newOwner) public virtual override onlyOwner
{
  _pendingOwner = newOwner;
  emit OwnershipTransferStarted(owner(), newOwner);
}
```

Affected code:

- src/EtherFiRedemptionManager.sol

```
# File: src/EtherFiRedemptionManager.sol

EtherFiRedemptionManager.sol:11: import
"@openzeppelin-upgradeable/contracts/access/OwnableUpgradeable.sol";
```

- src/LiquidityPool.sol

```
# File: src/LiquidityPool.sol
```

```
LiquidityPool.sol:11: import
"@openzeppelin-upgradeable/contracts/access/OwnableUpgradeable.sol";
```

- src/WithdrawRequestNFT.sol

```
# File: src/WithdrawRequestNFT.sol

WithdrawRequestNFT.sol:6: import
"@openzeppelin-upgradeable/contracts/access/OwnableUpgradeable.sol";
```

I-03. Function `requestWithdraw()` doesn't need to be `payable`

It never handles funds and the calling contracts from LiquidityPool are not `payable`.

If this is to save gas (23 units) then consider adding a comment.

I-04. Typos

- balanace

```
File: LiquidityPool.sol
574:          uint256 balanace = address(this).balance;
```

- onlyLiquidtyPool

```
File: WithdrawRequestNFT.sol
314:      modifier onlyLiquidtyPool() {
```

I-04. Duplicate import statements

Affected code:

- src/LiquidityPool.sol

```
# File: src/LiquidityPool.sol

LiquidityPool.sol:14: import "./interfaces/IStakingManager.sol";

LiquidityPool.sol:17: import "./interfaces/IStakingManager.sol";
```

I-05. Unused `error` definition

Affected code:

- src/LiquidityPool.sol

```
# File: src/LiquidityPool.sol

LiquidityPool.sol:100:     error InvalidParams();

LiquidityPool.sol:103:     error SendFail();
```

I-06. Event is never emitted

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

Affected code:

- src/LiquidityPool.sol

```
# File: src/LiquidityPool.sol

LiquidityPool.sol:90:     event UpdatedSchedulingPeriod(uint128
newPeriodInSeconds);
```

I-07. Event missing indexed field

Index event fields make the field more quickly accessible to off-chain tools that parse events. This is especially useful when it comes to filtering based on an address. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Where applicable, each `event` should use three `indexed` fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three applicable fields, all of the applicable fields should be indexed.

Affected code:

- src/LiquidityPool.sol

```
# File: src/LiquidityPool.sol

LiquidityPool.sol:81:      event Paused(address account);

LiquidityPool.sol:82:      event Unpaused(address account);

LiquidityPool.sol:86:      event UpdatedWhitelist(address userAddress, bool
value);

LiquidityPool.sol:87:      event UpdatedTreasury(address newTreasury);

LiquidityPool.sol:88:      event BnftHolderDeregistered(address user, uint256
index);

LiquidityPool.sol:89:      event BnftHolderRegistered(address user, uint256
index);

LiquidityPool.sol:90:      event UpdatedSchedulingPeriod(uint128
newPeriodInSeconds);

LiquidityPool.sol:94:      event Rebase(uint256 totalEthLocked, uint256
totalEEthShares);

LiquidityPool.sol:95:      event ProtocolFeePaid(uint128 protocolFees);

LiquidityPool.sol:96:      event WhitelistStatusUpdated(bool value);
```

- src/WithdrawRequestNFT.sol

```
# File: src/WithdrawRequestNFT.sol

WithdrawRequestNFT.sol:49:     event 
HandledRemainderOfClaimedWithdrawRequests(uint256 eEthAmountToTreasury, 
uint256 eEthAmountBurnt);

WithdrawRequestNFT.sol:51:     event Paused(address account);

WithdrawRequestNFT.sol:52:     event Unpaused(address account);
```

I-08. Events that mark critical parameter changes should contain both the old and the new value

This should especially be done if the new value is not required to be different from the old value

Affected code:

- src/LiquidityPool.sol

```
# File: src/LiquidityPool.sol

LiquidityPool.sol:485:     function setTreasury(address _treasury) external 
onlyOwner {
                               treasury = _treasury;
                               emit UpdatedTreasury(_treasury);

LiquidityPool.sol:515:     function updateWhitelistedAddresses(address[] 
calldata _users, bool _value) external onlyAdmin {
                               for (uint256 i = 0; i < _users.length; i++) {
                                   whitelisted[_users[i]] = _value;

                                   emit UpdatedWhitelist(_users[i], _value);

LiquidityPool.sol:523:     function updateWhitelistStatus(bool _value) 
external onlyAdmin {
                               whitelistEnabled = _value;
```

```
                        emit WhitelistStatusUpdated(_value);
```

I-09. Missing Event for critical parameters change

Events help non-contract tools to track changes, and events prevent users from being surprised by changes.

Affected code:

- src/EtherFiRedemptionManager.sol

```
# File: src/EtherFiRedemptionManager.sol

EtherFiRedemptionManager.sol:204:     function setCapacity(uint256 capacity)
external hasRole(PROTOCOL_ADMIN) {
                                          // max capacity = max(uint64) * 1e12
~= 16 * 1e18 * 1e12 = 16 * 1e12 ether, which is practically enough
                                          uint64 bucketUnit =
_convertToBucketUnit(capacity, Math.Rounding.Down);
                                          BucketLimiter.setCapacity(limit,
bucketUnit);

EtherFiRedemptionManager.sol:214:     function setRefillRatePerSecond(uint256
refillRate) external hasRole(PROTOCOL_ADMIN) {
                                          // max refillRate = max(uint64) *
1e12 ~= 16 * 1e18 * 1e12 = 16 * 1e12 ether per second, which is practically
enough
                                          uint64 bucketUnit =
_convertToBucketUnit(refillRate, Math.Rounding.Down);
                                          BucketLimiter.setRefillRate(limit,
bucketUnit);

EtherFiRedemptionManager.sol:224:     function setExitFeeBasisPoints(uint16
_exitFeeInBps) external hasRole(PROTOCOL_ADMIN) {
                                          require(_exitFeeInBps <=
BASIS_POINT_SCALE, "INVALID");
                                          exitFeeInBps = _exitFeeInBps;

EtherFiRedemptionManager.sol:229:     function
setLowWatermarkInBpsOfTvl(uint16 _lowWatermarkInBpsOfTvl) external
```

```
hasRole(PROTOCOL_ADMIN) {
                                          require(_lowWatermarkInBpsOfTvl <=
BASIS_POINT_SCALE, "INVALID");
                                          lowWatermarkInBpsOfTvl =
_lowWatermarkInBpsOfTvl;

EtherFiRedemptionManager.sol:234:     function
setExitFeeSplitToTreasuryInBps(uint16 _exitFeeSplitToTreasuryInBps) external
hasRole(PROTOCOL_ADMIN) {
                                          require(_exitFeeSplitToTreasuryInBps
<= BASIS_POINT_SCALE, "INVALID");
                                          exitFeeSplitToTreasuryInBps =
_exitFeeSplitToTreasuryInBps;
```

- src/LiquidityPool.sol

```
# File: src/LiquidityPool.sol

LiquidityPool.sol:491:     function setRestakeBnftDeposits(bool _restake)
external onlyAdmin {
                               restakeBnftDeposits = _restake;

LiquidityPool.sol:497:     function updateAdmin(address _address, bool
_isAdmin) external onlyOwner {
                               admins[_address] = _isAdmin;

LiquidityPool.sol:529:     function updateBnftMode(bool _isLpBnftHolder)
external onlyAdmin {
                               // Never toggle it in the process of
deposit-regiration
                               isLpBnftHolder = _isLpBnftHolder;
```

- src/WithdrawRequestNFT.sol

```
# File: src/WithdrawRequestNFT.sol

WithdrawRequestNFT.sol:227:     function updateAdmin(address _address, bool
_isAdmin) external onlyOwner {
                                    require(_address != address(0), "Cannot be
address zero");
                                    admins[_address] = _isAdmin;

WithdrawRequestNFT.sol:232:     function
updateShareRemainderSplitToTreasuryInBps(uint16
_shareRemainderSplitToTreasuryInBps) external onlyOwner {

require(_shareRemainderSplitToTreasuryInBps <= BASIS_POINT_SCALE, "INVALID");
                                    shareRemainderSplitToTreasuryInBps =
_shareRemainderSplitToTreasuryInBps;
```

I-10. Take advantage of Custom Error's return value property

An important feature of Custom Error is that values such as address, tokenID, msg.value can be written inside the () sign, this kind of approach provides a serious advantage in debugging and examining the revert details of dapps such as tenderly.

Affected code:

- src/LiquidityPool.sol

```
# File: src/LiquidityPool.sol

LiquidityPool.sol:115:         if (msg.value > type(uint128).max) revert
InvalidAmount();

LiquidityPool.sol:121:         if (_eEthAddress == address(0) ||
_stakingManagerAddress == address(0) || _nodesManagerAddress == address(0) ||
_membershipManagerAddress == address(0) || _tNftAddress == address(0)) revert
DataNotSet();

LiquidityPool.sol:193:         if (totalValueInLp < _amount || (msg.sender ==
address(withdrawRequestNFT) && ethAmountLockedForWithdrawal < _amount) ||
eETH.balanceOf(msg.sender) < _amount) revert InsufficientLiquidity();

LiquidityPool.sol:194:         if (_amount > type(uint128).max || _amount == 0
|| share == 0) revert InvalidAmount();

LiquidityPool.sol:215:         if (amount > type(uint96).max || amount == 0 ||
share == 0) revert InvalidAmount();
```

```
LiquidityPool.sol:249:         if (msg.sender != address(membershipManager))
revert IncorrectCaller();

LiquidityPool.sol:251:         if (amount > type(uint96).max || amount == 0 ||
share == 0) revert InvalidAmount();

LiquidityPool.sol:470:         if (msg.sender != address(membershipManager))
revert IncorrectCaller();

LiquidityPool.sol:478:         if (msg.sender !=
address(etherFiAdminContract)) revert IncorrectCaller();

LiquidityPool.sol:535:         if (!(msg.sender ==
address(etherFiAdminContract))) revert IncorrectCaller();

LiquidityPool.sol:541:         if (msg.sender !=
address(etherFiRedemptionManager) || msg.sender !=
address(withdrawRequestNFT)) revert IncorrectCaller();

LiquidityPool.sol:554:         if (amount > type(uint128).max || amount == 0
|| share == 0) revert InvalidAmount();
```

I-11. `override` function arguments that are unused should have the variable name removed or commented out to avoid compiler warnings

Affected code:

- src/EtherFiRedemptionManager.sol

```
# File: src/EtherFiRedemptionManager.sol

EtherFiRedemptionManager.sol:310:     function _authorizeUpgrade(address
newImplementation) internal override onlyOwner {}
```

- src/LiquidityPool.sol

```
# File: src/LiquidityPool.sol

LiquidityPool.sol:579:     function _authorizeUpgrade(address
newImplementation) internal override onlyOwner {}
```

- src/WithdrawRequestNFT.sol

```
# File: src/WithdrawRequestNFT.sol

WithdrawRequestNFT.sol:287:     function _beforeTokenTransfer(address from,
address to, uint256 firstTokenId, uint256 batchSize) internal override {

WithdrawRequestNFT.sol:294:     function _authorizeUpgrade(address
newImplementation) internal override onlyOwner {}
```

I-12. `public` functions not called by the contract should be declared `external` instead

Affected code:

- src/EtherFiRedemptionManager.sol

```
# File: src/EtherFiRedemptionManager.sol

EtherFiRedemptionManager.sol:88:     function redeemEEth(uint256 eEthAmount,
address receiver) public whenNotPaused nonReentrant {

EtherFiRedemptionManager.sol:97:     function redeemWeEth(uint256 weEthAmount,
address receiver) public whenNotPaused nonReentrant {

EtherFiRedemptionManager.sol:301:     function previewRedeem(uint256 shares)
public view returns (uint256) {
```

- src/LiquidityPool.sol

```
# File: src/LiquidityPool.sol
```

```
LiquidityPool.sol:248:     function requestMembershipNFTWithdraw(address
recipient, uint256 amount, uint256 fee) public whenNotPaused returns (uint256)
{

LiquidityPool.sol:423:     function registerAsBnftHolder(address _user) public
onlyAdmin {

LiquidityPool.sol:469:     function rebase(int128 _accruedRewards) public {

LiquidityPool.sol:618:     function amountForShare(uint256 _share) public view
returns (uint256) {
```

- src/WithdrawRequestNFT.sol

```
# File: src/WithdrawRequestNFT.sol

WithdrawRequestNFT.sol:197:     function isFinalized(uint256 requestId) public
view returns (bool) {
```

Invalidated Concerns

Invalid-01. treasury is an immutable variable set to address(0) in the deployment scripts

- https://github.com/etherfi-protocol/smart-contracts/blob/fd52a523caf8f34a1e79a6d9ca7eb95203b490cc/script/deploys/DeployPhaseTwo.s.sol#L84

The PR needs to be fixed as an immutable variable is a constant that can never be changed (even across proxies) :

```
File: WithdrawRequestNFT.sol
20:     address public immutable treasury;
...
56:     constructor(address _treasury) {
57:         treasury = _treasury;
58:
59:         _disableInitializers();
60:     }
```

```
File: DeployPhaseTwo.s.sol
78:     function deploy_WithdrawRequestNFT() internal {
...
84:         withdrawRequestNftImplementation = new
WithdrawRequestNFT(address(0));
```

```
File: DeployPhaseTwo.s.sol
78:     function deploy_WithdrawRequestNFT() internal {
...
84:         withdrawRequestNftImplementation = new
WithdrawRequestNFT(address(0));
```

ether.fi's response: The script is not relevant. It was just added so that compilation does not fail. This upgrade will use the script weEth_withdrawal_v2.s.sol in specialized folder inside script folder.

Invalid-02. Hardcoded 10k gas forwarded

```
File: EtherFiRedemptionManager.sol
156:          (bool success, ) = receiver.call{value: ethReceived, gas:
10_000}("");
```

If the receiver is guaranteed to be an EOA or a minimal contract that only implements receive() or fallback without any significant logic (perhaps not even an event emission), this can work.

However, if the target is a multisig proxy with a fallback that delegateCalls to an implementation contract, this may break. If there's any internal accounting, this may break too.

ether.fi's response: This is for additional security. We will keep this at 10_000.

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.