Security Assessment

Ether.Fi – Instant Withdrawal Merge into 2.49

March 2025

Prepared for EtherFi

Table of Contents

Project Summary

Project Scope

Project Name	Repository (link)	Commit Hashes	Platform
EtherFi smart contracts	etherfi-protocol/smart-contracts	Audit start: f86be238 Audit complete: f2c572e8	EVM

Project Overview

This document describes the manual code review of PR 230 related to v2.49 after merging the Instant Withdrawal feature from PR 207.
The work was a 1 day-effort undertaken from **05/03/2025** to **06/03/2025.**

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`

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	2	2	2
Total	2	2	2

Severity Matrix

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

Informational Severity Issues

I-01. Still writing to the `DEPRECATED_admins` mapping

Affected code:

- src/LiquidityPool.sol:L144

The `admins` mapping was deprecated and renamed to `DEPRECATED_admins`, but there is still a remaining instance where a value is assigned to it:

Relevant Code:

```
Unset
File: src/LiquidityPool.sol
144:             DEPRECATED_admins[_etherFiAdminContract] = true;   //
<----------- Still writing to a deprecated mapping
```

Since `DEPRECATED_admins` is otherwise unused, this assignment serves no functional purpose and should be removed to improve clarity and reduce contract size.

It should be noted that other DEPRECATED variables either remain unassigned of values, or are assigned their default value (false, 0, etc.).

ether.fi's response: Fixed in commit f2c572e8

Certora's response: Reviewed.

I-02. Unused `OwnableUpgradeable` Increasing Contract Size and Deployment Gas

Affected code:

- src/EtherFiRedemptionManager.sol#L30
- src/EtherFiRedemptionManager.sol#L72

The contract `EtherFiRedemptionManager` still inherits `OwnableUpgradeable`, even though the `onlyOwner` modifier is no longer used anywhere in the contract. Additionally, `__Ownable_init()` is called in the `initialize()` function, even though ownership functionality is now handled via `roleRegistry`.

Relevant Code:

```
Unset
File: src/EtherFiRedemptionManager.sol
30: contract EtherFiRedemptionManager is Initializable,
OwnableUpgradeable, PausableUpgradeable, ReentrancyGuardUpgradeable,
UUPSUpgradeable {  // <----------- OwnableUpgradeable is still
inherited but unused
```

```
Unset
File: src/EtherFiRedemptionManager.sol
72:         __Ownable_init();  // <----------- Unnecessary
initializer call, since Ownable is no longer used
```

Recommended Mitigation

- Remove `OwnableUpgradeable` from the contract inheritance.
- Remove the `__Ownable_init();` call in `initialize()`, as it is no longer needed.

ether.fi's response: Fixed in commit f2c572e8

Certora's response: Reviewed.

Appendix: Analysis of Instant Withdrawal Merge into 2.49

Introduction

This appendix provides an in-depth analysis of the instant withdrawal feature's integration into the 2.49 release. The goal of this review was to ensure that the merge did not introduce unintended security risks, permission misconfigurations, or logical inconsistencies. The findings presented here complement the main audit reports by assessing the impact of the merge rather than new functionalities.

The files affected by this merge are:

- `lib/BucketLimiter.sol` (untouched but reviewed for impact)
- `src/LiquidityPool.sol` (most complex as it contains changes from both 2.49 and instant withdrawal)
- `src/WithdrawRequestNFT.sol`
- `src/EtherFiRedemptionManager.sol`

By reviewing this merge in a structured manner, we ensure correctness, validate the new permission system, and check for unintended behaviors introduced by the merge.

File-by-File Breakdown

WithdrawRequestNFT.sol

Changes and Implementation Analysis

- **The `admins` state variable was renamed to `DEPRECATED_admins` and is now unused**
  - The `onlyAdmin` modifier was replaced by the use of `roleRegistry.hasRole(WITHDRAWAL_ADMIN_ROLE, msg.sender)`.
  - The `onlyPauser` modifier was removed, and the `pauseContract()` function now checks `roleRegistry.hasRole(roleRegistry.PROTOCOL_PAUSER(), msg.sender)`, ensuring permissions are correctly enforced.
  - `updateAdmin()` was deleted since `roleRegistry` is the one now handling permission assignments.
  - ✅ **This is a correctly aligned with the new permission model.**

- **Handling of Pausing and Unpausing**
  - The `pauser` state variable was removed
  - `initializeOnUpgrade` now initializes `roleRegistry` instead of `pauser`.
  - `pauseContract()` previously required `onlyPauser`; now, it checks for `PROTOCOL_PAUSER` via `roleRegistry`.
  - `unPauseContract()` previously required `onlyAdmin`; now, it checks for `PROTOCOL_UNPAUSER` via `roleRegistry`.
  - Additional checks ensure that the contract is actually paused before unpausing or unpaused before pausing, preventing redundant calls.
  - ✅ **The compiler would've caught most issues here (and any missing code needing refactoring) and the current changes actually improve the code's role-based security and prevent unnecessary state changes.**

Findings:

- **None.** The migration to `roleRegistry` is done correctly.

BucketLimiter.sol

- ✅ **No impact from the merge.**

EtherFiRedemptionManager.sol

Changes and Implementation Analysis

- **Updating the Upgrade Authorization Mechanism**
  - `onlyOwner` was removed from `_authorizeUpgrade()`.
  - Now calls `roleRegistry.onlyProtocolUpgrader(msg.sender)`, ensuring **protocol-wide upgrade control**.
  - ✅ **This aligns with the new permission system and improves security.**

Findings:

- **Potential Gas Optimization:**
  - The contract still inherits `OwnableUpgradeable`, even though `onlyOwner` is no longer used.
  - The function `initialize()` still calls `__Ownable_init();`, which now seems **redundant**.
  - 🔍 **Informational Concern:** Removing `OwnableUpgradeable` would reduce contract size and deployment gas costs.

LiquidityPool.sol

Changes and Implementation Analysis

- **New Role-Based Permissions**
  - Introduced `roleRegistry` and a new `LIQUIDITY_POOL_ADMIN_ROLE`.
  - The `admins` mapping was deprecated (`DEPRECATED_admins`) but **still has one remaining reference in `DEPRECATED_admins[_etherFiAdminContract] = true;`.**
  - ✅ **This transition to `roleRegistry` was largely successful. It could be further optimized by removing the remaining assignment to `DEPRECATED_admins`**
- **Whitelisting Mechanism Removed**
  - `_isWhitelisted()` and all references to whitelisted users were removed.
  - Previously, deposits required `_isWhitelisted(msg.sender)`, but now **any user can call `deposit()`**.
  - ⚠️ **Important Note:** During previous audits, whitelisted users were not considered "trusted," so removing this mechanism should not introduce a new risk. However, it does mean that deposit access is now fully permissionless.
- **Permission Change on `setTreasury`**
  - Previously restricted by `onlyOwner`, now governed by `LIQUIDITY_POOL_ADMIN_ROLE`.
  - **Uncertainty:** Either the previous `onlyOwner` usage was a misconfiguration, or this change reduces security.
    - **Ether.Fi team's answer:** Using admin is more appropriate as having this function behind a timelock who gets node operator/etherfi rewards is too strong.
    - ✅ **Certora's answer:** Agreed, this is an improvement.
- **Upgrade Security and Finalization**
  - `_authorizeUpgrade()` was changed from `onlyOwner` to `roleRegistry.onlyProtocolUpgrader(msg.sender)`, ensuring that **only designated upgraders can approve implementations.**
  - ✅ **This strengthens security by preventing unauthorized contract upgrades.**

Findings:

- 🔍 **Possible unnecessary assignation to `DEPRECATED_admins` remaining.**

Findings Summary

Finding	Severity	Description
`DEPRECATED_admins` lingering reference	🔍 **Informational**	Mapping was renamed but still has one remaining write operation.
Removal of `OwnableUpgradeable`	🔍 **Informational**	Now unused, removing it could optimize gas.

Conclusion

The merge successfully integrates the **instant withdrawal feature** into the **2.49 release**, ensuring compatibility with the new **role-based access control system**.

- ✅ **The refactoring to `roleRegistry` is correct and ensures security.**
- ✅ **The instant withdrawal functionality is preserved without issue.**
- ✅ **The permission change in `setTreasury()` is confirmed to be an improvement.**
- ⚠️ **The removal of whitelisting expands deposit access but does not introduce new security risks based on previous audits.**
- 🔍 **Optimizations (removing `OwnableUpgradeable` and `DEPRECATED_admins`) should be considered.**

This merge is **safe to proceed**, pending minor optimizations.

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.