Security Assessment

ether.fi - Safe Recovery Manager

March 2025

Prepared for ether.fi

Table of content

Project Summary....................3
Project Scope....................3
Project Overview....................3
Findings Summary....................3
Severity Matrix....................4
C-01 A single safe admin can drain the safe via a malicious bytes calldata data....................5
M-01 Not resetting incomingOwnerStartTime in _currentOwner might lead to unexpected state changes.... 7
M-02 RecoveryManager::overrideRecoverySigners doesn't correctly handle recoverySigners according to EIP712....................8
M-03 The incomingOwner doesn't get granted the safe admin role....................9
M-04 Previous owners don't get their safe admin role revoked.................... 10
Disclaimer.................... 11
About Certora.................... 11

Project Summary

Project Scope

Project Name	Repository (link)	Commit Hashes	Platform
EtherFi - cash-v3	etherfi-protocol/cash-v3	Audit start: PR 5 at 2b37e643 Audit end: PR 5 at df6d6c59 Last reviewed: 6bc9436	EVM

Project Overview

This document describes the manual code review of PR 5 related to "Recovery feature added to safe".

The work was a 5-day effort undertaken from **20/03/2025** to **26/03/2025**

The following contract list is included in our scope:

- src/data-provider/EtherFiDataProvider.sol
- src/hook/EtherFiHook.sol
- src/interfaces/IEtherFiSafe.sol
- src/interfaces/IModule.sol
- src/modules/openocean-swap/OpenOceanSwapModule.sol
- src/safe/EtherFiSafe.sol
- src/safe/ModuleManager.sol
- src/safe/MultiSig.sol
- src/safe/RecoveryManager.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	1	1	1
High	-	-	-
Medium	4	4	4
Low	-	-	-
Total	5	5	5

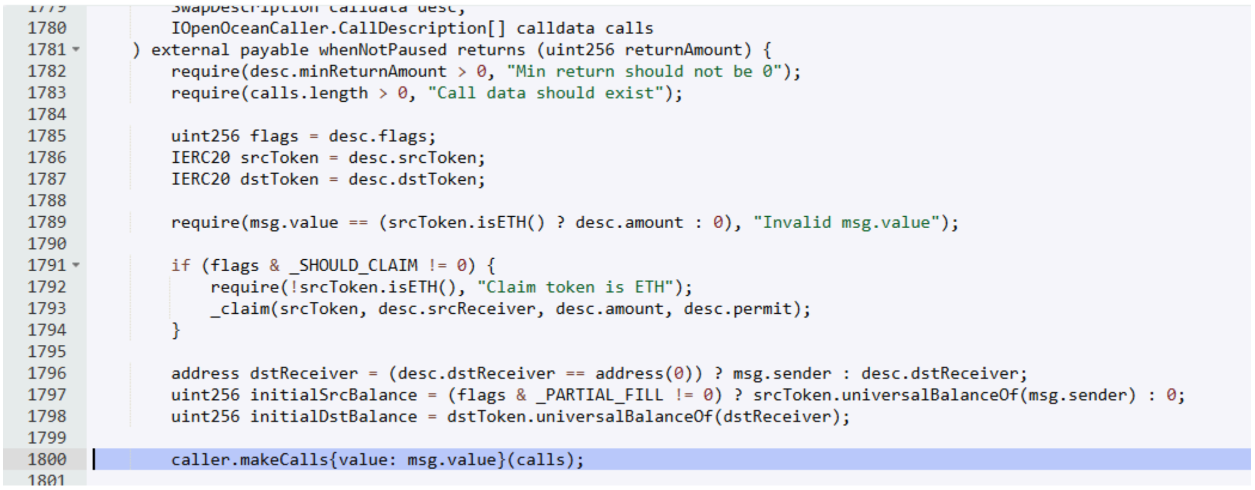
Severity Matrix

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

C-01 A single safe admin can drain the safe via a malicious `bytes calldata data`		
Severity: **Critical**	Impact: **High**	Likelihood: **High**
Files: OpenOceanSwapModule.sol	Status: Fixed	

Description: The `OpenOceanSwapModule::swap` function requires a single admin signature to execute swaps. If this admin supplies malicious data to the swap function he will be able to drain the safe. The admin should just call the swap function with the `minToAssetAmount = 1` (0 is not permitted) and inside the `data` he includes a custom made `executor` contract that will receive all of the `fromAssetAmount` of tokenA and will transfer 1 wei to the safe so that the slippage check passes.

This happens because underneath the OpenOcean router calls this executor contract and transfers it all of the funds that were sent from the safe - https://scrollscan.com/address/0xdec876911cbe9428265af0d12132c52ee8642a99#code



```
        SwapDescription calldata desc,
        IOpenOceanCaller.CallDescription[] calldata calls
    ) external payable whenNotPaused returns (uint256 returnAmount) {
        require(desc.minReturnAmount > 0, "Min return should not be 0");
        require(calls.length > 0, "Call data should exist");

        uint256 flags = desc.flags;
        IERC20 srcToken = desc.srcToken;
        IERC20 dstToken = desc.dstToken;

        require(msg.value == (srcToken.isETH() ? desc.amount : 0), "Invalid msg.value");

        if (flags & _SHOULD_CLAIM != 0) {
            require(!srcToken.isETH(), "Claim token is ETH");
            _claim(srcToken, desc.srcReceiver, desc.amount, desc.permit);
        }

        address dstReceiver = (desc.dstReceiver == address(0)) ? msg.sender : desc.dstReceiver;
        uint256 initialSrcBalance = (flags & _PARTIAL_FILL != 0) ? srcToken.universalBalanceOf(msg.sender) : 0;
        uint256 initialDstBalance = dstToken.universalBalanceOf(dstReceiver);

        caller.makeCalls{value: msg.value}(calls);
```

The custom executor contract needs to have this `makeCalls` function and will receive all of the funds.

But even if this executor is the expected `OpenOceanCaller` the admin can still include a malicious `CallDescription` and execute low-level calls of transferring the funds directly to an address of his choice.

Recommendations: It probably makes sense for this function to be executed only if the `threshold` of owners of the safe sign the transaction

Customer's response: Fixed in commit 5b96230

Fix Review: Fix confirmed. `OpenOceanSwapModule::swap` now requires a certain `threshold` of owners of the safe to sign the transaction.

M-01 Not resetting incomingOwnerStartTime in _currentOwner might lead to unexpected state changes		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: MultiSig.sol	Status: Fixed	

Description: The `_currentOwner` function is called every time an owner permissioned transaction happens and it removes all of the current owners from the safe and adds the incoming owner. Because the protocol doesn't reset the `incomingOwnerStartTime` once the `_currentOwner()` function is executed the first time, a scenario could occur where all of the current owners are removed without that being expected. For example:

Let's say after a safe recovery we set a single owner (`incomingOwner`) and after that this owner wants to add more owners so that it becomes a multi-sig again. Then he sets the threshold to be let's say 3 because he added 5 owners. Now before a certain owner operation is executed we call `_currentOwner()` that will remove all of the newly added 5 owners and will reset the threshold to 1 again.

Recommendations: Reset the `incomingOwnerStartTime` and `incomingOwner` once the `_currentOwner()` has been executed for the first time because the owner was already added to the `owners` mapping you don't need to keep track of him.

Customer's response: Fix confirmed 18eb41f

Fix Review: Fix confirmed

M-02 RecoveryManager::overrideRecoverySigners doesn't correctly handle recoverySigners according to EIP712

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: RecoveryManager.sol	Status: Fixed	

Description: Directly using the actual variable instead of hashing the concatenated values of the array goes against the EIP-712 specification.

In `overrideRecoverySigners`, this is not done correctly, which would result in a different signature than expected.

Recommendations: OpenSea's Seaport's example with offerHashes and considerationHashes can be used as a reference to understand how arrays should be encoded.

Customer's response: Fixed in commit 5c138b5

Fix Review: Fix confirmed

M-03 The `incomingOwner` doesn't get granted the safe admin role

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: MultiSig.sol	Status: Fixed	

Description: Usually when new owners are added we should add them to the safe admin role so they can execute admin restricted actions, however when a safe is recovered and if this incoming owner didn't have the admin role of the safe he won't be able to execute any admin operations.

Recommendations: Add the admin role to the incoming owner by calling the `_configureAdmin()` function after we add that owner to the mapping

Customer's response: Fixed in commit df6d6c5

Fix Review: Fix confirmed

M-04 Previous owners don't get their safe admin role revoked

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: MultiSig.sol	Status: Fixed	

Description: Once a safe is recovered the protocol removes all of the owners from the `owners` mapping however it doesn't revoke their safe admin role, which means they can still execute admin restricted functions to configure the safe as they wish. If these owners are malicious they could change some safe configurations which could result in an unexpected behavior

Recommendations: Revoke all of the previous owners' safe admin role by using the `_configureAdmin` function

Customer's response: Fixed in commit df6d6c5

Fix Review: Fix confirmed

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.