Security Assessment

ether.fi – Cash Module + Safe

March 2025

Prepared for ether.fi

certora

ether.fi

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Commit Hashes	Platform
EtherFi - cash-v3	etherfi-protocol/cash-v3	Audit start: PR 3 at 45dc3ee Audit end: PR 3 at d6aa4845	EVM

Project Overview

This document describes the manual code review of PR 3 related to “Cash Safe + Module”.

The work was a 10-day effort undertaken from **07/03/2025** to **20/03/2025**

The following contract list is included in our scope:

- src/beacon-factory/BeaconFactory.sol
- src/cashback-dispatcher/CashbackDispatcher.sol
- src/data-provider/EtherFiDataProvider.sol
- src/debt-manager/DebtManagerAdmin.sol
- src/debt-manager/DebtManagerCore.sol
- src/debt-manager/DebtManagerInitializer.sol
- src/debt-manager/DebtManagerStorage.sol
- src/hook/EtherFiHook.sol
- src/interfaces/ICashDataProvider.sol
- src/interfaces/ICashEventEmitter.sol
- src/interfaces/ICashLens.sol
- src/interfaces/ICashModule.sol
- src/interfaces/ICashbackDispatcher.sol
- src/interfaces/IDebtManager.sol
- src/interfaces/IEtherFiDataProvider.sol
- src/interfaces/IEtherFiHook.sol
- src/interfaces/IEtherFiSafe.sol
- src/interfaces/IEtherFiSafeFactory.sol

- src/interfaces/IModule.sol
- src/interfaces/IPermission.sol
- src/interfaces/IPriceProvider.sol
- src/interfaces/IRoleRegistry.sol
- src/libraries/ArrayDeDupLib.sol
- src/libraries/CashVerificationLib.sol
- src/libraries/EnumerableAddressWhitelistLib.sol
- src/libraries/SignatureUtils.sol
- src/libraries/SpendingLimitLib.sol
- src/libraries/TimeLib.sol
- src/modules/ModuleBase.sol
- src/modules/cash/CashEventEmitter.sol
- src/modules/cash/CashLens.sol
- src/modules/cash/CashModuleCore.sol
- src/modules/cash/CashModuleSetters.sol
- src/modules/cash/CashModuleStorageContract.sol
- src/role-registry/RoleRegistry.sol
- src/safe/EtherFiSafe.sol
- src/safe/EtherFiSafeErrors.sol
- src/safe/EtherFiSafeFactory.sol
- src/safe/ModuleManager.sol
- src/safe/MultiSig.sol
- src/top-up/TopUpDest.sol
- src/utils/ReentrancyGuardTransientUpgradeable.sol
- src/utils/StorageSlot.sol
- src/utils/UpgradeableProxy.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	2	2	2
Medium	6	6	4
Low	6	6	6
Total	16	16	14

Severity Matrix

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

Critical Severity Issues

C-01 `CashbackDispatcher::clearPendingCashback` lacks access control which allows anyone to drain the contract

Severity: **Critical**	Impact: **High**	Likelihood: **High**
Files: CashbackDispatcher.sol	Status: Fixed	

Description: The `clearPendingCashback` function in `CashbackDispatcher` lacks proper access control, allowing any user to call it. This function is intended to be executed by `CashModuleCore` to process cashback and reset the `pendingCashbackInUsd` mapping. However, due to the missing restriction, a malicious user can repeatedly call `clearPendingCashback` without resetting the mapping, enabling them to drain all available funds from the `CashbackDispatcher` contract.

Recommendations: Similarly to `CashbackDispatcher::cashback` allow only the cashModule to clear pending cashback

Customer's response: Fixed in commit d6aa484

Fix Review: Fix confirmed

C-02 `CashbackDispatcher::clearPendingCashback` sends the cashback directly to `msg.sender` instead of the account provided

Severity: **Critical**	Impact: **High**	Likelihood: **High**
Files: CashbackDispatcher.sol	Status: Fixed	

Description: The `clearPendingCashback` function in `CashbackDispatcher` incorrectly sends cashback funds to `msg.sender` instead of the intended recipient. Since this function is called by `CashModuleCore`, the cashback amount is transferred to the module itself rather than the designated safe/spender. As a result, the funds remain stuck within the `CashModuleCore` contract and are never properly distributed to the intended recipient, preventing cashback payouts.

Recommendations: send the funds to the `clearPendingCashback` argument provided instead of `msg.sender`

Customer's response: Fixed in commit d6aa484

Fix Review: Fix confirmed

High Severity Issues

H-01 `EtherFiSafe` cannot execute transactions on chains where the `hook = address(0)`

Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: EtherFiDataProvider.sol#L121-L122 EtherFiSafe.sol#L289	Status: Fixed	

Description: In `EtherFiDataProvider::initialize` we set the hook address for the current blockchain and a comment states that some blockchains will not have a hook - https://github.com/etherfi-protocol/cash-v2/blob/12cbd745e029d1941f6f60b8e4ad9ff3d2528d4a/src/data-provider/EtherFiDataProvider.sol#L121-L122.

So on these chains this address will be 0.
When a module tries to execute a transaction in `EtherFiSafe::execTransactionFromModule` the hook is called before and after the transaction, however because some chains don't have a hook `hook.preOpHook(msg.sender);` and `hook.postOpHook(msg.sender);` will revert - https://github.com/etherfi-protocol/cash-v2/blob/12cbd745e029d1941f6f60b8e4ad9ff3d2528d4a/src/safe/EtherFiSafe.sol#L289

Recommendations: In `EtherFiSafe::execTransactionFromModule` check if the hook is `address(0)` and don't call it if that is the case

Customer's response: Fixed in commit 34546ae

Fix Review: Fix confirmed

certora

ether.fi

H-02 Long-Term Insolvency of the Lending Market Due to Total Debt Compounding More Frequently than Individual User Debts

Severity: **High**	Impact: **Medium**	Likelihood: **High**
Files: DebtManagerCore.sol	Status: Fixed	

Description: The `DebtManagerCore` contract compounds interest for the total borrowing amount every time an action affecting the total debt amount occurs (borrowing or repaying). However, users only have their interest compounded when they interact individually. This means that so long as more than one open borrow position exists, for any debt-affecting interaction, the total borrowed amount increases by more than the sum of the total user borrowed amounts. The valuation of lending shares is based on the total borrowed amount. Thus, the lending market suffers from an exponentially compounding insolvency over time. Even if all users repay their debt, the claims of lenders on borrow tokens will exceed the liquidity in the market.

POC:

Place in test/safe/modules/cash and run with `forge test --mt testLongTermSolvency -vv`.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.28;

import { Test } from "forge-std/Test.sol";

import { ERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

import { DebtManagerAdmin } from
"../../../../src/debt-manager/DebtManagerAdmin.sol";
import { DebtManagerCore } from
"../../../../src/debt-manager/DebtManagerCore.sol";
import { DebtManagerInitializer } from
"../../../../src/debt-manager/DebtManagerInitializer.sol";
import { IDebtManager } from "../../../../src/interfaces/IDebtManager.sol";
import { UUPSProxy } from "../../../../src/UUPSProxy.sol";

contract TestToken is ERC20 {
    uint8 immutable DECIMALS;
```

```
    constructor(string memory name, string memory symbol, uint8 _decimals)
ERC20(name, symbol) {
        DECIMALS = _decimals;
    }

    function decimals() public view override returns (uint8) {
        return DECIMALS;
    }

    function mint(address to, uint256 amount) external {
        _mint(to, amount);
    }
}

contract MockPriceProvider {
    mapping (address => uint256) public price;
    function setPrice(address token, uint256 _price) external {
        price[token] = _price;
    }
}

contract MockCashLens {
    mapping (address => IDebtManager.TokenData[]) collateralTokens;

    function getUserTotalCollateral(address user) external view returns
(IDebtManager.TokenData[] memory) {
        return collateralTokens[user];
    }

    function setUserTotalCollateral(address user, IDebtManager.TokenData[]
calldata _collateralTokens) external {
        delete collateralTokens[user];
        for (uint256 i = 0; i < _collateralTokens.length; i++) {
            collateralTokens[user].push(_collateralTokens[i]);
        }
    }
}

contract MockCashModule {
    function getSettlementDispatcher() external pure returns (address) {
        return address(0x537713);
    }
}

contract MockEtherFiDataProvider {
    address immutable priceProvider;
```

```
    address immutable cashLens;
    address immutable cashModule;
    mapping (address => bool) public isEtherFiSafe;

    constructor(address _priceProvider, address _cashLens, address
_cashModule) {
        priceProvider = _priceProvider;
        cashLens = _cashLens;
        cashModule = _cashModule;
    }

    function getPriceProvider() external view returns (address) {
        return priceProvider;
    }

    function getCashLens() external view returns (address) {
        return cashLens;
    }

    function getCashModule() external view returns (address) {
        return cashModule;
    }

    function addSafe(address safe) external {
        isEtherFiSafe[safe] = true;
    }
}

contract DebtManager is DebtManagerCore, DebtManagerInitializer {}

contract DebtManagerTest is Test {
    uint64 constant BORROW_APY = 158_548_959_919;

    address constant LENDER_1 = address(0x1001);
    address constant LENDER_2 = address(0x1002);

    address constant SAFE_1 = address(0x2001);
    address constant SAFE_2 = address(0x2002);

    IDebtManager debtManager;

    MockPriceProvider mockPriceProvider;
    MockCashLens mockCashLens;
    MockCashModule mockCashModule;
    MockEtherFiDataProvider mockDataProvider;
    TestToken weth;
    TestToken usdc;
```

```
    function setUp() public virtual {
        mockPriceProvider = new MockPriceProvider();
        mockCashLens = new MockCashLens();
        mockCashModule = new MockCashModule();
        mockDataProvider = new MockEtherFiDataProvider(
            address(mockPriceProvider),
            address(mockCashLens),
            address(mockCashModule));

        address debtManagerImpl = address(new DebtManager());
        debtManager = IDebtManager(address(new UUPSProxy(debtManagerImpl,
"")));
        DebtManager(address(debtManager)).initialize(address(this),
address(mockDataProvider));
        address debtManagerAdmin = address(new DebtManagerAdmin());
        debtManager.setAdminImpl(debtManagerAdmin);

        weth = new TestToken("WETH", "WETH", 18);

        mockPriceProvider.setPrice(address(weth), 2_000e6);

        IDebtManager.CollateralTokenConfig memory wethCollateralConfig =
IDebtManager.CollateralTokenConfig({
            ltv: 80e18,
            liquidationThreshold: 90e18,
            liquidationBonus: 5e18

        });

        debtManager.supportCollateralToken(address(weth),
wethCollateralConfig);

        usdc = new TestToken("USDC", "USDC",  6);

        mockPriceProvider.setPrice(address(usdc), 1e6);

        IDebtManager.CollateralTokenConfig memory usdcCollateralConfig =
IDebtManager.CollateralTokenConfig({
            ltv: 95e18,
            liquidationThreshold: 99e18,
            liquidationBonus: 0.5e18

        });

        debtManager.supportCollateralToken(address(usdc),
usdcCollateralConfig);
```

```
        debtManager.supportBorrowToken(address(usdc), BORROW_APY, 1);  //
zero minShares for simplicity

        mockDataProvider.addSafe(SAFE_1);
        mockDataProvider.addSafe(SAFE_2);
    }

    function testLongTermSolvency() public {
        usdc.mint(LENDER_1, 1_000_000e6);

        // LENDER_1 supplies one million USDC to the market for borrowing.
        vm.startPrank(LENDER_1);
        usdc.approve(address(debtManager), type(uint256).max);
        debtManager.supply(LENDER_1, address(usdc), 1_000_000e6);
        vm.stopPrank();

        usdc.mint(LENDER_2, 1_000_000e6);

        // LENDER_2 also supplies one million USDC to the market for
borrowing.
        vm.startPrank(LENDER_2);
        usdc.approve(address(debtManager), type(uint256).max);
        debtManager.supply(LENDER_2, address(usdc), 1_000_000e6);
        vm.stopPrank();

        IDebtManager.TokenData[] memory safe1Collaterals = new
IDebtManager.TokenData[](1);
        safe1Collaterals[0].token = address(weth);
        safe1Collaterals[0].amount = 10_000_000e18;
        mockCashLens.setUserTotalCollateral(SAFE_1, safe1Collaterals);

        vm.startPrank(SAFE_1);
        debtManager.borrow(address(usdc), 1_480_000e6);
        vm.stopPrank();

        vm.warp(block.timestamp + 13 weeks);

        // Give the second safe the same amount of collateral as the first.
        mockCashLens.setUserTotalCollateral(SAFE_2, safe1Collaterals);

        for (uint256 i = 0; i < 52; i++) {
            vm.warp(block.timestamp + 1 weeks);

            // With every weekly borrowing, the original borrow by SAFE_1
compounds
            // in the total borrowings, but not in SAFE_1's account.
            vm.startPrank(SAFE_2);
```

```
            debtManager.borrow(address(usdc), 10_000e6);
            vm.stopPrank();
        }

        // SAFE_1 does something to update their borrow state.
        usdc.mint(SAFE_1, 1_000e6);
        vm.startPrank(SAFE_1);
        usdc.approve(address(debtManager), type(uint256).max);
        debtManager.repay(SAFE_1, address(usdc), 1_000e6);
        vm.stopPrank();

        // Total supplies (debt + token balance)
        uint256 totalSupplies = debtManager.totalSupplies(address(usdc));
        emit log_named_uint("totalSupplies", totalSupplies);

        uint256 debtManagerUsdcBal = usdc.balanceOf(address(debtManager));
        emit log_named_uint("debtManagerUsdcBal", debtManagerUsdcBal);

        uint256 safe1Debt = debtManager.borrowingOf(SAFE_1, address(usdc));
        emit log_named_uint("safe1Debt", safe1Debt);

        uint256 safe2Debt = debtManager.borrowingOf(SAFE_2, address(usdc));
        emit log_named_uint("safe2Debt", safe2Debt);

        emit log_named_uint("safe debts plus USDC bal", debtManagerUsdcBal +
safe1Debt + safe2Debt);

        emit log_named_uint("shortfall", totalSupplies - debtManagerUsdcBal
- safe1Debt - safe2Debt);

        uint256 lender1Balance = debtManager.supplierBalance(LENDER_1,
address(usdc));
        emit log_named_uint("lender1Balance", lender1Balance);

        uint256 lender2Balance = debtManager.supplierBalance(LENDER_2,
address(usdc));
        emit log_named_uint("lender2Balance", lender2Balance);

        emit log_named_uint("sum of lender balances", lender1Balance +
lender2Balance);
    }
}
```

Recommendations: Modify the accounting model to one that can accurately track compounding for all users even when they don't interact; refer to any of the numerous existing lending market codebases for examples of how to do this. A simpler fix is to only increase the total debt by the per-user compounded amount in each interaction step, but this is unfair to users with higher interaction frequencies.

Customer's response: Fixed in commit 104f591

Fix Review: Fix confirmed. The protocol introduced a new accounting model that compounds the interest index instead of the total borrowed amount. That model fixes the issue

Medium Severity Issues

M-01 `CONFIGURE_MODULES_TYPEHASH` doesn't include the `bytes[] calldata moduleSetupData`

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: EtherFiSafe.sol#L49-L51 EtherFiSafe.sol#L250-L251	Status: Fixed	

Description: The `CONFIGURE_MODULES_TYPEHASH` in `EtherFiSafe` does not include the `bytes[] calldata moduleSetupData` parameter. This omission results in an incorrect hash being generated, causing signature mismatches when verifying off-chain signatures that were signed with `moduleSetupData`.

Recommendations: Include the `bytes[] moduleSetupData` in the type hash

Customer's response: Fixed in commit 34546ae

Fix Review: Fix confirmed

M-02 `EtherFiSafe::configureModules` doesn't correctly hash the `moduleSetupData`

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: EtherFiSafe.sol#251	Status: Fixed	

Description: According to EIP712 `bytes` and `string` types should be `keccak256` and then be included in the hash, however `configureModules` doesn't do that and just hashes the array struct while it should hash each element of that array and then concatenate the hashes and hash again. Quoting from the EIP:

"The dynamic values bytes and string are encoded as a keccak256 hash of their contents.
The array values are encoded as the keccak256 hash of the concatenated encodeData of their contents (i.e. the encoding of SomeType[5] is identical to that of a struct containing five members of type SomeType)."

Recommendations: In our case `moduleSetupData` is `bytes[]` so each element is of type `bytes` that should be hashed and then hash again the concatenated elements of that array. So that is the fix, to loop over the `moduleSetupData` and hash each element individually, then concatenate and hash again.

Customer's response: Fixed in commit a773098

Fix Review: Fix confirmed

M-03 No incentive to liquidate small positions

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: DebtManagerCore.sol #L335	Status: Acknowledged	

Description: There is no min borrowing limit which means that users can create such small positions that aren't worth liquidating because the liquidator would lose money. For example if the borrowed amount is 5$ but the liquidation transaction costs 7$ worth of gas no one would want to liquidate it. This however is very limited because 1 user can only create 1 such small debt because 1 user only has 1 safe. However if a lot of people do it, it can still create bad debt

Recommendations: Require a min amount of borrowed of USD to be borrowed

Customer's response: Acknowledged

M-04 Users can be unfairly liquidated

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: DebtManagerCore.sol#L353	Status: Acknowledged	

Description: A user might appear to be liquidatable when he has a pending withdrawal, however after canceling that pending withdrawal in `CashModuleCore::preLiquidate` this user might have a healthy factor but he will be liquidated anyways. This is because `CashLens::getUserTotalCollateral` subtracts the pending borrow request from the collateral and in `cashModule.preLiquidate` we cancel that withdrawal request so the user can be with a healthy factor after the cancel but he will still get liquidated

Recommendations: After the call to `preLiquidate` check if the user is still liquidatable and if he is not, return the call without liquidating

Customer's response: Acknowledged

M-05 Repay might leave unused approval which causes problems with USDT

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: CashModuleCore.sol#L436	Status: Fixed	

Description: The `_repay` function in `CashModuleCore` may leave an unused approval if a user attempts to repay more than they owe. This behavior is particularly problematic for USDT, which requires an explicit reset of allowance to zero before setting a new value.

If `_repay` is called a second time, the transaction will revert due to USDT's strict allowance mechanism, as the existing non-zero approval to the `debtManager` will prevent re-approval.

Recommendations: Add a zero approval transaction to the debtManager before approving to the real amount

Customer's response: Fixed in commit 0abbebb

Fix Review: Fix confirmed

M-06 Inconsistent Minimum Shares Validation in `DebtManagerCore::supply` and `DebtManagerCore::withdraw`

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: DebtManagerCore.sol #L268	Status: Fixed	

Description: The `supply` function in `DebtManagerCore` restricts users from depositing an amount that results in shares below `config.minShares`. However, the `withdrawBorrowToken` function only ensures that the **total shares** remain above `config.minShares`, rather than enforcing this constraint on individual withdrawals.

This inconsistency limits user deposits, preventing small contributions that would yield fewer than `minShares`. The likely intent of the restriction is to prevent a "first depositor inflation attack," where an initial small deposit could gain disproportionate shares. To align with this intent, the validation should be performed against `totalSharesOfBorrowTokens` instead of individual deposits.

Recommendations: In `DebtManager::supply` first add the shares to the total:
`_borrowTokenConfig[borrowToken].totalSharesOfBorrowTokens += shares;`

and only then check :
```
if (_borrowTokenConfig[borrowToken].totalSharesOfBorrowTokens <
_borrowTokenConfig[borrowToken].minShares) { revert
SharesCannotBeLessThanMinShares(); }
```

Customer's response: Fixed in commit 696fc6a

Fix Review: Fix confirmed

Low Severity Issues

L-01 Liquidations May Fail Unexpectedly If Some Collateral Amounts Are Zero

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: CashModuleCore.sol	Status: Fixed	

Description:

```
       uint256 counter = 0;

        for (uint256 i = 0; i < len;) {
            if (tokensToSend[i].amount > 0) {
                to[i] = tokensToSend[i].token;
                data[i] = abi.encodeWithSelector(IERC20.transfer.selector, liquidator,
tokensToSend[i].amount);
                unchecked {
                    ++counter;
                }
            }

            unchecked {
                ++i;
            }
        }

        assembly ("memory-safe") {
            mstore(to, counter)
            mstore(data, counter)
        }
```

The `postLiquidate()` function can revert due to a call to the zero address if some amounts are zero. The loop over the liquidated tokens writes to the `to` and `data` arrays at the index of the loop counter (`i`) when it should actually write at the index of the `counter` variable, which counts tokens with non-zero amounts. These two arrays later have their lengths manually set to the final value of the `counter`. As an example, consider a case where `tokensToSend` contains

two tokens, the first of which has a zero amount, and the second of which is non-zero. The `to` and `data` arrays will have a length of 1 but the first entry was never initialized--the attempt to call the zero address in `execTransactionFromModule` will revert, causing the liquidation to fail.

Recommendations: Liquidators should generally be able to avoid this by using only tokens in the `collateralTokensPreference` argument to `liquidate()` that the user actually possess, so there's little impact in practice; however, it is still recommended to fix the code as doing so is straightforward and simplifies the work of integrators.

Customer's response: Fixed in commit 5d699a6

Fix Review: Fix confirmed

L-02 Some Contracts Do Not Disable Initializers in Their Constructors

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: CashLens.sol#L44 CashModuleCore.sol#L36 EtherFiSafe.sol#L114 EtherFiDataProvider.sol	Status: Fixed	

CashLens.sol#L44
CashModuleCore.sol#L36
EtherFiSafe.sol#L114
EtherFiDataProvider.sol

These contracts don't disable the initializers in their constructor and allow anyone to initialize the implementations with unwanted data.

Customer's response: Fixed in commit 2d3fed3

Fix Review: Fix confirmed

L-03 `CashLens::canSpend` incorrectly accounts for the mode change

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: CashLens.sol#L88	Status: Acknowledged	

Description: We should compare the `incomingCreditModeStartTime` against `block.timestamp` before switching the mode to Credit in `CashLens::canSpend`. Otherwise, it may report that transaction can succeed when it will not, which could lead to unexpected failures and bad user experience

Customer's response: Acknowledged

L-04 `CashLens::_calculateDebitModeAmount` passes a wrong argument to `getBorrowingPowerAndTotalBorrowing`

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: CashLens.sol#L308	Status: Fixed	

Description: The first argument to getBorrowingPowerAndTotalBorrowing() is the user to get the total borrowings of; here this is being passed as address(0) but it should be the safe. Because address(0) has no borrowing position, the totalBorrowings return will always be zero and the subsequent logic will never correctly account for the collateral requirements of the user's borrowings, potentially overestimating the maximum spend amount.

Customer's response: Fixed in commit 6e71c14

Fix Review: Fix confirmed

L-05 `CashLens::_calculateDebitModeAmount` may overestimate the maximum Debit mode withdrawal if the safe is in Credit mode

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: CashLens.sol#L300	Status: Fixed	

Description:

In CashLens._calculateDebitModeAmount, _getCollateralBalanceWithTokenSubtracted is invoked with a non-zero amount argument. The handling of this argument depends on the current mode of the safe:
https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/modules/cash/CashLens.sol#L352-L355
Because this logic is intended to calculate its result under the assumption of mode == Debit, the mode field of the safeData argument should be set to Debit before the call to ensure a correct result. Otherwise, maxCanSpend may overestimate the maximum Debit mode withdrawal.

Customer's response: Fixed in commit 6e71c14

Fix Review: Fix confirmed

L-06 CashLens._calculateCreditModeAmount doesn't ensure the mode is Credit

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: CashLens.sol#L264	Status: Fixed	

Description:

In CashLens._calculateCreditModeAmount, _getCollateralBalanceWithTokenSubtracted is invoked without first ensuring that the mode of the safeData argument is Credit; this may result in an underestimate of the maximum spend amount with there is a pending withdrawal for the target token equal to its balance:
https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/modules/cash/CashLens.sol#L351-L353
https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/modules/cash/CashLens.sol#L267-L269
The mode of the safeData argument should be set to Credit to ensure accurate results.

Customer's response: Fixed in commit 6e71c14

Fix Review: Fix confirmed

Informational Severity Issues

I-01. Unused Imports

Description:

- https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/modules/cash/CashModuleCore.sol#L15
- https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/modules/cash/CashModuleStorageContract.sol#L15
- https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/modules/cash/CashModuleSetters.sol#L13

UpgradeableBeacon only:

- https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/EtherFiSafeFactory.sol#L4
- https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/EtherFiSafeFactory.sol#L7-L8

Customer's response: Fixed in commit ef608f4

Fix Review: Fix confirmed

I-02. Inconsistencies In Documenting Errors That Can Be Thrown

Description:
Some natspec comments exhaustively document the errors that can be thrown by various functions; others do not. In the cases where this is done, the comments are sometimes inaccurate.

https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/MultiSig.sol#L112
The comment states that InvalidOwnerAddress is thrown if an input address is zero, but the actual error is InvalidAddress(uint256 index) (https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/libraries/EnumerableAddressWhitelistLib.sol#L23). Further, this function can also throw DuplicateElementFound if an address is repeated, but this is not documented in the comments.

https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/ModuleManager.sol#L53-L55
ModuleManager._setupModules does not correctly document all the errors that it can throw. In addition to the errors that it lists, it can also throw:

- ModulesAlreadySetup (https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/ModuleManager.sol#L60)
- ArrayLengthMismatch (https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/ModuleManager.sol#L64)
- DuplicateElementFound (https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/ModuleManager.sol#L65)

If the call to setupModule() fails, it is likely intended that this should throw ModuleSetupFailed based on a comment on _configureModules():
https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/ModuleManager.sol#L92

https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/ModuleManager.sol#L100

ModuleManager._configureModules can throw DuplicateElementFound but does not document this in its natspec comments.

https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/MultiSig.sol#L59
This comment states that AlreadySetup is thrown, but actually the name of the error is MultisigAlreadySetup.

https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/EtherFiSafe.sol#L160
https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/EtherFiSafe.sol#L175
https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/EtherFiSafe.sol#L210
https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/EtherFiSafe.sol#L232
https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/EtherFiSafe.sol#L254
Functions in EtherFiSafe.sol that invoke the checkSignatures() function fail to document the various errors that can be thrown by checkSignatures().

https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/safe/ModuleManager.sol#L88
This comment should mention that the length consistency check applies to _moduleSetupData as well.

Customer's response: Fixed in commit ef608f4

Fix Review: Fix confirmed

I-03. `TopUpDest::etherFiDataProvider` can be stored as immutable

Description:
This can save a considerable amount of gas:
https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/top-up/TopUpDest.sol#L32

Customer's response: Fixed in commit ef608f4

Fix Review: Fix confirmed

I-04. `TopUpDest::topUpUserSafeBatch` `expectedCumulativeTopUps` can be a different length

Description:
https://github.com/etherfi-protocol/cash-v2/blob/8f70cdb90511e6e0b93cdfd1aa040950f175431b/src/top-up/TopUpDest.sol#L167

Customer's response: Fixed in commit ef608f4

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.