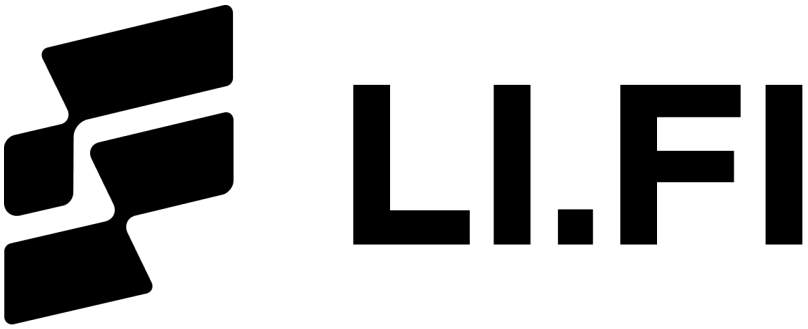

# LI.FI Security Review

LibAsset(v2.0.0), GenericErrors(v1.0.1), HopFacet(v2.0.1)


**Security Researcher**

Sujith Somraaj (somraajsujith@gmail.com)


**Report prepared by:** Sujith Somraaj

May 6, 2025

# Contents

# 1 About Researcher

Sujith Somraaj is a distinguished security researcher and protocol engineer with over eight years of comprehensive experience in the Web3 ecosystem.

In addition to working as a Security researcher at Spearbit, Sujith is also the security researcher and advisor for leading bridge protocol LI.FI and also is a former founding engineer and current CISO at Superform, a yield aggregator with over $170M in TVL.

Sujith has experience working with protocols including Berachain, Optimism, Fantom, Monad, Blast, ZkSync, Decent, Drips, SuperSushi Samurai, DistrictOne, Omni-X, Centrifuge, Superform-V2, Tea.xyz, Paintswap, Bitcorn, Sweep n' Flip, Byzantine Finance, Variational Finance, Satsbridge, Earthfast and Angles

Learn more about Sujith on sujithsomraaj.xyz or on cantina.xyz

# 2 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of that given smart contract(s) or blockchain software. i.e., the evaluation result does not guarantee against a hack (or) the non existence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, I always recommend proceeding with several audits and a public bug bounty program to ensure the security of smart contract(s). Lastly, the security audit is not an investment advice.

This review is done independently by the reviewer and is not entitled to any of the security agencies the researcher worked / may work with.

# 3 Scope

- src/Libraries/LibAsset.sol(v2.0.0)
- src/Facets/HopFacet.sol(v2.0.1)
- src/Errors/GenericErrors.sol(v1.0.1)

# 4 Risk classification

| Severity level | Impact: High | Impact: Medium | Impact: Low |
|---|---|---|---|
| **Likelihood: high** | Critical | High | Medium |
| **Likelihood: medium** | High | Medium | Low |
| **Likelihood: low** | Medium | Low | Low |

## 4.1 Impact

**High** leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.

**Medium** global losses <10% or losses to only a subset of users, but still unacceptable.

**Low** losses will be annoying but bearable — applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

### 4.2 Likelihood

**High** almost certain to happen, easy to perform, or not easy but highly incentivized

**Medium** only conditionally possible or incentivized, but still relatively likely

**Low** requires stars to align, or little-to-no incentive

### 4.3 Action required for severity levels

**Critical** Must fix as soon as possible (if already deployed)

**High** Must fix (before deployment if not already deployed)

**Medium** Should fix

**Low** Could fix

## 5 Executive Summary

Over the course of 1 hours in total, LI.FI engaged with the researcher to audit the contracts described in section 3 of this document ("scope").

In this period of time a total of 7 issues were found. This review focussed only on the changes made from the previous version, not the code on its entirety.

| Project Summary | |
|---|---|
| Project Name | LI.FI |
| Repository | lifinance/contracts |
| Commit | 648d4724eb7b.....575705dacc3 |
| Audit Timeline | April 30, 2025 |
| Methods | Manual Review |
| Documentation | High |
| Test Coverage | Low |

| Issues Found | |
|---|---|
| Critical Risk | 0 |
| High Risk | 0 |
| Medium Risk | 2 |
| Low Risk | 2 |
| Gas Optimizations | 1 |
| Informational | 2 |
| **Total Issues** | **7** |

# 6 Findings

## 6.1 Medium Risk

### 6.1.1 Unchecked `msg.value` for multiple deposits including native tokens

**Context:** LibAsset.sol#L120

**Description:** The `depositAssets()` function in the LibAsset library contains an issue related to how it handles multiple deposits involving native ETH (Ether).

When processing a batch of swaps that require deposits, the function fails to properly validate that the total amount of ETH needed for all swaps matches the actual ETH sent with the transaction.

**Double-counting of ETH:** Since each call to `depositAsset()` with a native asset checks that `msg.value >= amount`, the same ETH can be "counted" multiple times for different swaps.

**Successful Execution with Insufficient Funds:** Suppose there are multiple swaps requiring ETH deposits with a combined value greater than the provided `msg. value`, the function will still execute successfully as long as the largest individual swap has sufficient ETH. This could happen if the inheriting contract has enough ETH to cover the transaction.

**Accounting Errors:** The contract will record deposits as successful even when the actual transferred value is less than the sum of all intended deposits.

**PoC:**

```
    function test_depositValues_Audit() public {
        LibSwap.SwapData[] memory swaps = new LibSwap.SwapData[](2);

        swaps[0] = LibSwap.SwapData(
            address(100),
            address(101),
            address(0),
            address(103),
            1 ether,
            bytes(""),
            true
        );

        swaps[1] = LibSwap.SwapData(
            address(100),
            address(101),
            address(0),
            address(103),
            1 ether,
            bytes(""),
            true
        );

        implementer.depositAssets{value: 1 ether}(swaps);
    }
}
```

**Recommendation:** The proper approach is to validate the total ETH required against `msg.value` before processing individual deposits.

**LI.FI:** Acknowledged.

**Researcher:** Acknowledged. This issue still exists and inheriting contracts shouldn't use it on a loop as mentioned above.

### 6.1.2 Function `transferFromERC20` fails to revert for native asset

**Context:** LibAsset.sol#L95

**Description:** The function `transferFromERC20` is used to transfer ERC20 tokens from the `from` account to the `recipient` account.

This function, instead of reverting for an invalid ERC20 address (like address(0)), succeeds silently without any errors.

**PoC:**

```
function testRevert_approveERC20NativeAudit() public {
        implementer.transferFromERC20(
            address(0),
            makeAddr("Alice"),
            payable(makeAddr("Bob")),
            defaultUSDCAmount
        );
    }
```

```
  [16289] USER_DIAMOND_OWNER::testRevert_approveERC20NativeAudit()
    [0] VM::addr(<pk>) [staticcall]
      ← [Return] Alice: [0xBf0b5A4099F0bf6c8bC4252eBeC548Bae95602Ea]
    [0] VM::label(Alice: [0xBf0b5A4099F0bf6c8bC4252eBeC548Bae95602Ea], "Alice")
      ← [Return]
    [0] VM::addr(<pk>) [staticcall]
      ← [Return] Bob: [0x4dBa461cA9342F4A6Cf942aBd7eacf8AE259108C]
    [0] VM::label(Bob: [0x4dBa461cA9342F4A6Cf942aBd7eacf8AE259108C], "Bob")
      ← [Return]
    [3359] LibAssetImplementer::transferFromERC20(0x0000000000000000000000000000000000000000, Alice:
    ↪ [0xBf0b5A4099F0bf6c8bC4252eBeC548Bae95602Ea], Bob:
    ↪ [0x4dBa461cA9342F4A6Cf942aBd7eacf8AE259108C], 100000000 [1e8])
      [0] 0x0000000000000000000000000000000000000000::transferFrom(Alice:
    ↪ [0xBf0b5A4099F0bf6c8bC4252eBeC548Bae95602Ea], Bob:
    ↪ [0x4dBa461cA9342F4A6Cf942aBd7eacf8AE259108C], 100000000 [1e8])
        ← [Stop]
      ← [Stop]
    ← [Stop]
```

**Recommendation:** Consider adding a defensive check to revert to the native asset to avoid breaking changes from the previous version:

```
function transferFromERC20(
        address assetId,
        address from,
        address recipient,
        uint256 amount
    ) internal {
      if (isNativeAsset(assetId)) {
        revert NullAddrIsNotAnERC20Token();
      }
      ....
}
```

**LI.FI:** Fixed in b1d0a6e248

**Researcher:** Verified fix

## 6.2 Low Risk

### 6.2.1 Lack of native asset validation in `approveERC20` function

**Context:** LibAsset.sol#L158

**Description:** The function `approveERC20()` deviates from the older implementation by reverting on native tokens, instead of returning.

Older implementation (v1.0.2):

```
function maxApproveERC20(
        IERC20 assetId,
        address spender,
        uint256 amount
    ) internal {
    if (isNativeAsset(address(assetId))) {
        return;
    }
    .....
}
```

This discrepancy in behavior between facets/contracts and the older version.

**Recommendation:** Consider maintaining uniform behavior, by adding a return for native asset:

```
function approveERC20(
        IERC20 assetId,
        address spender,
        uint256 requiredAllowance,
        uint256 setAllowanceTo
    ) internal {
+     if (isNativeAsset(address(assetId))) {
+        return;
+    }
    ....
}
```

**LI.FI:** Fixed in 84a413732

**Researcher:** Verified fix

### 6.2.2 EIP-7702 delegation designator spoofing

**Context:** LibAsset.sol#L206

**Description:** The `isContract()` function returns **true** as soon as it sees the `0xef0100` prefix, without verifying that the address has deployed code.

This means:

An address with just the delegation designator prefix but no actual contract code would be identified as a contract

- This could include `address(0)` or other addresses without bytecode
- Any calls to such addresses would be handled as EOAs in reality, creating a mismatch between the function's result and actual behavior.

This could lead to serious issues in systems that rely on this function for distinguishing between contracts and EOAs for access control or other security-critical operations.

**PoC:** The following PoC explains how by delegating `address(0)` as implementation is wrongfully considered as a valid contract by the code:

```solidity
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import  "forge-std/Test.sol";

contract EOAImplementation {
    event Count();

    function counter() external {
        emit Count();
    }
}

contract DestructedContract {
    constructor() {
        selfdestruct(payable(address(this)));
    }
}

contract SetEOATest is Test {
    EOAImplementation public implementation;
    DestructedContract public sdImpl;

    uint256 constant ALICE_PK = 1020321312302131;
    address payable ALICE_ADDRESS = payable(vm.addr(ALICE_PK));

    function setUp() external {
        vm.createSelectFork("https://sepolia.infura.io/v3/8873097743b54814abaf83f9ae0f5520");
        implementation = new EOAImplementation();
        sdImpl = new DestructedContract();
    }

    function test_7702Contract() public {
        console.log(isContract(ALICE_ADDRESS));

        Vm.SignedDelegation memory signedDelegation = vm.signDelegation(address(0), ALICE_PK);
        vm.attachDelegation(signedDelegation);

        console.log(isContract(ALICE_ADDRESS));
        EOAImplementation(ALICE_ADDRESS).counter();
    }

    function isContract(address account) internal view returns (bool) {
        uint256 size;
        bytes32 codehash;
        bytes3 delegationDesignator = 0xef0100;
        bytes memory code = new bytes(3);
        bytes32 emptyHash = keccak256("");

        // use assembly to get codesize and codehash
        assembly {
            size := extcodesize(account)
            extcodecopy(account, add(code, 32), 0, 3)
            codehash := extcodehash(account)
        }

        // Check for delegation designator prefix (0xef0100) >> EIP7702
        bytes3 prefix = bytes3(code);

        console.logBytes3(prefix);
        if (prefix == delegationDesignator) {
```

```
            return true;
        }

        // Traditional check for contract code
        return (size > 0 && codehash != emptyHash);
    }
}
```

**Recommendation:** Valid the first level of delegation to ensure the delegated contract has a size greater than zero.

**LI.FI:** Fixed in dcb3125546 and 5927e648

**Researcher:** Verified fix

## 6.3 Gas Optimization

### 6.3.1 Variables `delegationDesignator` and `emptyHash` could be declared constants

**Context:** LibAsset.sol#L195, LibAsset.sol#L193

**Description:** The variables `delegationDesignator` and `emptyHash` are hardcoded inside the `isContract()` function. These variables could be declared as constants to save gas.

**Recommendation:** Consider declaring `delegationDesignator` and `emptyHash` variables as constants.

**LI.FI:** Fixed in 23b0dd1621

**Researcher:** Verified fix

## 6.4 Informational

### 6.4.1 Increase test coverage

**Context:** LibAsset

**Description:** There is incomplete test coverage of the key contract (LibAsset) under review. Adequate test coverage and regular reporting are essential processes in ensuring the codebase works as intended. Insufficient code coverage may lead to unexpected issues and regressions arising due to changes in the underlying smart contract implementation.

**Recommendation:** Add to test coverage to ensure all execution paths are covered.

**LI.FI:** Fixed in e302ab475ab2

**Researcher:** The fixes only increased code coverage partially. There are still a lot of missing paths that remain untested.

### 6.4.2 Incorrect empty codehash validation

**Context:** LibAsset.sol#L211

**Description:** The `isContract()` function uses `keccak256("")` to determine an "empty hash" for comparison, but this approach is incomplete.

The extcodehash operation can return different values for non-contract addresses:

- bytes32(0) for addresses that have never received any transactions
- keccak256("") for addresses that have received ETH but have no code

The current implementation only checks against keccak256(""), which means that addresses that have never interacted on-chain will be incorrectly identified as contracts when they are not.

**Recommendation:** Check against both possible empty hash values:

```
bytes32 emptyHash = keccak256("");
return (size > 0 && codehash != bytes32(0) && codehash != emptyHash);
```

**LI.FI:** Code has check is removed in dcb3125546

**Researcher:** Verified fix