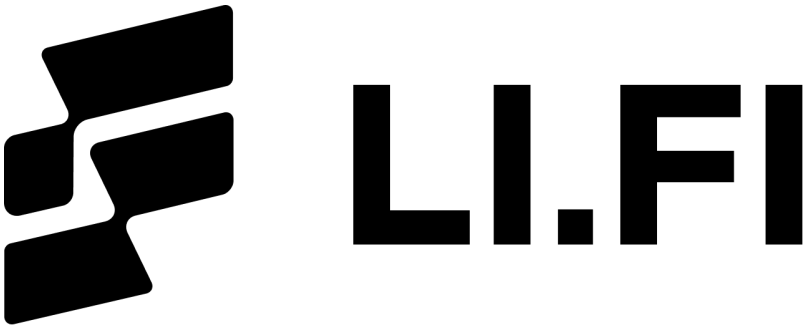

LI.FI Security Review

WhitelistManagerFacet.sol(v1.0.0)

Security Researcher

Sujith Somraaj (somraajsujith@gmail.com)

Report prepared by: Sujith Somraaj

November 4, 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 Lead 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 security advisor at Superform, a yield aggregator with over $170M in TVL.

Sujith has experience working with protocols / funds including Coinbase, Layerzero, Edge Capital, Berachain, Optimism, Ondo, Sonic, 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, Rova, Horizen, 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/Facets/WhitelistManagerFacet.sol(v1.0.0)
- src/Interfaces/IWhitelistManagerFacet.sol(v1.0.0)
- src/Libraries/LibAllowList.sol(v2.0.0)

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 12 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 5 issues were found.

Project Summary	
Project Name	LI.FI
Repository	lifinance/contracts
Commit	e87f69019
Audit Timeline	November 3, 2025 - November 4, 2025
Methods	Manual Review
Documentation	High
Test Coverage	High

Issues Found	
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	0
Gas Optimizations	1
Informational	4
Total Issues	**5**

6 Findings

6.1 Gas Optimization

6.1.1 Deleting storage variable could save more gas

Context: WhitelistManagerFacet.sol#L182, WhitelistManagerFacet.sol#L192

Description: For resetting state during migration, the whitelist values are changed from true -> false. Instead of assigning false, deleting the mapping will result in additional gas savings.

Recommendation: Consider changing the code to following:

```
- als.contractAllowList[als.contracts[i]] = false;
+ delete als.contractAllowList[als.contracts[i]];

- als.selectorAllowList[_selectorsToRemove[i]] = false;
+ delete als.selectorAllowList[_selectorsToRemove[i]]
```

LI.FI: Fixed in 7185e03

Researcher: Verified fix.

6.2 Informational

6.2.1 Unbounded array growth

Context: LibAllowList.sol

Description: Arrays **contracts**, **selectors**, **whitelistedSelectorsByContract** can grow unbounded. While there's efficient removal logic, if many contracts/selectors are added, gas costs for:

1. getAllowedContracts()
2. getAllowedSelectors()
3. getWhitelistedSelectorsForContract()

Could exceed block gas limits, making these functions unusable.

Recommendation: Though warnings about this behavior exists, consider adding pagination if required.

LI.FI: Acknowledged.

Researcher: Acknowledged.

6.2.2 Remove unused events

Context: IWhitelistManagerFacet.sol#L15-L22

Description: The two events `AddressWhitelisted` and `FunctionSelectorWhitelistChanged` remain unused, which could be removed to improve code quality.

LI.FI: Fixed in 6b11b41

Researcher: Verified fix.

6.2.3 Functions `migrate()` and `isMigrated()` violate conventions

Context: WhitelistManagerFacet.sol#L158, WhitelistManagerFacet.sol#L250

Description: The `migrate()` and `isMigrated()` functions in `WhitelistManagerFacet.sol` are placed in the "Internal Logic" section despite being external/public functions. This violates the established code organization pattern, making the code harder to maintain and review.

Recommendation: Consider moving the function up and group it with other "external functions"

LI.FI: Fixed in 7c0c3fa and 1c630c5

Researcher: Verified fix.

6.2.4 Insufficient migration testing from v1 to v2 whitelist system

Context: Global

Description: The migration from the legacy whitelist system (V1) to the new granular whitelist system (V2) lacks comprehensive testing, resulting in critical bugs that corrupt contract state and cause data loss. The test suite does not adequately verify that the migration function properly clears all legacy state mappings before rebuilding the new state structure.

Recommendation: Consider adding to the tests and covering all possible cases of migration.

1. Migration Completeness Tests: - Verify all V1 state is properly cleared (mappings and arrays) - Confirm migrated data is immediately queryable via getter functions - Validate reference counts and indices are correctly initialized
2. State Consistency Tests: - After every state-changing operation, verify internal mappings match arrays - Ensure contractToIndex[contract] points to correct position in contracts

LI.FI: Fixed in bb32542

Researcher: Verified fix.