LI.FI Security Review

WhitelistManagerFacet(v1.0.0)

Security Researcher

Sujith Somraaj (somraajsujith@gmail.com)

Report prepared by: Sujith Somraaj

August 18, 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 / funds including 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 7 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 2 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	5665c5f9
Audit Timeline	August 10, 2025
Methods	Manual Review
Documentation	Medium-High
Test Coverage	Medium

Issues Found	
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	0
Gas Optimizations	0
Informational	2
Total Issues	**2**

6 Findings

6.1 Informational

6.1.1 Functions `getWhitelistedFunctionSelectors()` and `getWhitelistedAddresses()` might go out of block gas limit

Context: WhitelistManagerFacet.sol#L91, WhitelistManagerFacet.sol#L107

Description: Although not an immediate concern, if the whitelisted address/selector list expands over time, the function might exceed the block gas limit.

This might do any potential integrators. But for this to happen, the list has to be excessively huge.

Recommendation: No code changes are expected from the project team. However, adding warning documentation about the getter functions will be helpful for integrators.

LI.FI: Acknowledged.

Researcher: Acknowledged.

6.1.2 Selectors not removed during migration can never be removed

Context: WhitelistManagerFacet.sol#L146

Description: The `migrate()` function in **WhitelistManagerFacet** is designed to clear all current whitelisted addresses and selectors before adding a new set of selectors and addresses.

However, if a selector is not removed during migration, it will persist in the contract indefinitely, and attempting to remove it using `setFunctionWhitelistBySelector` will have no effect.

Recommendation: This issue can be avoided by ensuring the migration script has all the existing selectors added to the script, in path `/config/whitelistManager.json`.

Else, gate the read path for selectors after migration: require an index hit as well as the boolean. This makes any legacy-true-but-unindexed selector read as not allowed after migration.

```
function selectorIsAllowed(bytes4 sel) internal view returns (bool) {
    AllowListStorage storage als = _getStorage();
    if (!als.migrated) {
        // legacy behavior
        return als.selectorAllowList[sel];
    }
    // post-migration: require both the flag AND a nonzero index
    return als.selectorAllowList[sel] && als.selectorToIndex[sel] != 0;
}
```

LI.FI: Acknowledged.

Researcher: Acknowledged.