LI.FI Security Review

WhitelistManagerFacet.sol(v1.0.0), IWhitelistManagerFacet.sol(v1.0.0), LibAllowList.sol(v1.0.1)

Security Researcher

Sujith Somraaj (somraajsujith@gmail.com)

Report prepared by: Sujith Somraaj

June 2, 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/Facets/WhitelistManagerFacet.sol(v1.0.0)
- src/Interfaces/IWhitelistManagerFacet.sol(v1.0.0)
- src/Libraries/LibAllowList.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 4 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 6 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	de2592e930.....d651015570d
Audit Timeline	May 29, 2025
Methods	Manual Review
Documentation	High
Test Coverage	Medium-High

Issues Found	
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	0
Gas Optimizations	0
Informational	6
Total Issues	**6**

6 Findings

6.1 Informational

6.1.1 Typos

Context: LibAllowList.sol#L8, IWhitelistManagerFacet.sol#L22

Description: Multiple typos were found across multiple files under the scope of this audit:

1. LibAllowList.sol

```
- /// @notice Library for managing and accessing the conract address allow list
+ /// @notice Library for managing and accessing the contract address allow list
```

2. IWhitelistManagerFacet.sol

```
- /// @param _contractAddress The contractaddress to be whitelisted.
+ /// @param _contractAddress The contract address to be whitelisted.
```

Recommendation: Consider fixing the typos mentioned above.

LI.FI: Fixed in 70bd9fdb820ac4dcb044691a15fa09a5731df509

Researcher: Verified fix.

6.1.2 Implement sanity checks in `_setFunctionApproval` function

Context: WhitelistManagerFacet.sol#L143

Description: The `_setFunctionApproval` function is used to add/remove function selectors from the whitelist manager facet. However, this function does not perform any sanity checks on the input and processes it optimistically, leading to confusion and the emission of incorrect events.

For example, trying to remove a non-existent function selector succeeds with an invalid event.

Recommendation: Consider implementing existence checks before emitting events in `_setFunctionApproval` function:

```
function _setFunctionApproval(bytes4 _selector, bool _approval) internal {
    if (_approval) {
+       if(LibAllowList.selectorIsAllowed(_selector)) return;
        LibAllowList.addAllowedSelector(_selector);
     } else {
+       if(!LibAllowList.selectorIsAllowed(_selector)) return;
        LibAllowList.removeAllowedSelector(_selector);
     }
     emit FunctionSelectorApprovalChanged(_selector, _approval);
    }
```

LI.FI: Fixed in 7e134a8f5bf30d63e3e6ee2ea96c17317c29dd7c

Researcher: Verified fix.

6.1.3 Inaccurate contract documentation in interface `IWhitelistManagerFacet`

Context: IWhitelistManagerFacet.sol#L6

Description: The contract documentation states that the WhitelistManagerFacet is for managing approved contracts and addresses. However, both contracts and addresses mean the same thing, it should be changed to following:

```
- /// @notice Interface for WhitelistManagerFacet facet for managing approved contracts and addresses.
+/// @notice Interface for WhitelistManagerFacet facet for managing approved contracts and function
↪  selectors.
```

Recommendation: Consider updating the interface notice to reflect exact contract behavior.

LI.FI: Fixed in a61930459e8472884fd6c20ce29d6a95bfd5bed7

Researcher: Verified fix.

6.1.4 Remove unchecked loop increments

Context: WhitelistManagerFacet.sol#L34-L36, WhitelistManagerFacet.sol#L55-L57, WhitelistManagerFacet.sol#L86-L88

Description: The contract implements unchecked `{ ++i; }` increments in for-loops under the assumption that this provides gas savings by bypassing overflow checks. However, empirical testing demonstrates that this optimization provides no measurable reduction in gas, making the added code complexity unjustified.

Recommendation: Replace all instances of the complex loop pattern with standard for-loop syntax:

```
// Current (unnecessary complexity)
for (uint256 i = 0; i < length; ) {
    // loop body
    unchecked {
        ++i;
    }
}

// Recommended (clear and equivalent)
for (uint256 i; i < length; ++i) {
    // loop body
}
```

LI.FI: Fixed in 357f2149abcaf5a3206eb7ce599b858a39c54c7d

Researcher: Verified fix.

6.1.5 Exponential gas cost increase for `getWhitelistedAddresses` and `getApprovedFunctionSelectors` could lead to permanent DoS

Context: WhitelistManagerFacet.sol#L100, WhitelistManagerFacet.sol#L116

Description: The `getWhitelistedAddresses()` and `getApprovedFunctionSelectors()` functions become unusable due to excessive gas consumption when their respective arrays contain large numbers of entries.

With approximately 45,000+ addresses or function selectors, these functions exceed practical gas limits, effectively creating a denial-of-service condition that prevents users from retrieving complete data.

Recommendation: Consider adding a reasonable limit and preventing adding addresses (or) selectors beyond that limit. Since the 45,000 figure is well within the operable limits of the protocol, add a warning to the documentation to integrators about the limitations of the function.

LI.FI: Fixed in b1d4f0af95b63de0c874f85b874afe38b23ab590

Researcher: Verified fix.

6.1.6 Misleading event emission while removing an non-existent facet

Context: WhitelistManagerFacet.sol#L137

Description: The `WhitelistManagerFacet.sol` contract contains a logic inconsistency where the AddressRemoved event is emitted even when attempting to remove an address that doesn't exist in the whitelist. This creates misleading event logs that could confuse off-chain monitoring systems and users.

```
function _removeFromWhitelist(address _address) internal {
    LibAllowList.removeAllowedContract(_address);
    emit AddressRemoved(_address);  // ← Always emitted regardless of actual removal
}
```

Recommendation: Consider adding a check in `_removeFromWhitelist` to ensure the facet is removed before emitting the `AddressRemoved` event.

```
function _removeFromWhitelist(address _address) internal {
+       if(!LibAllowList.contractIsAllowed(_address)) return;

        LibAllowList.removeAllowedContract(_address);
        emit AddressRemoved(_address);
    }
```

LI.FI: Fixed in 7505729b69ba30aa3b8a429c7d4e12a39c13dfdc

Researcher: Verified fix.