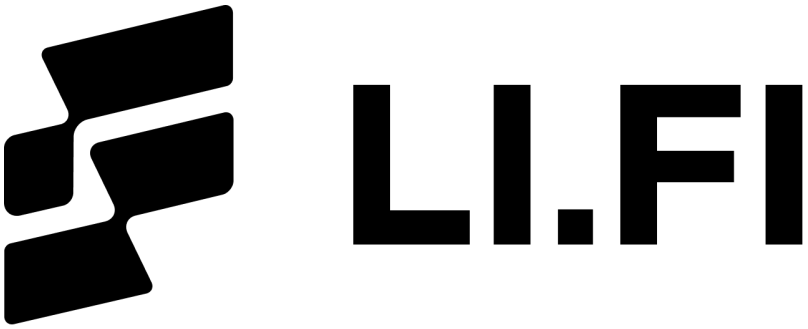

LI.FI Security Review

MayanFacet(v1.2.2)

Security Researcher

Sujith Somraaj (somraajsujith@gmail.com)

Report prepared by: Sujith Somraaj

August 25, 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/MayanFacet.sol(v1.2.2)

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 3 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 1 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	b7e4b66
Audit Timeline	August 25, 2025
Methods	Manual Review
Documentation	High
Test Coverage	Medium-High

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

6 Findings

6.1 Medium

6.1.1 Incorrect decoding of receiver address

Context: MayanFacet.sol#L259-L266

Description: The `MayanFacet.sol` added support for Hypercore routes, which introduced receiver address decoding for two new selectors deposit(0xe27dce37) and fastDeposit(0x4d1ed73b). However these two selector decodes the `trader` address as the final receiver address, which is incorrect.

The deposit function from `HCDepositInitiator.sol`:

```
function deposit(
    address tokenIn,
    uint256 amountIn,
    address trader,
    uint64 gasDrop,
    uint256 bridgeAmount,
    DepositPayload calldata depositPayload
    ) external nonReentrant {
      ....
      IMayanCircle(mayanCircle).bridgeWithFee(
            tokenIn,
            bridgeAmount,
            0, // redeemFee
            gasDrop,
            hcProcessor,
            hcDomain,
            2, // payloadType
            encodeDepositPayload(depositPayload)
        );
    }
```

where it is clear that the receiver is not the trader address, but is encoded in the depositPayload parameter. This inconsistency will lead to permanent DoS if the receiver address is different from the sender address. Similar issue exists in the fastDeposit function too.

Recommendation: Consider fixing the decoding by fetching receiver address from the depositPayload parameter.

LI.FI: Fixed in 2e9eeda, 74a57f9 and b1f7ea4

Researcher: Verified fix