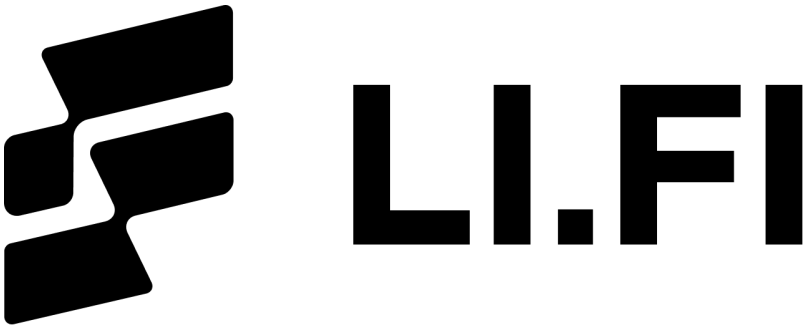

LI.FI Security Review

Permit2Proxy.sol(v1.0.3)

Security Researcher

Sujith Somraaj (somraajsujith@gmail.com)

Report prepared by: Sujith Somraaj

April 15, 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 bridge protocols, including LI.FI & Garden.Finance (over $16B in combined volume) 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, 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 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/Periphery/Permit2Proxy.sol(v1.0.3)

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 1 issues were found. This review focussed only on the patch of an issue in the callDiamondWithEIP2612Signature function of the Permit2Proxy, not the contract on its entirety.

Project Summary	
Project Name	LI.FI
Repository	lifinance/contracts
Commit	36803ba526.....f0b7b088b0
Audit Timeline	April 13, 2025
Methods	Manual Review

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

6 Findings

6.1 Informational

6.1.1 Add more test cases to ensure all errors are handled properly in the `try/catch` of `callDiamondWithEIP2612Signature()` function

Context: Permit2Proxy.sol#L77

Description: The test cases for the `callDiamondWithEIP2612Signature()` function in the `Permit2Proxy.sol` periphery contract do not include multiple scenario where the permit call to `tokenAddress()` could revert, including:

- a panic (assertion errors (or) division by zero).
- an empty revert()
- a revert with string message
- a revert inside require function

Though the function works as expected in these case, adding the explicit test case will help reduce errors in production and boost function coverage.

Recommendation: Consider adding the test cases for individual case as shown below:

```
contract MockPermitTokenPanic is ERC20, ERC20Permit {
    constructor(
        string memory name,
        string memory symbol
    ) ERC20(name, symbol) ERC20Permit(name) {}

    function mint(address to, uint256 amount) external {
        _mint(to, amount);
    }

    // intentionally revert with custom error (not catchable by `Error(string)` only catchable by
    ↪  Error(bytes))
    function permit(
        address,
        address,
        uint256 amount,
        uint256,
        uint8,
        bytes32,
        bytes32
    ) public pure override {
        assert(amount == 0);
    }

    error CustomPermitError();
}

function testRevert_FailsOnPanicDuringAllowance() public {
        MockPermitTokenPanic token = new MockPermitTokenPanic("Mock", "MCK");

        address tokenAddress = address(token);

        vm.startPrank(permit2User);

        bytes memory callData = _getCalldataForBridging();

        // we don't care about the signature since permit will revert anyway
        permit2Proxy.callDiamondWithEIP2612Signature(
            tokenAddress,
```

```
            defaultUSDCAmount,
            block.timestamp + 1000,
            27, // dummy v
            bytes32(0), // dummy r
            bytes32(0), // dummy s
            callData
        );

        vm.stopPrank();
}
```

LI.FI: Fixed in 5b212bd302a7ec0c027c51753d6d5a9937f6769e

Researcher: Verified fix