LI.FI Security Review

Permit2Proxy

Independent Review By:

Sujith Somraaj (somraajsujith@gmail.com)

October 3, 2024

Contents

1 About Researcher

Sujith Somraaj is a distinguished security researcher and protocol engineer with over seven years of comprehensive experience in the Web3 ecosystem.

In addition to working as an external auditor/security researcher with LI.FI, Sujith is a protocol engineer and security researcher at Superform and Spearbit.

Learn more about Sujith on sujithsomraaj.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

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 days 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 Hashes	0e3debb78a.....1e16b039a0 26f54d4411.....825757fc18
Type of Project	
Audit Timeline	September 16, 2024
Methods	Manual Review

Issues Found	
Critical Risk	0
High Risk	1
Medium Risk	0
Low Risk	3
Gas Optimizations	0
Informational	1
Total Issues	**5**

6 Findings

6.1 High Risk

6.1.1 Frontrunning callDiamondWithEIP2612Signature leads to permanent loss of user funds

Context: Permit2Proxy.sol#L64

Description: The `callDiamondWithEIP2612Signature` function is susceptible to a frontrunning attack. Frontrunning occurs when an attacker observes a pending transaction in the transaction pool and attempts to execute their transaction with a higher gas price, effectively "frontrunning" the original transaction.

Here's the relevant code snippet:

```
function callDiamondWithEIP2612Signature(
    address tokenAddress,
    address owner,
    uint256 amount,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s,
    address diamondAddress,
    bytes calldata diamondCalldata
) public payable {
    // ...
}
```

The `callDiamondWithEIP2612Signature` function allows users to bridge tokens through the LI.FI diamond contract using an EIP2612 gasless permit. However, the function parameters `diamondAddress` and `diamondCalldata` are not signed by the user and can be modified by an attacker.

An attacker can observe a pending transaction that calls `callDiamondWithEIP2612Signature` and extract the `tokenAddress`, `owner`, `amount`, `deadline`, `v`, `r`, and `s` parameters. The attacker can then create their transaction with the same parameters but modify the `diamondAddress` and `diamondCalldata` to execute a different action or transfer the tokens to a different address.

By submitting the attacker's transaction with a higher gas price, the attacker can potentially frontrun the original transaction and execute their malicious action before it is processed.

Recommendation: Consider accepting a signature from the `owner` approving the `diamondAddress` / `diamondCallData`.

LI.FI: Fixed in 0e3debb78abcdf9a9f934115338b611e16b039a0

Researcher: Verified.

6.2 Low Risk

6.2.1 Witness typehash contains non-existent fields

Context: Permit2Proxy.sol#L25

Description: The witness type hash is used by the Permit2Proxy contract to process witness-based approvals. The witness type hash is generated by keccak256 hashing of `LiFiCall(address tokenReceiver,address diamondAddress,bytes32 diamondCalldataHash)`; however, the `tokenReceiver` is not required in the newer implementation as it is unused.

Recommendation: Consider removing the `tokenReceiver` parameter from the witness typehash.

LI.FI: Fixed in 6ab55d42168e4d58e2b1ffd24d60d7434a7a9ca6

Researcher: Verified.

6.2.2 `Permit2Proxy` is in-compatible with multiple facets that refunds native tokens to the `msg.sender`

Context: Permit2Proxy.sol#L17

Description: The `Permit2Proxy` contract does not have a `receive()` function to accept native refunds from the Diamond contract. Therefore, the contract might be incompatible with some facets that refund excess native.

Facets that refund native tokens to `msg.sender` include, `DeBridgeDlnFacet`, `DeBridgeFacet`, `MakerTeleportFacet`, `MultichainFacet`, `NXTPFacet`, `SynapseBridgeFacet`, `WormholeFacet`, `AcrossFacet`, `AllBridgeFacet`, `ArbitrumBridgeFacet`, `CBridgeFacet`, `CelerCircleBridgeFacet`, `CircleBridgeFacet`, `GenericSwapFacet`, `GnosisBridgeFacet`, `GnosisBridgeL2Facet`, `HopFacet`, `HyphenFacet`, `LIFuelFacet`, `MayanFacet`, `OmniBridgeFacet`, `PolygonBridgeFacet`, `SquidFacet`, `StargateFacet`, `StargateFacetV2`, `SymbiosisFacet`, `ThorSwapFacet` and `SwapperV2`.

Hence, the user's native token estimates should be 100% accurate for using the `Permit2Proxy` contract rather than the built-in refunding mechanism of the `Diamond` contract.

Recommendation: Consider adding a `receive()` function and forward the refunded native tokens to the user.

LI.FI: Fixed in 976966de7ba14d1782904ebe7bad1b3fd2e79281

Researcher: Verified. The refunding mechanism added can help retrieve refunded tokens & also accept refunds from the diamond contract.

6.2.3 Refund unused tokens in `callDiamondWithPermit2Witness`, `callDiamondWithPermit2` and `callDiamondWithEIP2612Signature` functions

Context: Permit2Proxy.sol#L72, Permit2Proxy.sol#L115, Permit2Proxy.sol#L147

Description: The `Permit2Proxy` contract transfers assets from the users to themselves before forwarding arbitrary call data to the LI.FI diamond. However, the contract doesn't account for the dust left in it, which a new user could steal.

Recommendation: Consider refunding any dust after the execution of `calldata` to all the functions mentioned above.

```
function callDiamondWithEIP2612Signature(
    address tokenAddress,
    uint256 amount,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s,
    bytes calldata diamondCalldata
) public payable returns (bytes memory) {
+ uint256 balanceBefore = IERC20(tokenAddress).balanceOf(address(this));
    .....
+  uint256 balanceAfter = IERC20(tokenAddress).balanceOf(address(this));
+  IERC20(tokenAddress).transfer(msg.sender, balanceAfter - balanceBefore);
}
```

LI.FI: Acknowledged.

6.3 Informational

6.3.1 Remove unused imports in `Permit2Proxy`

Context: Permit2Proxy.sol#L5, Periphery/Permit2Proxy.sol#L8

Description: The `Permit2Proxy` contract imports two contracts, `TransferrableOwnership` and`SafeERC20`, but is not used anywhere in the code.

Recommendation: Consider removing the unused import.

LI.FI: Fixed in 3eaf8ba32972a9bfd78ab0c16d6998bfa97a46c0

Researcher: Verified.