LI.FI Security Review

TokenWrapper.sol(v1.2.0)

Security Researcher

Sujith Somraaj (somraajsujith@gmail.com)

Report prepared by: Sujith Somraaj

December 26, 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 Lead 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 security advisor at Superform, a yield aggregator with over $170M in TVL.

Sujith has experience working with protocols / funds including Coinbase, Uniswap, 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/Periphery/TokenWrapper.sol(1.2.0)

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 3 hours in total, LI.FI engaged with the researcher to audit the contracts described in section 3 of this document ("scope"). This is a differential review focussed on the changes from previous version.

In this period of time a total of 4 issues were found.

Project Summary	
Project Name	LI.FI
Repository	lifinance/contracts
Commit	6c46300
Audit Timeline	December 25, 2025
Methods	Manual Review
Documentation	High
Test Coverage	High

Issues Found	
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	0
Gas Optimizations	2
Informational	2
Total Issues	**4**

6 Findings

6.1 Gas Optimization

6.1.1 Function `withdraw()` could be optimized

Context: TokenWrapper.sol#L86-L107

Description: The `withdraw()` function in the TokenWrapper contract contains the same gas inefficiency pattern as the `deposit()` function, with an unnecessary variable declaration and redundant else branch.

Recommendation: Simplify the function by reusing the amount variable and removing the else branch:

```
function withdraw() external {
      // While in a general purpose contract it would make sense
      // to have `amount` equal to the minimum between the balance and the
      // given allowance, in our specific usecase allowance is always
      // nearly MAX_UINT256. Using the balance only is a gas optimisation.
      uint256 amount = IERC20(WRAPPED_TOKEN).balanceOf(msg.sender);
      SafeTransferLib.safeTransferFrom(
          WRAPPED_TOKEN,
          msg.sender,
          address(this),
          amount
      );

      WETH(payable(CONVERTER)).withdraw(amount);

      if (USE_CONVERTER) {
          amount = (amount * BASE_DENOMINATOR) / SWAP_RATIO_MULTIPLIER;
      }

      SafeTransferLib.safeTransferETH(msg.sender, amount);
  }
```

LI.FI: Fixed in 8d177d02

Researcher: Verified fix.

6.1.2 Function `deposit()` could be optimized

Context: TokenWrapper.sol#L69-L81

Description: The `deposit()` function in the TokenWrapper contract contains unnecessary gas overhead from a redundant variable declaration and an else branch that can be eliminated.

Recommendation: Simplify the function by reusing the amount variable and removing the else branch:

```
function deposit() external payable {
      uint256 amount = msg.value;
      WETH(payable(CONVERTER)).deposit{ value: amount }();

      if (USE_CONVERTER) {
          amount =
              (amount * SWAP_RATIO_MULTIPLIER) /
              BASE_DENOMINATOR;
      }

      SafeTransferLib.safeTransfer(WRAPPED_TOKEN, msg.sender, amount);
  }
```

LI.FI: Fixed in 8d177d02

Researcher: Verified fix.

6.2 Informational

6.2.1 Unused error code `WithdrawFailure`

Context: TokenWrapper.sol#L26

Description: The contract defines an error code called `WithdrawFailure`, but it is not referenced or utilized anywhere.

Recommendation: Consider removing the unused error code to improve code quality.

LI.FI: Fixed in 8d177d02

Researcher: Verified fix.

6.2.2 Add extensive documentation about fee assumptions and native token decimal requirements

Context: TokenWrapper.sol

Description: The **TokenWrapper** contract lacks documenting two important assumptions that are essential for correct operation:

1. The constant BASE_DENOMINATOR = 1 ether (line 22) implicitly assumes 18 decimals for native tokens.
2. The converter contract (if used) MUST NOT charge any fees and should only perform decimal conversion, without rounding losses. Stable GasUSDT0Converter handles this properly.

Recommendation: Add documentation to clarify these two assumptions:

```
/// @notice IMPORTANT: This contract assumes the native token has 18 decimals (standard for all EVM
↪ chains)
/// @notice IMPORTANT: The converter contract (if used) MUST NOT charge any fees and should only perform
↪ decimal conversion
```

LI.FI: Fixed in 8d177d02

Researcher: Verified fix.