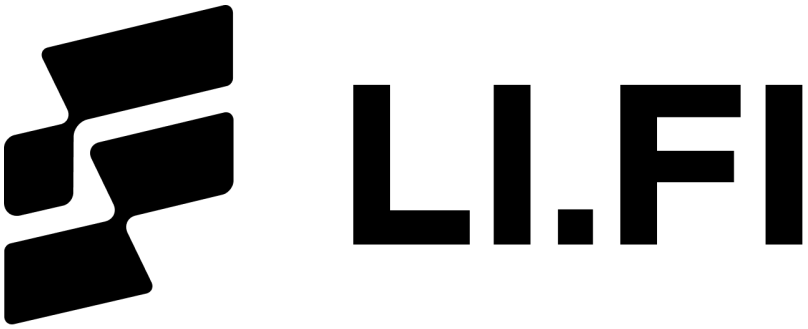

LI.FI Security Review

LiFiDEXAggregator.sol(v1.11.0)

Security Researcher

Sujith Somraaj (somraajsujith@gmail.com)

Report prepared by: Sujith Somraaj

June 30, 2025

Contents

1 About Researcher **2**

2 Disclaimer **2**

3 Scope **2**

4 Risk classification **2**
4.1 Impact . . . 2
4.2 Likelihood . . . 3
4.3 Action required for severity levels . . . 3

5 Executive Summary **3**

6 Findings **4**
6.1 Medium Risk . . . 4
6.1.1 Unsafe amount parameter casting can lead to locking of user funds . . . 4
6.2 Low Risk . . . 5
6.2.1 Incorrect price point boundaries used in `swapIzumiV3()` . . . 5
6.3 Gas Optimization . . . 5
6.3.1 Decode tokenIn only if the amount to transfer is valid in `_handleIzumiV3SwapCallback()` function . . . 5
6.4 Informational . . . 5
6.4.1 Sanity check withdrawMode in `swapSyncSwap()` . . . 5
6.4.2 Inconsistent `INTERNAL_INPUT_SOURCE` handling in `swapSyncSwap()` function . . . 6
6.4.3 Inaccurate inline documentation for izumi swap handlers . . . 6
6.4.4 Validate decoded stream parameters in `swapIzumiV3()` function . . . 6

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 Edge Capital, Berachain, Optimism, 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, 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/LiFiDEXAggregator.sol(v1.11.0)
- src/Interfaces/IiZiSwapPool.sol(v1.0.0)
- src/Interfaces/ISyncSwapVault.sol(v1.0.0)
- src/Interfaces/ISyncSwapPool.sol(v1.0.0)

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 6 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 7 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	6561ef4cc
Audit Timeline	June 20 2025 - June 30, 2025
Methods	Manual Review
Documentation	High
Test Coverage	Medium-High

Issues Found	
Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	1
Gas Optimizations	1
Informational	4
Total Issues	**7**

6 Findings

6.1 Medium Risk

6.1.1 Unsafe amount parameter casting can lead to locking of user funds

Context: LiFiDEXAggregator.sol#L780, LiFiDEXAggregator.sol#L787

Description: The parameter `amount` is passed in as **uint256** to the `swapIzumiV3()` function. Later, this value is cast to **uint128** before passing it to the DEX.

The primary issue here is that the aggregator contract transfers a **uint256** value but only swaps a **uint128** value without refunding any excess, resulting in a permanent lock of funds.

PoC: The following PoC file can be pasted into the `test/solidity/Periphery/LiFiDEXAggregator.t.sol` file to reproduce this issue locally.

So, the user attempted to swap a value exceeding the maximum value of uint128, and the function executed successfully, locking the excess funds within the contract itself.

```
    function test_izumi() public {
        deal(address(WETH), USER_SENDER, type(uint256).max);

        vm.startPrank(USER_SENDER);
        IERC20(WETH).approve(address(liFiDEXAggregator), type(uint256).max);

        // fix the swap data encoding
        bytes memory swapData = _buildIzumiV3Route(
            CommandType.ProcessUserERC20,
            WETH,
            uint8(SwapDirection.Token0ToToken1),
            IZUMI_WETH_USDC_POOL,
            USER_RECEIVER
        );

        liFiDEXAggregator.processRoute(
            WETH,
            type(uint216).max,
            USDC,
            0,
            USER_RECEIVER,
            swapData
        );

        assert(IERC20(WETH).balanceOf(address(liFiDEXAggregator)) > 0);

        vm.stopPrank();
    }
```

Recommendation: Consider validating the amount parameter as follows:

```
function swapIzumiV3(
   uint256 stream,
   address from,
   address tokenIn,
   uint256 amountIn
) private {
    ....
+   if(amountIn > type(uint128).max) revert InvalidCallData();
    ....
}
```

LI.FI: Fixed in d8935ac

Researcher: Verified fix

6.2 Low Risk

6.2.1 Incorrect price point boundaries used in `swapIzumiV3()`

Context: LiFiDEXAggregator.sol#L781, LiFiDEXAggregator.sol#L788

Description: The `swapIzumiV3()` function, utilized by LiFiDEXAggregator, handles token swaps on the Izumi protocol. To perform swaps, the `swapX2Y()` and `swapY2X()` functions require price boundary parameters. These price points can be set within a range of -79999 to 79999, but the protocol often uses -80000 and 80000, which can cause unpredictable results.

Recommendation: Consider fixing this issue by passing the right price boundary values to the respective izumi swap functions.

LI.FI: Fixed in 36c3bbc

Researcher: Verified fix

6.3 Gas Optimization

6.3.1 Decode tokenIn only if the amount to transfer is valid in `_handleIzumiV3SwapCallback()` function

Context: LiFiDEXAggregator.sol#L870

Description: The `_handleIzumiV3SwapCallback()` function is used by the dex aggregator contract to handle callbacks from Izumi dex.

This function can be further optimized to decode `tokenIn` only when there is a valid transfer amount. This change can reduce gas costs in certain scenarios and align it with other callback handlers.

Recommendation: Consider fixing the function as follows:

```
    function _handleIzumiV3SwapCallback(
        uint256 amountToPay,
        bytes calldata data
    ) private {
        if (msg.sender != lastCalledPool) {
            revert IzumiV3SwapCallbackUnknownSource();
        }

-       address tokenIn = abi.decode(data, (address));

        if (amountToPay == 0) {
            revert IzumiV3SwapCallbackNotPositiveAmount();
        }

        lastCalledPool = IMPOSSIBLE_POOL_ADDRESS;

+       address tokenIn = abi.decode(data, (address));
        IERC20(tokenIn).safeTransfer(msg.sender, amountToPay);
    }
```

LI.FI: Fixed in eb53e17

Researcher: Verified fix

6.4 Informational

6.4.1 Sanity check withdrawMode in `swapSyncSwap()`

Context: LiFiDEXAggregator.sol#L822

Description: The **withdrawMode** values should be bounded between 0 and 2. However, due to a lack of sanity checks, any value could be passed into the function, leading to unexpected behavior.

Recommendation: Consider validating the **withdrawMode** value as follows:

```
uint8 withdrawMode = stream.readUint8();
if(withdrawMode > 2) revert InvalidCallData();
```

LI.FI: Fixed in ec277e6

Researcher: Verified fix

6.4.2 Inconsistent `INTERNAL_INPUT_SOURCE` handling in `swapSyncSwap()` function

Context: LiFiDEXAggregator.sol#L849

Description: For V1 pools, `INTERNAL_INPUT_SOURCE` is explicitly handled and the function reverts, but for V2 pools, it's only handled in a comment with no explicit validation.

Recommendation: Consider adding explicit validations for v2 pools as well:

```
} else if (from == INTERNAL_INPUT_SOURCE) {
    // Tokens already in pool, no transfer needed
} else {
    revert InvalidCallData();
}
```

LI.FI: Fixed in ad4b814

Researcher: Verified fix

6.4.3 Inaccurate inline documentation for izumi swap handlers

Context: LiFiDEXAggregator.sol#L883, LiFiDEXAggregator.sol#L897

Description: The izumi swap handler functions `swapX2YCallback()` and `swapY2XCallback()` only validates if the **msg.sender** equal the **lastCalledPool** state variable stored during the swap initialization.

However, the inline documentation of these functions suggests that the handlers should verify if the caller is an iZiSwap pool deployed by the canonical iZiSwap factory, which is not the case here.

Recommendation: Consider updating the correct inline documentation.

LI.FI: Fixed in bed2e7a

Researcher: Verified fix

6.4.4 Validate decoded stream parameters in `swapIzumiV3()` function

Context: LiFiDEXAggregator.sol#L766

Description: The `swapIzumiV3()` function is used to swap tokens using the Izumi pools. This function decodes the pool, direction, and recipient from the stream. But does not validate those parameters, exhibiting inconsistency.

Also, the parameter `to` can be renamed to `recipient` for consistency with other DEX interactions.

Recommendation: Consider validating the stream parameters as follows:

```
function swapIzumiV3(
    uint256 stream,
    address from,
    address tokenIn,
    uint256 amountIn
) private {
  address pool = stream.readAddress();
  uint8 direction = stream.readUint8(); // 0 = Y2X, 1 = X2Y
-  address to = stream.readAddress();
+ address recipient = stream.readAddress();
+ if(
+     pool == address(0) ||
+     pool == IMPOSSIBLE_POOL_ADDRESS ||
+     recipient == address(0)
+  ) revert InvalidCallData();
}
```

LI.FI: Fixed in 36c3bbc

Researcher: Verified fix