LI.FI Security Review

LiFiDexAggregator(v1.7.0), IVelodromeV2Pool(v1.0.0), IVelodromeV2PoolCallee(v1.0.0), IVelodromeV2Router(v1.0.0)

Security Researcher

Sujith Somraaj (somraajsujith@gmail.com)

Report prepared by: Sujith Somraaj

April 11, 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/LiFiDEXAggregator.sol(v1.7.0)
- src/Interfaces/IVelodromeV2Pool.sol(v1.0.0)
- src/Interfaces/IVelodromeV2PoolCallee.sol(v1.0.0)
- src/Interfaces/IVelodromeV2Router.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 2 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 8 issues were found. This review focussed only on the Velodrome v2 swap integration to the LiFiDexAggregator and other small changes made in the contract under scope in the audit commit hash and the rest of the code is assumed to be safe from previous audits.

Project Summary	
Project Name	LI.FI
Repository	lifinance/contracts
Commit	f49cfdf2bfc.....dac975040
Audit Timeline	April 9, 2025 - April 10, 2025
Methods	Manual Review

Issues Found	
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	2
Gas Optimizations	1
Informational	5
Total Issues	**8**

6 Findings

6.1 Low Risk

6.1.1 MEV vulnerability in `swapVelodromeV2()` if callback alters token balances

Context: LiFiDEXAggregator.sol#L806

Description: The contract's balance verification mechanism is vulnerable to manipulation when using Velodrome V2 pools with **callbacks** enabled. The `processRouteInternal()` function verifies the recipient's final balance after all operations have been completed but does not account for potential balance manipulations that could occur during callbacks from the Velodrome V2 pool.

When a swap is executed with callbacks enabled (callback == true), the IVelodromeV2Pool(pool).swap() function can trigger a callback to the recipient contract. The recipient is unaware of the **minAmount** specified in the LDA (LiFiDexAggregator) contract. Hence, if the recipient is designed in such a way that it does an operation that increases by the token out balance, then the entire swap through LDA is vulnerable to MEV.

The balance verification check passes despite the swap providing less than **amountOutMin** tokens.

Proof Of Concept:

1. A victim actor creates a contract that implements the VelodromeV2 callback interface and does some operations that lead to increased output token balance post-swap.
2. When using this contract as the recipient, the slippage checks are redundant as the recipient accepts a value less than their expected **amountOutMin**, violating the slippage protection.

Recommendation: Implement balance checks based on the pool balance difference (or) add more caution against doing any operations that might lead to increased balance in the callback.

LI.FI: Acknowledged This is out of our scope. But we added comments to warn developers and integrators in a507f1e795f9e50b9d02655c3c06f126975f73b6

Researcher: The documentation added looks good.

6.1.2 Inaccurate callback flag validation

Context: LiFiDEXAggregator.sol#L774

Description: The `swapVelodromeV2()` function incorrectly validates the callback flag from the input stream. The function currently uses `stream.readUint8() > 0` to determine whether a callback should be executed. This implementation accepts any non-zero value (1-255) as valid for enabling callbacks when, according to the documentation, only the specific value `1` should trigger a callback.

Recommendation: Replace the loose comparison with a strict equality check:

```diff
- bool callback = stream.readUint8() > 0;
+ bool callback = stream.readUint8() == 1;
```

Alternatively, update the documentation in the tests and clearly state that any flag value > 0 will trigger a callback.

LI.FI: Fixed in 92302f00d184e89f9683208fd48e02a8bf0a4a5f

Researcher: Verified fix

6.2 Gas Optimization

6.2.1 Revert early if `amountOut` from velodrome pool is less than `amountOutMin`

Context: LiFiDEXAggregator.sol#L778

Description: The `swapVelodromeV2()` accepts any output amount from the velodrome pool as a valid output and continues the execution flow. Later, in the `processRouteInternal()` function, the output amounts are verified, and the call will be reverted. However, reverting to the `swapVelodromeV2()` will save the user gas before executing the swap.

Recommendation: Consider validating the **amountOutMin** inside the `swapVelodromeV2()` function to save user gas.

LI.FI: Acknowledged. While the proposed change makes sense, the amountOutMin value is not easily available in the swapVelodromeV2() function. We accept the potential gas loss in error case as this is a rare case anyway.

Researcher: Acknowledged.

6.3 Informational

6.3.1 Function `swapVelodromeV2()` can be used to make unauthorized calls to the recipient hook

Context: LiFiDEXAggregator.sol#L806

Description: The `swapVelodromeV2()` function accepts arbitrary pool addresses from the stream supplied by the user. Hence, the pool address can be manipulated to call a victim recipient's `hook()` function without sender validation. In most cases, it won't result in financial losses, but it totally depends on the hook design on the recipient contract.

Recommendation: Add documentation on the callback behavior and inform users that they should not trust the hook calls or validate whether the sender is a valid Velodrome pool contract.

LI.FI: Documentation about this behavior added in 778d22b1bbf1133bc2d583cb9a1d38b1fcf50ee4

Researcher: The documentation added looks good.

6.3.2 Improve test coverage

Context: Global

Description: The `LiFiDEXAggregator.sol` contract has less than complete test coverage. Adequate test coverage and regular reporting are essential to ensuring the codebase works as intended. Insufficient code coverage may lead to unexpected issues and regressions arising due to underlying smart contract implementation changes.

Cases like re-entrancy, pausing, unauthorized calls to previlaged functions are not unit tested,

Recommendation: Add to test coverage ensuring all execution paths are covered. See also Paul R Berg's BTT thread and talk.

LI.FI: Acknowledged. We took over the fully audited contract from Sushiswap (RouteProcessor4.sol) and currently only write tests for the code we change or add.

Researcher: Acknowledged.

6.3.3 typos

Context: IVelodromeV2Pool.sol#L11

Description: Typos can reduce code quality.

1. IVelodromeV2Pool.sol

```
-  /// @param to           Address to recieve the swapped output
+  /// @param to           Address to receive the swapped output
```

Recommendation: Consider fixing the typos mentioned above

LI.FI: Fixed in aae077788359172c4fa9eddc519dc91af6a354b3

Researcher: Verified fix

6.3.4 Misplaced comment in `swapVelodromeV2()` function

Context: LiFiDEXAggregator.sol#L775

Description: The explanatory comment `// we don't need 'fee' and 'stable' flags since the pool handles that internally` is incorrectly positioned in the code. It appears after the stream reads operations it's describing, rather than before them, making the code harder to understand and maintain.

Recommendation: Reposition the comment to appear before the related stream read operations:

```
+ // we don't need 'fee' and 'stable' flags since the pool handles that internally
stream.readUint24(); // `pool fee` flag
- stream.readUint8();
+ stream.readUint8(); // `stable` flag
bool callback = stream.readUint8() > 0; // if true then run callback after swap with tokenIn as
↪  flashloan data. Will revert if contract (to) does not implement IVelodromeV2PoolCallee

- // we don't need 'fee' and 'stable' flags since the pool handles that internally
```

LI.FI: Fixed in 10488bdb4e5c91c13a719e385b2a66c951fc83a8

Researcher: Verified fix

6.3.5 Missing zero address validation for `pool` and `to` address in `swapVelodromeV2()` Function

Context: LiFiDEXAggregator.sol#L769

Description: The `swapVelodromeV2()` function reads the `pool` address and `to` address from the input stream but does not validate that this address is non-zero before interacting with it. A zero or invalid address could be passed through the stream, leading to transaction failures or, in the worst case, loss of funds for users.

Recommendation: Add explicit validation to ensure the `pool` and `to` addresses are not the zero address:

```
address pool = stream.readAddress();
+ if (pool == address(0)) revert InvalidPoolAddress();

uint8 direction = stream.readUint8();
address to = stream.readAddress();
+ if (to == address(0)) revert InvalidToAddress();
```

LI.FI: Fixed in 97781cd4f0c7fd7286401fc22f69f0c2fe22317f

Researcher: Verified fix