AstraSec

Orbs Liquidity Hub Security Audit Report

October 24, 2024

Contents

1 Introduction

1.1 About Liquidity Hub

Liquidity Hub is a decentralized optimization layer that operates above Automated Market Makers (AMMs). This layer mitigates the problem of fragmented liquidity in DeFi, enabling DEXs to tap into external liquidity sources in order to provide better prices on swaps.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/orbs-network/liquidity-hub.git
- CommitID: cc5aed4

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/orbs-network/liquidity-hub.git
- CommitID: 3fcc14e

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the Liquidity Hub protocol. Throughout this audit, we identified a total of 3 issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won’t Do	Addressed
Critical	–	–	–	–
High	–	–	–	–
Medium	1	–	–	1
Low	1	–	–	1
Informational	1	1	–	–
Undetermined	–	–	–	–

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~L-1~~	Unexpected Revert in LiquidityHub::_verifyOutAmountSwapper()
~~L-2~~	Revisited Logic of LiquidityHub::_handleOrderOutputs()
I-1	Meaningful Events for Key Operations

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

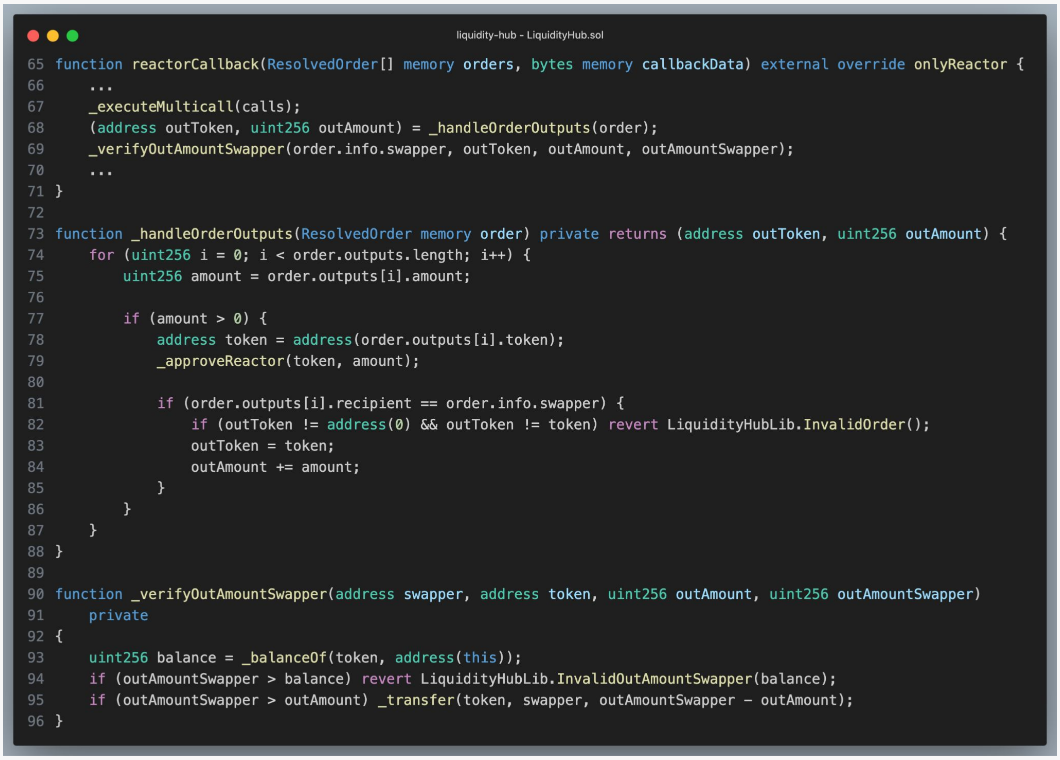
Severity	Acknowledged
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

3.3.1 [L-1] Unexpected Revert in LiquidityHub::_verifyOutAmountSwapper()

Target	Category	IMPACT	LIKELIHOOD	STATUS
LiquidityHub.sol	Business Logic	Low	Medium	Addressed

In the reactorCallback() function, the _verifyOutAmountSwapper() function is executed (line 69) after the token exchange (line 67) to ensure that the amount of outToken held by the LiquidityHub contract is greater than or equal to the input outAmountSwapper parameter. However, the current implementation overlooks the fact that if the outToken is ETH, the ETH is transferred directly to the PartialOrderReactor contract before _verifyOutAmountSwapper() is called. This causes the balance check for ETH to fail, leading to an unexpected revert of the transaction.


liquidity-hub - LiquidityHub.sol

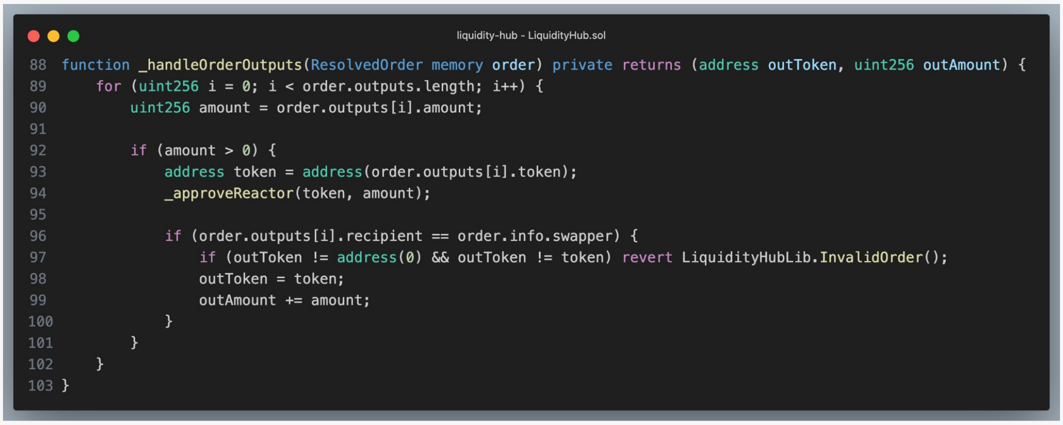
```
65 function reactorCallback(ResolvedOrder[] memory orders, bytes memory callbackData) external override onlyReactor {
66     ...
67     _executeMulticall(calls);
68     (address outToken, uint256 outAmount) = _handleOrderOutputs(order);
69     _verifyOutAmountSwapper(order.info.swapper, outToken, outAmount, outAmountSwapper);
70     ...
71 }
72
73 function _handleOrderOutputs(ResolvedOrder memory order) private returns (address outToken, uint256 outAmount) {
74     for (uint256 i = 0; i < order.outputs.length; i++) {
75         uint256 amount = order.outputs[i].amount;
76
77         if (amount > 0) {
78             address token = address(order.outputs[i].token);
79             _approveReactor(token, amount);
80
81             if (order.outputs[i].recipient == order.info.swapper) {
82                 if (outToken != address(0) && outToken != token) revert LiquidityHubLib.InvalidOrder();
83                 outToken = token;
84                 outAmount += amount;
85             }
86         }
87     }
88 }
89
90 function _verifyOutAmountSwapper(address swapper, address token, uint256 outAmount, uint256 outAmountSwapper)
91     private
92 {
93     uint256 balance = _balanceOf(token, address(this));
94     if (outAmountSwapper > balance) revert LiquidityHubLib.InvalidOutAmountSwapper(balance);
95     if (outAmountSwapper > outAmount) _transfer(token, swapper, outAmountSwapper - outAmount);
96 }
```

Remediation Improve the implementation of the _verifyOutAmountSwapper() function to handle this scenario.

3.3.2 [L-2] Revisited Logic of LiquidityHub::_handleOrderOutputs()

Target	Category	IMPACT	LIKELIHOOD	STATUS
LiquidityHub.sol	Business Logic	Low	Low	Addressed

The _handleOrderOutputs() function processes the order.outputs and accumulates the outToken and outAmount for the order.info.swapper. The statement (if (outToken != address(0) && outToken != token) revert LiquidityHubLib.InvalidOrder()) (line 97) is designed to ensure that the tokens provided to the order.info.swapper in the order.outputs are consistent. However, the function fails to account for a specific scenario where the first token in the order.outputs is ETH (i.e., address(0)), and the second token is a non-ETH token (e.g., USDT). In this case, the function mishandles the outToken and outAmount, potentially leading to incorrect token accumulation and reverting the transaction incorrectly.



```
88  function _handleOrderOutputs(ResolvedOrder memory order) private returns (address outToken, uint256 outAmount) {
89      for (uint256 i = 0; i < order.outputs.length; i++) {
90          uint256 amount = order.outputs[i].amount;
91
92          if (amount > 0) {
93              address token = address(order.outputs[i].token);
94              _approveReactor(token, amount);
95
96              if (order.outputs[i].recipient == order.info.swapper) {
97                  if (outToken != address(0) && outToken != token) revert LiquidityHubLib.InvalidOrder();
98                  outToken = token;
99                  outAmount += amount;
100             }
101         }
102     }
103 }
```

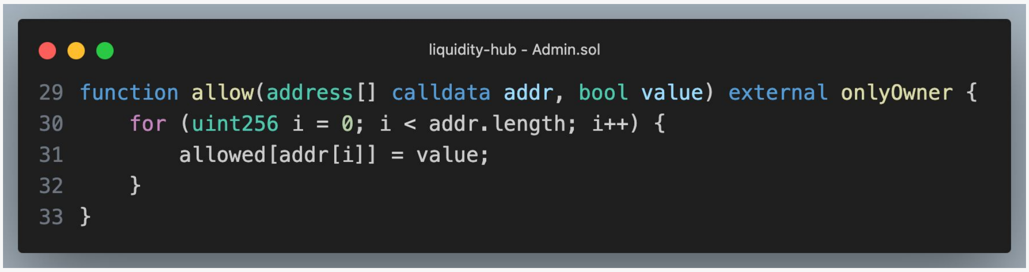
Remediation Improve the implementation of the _handleOrderOutputs() function to handle this scenario.

3.3.3 [I-1] Meaningful Events for Key Operations

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Coding Practice	NA	NA	Acknowledged

The event feature is vital for capturing runtime dynamics in a contract. Upon emission, events store transaction arguments in logs, supplying external analytics and reporting tools with crucial information. They play a pivotal role in scenarios like modifying system-wide parameters or handling token operations.

However, in our examination of protocol dynamics, we observed that certain privileged routines lack meaningful events to document their changes. We highlight the representative routines below.


liquidity-hub - Admin.sol

```
function allow(address[] calldata addr, bool value) external onlyOwner {
    for (uint256 i = 0; i < addr.length; i++) {
        allowed[addr[i]] = value;
    }
}
```

Remediation Ensure the proper emission of meaningful events containing accurate information to promptly reflect state changes.

4 Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

4.3 Contact

Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI