# Reya Network Security Review

## Pashov Audit Group

Conducted by: rvierdiiev, SpicyMeatball, Shaka

July 15th 2024 - July 17th 2024

# Contents

# 1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

# 2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

# 3. Introduction

A time-boxed security review of the **Reya-Labs/reya-network** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

# 4. About Reya Network

Reya Network is a trading-optimised modular L2 for perpetuals. The chain layer is powered by Arbitrum Orbit and is gas-free, with transactions ordered on a FIFO basis. The protocol layer directly tackles the vertical integration of DeFi applications by breaking the chain into modular components to support trading, such as PnL settlements, margin requirements, liquidations.

# 5. Risk Classification

| Severity | Impact: High | Impact: Medium | Impact: Low |
|---|---|---|---|
| Likelihood: High | Critical | High | Medium |
| Likelihood: Medium | High | Medium | Low |
| Likelihood: Low | Medium | Low | Low |

## 5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

## 5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

## 5.3. 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

# 6. Security Assessment Summary

*review commit hash* - 71f999550540da759355f042c7efdf15fd19ed63

*fixes review commit hash* - accd2420aa24a8424172b175ef4cf253a610358b

## Scope

The following smart contracts were in scope of the audit:

- `OracleAdaptersProxy`
- `DataTypes`
- `Errors`
- `Events`
- `FeatureFlagSupport`
- `ConfigurationModule`
- `FeatureFlagModule`
- `OwnerUpgradeModule`
- `StorkERC7412WrapperModule`
- `StorkPriceInformationModule`
- `Configuration`
- `StorkPrice`
- `NodeModule`
- `FallbackReducerNode`
- `StorkOffchainLookupNode`
- `NodeDefinition`

# 7. Executive Summary

Over the course of the security review, rvierdiiev, SpicyMeatball, Shaka engaged with Reya Network to review Reya Network. In this period of time a total of **3** issues were uncovered.

## Protocol Summary

| | |
|---|---|
| **Protocol Name** | Reya Network |
| **Repository** | https://github.com/Reya-Labs/reya-network |
| **Date** | July 15th 2024 - July 17th 2024 |
| **Protocol Type** | Perpetuals Trading L2 |

## Findings Count

| **Severity** | **Amount** |
|---|---|
| Medium | 1 |
| Low | 2 |
| **Total Findings** | **3** |

# Summary of Findings

| ID | Title | Severity | Status |
|---|---|---|---|
| [M-01] | Stork Oracle price can be arbitrated | Medium | Acknowledged |
| [L-01] | RedstoneNode will proceed with price calculation if price fetching reverts | Low | Resolved |
| [L-02] | Ability to set max stale duration for non-existent node | Low | Resolved |

# 8. Findings

## 8.1. Medium Findings

## [M-01] Stork Oracle price can be arbitrated

### Severity

**Impact:** Medium

**Likelihood:** Medium

### Description

`StorkERC7412WrapperModule.fulfillOracleQuery` is used to update the price of the Stork oracle.

Given that the oracle works with a push model, this opens the opportunity to arbitrage a price update by performing a trade, setting the new price, and closing the position, all in the same transaction.

What is more, the price can be updated multiple times in the same block, so the attacker can choose both the initial and final price, as long as they satisfy the following conditions:

- The initial price timestamp is equal to or greater than the previous price timestamp.
- The final price timestamp is equal to or greater than the initial price timestamp.

This increases the chances of performing a profitable arbitrage, especially in moments of high volatility.

Note that while `FeatureFlagSupport.ensureExecutorAccess()` ensures that the caller is allowed to perform the price update, and the protocol will start restricting the update of the price to trusted entities, it is planned to allow

anyone to perform this action in the future, which can be done by setting `allowAll` to `true` for the executor feature flag.

# Recommendations

Ensure that the Stork price is not updated more than once per block and that the new price is not stale. This is a reference implementation of how this could be done:

```
File: StorkPrice.sol

library StorkPrice {
    struct Data {
        StorkPricePayload latestPrice;
+       uint256 updatedAtBlock;
    }

(...)

+   function getLatestPriceUpdatedAtBlock
+ (string memory assetPairId) internal view returns (uint256) {
+       return load(assetPairId).updatedAtBlock;
+   }
}
```

```
File: StorkERC7412WrapperModule.sol

        if (latestPrice.timestamp > pricePayload.timestamp) {
            revert Errors.StorkPayloadOlderThanLatest
              (pricePayload, latestPrice);
        }
+


+
+        // Do not revert if the price has already been updated in the current block,
+
+        // do a multicall to update the price and perform a trade in the same transac
+       // been updated in the current block, we can continue.
+       if (latestPriceUpdatedAtBlock == block.number) {
+           return;
+       }
+
+       if (pricePayload.timestamp + MAX_PAYLOAD_DELAY < block.timestamp) {
+           revert Errors.StorkPayloadStaleData(pricePayload);
+       }
+
-       StorkPrice.load(pricePayload.assetPairId).latestPrice = pricePayload;
+       StorkPrice storage storkPrice = StorkPrice.load
+ (pricePayload.assetPairId);
+       storkPrice.updatedAtBlock = block.number;
+       storkPrice.latestPrice = pricePayload;
```

When the Stork oracle price is consumed, ensure that the price has been updated in the current block.

```
File: NodeDefinition.sol

    function isPriceFresh(bytes32 id, uint256 timestamp) internal view returns
      (bool) {
+       NodeDefinition.Data memory nodeDefinition = NodeDefinition.load(id);
+       if (nodeDefinition.nodeType == NodeType.STORK_OFFCHAIN_LOOKUP) {

+           return latestPriceUpdatedAtBlock == block.number;
+       }
+
        Data storage storedConfig = load(id);
```

# 8.2. Low Findings

## [L-01] RedstoneNode will proceed with price calculation if price fetching reverts

`RedstoneNode.getAveragePrice` function should calculate the twap price during the time interval. In order to do so, it fetches information about rounds and if they are not stale, their price is included to calculate the average.

```
while (latestRoundId > 0) {
    try redstone.getRoundData(--latestRoundId) returns (
        uint80, int256 answer, uint256, uint256 updatedAt, uint80
    ) {
        if (updatedAt < startTime) {
            break;
        }
        if (answer < 0) {
            revert NegativePrice(answer, redstone);
        }
        priceSum += answer.toUint();
        priceCount++;
    } catch {
        break;
    }
}
```

In case `redstone.getRoundData(--latestRoundId)` reverts then the function breaks the loop through rounds instead of revert. As a result, the price will be calculated on less amount of sources than it should.

## [L-02] Ability to set max stale duration for non-existent node

The `owner` has the capability to set the `maxStaleDuration` for a non-existent node:

```
function setMaxStaleDuration
      (bytes32 nodeId, uint256 maxStaleDuration) external {
        OwnableStorage.onlyOwner();
        NodeDefinition.setMaxStaleDuration(nodeId, maxStaleDuration);

        emit MaxStaleDurationUpdated(nodeId, maxStaleDuration);
    }
```

This allows a newly created node to have a `maxStaleDuration` value other than zero. It is recommended to validate the existence of nodes before modifying their parameters.