RENASCENCE

Dinero (Arbitrum LST) Audit Report

Version 2.0

Audited by:

MiloTruck

bytes032

September 25, 2024

Contents

1 Introduction

1.1 About Renascence

Renascence Labs was established by a team of experts including HollaDieWaldfee, MiloTruck, alexxander and bytes032.

Our founders have a distinguished history of achieving top honors in competitive audit contests, enhancing the security of leading protocols such as Reserve Protocol, Arbitrum, MaiaDAO, Chainlink, Dodo, Lens Protocol, Wenwin, PartyDAO, Lukso, Perennial Finance, Mute and Taurus.

We strive to deliver tailored solutions by thoroughly understanding each client's unique challenges and requirements. Our approach goes beyond addressing immediate security concerns; we are dedicated to fostering the enduring success and growth of our partners.

More of our work can be found here.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an 'as-is' and 'as-available' basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

1.3.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol's functionality
- Low - Funds are **not** at risk

1.3.2 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

2 Executive Summary

2.1 About Dinero

Dinero is an experimental protocol which capitalizes on the premium blockspace market by introducing:

1. An ETH liquid staking token (“LST”) which benefits from staking yield and the Dinero protocol
2. A decentralized stablecoin (DINERO) as a medium of exchange on Ethereum
3. A public and permissionless RPC for users

2.2 Overview

Project	Dinero (Arbitrum LST)
Repository	dinero-pirex-eth
Commit Hash	6f4fbed86674...
Mitigation Hash	a341d24a0f67...
Date	16 September 2024 - 18 September 2024

2.3 Issues Found

Severity	**Count**
High Risk	0
Medium Risk	0
Low Risk	0
Informational	2
Total Issues	**2**

3 Findings Summary

ID	Description	Status
I-1	Missing `tokenIn != Constants.ETH_ADDRESS` in `L1ArbReceiverETH.onMessageReceived()`	Acknowledged
I-2	`LiquidStakingToken` and `LiquidStakingTokenArb` can be redesigned	Resolved

4 Findings

Informational

[I-1] Missing `tokenIn != Constants.ETH_ADDRESS` in `L1ArbReceiverETH.onMessageReceived()`

Context:

- L1OPReceiverETH.sol#L51
- L1ArbReceiverETH.sol#L37-L65

Description: In `L1OPReceiverETH`, the `onMessageReceived()` function checks that `tokenIn` is the `ETH_ADDRESS` constant:

```
if (tokenIn != Constants.ETH_ADDRESS) revert Errors.OnlyETH();
```

However, `onMessageReceived()` in `L1ArbReceiverETH` does not contain this check, which is inconsistent.

Although `tokenIn` will never be an address other than `ETH_ADDRESS` if the `LiquidStakingToken` contract on L2 is configured correctly, it is best to add this check on L1.

Recommendation: Consider adding the `tokenIn != Constants.ETH_ADDRESS` check to `L1ArbReceiverETH.onMessageReceived()`.

Redacted: Acknowledged. On Arbitrum, the `tokenIn` when it comes from L3 (Orbit) might not be ETH.

Renascence: Acknowledged.

[I-2] `LiquidStakingToken` **and** `LiquidStakingTokenArb` **can be redesigned**

Context:

- LiquidStakingToken.sol#L793-L874
- LiquidStakingTokenArb.sol#L38-L115

Description: In the `LiquidStakingToken` contract, `_sync()` sends a slow sync message from Optimism to L1:

```
// Some logic here...

// send slow sync message
ICrossDomainMessenger(getMessenger()).sendMessage{value: _amountIn}(
    getReceiver(),
    message,
    _minGasLimit()
);

// Some logic here...
```

To deploy the LST on Arbitrum, a new `ArbitrumLST` contract that inherits `LiquidStakingToken` has been added in `LiquidStakingTokenArb.sol`. `_sync()` was overridden with the same logic, with the only difference being that it sends an L2 → L1 message from Arbitrum to L1 instead:

```
// Some logic here...

IArbitrumMessenger(getMessenger()).sendTxToL1{value: _amountIn}(
    getReceiver(),
    message
);

// Some logic here...
```

However, this inheritance pattern is not ideal as `LiquidStakingToken._sync()` and `ArbitrumLST._sync()` contains duplicated logic.

Recommendation: Consider making `LiquidStakingToken` an abstract contract, with both `LiquidStakingTokenArb` and a new `LiquidStakingTokenOp` contract inheriting it for deployment on Arbitrum and Optimism respectively.

For example:

1. Declare the `LiquidStakingToken` contract as abstract:

```
- contract LiquidStakingToken is
+ abstract contract LiquidStakingToken is
```

2. Add a virtual function that sends an L2 → L1 message in `LiquidStakingToken`:

```
function _sendSlowSyncMessage(uint256 value, bytes memory message) internal virtual;
```

3. In `LiquidStakingToken._sync()`, call `_sendSlowSyncMessage()` instead of Optimism's bridge:

```
  // Some logic here...

  // send slow sync message
- ICrossDomainMessenger(getMessenger()).sendMessage{value: _amountIn}(
-     getReceiver(),
-     message,
-     _minGasLimit()
- );
+ _sendSlowSyncMessage(_amountIn, message);

  // Some logic here...
```

4. Instead of overriding `_sync()` in `ArbitrumLST`, override `_sendSlowSyncMessage()` with logic to send an L2 → L1 transaction from Arbitrum:

```
contract ArbitrumLST is LiquidStakingToken {
    constructor(
        address _endpoint,
        uint32 _srcEid
    ) LiquidStakingToken(_endpoint, _srcEid) {}

    function _sendSlowSyncMessage(uint256 value, bytes memory message) internal
override {
        IArbitrumMessenger(getMessenger()).sendTxToL1{value: value}(
            getReceiver(),
            message
        );
    }
}
```

The same can be done for `LiquidStakingTokenOp`.

Redacted: Fixed in commit d415cc3.

Renascence: Verified, the recommended fix was implemented.