# Layer Zero Orbit Security Review

## Pashov Audit Group

Conducted by: defsec, ast3ros, pontifex

September 18th - September 20th

# 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 **LayerZero-Labs/orbit-adapter** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

# 4. About Orbit

Layer Zero Orbit integrates Omnichain Fungible Tokens (OFT) into Arbitrum’s native bridge to enable smooth asset transfers between Layer 3 (L3) and Arbitrum (L2) networks without issues like liquidity splits. Users can transfer assets using either the Arbitrum bridge or OFT seamlessly.

# 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* - baf2f49c3c4b53d7c733e49c111f39630609cda2

## Scope

The following smart contracts were in scope of the audit:

- `OrbitProxyOFT1_2.sol`
- `OrbitNativeOFT1_2.sol`

# 7. Executive Summary

Over the course of the security review, defsec, ast3ros, pontifex engaged with Layer Zero to review Orbit. In this period of time a total of **5** issues were uncovered.

## Protocol Summary

| | |
|---|---|
| **Protocol Name** | Orbit |
| **Repository** | https://github.com/LayerZero-Labs/orbit-adapter |
| **Date** | September 18th - September 20th |
| **Protocol Type** | Messaging protocol |

## Findings Count

| **Severity** | **Amount** |
|---|---|
| Medium | 2 |
| Low | 3 |
| **Total Findings** | **5** |

# Summary of Findings

| ID | Title | Severity | Status |
|---|---|---|---|
| [M-01] | Unexpected token divisibility issues on non-evm chains | Medium | Acknowledged |
| [M-02] | Lack of global outbound limit | Medium | Acknowledged |
| [L-01] | Missing pause mechanism | Low | Acknowledged |
| [L-02] | Incorrect circulating amount of tokens on the current chain | Low | Acknowledged |
| [L-03] | Ignoring return value | Low | Acknowledged |

# 8. Findings

## 8.1. Medium Findings

## [M-01] Unexpected token divisibility issues on non-evm chains

### Severity

**Impact:** High

**Likelihood:** Low

### Description

The OrbitProxyOFT1_2 contract, which inherits from BaseOFTV2, uses a `_sharedDecimals` parameter in its constructor. However, this value is not explicitly set to accommodate potential differences in decimal places across various blockchain networks, especially non-EVM chains.

```
constructor(
    address _token,
    uint8 _sharedDecimals,
    address _lzEndpoint,
    IERC20Bridge _bridge
) BaseOFTV2(_sharedDecimals, _lzEndpoint) {
    // ... other initialization logic ...
    if (_sharedDecimals > decimals) {
        revert SharedDecimalsTooBig();
    }
    ld2sdRate = 10**(decimals - _sharedDecimals);
}
```

While the contract does check if `_sharedDecimals` is not greater than the token's decimals, it doesn't ensure that the lowest common denominator across all potential chains is used. If OrbitProxyOFT1_2 is deployed on both EVM (typically 18 decimals) and non-EVM chains (e.g., Aptos with 6 decimals), using 18 as `_sharedDecimals` could lead to precision loss or overflow issues on chains with fewer decimal places.

## Recommendation

Determine the lowest decimal count among all supported chains.

# [M-02] Lack of global outbound limit

## Severity

**Impact:** High

**Likelihood:** Low

## Description

The current implementation checks that individual transactions do not exceed the uint64 max value in shared decimals:

```
function _debitFrom(
        address _from,
        uint16,
        bytes32,
        uint256 _amount
    ) internal virtual override returns (uint) {
        ...

        uint256 cap = _sd2ld(type(uint64).max);
        if (amount > cap) {
            revert AmountOverflow();
        }

        return amount;
    }
```

From the architecture, the gas token is bridge not only between L2 and L3 but also between other chains. However, there is no global tracking of the total amount bridged to non-EVM chains. This could allow the cumulative outbound amount across many transactions to exceed `uint64.max` in shared decimals. Many non-EVM chains use `uint64` to represent token balances. If the total bridged amount exceeds this limit, it could cause balance overflows on those chains, potentially locking tokens or creating other unexpected behaviors.

## Recommendations

Tracking total `outboundAmount` for other chains except L3 and if the total amount is greater than the cap `_sd2ld(type(uint64).max)`, revert the

transaction.

# 8.2. Low Findings

## [L-01] Missing pause mechanism

The OrbitNativeOFT1_2 contract currently lacks a pause mechanism. This feature is crucial for smart contracts, especially those handling cross-chain token transfers and native currency wrapping/unwrapping. A pause functionality allows the contract owner or designated parties to temporarily halt contract operations in case of emergencies, such as discovered vulnerabilities, unexpected behavior, or during upgrades.

Implement the Pausable pattern on the contract.

## [L-02] Incorrect circulating amount of tokens on the current chain

The circulating amount of tokens on the current chain includes the bridged tokens.

```
function circulatingSupply() public view virtual override returns (uint) {
        return innerToken.totalSupply();
    }
```

Consider tracking a separate variable for bridged tokens to subtract it from `innerToken.totalSupply()`.

## [L-03] Ignoring return value

The `OrbitProxyOFT1_2` ignores `IBridge.executeCall` function return.

```
function executeCall(
        address to,
        uint256 value,
        bytes calldata data
>>  ) external returns (bool success, bytes memory returnData);
```

There is no impact for the current `IBridge` implementation in `ERC20Bridge` contract but can cause asset losses for custom implementations. Consider checking the return value from `IBridge.executeCall` calls. There are two instances of the issue:

```
function _debitFrom(
        address _from,
        uint16,
        bytes32,
        uint256 _amount
    ) internal virtual override returns (uint) {
        if (_from != _msgSender()) {
            revert OwnerNotSendCaller();
        }

        _amount = _transferFrom(_from, address(bridge), _amount);

        // _amount still may have dust if the token has transfer fee, then give
        // the dust back to the sender
        (uint256 amount, uint256 dust) = _removeDust(_amount);
>>      if (dust > 0) bridge.executeCall(_from, dust, "");
<...>
    function _transferFrom(
        address _from,
        address _to,
        uint256 _amount
    ) internal virtual override returns (uint) {
        uint256 before = innerToken.balanceOf(_to);
        if (_from == address(bridge)) {
>>          bridge.executeCall(_to, _amount, "");
        } else if (_from == address(this)) {
            innerToken.safeTransfer(_to, _amount);
        } else {
            innerToken.safeTransferFrom(_from, _to, _amount);
        }
        return innerToken.balanceOf(_to) - before;
    }
```