SALUS SECURITY

JAN 2026

# CODE SECURITY ASSESSMENT

LORENZO

# Overview

## Project Summary

- Name: Lorenzo - Airdrop
- Platform: BSC
- Language: Solidity
- Repository:
  - https://github.com/Lorenzo-Protocol/Unified-Merkle-Airdrop
- Audit Range: See Appendix - 1

# Project Dashboard

## Application Summary

| | |
|---|---|
| **Name** | Lorenzo - Airdrop |
| **Version** | v2 |
| **Type** | Solidity |
| **Dates** | Jan 19 2026 |
| **Logs** | Jan 15 2026; Jan 19 2026 |

## Vulnerability Summary

| | |
|---|---|
| **Total High-Severity issues** | 0 |
| **Total Medium-Severity issues** | 0 |
| **Total Low-Severity issues** | 1 |
| **Total informational issues** | 1 |
| **Total** | 2 |

## Contact

E-mail: support@salusec.io

## Risk Level Description

| | |
|---|---|
| **High Risk** | The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users. |
| **Medium Risk** | The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact. |
| **Low Risk** | The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances. |
| **Informational** | The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth. |

# Content

# Introduction

## 1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

## 1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

## 1.3 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 with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issue

# Findings

## 2.1 Summary of Findings

| ID | Title | Severity | Category | Status |
|---|---|---|---|---|
| 1 | Inconsistency between code and comments | Low | Business Logic | Resolved |
| 2 | Use of floating pragma | Informational | Configuration | Acknowledged |

## 2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

| **1. Inconsistency between code and comments** | |
|---|---|
| Severity: Low | Category: Business Logic |
| Target:<br>- contracts/MerkleAirdrop.sol | |

### Description

The computeLeaf implementation is inconsistent with the documentation comments: the comments state `` `leaf node hash = keccak256(abi.encode(task_type, epoch, user, token, amount))` ``, but the code wraps `` `keccak256(abi.encode(...))` `` in another layer of `` `keccak256(bytes.concat(...))` ``, forming a double hash.

contracts/MerkleAirdrop.sol

```
function computeLeaf(
    uint256 taskType,
    uint256 epoch,
    address user,
    address token,
    uint256 amount
) public pure returns (bytes32) {
    bytes32 leafNode = keccak256(
        bytes.concat(
            keccak256(abi.encode(taskType, epoch, user, token, amount))
        )
    );
    return leafNode;
}
```

### Recommendation

Unified as a double hash.

### Status

The team has resolved this issue in commit 8727f50.

## 2.3 Informational Findings

| 2. Use of floating pragma | |
|---|---|
| Severity: Informational | Category: Configuration |
| Target:<br>- All | |

### Description

```
pragma solidity ^0.8.28;
```

For example, the `LRZToken` and `MerkleAridrop` use a floating compiler version ^0.8.28.

Using a floating pragma ^0.8.28 statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

### Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

### Status

This issue has been acknowledged by the team.

# Appendix

## Appendix 1 - Files in Scope

This audit covered the following files in commit f8bb458:

| File | SHA-1 hash |
|---|---|
| MerkleAirdrop.sol | 66690ed9936e29be7d9cd7491a7e2a9cc9b3589d |
| LRZToken.sol | 264c5258d871eea33f94e01ffec4acfab6fa648a |