SALUS SECURITY

JAN 2026

CODE SECURITY ASSESSMENT

LORENZO

Overview

Project Summary

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

Project Dashboard

Application Summary

Name	Lorenzo - ListaEarn
Version	v3
Type	Solidity
Dates	Jan 19 2026
Logs	Jan 15 2026; Jan 19 2026; Jan 19 2026

Vulnerability Summary

Total High-Severity issues	2
Total Medium-Severity issues	0
Total Low-Severity issues	1
Total informational issues	3
Total	6

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	Mint() logic is incorrect and will always revert when feeRate > 0	High	Denial of Service	Resolved
2	Withdraw() uses net-of-fee previews but withdraws the pre-fee amount, causing funds loss	High	Business Logic	Resolved
3	Changing underlyingAsset/listaVault can strand accrued fees and break asset accounting	Low	Business Logic	Acknowledged
4	Mint() emits incorrect Deposit event parameters	Informational	Logging	Resolved
5	Function Visibility Overly Permissive	Informational	Code Quality	Acknowledged
6	Use of floating pragma	Informational	Configuration	Acknowledged

2.2 Notable Findings

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

1. Mint() logic is incorrect and will always revert when feeRate > 0

Severity: High	Category: Denial of Service
Target: - contracts/ListaEarn.sol	

Description

The `ListaEarn` contract charges an entry fee by reducing the assets forwarded into `listaVault`. The `deposit()` path applies the fee first and then mints shares based on the net assets deposited into `listaVault`, which is internally consistent.

However, `mint()` computes `assetsGross = listaVault.previewMint(shares)` for the user-requested shares, then deducts a fee and only forwards/approves `assetsNet = assetsGross - fee`, while still calling `listaVault.mint(shares, ...)` to mint the original shares. With listaVault's ERC4626 implementation, `mint(shares)` requires pulling assetsGross from the caller, so approving only assetsNet causes `mint()` to revert whenever `feeRate > 0`.

contracts/ListaEarn.sol:L282-L311

```
function mint(uint256 shares, address receiver) public override nonReentrant returns
(uint256) {
        uint256 assets = listaVault.previewMint(shares);
        require(assets >= minDeposit, "ListaEarn: less than min deposit");
        SafeERC20.safeTransferFrom(IERC20(underlyingAsset), msg.sender, address(this),
assets);

        // calculate fee
        assets = chargeFee(assets);

        // Approve listaVault to spend assets
        SafeERC20.forceApprove(IERC20(underlyingAsset), address(listaVault), assets);
        uint256 assetsMinted = listaVault.mint(shares, address(this));
        _mint(receiver, shares);
        emit Deposit(msg.sender, receiver, underlyingAsset, assetsMinted, assets);
        return assetsMinted;
    }
```

More importantly, even if this revert is "fixed" naively (e.g., approving/forwarding assetsGross), the fee accounting becomes economically incorrect: the user would still receive shares even though only assetsNet is considered after fees, breaking the expected equivalence between `deposit()` and `mint()`. This turns `mint()` into a potential fee-bypass / share-pricing inconsistency path (i.e., obtaining the same shares with less effective net assets than intended).

Recommendation

Record `assets = fee + assets(listaVault.previewMint(shares))` then transfer to `ListaEarn` contract, and revise the following logic.

Status

The team has resolved this issue in commit 721ea68.

2. Withdraw() uses net-of-fee previews but withdraws the pre-fee amount, causing funds loss

Severity: High	Category: Business Logic

Target:
- contracts/ListaEarn.sol

Description

`previewWithdraw(assets)` treats `assets` as the net amount the user should receive by grossing it up. However, `withdraw(assets, ...)` calls the underlying vault with the ungrossed amount.

contracts/ListaEarn.sol:L323-L361

```
function withdraw(
        uint256 assets,
        address receiver,
        address owner
    ) public override nonReentrant returns (uint256) {
        require(receiver != address(0), "ListaEarn: zero address");

        uint256 shares = previewWithdraw(assets);
        require(shares <= balanceOf(owner), "ListaEarn: insufficient shares");
        ...
        // Withdraw from listaVault first
        uint256 sharesBurned = listaVault.withdraw(
            assets,
            address(this),
            address(this)
        );

        // Burn user's shares
        _burn(owner, shares);

        // calculate fee
        uint256 fee = previewFee(assets);
        totalFeeCharged += fee;
        uint256 returnAmount = assets - fee;

        // Transfer assets to receiver
        SafeERC20.safeTransfer(IERC20(underlyingAsset), receiver, returnAmount);

        emit Withdraw(msg.sender, receiver, owner, underlyingAsset, returnAmount,
shares);
        return sharesBurned;
    }
```

As a result, users can be over-charged the fee while the contract only withdraws assets (without the fee) from the vault. This breaks the intended fee semantics and can cause deterministic value loss for withdrawers.

Recommendation

In `withdraw()`, use the same grossed-up amount as `previewWithdraw()`:

- Compute `realAssets` (gross) from the requested net assets and call `listaVault.withdraw(realAssets, ...)` (use ceil division to avoid under-withdraw due to rounding).

- Compute the fee as `fee = realAssets - assets` and transfer exactly assets (net) to the receiver.

Status

The team has resolved this issue in commit bb180c0.

3. Changing underlyingAsset/listaVault can strand accrued fees and break asset accounting

Severity: Low	Category: Business Logic
Target: - contracts/TokenVesting.sol	

Description

`ListaEarn` accumulates protocol fees in `totalFeeCharged` as a single scalar value, without tracking which token those fees were collected in. Meanwhile, `withdrawFee()` always transfers the current underlyingAsset. Since the owner can change either underlyingAsset (`setUnderlyingAsset()`) or the underlying ERC4626 vault (`setListaVault()`, which implicitly defines the vault asset), this creates a fee/accounting mismatch:

Fees collected under the old asset/vault are still counted in `totalFeeCharged`, but after an asset/vault change, `withdrawFee()` attempts to pay them out in the new underlyingAsset.

This can permanently strand fees collected in the old token, or incorrectly pay “fees” out of the new token balance (breaking accounting and potentially extracting non-fee funds held by the contract).

Because `setUnderlyingAsset()` and `setListaVault()` are admin-facing configuration functions, this is a realistic operational footgun that can lead to irrecoverable balances and incorrect fee withdrawals even without malicious intent.

Recommendation

Disallow changing underlyingAsset and vault.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

4. Mint() emits incorrect Deposit event parameters

Severity: Informational	Category: Logging
Target: - contracts/ListaEarn.sol	

Description

`mint()` emits `Deposit(msg.sender, receiver, underlyingAsset, assetsMinted, assets)`, but the last parameter of `Deposit` is `shares`. It incorrectly logs assets (net assets after fees) as `shares`, producing misleading event data.

contracts/ListaEarn.sol:L282-L311

```
function mint(uint256 shares, address receiver) public override nonReentrant returns
(uint256) {
        uint256 assets = listaVault.previewMint(shares);
        ...
        uint256 assetsMinted = listaVault.mint(shares, address(this));
        _mint(receiver, shares);
        emit Deposit(msg.sender, receiver, underlyingAsset, assetsMinted, assets);
        return assetsMinted;
    }
```

Recommendation

Emit `shares` in the event's `shares` field.

Status

The team has resolved this issue in commit 721ea68.

5. Function Visibility Overly Permissive

Severity: Informational	Category: Code Quality
Target: - contracts/ListaEarn.sol	

Description

The `deposit()`, `mint()`, `withdraw()` and `redeem()` functions in the contract are declared with broader visibility than necessary (e.g., public instead of external). Overly permissive visibility can expose internal logic to unintended callers, increase the attack surface, and slightly increase gas costs. Functions intended to be called externally only do not need public visibility and could be declared external to better communicate intended usage and optimize gas.

Recommendation

Consider changing the visibility of the above function.

Status

This issue has been acknowledged by the team.

6. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.28;
```

For example, the `ListaEarn` uses 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 48ea4b0:

File	SHA-1 hash
ListaEarn.sol	7f45b8ce3c5889832a3378273e3bfbbdbf6529af