SALUS SECURITY

SEP 2024

CODE SECURITY ASSESSMENT

LISTA DAO

Overview

Project Summary

- Name: Lista DAO - restake autoCompound
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/lista-dao/lista-token/tree/restake-autoCompound-audit-1.0
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Lista DAO - restake autoCompound
Version	v2
Type	Solidity
Dates	Sep 26 2024
Logs	Sep 25 2024; Sep 26 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	2
Total Low-Severity issues	2
Total informational issues	3
Total	7

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 issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Error return value in getStakeClaimableReward() function	Medium	Business Logic	Resolved
2	Centralization risk	Medium	Centralization	Mitigated
3	Does not support fee on transfer tokens	Low	Business Logic	Acknowledged
4	Missing events for functions that change critical state	Low	Logging	Partially resolved
5	Incorrect gap size	Informational	Code Quality	Acknowledged
6	Redundant Code	Informational	Redundancy	Partially resolved
7	Use of floating pragma	Informational	Configuration	Acknowledged

2.2 Notable Findings

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

1. Error return value in getStakeClaimableReward() function

Severity: Medium	Category: Business Logic
Target: - contracts/dao/ERC20LpListaDistributor.sol	

Description

contracts/dao/ERC20LpListaDistributor.sol:L223-L237

```
function getStakeClaimableReward(address account) external view returns (uint256) {
    uint256 balance = lpBalanceOf[account];
    uint256 supply = lpTotalSupply;
    uint256 updated = stakePeriodFinish;
    if (updated > block.timestamp) updated = block.timestamp;
    uint256 duration = updated - stakeLastUpdate;
    if (duration > 0 && supply > 0) {
        uint256 rewardIntegral = stakeRewardIntegral + (duration * stakeRewardRate *
1e18) / supply;
        uint256 integralFor = stakeRewardIntegralFor[account];
        if (rewardIntegral > integralFor) {
            return (balance * (rewardIntegral - integralFor)) / 1e18;
        }
    }
    return 0;
}
```

The `getStakeClaimableReward()` function is used to retrieve the claimable reward for users. The actual value consists of two parts: `stakeStoredPendingReward` and the not updated reward.

However, this function only returns the not updated reward value, which causes the return value of the function to be incorrect.

Recommendation

Consider modifying the return value to be the sum of the two parts: `stakeStoredPendingReward` and the not updated reward value.

Status

The team has resolved this issue in commit d62c1ca.

2. Centralization risk

Severity: Medium	Category: Centralization
Target: - All	

Description

All contracts have privileged accounts such as `owner`, `manager` and `DEFAULT_ADMIN`. These privileged accounts can modify critical contract parameters, such as `maxFee`, `feeRate`, and perform actions like `withdrawFee()` and `emergencyWithdraw()`. Additionally, `DEFAULT_ADMIN` can grant any permission to any address. If the private keys of any privileged accounts are compromised, an attacker could perform privileged operations, causing significant damage to the contracts and users.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

The team has mitigated this problem by setting `owner` and `manager` to multisig wallet

3. Does not support fee on transfer tokens

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

Description

contracts/dao/StakingVault.sol:L85-L104

```
function sendRewards(address distributor, uint256 amount) external onlyStaking {
    IERC20(rewardToken).safeTransferFrom(staking, address(this), amount);

    uint256 rewardAmount = amount;
    uint256 fee;
    if (feeRate > 0) {
        fee = Math.mulDiv(amount, feeRate, FEE_PRECISION);

        fees += fee;
        rewardAmount -= fee;
    }

    if (rewardAmount > 0) {
        allocated[distributor] += rewardAmount;

        IDistributor(distributor).notifyStakingReward(rewardAmount);
    }

    emit AddRewards(distributor, rewardAmount, fee);
}
```

The `sendRewards()` function is responsible for transferring rewards when users perform `depositLp` or `withdrawLp` actions in the `distributor` contract. These rewards are first sent to the `stakingVault` contract, then transferred to the `distributor` contract by `sendRewards()`. However, the function assumes that the `rewardAmount` transferred to the `stakingVault` is the exact amount being transferred.

If the `rewardToken` is a fee-on-transfer token, the function's logic will be incorrect, as fee-on-transfer tokens deduct a portion of the transfer as a fee. This means that the amount received by the `stakingVault` will be less than the intended `rewardAmount`, leading to incorrect calculations for both the `fee` and the actual `rewardAmount`.

Recommendation

Consider adjusting the logic so that it calculates and uses the actual received amount rather than assuming the `rewardAmount` equals the transferred amount.

Status

This issue has been acknowledged by the team. The team stated that only `Cake` and `The` will be supported as reward tokens.

4. Missing events for functions that change critical state

Severity: Low	Category: Logging

Target:
- contracts/VeListaAutoCompounder.sol
- contracts/dao/ERC20LpListaDistributor.sol
- contracts/dao/ListaVault.sol
- contracts/dao/StakingVault.sol
- contracts/dao/LpProxy.sol
- contracts/dao/PancakeStaking.sol
- contracts/dao/ThenaStaking.sol

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In the aforementioned contract files, modifications to certain key variables do not trigger events.For example, `registerPool()`, `unregisterPool()`,`toggleDefaultStatus()`, `file()`, `setStakeRewardToken()`, `setWeeklyDistributorPercent()`, etc.

Recommendation

It is recommended to emit events for critical state changes.

Status

The team has partially resolved this issue in commit d62c1ca.

2.3 Informational Findings

5. Incorrect gap size

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

Description

contracts/dao/CommonListaDistributor.sol:L243

```
uint256[49] __gap;
```

According to the documentation, the OpenZeppelin team has calculated that having a total of 50 storage slots for upgradeable contracts is more convenient for contract use. `CommonListaDistributor` contract has variables that fill 13 storage slots, so `__gap` should not occupy 49 slots.

Recommendation

Consider changing `uint256[49] __gap` to `uint256[37] __gap`.

Status

This issue has been acknowledged by the team. The contract has already been deployed, so this variable cannot be modified.

6. Redundant Code

Severity: Informational	Category: Redundancy

Target:
- contracts/VeListaAutoCompounder.sol
- contracts/dao/LpProxy.sol
- contracts/dao/ListaVault.sol

Description

1. contracts/VeListaAutoCompounder.sol:L114-L119

```
function initialize(
    ...
) public initializer {
    ...

    feeRate = 300; // 3%
    maxFeeRate = 1000; // 10%
    require(
        feeRate <= maxFeeRate && maxFeeRate <= 10000,
        "Invalid fee rate"
    );

    ...
}
```

Based on the hard-coded values, the comparison here is unnecessary.

2. contracts/dao/LpProxy.sol:L5

```
import "./interfaces/IV2Wrapper.sol";
```

Import the file but not use it.

3. contracts/dao/ListaVault.sol:L22

```
event Deposit(address indexed account, uint256 amount);
```

Defined the event but it has not been used.

Recommendation

Consider removing the redundant code.

Status

The team has partially resolved this issue in commit d62c1ca.

7. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.10;
```

All contracts use a floating compiler version ^0.8.10.

Using a floating pragma ^0.8.10 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. The team has stated that they will use a fixed compiler version 0.8.19 for deployment.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 700b1b9:

File	SHA-1 hash
contracts/VeListaAutoCompounder.sol	3ddf0216be1125e933e5c5751cb388e4ffcb0e9c
contracts/VeLista.sol	437e91ea5f2c35754fd713524ce42f887ffff6d1
contracts/VeListaDistributor.sol	7a11b2d261832f4288363ddd6ffbf0ec3d47bf17
contracts/dao/ERC20LpListaDistributor.sol	42bbbb72bc909c2ab0cdb549d1b73ca145ce5959
contracts/dao/CommonListaDistributor.sol	2749cc60b7e95aef8ef7ef233186c249a435a0b4
contracts/dao/ListaVault.sol	8f59c99c21d94ab657a179758788aef613334bd5
contracts/dao/LpProxy.sol	bde09e1294e3e12b7b9131875b8a0bab113481e3
contracts/dao/PancakeStaking.sol	5567c1f398242283d86939ec7c6e6ea421ff4cc5
contracts/dao/StakingVault.sol	819bcf9da2b0b258bbf525183e2e370fdb94b23e
contracts/dao/ThenaStaking.sol	5ae333c18ccaf41de8896ec98a8ddac7b35cd915