Smart Contract
Security Audit Report

Prepared for Lista DAO

Prepared by Supremacy

April 03, 2024

Contents

1 Introduction

Given the opportunity to review the design document and related codebase of the Lista DAO Airdrop, we outline in the report our systematic approach to evaluate potential security issues in the smart contract(s) implementation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issue related to either security or performance. This document outlines our audit results.

1.1 About Client

Lista DAO functions as the open-source decentralized stablecoin lending protocol powered by LSDfi. Users can undergo staking and liquid staking on Lista, as well as borrow lisUSD against a variety of decentralized collateral. Present on the BNB Chain, Lista aims to position lisUSD as the number one stablecoin in the crypto space, leveraging on innovative liquid staking solutions.

Item	Description
Client	Lista DAO
Website	https://lista.org
Type	Smart Contract
Languages	Solidity
Platform	EVM-compatible

1.2 Audit Scope

In the following, we show the Git repository of reviewed file and the commit hash used in this security audit:

- Repository: https://github.com/lista-dao/lista-token/blob/airdrop/contracts
- Commit Hash: 16a7f43bdee2a81a0165433e3430aec99556f780

Below are the files in scope for this security audit and their corresponding MD5 hashes.

Filename	MD5
./ListaAirdrop.sol	5292faaac5e847ac05e1dc432c3fe424
./MerkleVerifier.sol	66c7640452ccc4194103af2c42b52f89

And this is the commit hash after all fixes for the issues found in the security audit have been checked in:

- Repository: https://github.com/lista-dao/lista-token/blob/airdrop/contracts
- Commit Hash: 19f279c8ff0efd6adaf7b7e4f806dab029937cd9

Filename	MD5
./ListaAirdrop.sol	54bdbcef80a942d8e060ec30137bf85c
./MerkleVerifier.sol	66c7640452ccc4194103af2c42b52f89

1.3 Changelogs

Version	Date	Description
0.1	April 01, 2024	Initial Draft
1.0	April 03, 2024	Final Release

1.4 About Us

Supremacy is a leading blockchain security firm, composed of industry hackers and academic researchers, provide top-notch security solutions through our technology precipitation and innovative research.

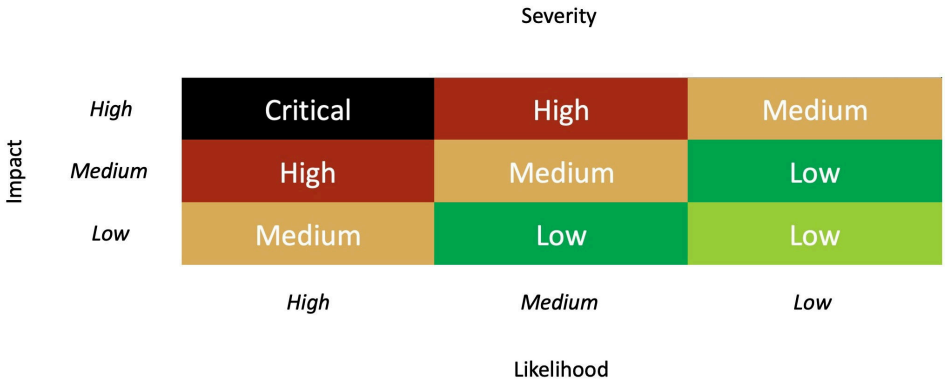
We are reachable at Twitter (https://twitter.com/SupremacyHQ), or Email (contact@supremacy.email).

1.5 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- Likelihood represents the likelihood of a finding to be triggered or exploited in practice
- Impact specifies the technical and business-related consequences of a finding
- Severity is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.



As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

2 Findings

The table below summarizes the findings of the audit, including status and severity details.

ID	Severity	Description	Status
1	Medium	The potential airdrop termination	Fixed
3	Low	Lack of `_endBlock` validation	Fixed
3	Informational	Lack of address validation	Fixed
4	Informational	Lack of comments	Fixed
5	Informational	The potential denial-of-service	Fixed

2.1 Medium

1. The potential airdrop termination [Medium]

`Severity: Medium` `Likelihood: Low` `Impact: High`

`Status: Fixed`

Description:

In the `ListaAirdrop::reclaim()` functions, this is used by Ownership privileged accounts to extract tokens that have not been claimed after `endBlock`; however, there is a lack of validation of `reclaimDelay` in the `constructor()` function. Also, the `reclaimPeriod` variable is not on the same timeline as `endBlock`, which in some extreme cases may result in the final `reclaimPeriod` being much less than the `endBlock` time. In the case of a malicious owner, this could result in the user not being able to claim the Lista token.

```
60      function reclaim(uint256 amount) external onlyOwner {
61          require(block.timestamp > reclaimPeriod, "Tokens cannot be reclaimed");
62          require(IERC20(token).transfer(msg.sender, amount), "Transfer failed");
63      }
```

ListaAirdrop.sol

```
20      constructor(address _token, bytes32 _merkleRoot, uint256 reclaimDelay,
    uint256 _startBlock, uint256 _endBlock) {
21          require(_startBlock >= block.number, "Invalid start block");
22          require(_endBlock > _startBlock, "Invalid end block");
23          token = _token;
24          merkleRoot = _merkleRoot;
25          reclaimPeriod = block.timestamp + reclaimDelay;
26          startBlock = _startBlock;
27          endBlock = _endBlock;
28      }
```

ListaAirdrop.sol

Recommendation: Use the same time calculation in preference and check that it is always greater than `endBlock`. Alternatively, in `reclaim()` directly calculate that the current block is greater than `endBlock`.

2.2 Low

2. Lack of `_endBlock` validation [Low]

`Severity: Medium` `Likelihood: Low` `Impact: Medium`

`Status: Fixed`

Description:

In the `ListaAirdrop::setEndBlock()` function, lack of `_endBlock` validation and check it in the constructor settings of `#L22`. If `ListaAirdrop::setEndBlock()` is called after deploying the contract, unexpected problems may occur.

```
39      function setEndBlock(uint256 _endBlock) external onlyOwner {
40          require(_endBlock != endBlock, "End block already set");
41          endBlock = _endBlock;
42      }
```

ListaAirdrop.sol

```
20      constructor(address _token, bytes32 _merkleRoot, uint256 reclaimDelay,
    uint256 _startBlock, uint256 _endBlock) {
21          require(_startBlock >= block.number, "Invalid start block");
22          require(_endBlock > _startBlock, "Invalid end block");
23          token = _token;
24          merkleRoot = _merkleRoot;
25          reclaimPeriod = block.timestamp + reclaimDelay;
26          startBlock = _startBlock;
27          endBlock = _endBlock;
28      }
```

ListaAirdrop.sol

Recommendation: Revise the `setEndBlock()` code logic as `require(_endBlock > startBlock, "Invalid end block");`

2.3 Informational

3. Lack of address validation [Informational]

`Status: Fixed`

Description:

The `ListaAirdrop::constructor()` function lacks zero address validation for the `_token` parameter.

```
20      constructor(address _token, bytes32 _merkleRoot, uint256 reclaimDelay,
    uint256 _startBlock, uint256 _endBlock) {
21          require(_startBlock >= block.number, "Invalid start block");
22          require(_endBlock > _startBlock, "Invalid end block");
23          token = _token;
24          merkleRoot = _merkleRoot;
25          reclaimPeriod = block.timestamp + reclaimDelay;
26          startBlock = _startBlock;
27          endBlock = _endBlock;
28      }
```

ListaAirdrop.sol

4. Lack of comments [Informational]

`Status: Fixed`

Description:

In the `ListaAirdrop::setStartBlock()` and `ListaAirdrop::setEndBlock()` functions, there is ambiguity in the comments. Also, throughout the codebase there are numerous functions missing or lacking documentation. This hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, comments improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned and the events emitted.

Recommendation: Consider thoroughly documenting all functions (and their parameters) that are part of the smart contracts' public interfaces. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing comments, consider following the Ethereum Natural Specification Format (NatSpec).

5. The potential denial-of-service [Informational]

`Status: Fixed`

Description:

In the `ListaAirdrop` contract, `merkleRoot` is a core variable that supports the operation of the contract. However, in addition to the `constructor()`, there is a `setMerkleRoot()` function that assigns a value to `merkleRoot`, and this extra room for maneuvering may lead to denial-of-service in the future.

Recommendation: Consider removing the function altogether.

Feedback: We didn't remove `setMerkleRoot`, instead we added a validation that disallow changes of merkle root once airdrop claim started.

3 Disclaimer

This security audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. This security audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues, also cannot make guarantees about any additional code added to the assessed project after the audit version. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contract(s). Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.