Smart Contract
Security Audit Report

Prepared for Listapie

Prepared by Supremacy

August 27, 2024

Contents

1 Introduction

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

1.1 About Client

Magpie XYZ is an ecosystem of DeFi protocols that provide yield and veTokenomics boosting services across multiple blockchain networks.

Listapie is an advanced SubDAO developed by Magpie to enhance the enduring viability of Lista DAO's CDP and liquid staking services. As a yield and veTokenomics service provider, Listapie's primary function is to lock LISTA tokens as veLISTA on Lista DAO. This action empowers Listapie to attain superior yields and increased voting power within Lista DAO, delivering exceptional opportunities for LSDfi participants.

Item	Description
Client	Magpiexyz
Project	Listapie
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/magpiexyz/listapie_contract/tree/rewarding/contracts
- Commit Hash: 5e1497e3e6196df66e62bc29e9986f4ba7285b90

1.3 Changelogs

Version	Date	Description
0.1	August 10, 2024	Initial Draft
1.0	August 27, 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.

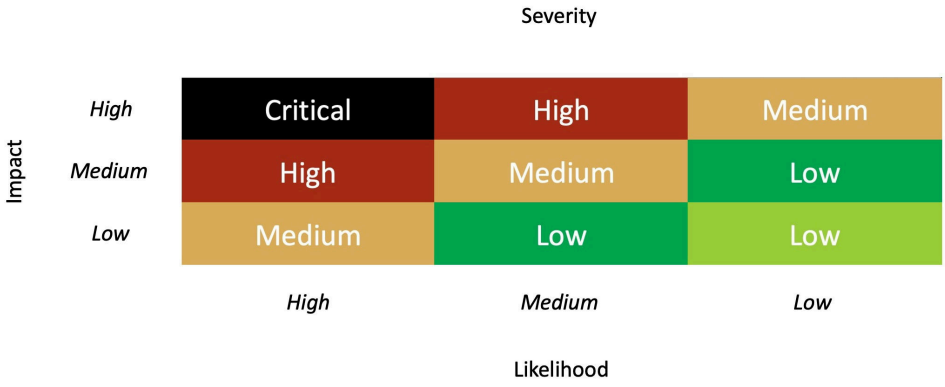
We are reachable at Twitter (https://twitter.com/SupremacyHQ), or Email (contact@supremacy.email).

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	Informational	Lack of comments	Fixed
2	Informational	Follow Check-Effects-Interactions Pattern	Fixed
3	Informational	Redundant code	Fixed

2.1 Informational

1. Lack of comments [Informational]

`Status: Fixed`

Description

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).

Feedback: Fixed at commit `89e6acd`.

2. Follow Check-Effects-Interactions Pattern [Informational]

`Status: Fixed`

Description

In the `listaRush::withdraw()`, the withdraw of `Lista` does not follow the Check-Effects-Interactions Pattern.

```
101     function withdraw(
102         uint256 _amount
103     ) external whenNotPaused nonReentrant {
104         if(_amount >   userDeposited[msg.sender]) revert
    withdrawAmountExceedDepositBalance();
105         if (_amount == 0) revert InvalidAmount();
106 
107         IERC20(Lista).safeTransfer(msg.sender, _amount);
108 
```

```
109             userDeposited[msg.sender] -= _amount;
110             totalDeposited -= _amount;
111
112             emit Withdraw(msg.sender, _amount);
113         }
```

listaRush.sol

Recommendation: Revise the code logic accordingly.

```
101         function withdraw(
102             uint256 _amount
103         ) external whenNotPaused nonReentrant {
104             if(_amount >   userDeposited[msg.sender]) revert
    withdrawAmountExceedDepositBalance();
105             if (_amount == 0) revert InvalidAmount();
106
107             userDeposited[msg.sender] -= _amount;
108             totalDeposited -= _amount;
109
110             IERC20(Lista).safeTransfer(msg.sender, _amount);
111
112             emit Withdraw(msg.sender, _amount);
113         }
```

listaRush.sol

Feedback: Fixed at commit `a1dd5e4`.

3. Redundant code [Informational]

`Status: Fixed`

Description

In the `StreamRewarder`, and `RewardDistributor` contract, `MasterListapieUpdated`, `RemoveLTPOrListaFee`, `SmartListaConvertUpdated`, `AddListaFees` and `SetListaFee` are unused events.

```
60      /* ============ Events ============ */
61
62      // Fee
63      event AddVeListaFees(address _to, uint256 _value, bool _isForVeLista, bool
    _isAddress);
64      event AddListaFees(address _to, uint256 _value, bool _isAddress);
65      event SetVeListaOrRevenueFee(address _to, uint256 _value, bool
    _isForVeLista);
66      event SetListaFee(address _to, uint256 _value);
67      event RemoveVeListaOrRevenueFee(uint256 value, address to, bool _isAddress,
    bool _isForVeLista);
68      event RemoveLTPOrListaFee(uint256 value, address to, bool _isAddress);
69
70      event RewardPaidTo(
71          address _to,
72          address _rewardToken,
73          uint256 _feeAmount
74      );
```

```
75
76      event VeRewardPaidTo(
77          address _to,
78          address _rewardToken,
79          uint256 _feeAmount
80      );
81
82      event RewardFeeDustTo(address _reward, address _to, uint256 _amount);
83
84      event SmartListaConvertUpdated(address _OldSmartListaConvert, address
    _smartListaConvert);
```

RewardDistributor.sol

Recommendation: Consider making the most of them or remove them.

Feedback: Fixed at commit `b3dc8f0`.

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.