AstraSec

Listapie

Security Audit Report

July 24, 2024

Contents

1 Introduction

1.1 About Listapie

Listapie is an advanced SubDAO developed by Magpie to enhance the enduring viability of Lista DAO's CDP and liquid staking services. The goal of launching Listapie is to leverage on the veTokenomics mechanism of Lista DAO. LISTA tokens will be gathered and locked into veLISTA via Listapie, which will be used to boost yields and increase governance authority on Lista DAO.

1.2 Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/listapie_contract/tree/rewarding
- Commit ID: daf4229

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/magpiexyz/listapie_contract/tree/rewarding
- Commit ID: 5e1497e

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `Listapie` project. Throughout this audit, we identified several issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	1	-	-	1
Medium	1	1	-	-
Low	2	1	-	1
Informational	1	-	-	1
Total	5	2	-	3

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~H-1~~ Possible rewardRate Manipulation in StreamRewarder::donateRewards()

M-2 Potential Risks Associated with Centralization

~~L-1~~ Potential Reentrancy Risk in MasterListapie

L-2 Potential Funds Locking Risk in RewardDistributor::sendVeListaRewards()

~~I-1~~ Improved Implementation Logic in MasterListapie::_multiClaim()

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Description
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

[H-1] Possible rewardRate Manipulation in StreamRewarder::donateRewards()

Target	Category	IMPACT	LIKELIHOOD	STATUS
StreamRewarder.sol	Business Logic	High	High	Addressed

The internal function `_provisionReward()` in the `StreamRewarder` contract is used to manage the distribution of rewards. Its logic is as follows: if the current time has surpassed the end of the reward distribution period (`rewardInfo.periodFinish`), a new reward rate is recalculated. If the reward distribution period has not yet ended, the remaining reward amount is calculated and added to the new reward amount, followed by recalculating the new reward rate.

When examining its implementation logic, we notice that the update for `rewardInfo.rewardPerTokenStored` (line 276) is incorrect. Specifically, `rewardInfo.rewardPerTokenStored` should be calculated using the `rewardRate` before it is updated, not after. Additionally, since this function can be executed by anyone by calling `donateRewards()`, a malicious user could manipulate the `rewardRate` by donating 1 wei of reward tokens to the contract. This could result in users who have staked assets in `Listapie` receiving less reward than expected during a specific period.

StreamRewarder::_provisionReward()

```
253 function _provisionReward(uint256 _rewards, address _rewardToken) internal {

255     _rewards = _rewards * DENOMINATOR; // to support small deciaml rewards

257     Reward storage rewardInfo = rewards[_rewardToken];

259     if (totalStaked() == 0) {
260         rewardInfo.queuedRewards = rewardInfo.queuedRewards + _rewards;
261         return ;
262     }

264     _rewards = _rewards + rewardInfo.queuedRewards;
265     rewardInfo.queuedRewards = 0;

267     if (block.timestamp >= rewardInfo.periodFinish) {
268         rewardInfo.rewardRate = _rewards / duration;
269     } else {
270         uint256 remaining = rewardInfo.periodFinish - block.timestamp;
271         uint256 leftover = remaining * rewardInfo.rewardRate;
272         _rewards = _rewards + leftover;
273         rewardInfo.rewardRate = _rewards / duration;
274     }
```

```
    rewardInfo.rewardPerTokenStored = rewardPerToken(_rewardToken);
    rewardInfo.lastUpdateTime = block.timestamp;
    rewardInfo.periodFinish = block.timestamp + duration;

}
```

Remediation Calculate the update for `rewardInfo.rewardPerTokenStored` using the old `rewardRate`, and restrict the access to the `donateRewards()` function to prevent unauthorized manipulation.

[M-1] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Medium	Medium	Acknowledged

In the `Listapie` protocol, the existence of the privileged `owner/minter` accounts introduces centralization risks, as it holds significant control and authority over critical operations governing the protocol. In the following, we show the representative functions potentially affected by the privileges associated with these privileged accounts.

Example Privileged Operations in `Listapie protocol`

```
function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
    if (!Address.isContract(msg.sender)) revert NonContractCaller();
    _mint(to, amount);
}

function setRewardDistributor(
  address _rewardDistributor
) external onlyOwner {
    if (_rewardDistributor == address(0)) revert InvalidAddress();
    rewardDistributor = IRewardDistributor(_rewardDistributor);
}

function createRewarder(
  address _receiptToken,
  address _rewardDistributor,
  uint256 _duration
) public onlyOwner returns (address) {
    address rewarder = ListapieUitilLib.createRewarder(
        address(this),
        _rewardDistributor,
        _receiptToken,
        _duration,
        streamRewarderBeacon
    );
```

```
    return rewarder;
}

function addListaFees(
    uint256 _value,
    address _to,
    bool _isMLISTA,
    bool _isAddress,
    bool _isVeListaFee
) external onlyOwner {
    if (_isVeListaFee && totalVeListaFee + _value > DENOMINATOR) revert
        ExceedsDenominator();
    if (!_isVeListaFee && totalRevenueShareFee + _value > DENOMINATOR) revert
        ExceedsDenominator();

    Fees[] storage targetFeeInfos = _isVeListaFee ? veListaFeeInfos :
        revenueShareFeeInfo;

    _addfee(targetFeeInfos, _value, _isMLISTA, _to, _isAddress);

    if (_isVeListaFee) {
        totalVeListaFee += _value;
    } else {
        totalRevenueShareFee += _value;
    }

    emit AddVeListaFees(_to, _value, _isVeListaFee, _isAddress);
}
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged accounts. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

[L-1] Potential Reentrancy Risk in MasterListapie

Target	Category	IMPACT	LIKELIHOOD	STATUS
MasterListapie.sol	Time and State	Low	Low	Addressed

The functions `depositMListaSVFor()` and `withdrawMListaSVFor()` in the `MasterListapie` contract are used to handle deposit and withdrawal requests for staking tokens from `mListaSV`. Upon reviewing their implementation, we notice that these functions lack reentrancy protection, which may introduce a potential reentrancy risk.

MasterListapie.sol

```
function depositMListaSVFor(uint256 _amount, address _for) external
    whenNotPaused _onlyMListaSV {
    _deposit(address(mListaSV), msg.sender, _for, _amount, true);
}

function withdrawMListaSVFor(uint256 _amount, address _for) external
    whenNotPaused _onlyMListaSV {
    _withdraw(address(mListaSV), _for, _amount, true);
}
```

Remediation From a security and code best practices perspective, it is recommended to add reentrancy protection mechanism to the above mentioned two functions.

[L-2] Potential Funds Locking Risk in RewardDistributor::sendVeListaRewards()

Target	Category	IMPACT	LIKELIHOOD	STATUS
RewardDistributor.sol	Business Logic	Low	Low	Acknowledged

The `sendVeListaRewards()` function in the `RewardDistributor` contract is used to distribute the reward token. Its distribution logic involves iterating through each active fee entry in the `veListaFeeInfos` array, calculating the reward amount for each fee, and deducting it from the total reward. If the reward amount is greater than 0, the `_distributeReward()` function is called to distribute the rewards.

Upon reviewing its implementation, we notice that if there is any remaining reward amount, i.e., `_leftRewardAmount` > 0, these remaining assets will be locked in the current contract and cannot be withdrawn.

RewardDistributor::sendVeListaRewards()

```
function sendVeListaRewards(
    address _rewardToken,
    uint256 _amount
) external nonReentrant _onlyRewardQeuer {
    IERC20(_rewardToken).safeTransferFrom(msg.sender, address(this), _amount);
    uint256 _leftRewardAmount = _amount;

    Fees[] memory feeInfo = veListaFeeInfos;

    for (uint256 i = 0; i < feeInfo.length; i++) {
        if (feeInfo[i].isActive) {
            address rewardToken = _rewardToken;
            uint256 feeAmount = (_amount * feeInfo[i].value) / DENOMINATOR;
            uint256 feeToSend = feeAmount;
```

```
                _leftRewardAmount -= feeToSend;
                if (feeToSend > 0){
                    _distributeReward(feeInfo[i], rewardToken, feeToSend, true);
                }
            }
        }

        // if (_leftRewardAmount > 0) {
        //     IERC20(_rewardToken).safeTransfer(owner(), _leftRewardAmount);

        //     emit RewardFeeDustTo(_rewardToken, owner(), _leftRewardAmount);
        // }
}
```

Remediation Add additional handling logic to transfer these remaining `_rewardToken` to the contract owner's address when `_leftRewardAmount` > 0.

[I-1] Improved Implementation Logic in MasterListapie::_multiClaim()

Target	Category	IMPACT	LIKELIHOOD	STATUS
MasterListapie.sol	Business Logic	N/A	N/A	Addressed

The internal helper function `_multiClaim()` in the `MasterListapie` contract is used to handle the logic for claiming rewards for multiple staking tokens. According to the current implementation, when the input parameter `_withLtp` is true and `_stakingToken` equals the address of `vlListapie`, the function throws an `InvalidToken()` error. This means that if the `vlListapie` token is included in the `_stakingTokens` input parameters for the functions `multiclaimSpecLtp()`, `multiclaimSpec()`, `multiclaimFor()`, or `multiclaim()`, the function execution will revert, which is unnecessary.

MasterListapie::_multiClaim()

```
function _multiClaim(
    address[] calldata _stakingTokens,
    address _user,
    address _receiver,
    address[][] memory _rewardTokens,
    bool _withLtp
) internal nonReentrant {
    uint256 length = _stakingTokens.length;
    if (length != _rewardTokens.length) revert LengthMismatch();

    uint256 defaultPoolAmount;

    for (uint256 i = 0; i < length; ++i) {
```

```
        address _stakingToken = _stakingTokens[i];
        UserInfo storage user = userInfo[_stakingToken][_user];

        updatePool(_stakingToken);
        uint256 claimableListapie = _calNewListapie(_stakingToken, _user) + user
            .unClaimedListapie;

        if (_withLtp) {
            if (_stakingToken == address(vlListapie)) {
                revert InvalidToken();
            } else {
                defaultPoolAmount += claimableListapie;
            }
            user.unClaimedListapie = 0;
        } else {
            user.unClaimedListapie = claimableListapie;
        }

        user.rewardDebt =
            (user.amount * tokenToPoolInfo[_stakingToken].accListapiePerShare) /
            1e12;

        _claimBaseRewarder(_stakingToken, _user, _receiver, _rewardTokens[i]);
    }

    if (!_withLtp) return;

    _sendListapie(_user, _receiver, defaultPoolAmount);
}
```

Remediation When the input parameter `_withLtp` is true and `_stakingToken` equals the address of `vlListapie`, use the continue statement in the for loop instead of throwing an `InvalidToken()` error.

4 Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI