AstraSec

Listapie

Security Audit Report

February 12, 2026

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

First 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

Second Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/listapie_contract/pull/22

- Commit ID: dfc406e

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

- https://github.com/magpiexyz/listapie_contract/pull/22

- Commit ID: b1d87a1

Third Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/listapie_contract/pull/20

- Commit ID: 73e33e8

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

- https://github.com/magpiexyz/listapie_contract

- Commit ID: 9bd8415

Fourth Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/listapie_contract/pull/24

- Commit ID: b7f97fc

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

- https://github.com/magpiexyz/listapie_contract

- Commit ID: 50361da

Fifth Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/listapie_contract/pull/39

- Commit ID: 257029b

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

- https://github.com/magpiexyz/listapie_contract

- Commit ID: 8904fac

Sixth Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/listapie_contract/pull/55

- Commit ID: 357da7e

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

- https://github.com/magpiexyz/listapie_contract

- Commit ID: 3f7bede

Seventh Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/listapie_contract/pull/65

- Commit ID: c24235a

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

- https://github.com/magpiexyz/listapie_contract

- Commit ID: 8335127

Eighth Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/listapie_contract/pull/66

- Commit ID: 21e211e

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

- https://github.com/magpiexyz/listapie_contract

- Commit ID: 5795f23

1.3   Revision History

Version	Date
First Audit	July 24, 2024
Second Audit	November 4, 2024
Third Audit	November 20, 2024
Fourth Audit	January 6, 2025
Fifth Audit	January 15, 2025
Sixth Audit	August 3, 2025
Seventh Audit	January 15, 2026
Eighth Audit	February 12, 2026

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	4	1	-	3
Informational	1	-	-	1
Undetermined	-	-	-	-

3 | Vulnerability Summary

3.1 Overview

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

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      }
```

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

280  }
```

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`

```
41  function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
42      if (!Address.isContract(msg.sender)) revert NonContractCaller();
43      _mint(to, amount);
44  }

46  function setRewardDistributor(
47     address _rewardDistributor
48  ) external onlyOwner {
49      if (_rewardDistributor == address(0)) revert InvalidAddress();
50      rewardDistributor = IRewardDistributor(_rewardDistributor);
51  }

53  function createRewarder(
54     address _receiptToken,
55     address _rewardDistributor,
56     uint256 _duration
57  ) public onlyOwner returns (address) {
58      address rewarder = ListapieUitilLib.createRewarder(
59          address(this),
60          _rewardDistributor,
61          _receiptToken,
62          _duration,
63          streamRewarderBeacon
64      );
```

```
66        return rewarder;
67  }

69  function addListaFees(
70        uint256 _value,
71        address _to,
72        bool _isMLISTA,
73        bool _isAddress,
74        bool _isVeListaFee
75  ) external onlyOwner {
76        if (_isVeListaFee && totalVeListaFee + _value > DENOMINATOR) revert
               ExceedsDenominator();
77        if (!_isVeListaFee && totalRevenueShareFee + _value > DENOMINATOR) revert
               ExceedsDenominator();

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

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

83        if (_isVeListaFee) {
84            totalVeListaFee += _value;
85        } else {
86            totalRevenueShareFee += _value;
87        }

89        emit AddVeListaFees(_to, _value, _isVeListaFee, _isAddress);
90  }
```

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

```
336  function depositMListaSVFor(uint256 _amount, address _for) external
         whenNotPaused _onlyMListaSV {
337      _deposit(address(mListaSV), msg.sender, _for, _amount, true);
338  }

340  function withdrawMListaSVFor(uint256 _amount, address _for) external
         whenNotPaused _onlyMListaSV {
341      _withdraw(address(mListaSV), _for, _amount, true);
342  }
```

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()

```
108  function sendVeListaRewards(
109      address _rewardToken,
110      uint256 _amount
111  ) external nonReentrant _onlyRewardQeuer {
112      IERC20(_rewardToken).safeTransferFrom(msg.sender, address(this), _amount);
113      uint256 _leftRewardAmount = _amount;

115      Fees[] memory feeInfo = veListaFeeInfos;

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

```
122             _leftRewardAmount -= feeToSend;
123             if (feeToSend > 0){
124                 _distributeReward(feeInfo[i], rewardToken, feeToSend, true);
125             }
126         }
127     }

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

132     //     emit RewardFeeDustTo(_rewardToken, owner(), _leftRewardAmount);
133     // }
134 }
```

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

[L-3] Revisited Logic in V2LiquidityPoolHelper::withdrawAndClaim()

Target	Category	IMPACT	LIKELIHOOD	STATUS
V2LiquidityPoolHelper.sol	Business Logic	Low	Low	🔗Addressed

The `V2LiquidityPoolHelper` contract is designed to facilitate user interactions such as depositing, withdrawing, and claiming rewards. When reviewing the implementation of the `withdrawAndClaim()` function, we notice that the current logic needs to be revisited. Specifically, the `withdrawAndClaim()` function has a potential issue related to the sequence of operations: `receiptToken` is burned (line 130) before the harvest operation is executed (line 135). This could lead to a situation where a user's `receiptTokens` are burned without ensuring the harvest logic is executed properly, resulting in the user not receiving the expected harvest rewards.

V2LiquidityPoolHelper::withdrawAndClaim()

```
127  function withdrawAndClaim(address _pool, uint256 _amount, bool _isClaim)
         external nonReentrant whenNotPaused {
128      Pool memory poolInfo = pools[_pool];
129      bool _harvest = false;
130      IMintableERC20(poolInfo.receiptToken).burn(msg.sender, _amount);
131      if (poolInfo.lastHarvestTime + harvestTimeGap < block.timestamp) {
132          _harvest = true;
133          pools[_pool].lastHarvestTime = block.timestamp;
134      }
135      IListaStaking(listaStaking).withdrawV2LPFor(msg.sender, _pool, _amount,
             _harvest);
136      if (_isClaim) _claimRewards(msg.sender, poolInfo.depositToken, _pool);
```

```
138        emit NewWithdraw(msg.sender, _pool, _amount);
139   }
```

Remediation   Conduct the burn operation for the user's `receiptToken` after executing the `withdrawV2LPFor()` function.

[L-4] Improved Implementation Logic in ListapieBribeManager

Target	Category	IMPACT	LIKELIHOOD	STATUS
ListapieBribeManager.sol	Business Logic	Low	Low	🔗Addressed

The `ListapieBribeManager` contract introduces a new feature which allows anyone to add bribes for a specific pool and epoch. At the end of the epoch, these bribes will be transferred to the corresponding rewarder contracts of the pool via the `batchHarvestVLLTPBribes()` function, and users can claim them afterward. These bribes contribute to an increase in the APR for the pools, motivating users to vote for the corresponding pools by boosting the APR, as they will receive additional rewards for their votes. While reviewing the code implementation of this new feature, we notice that the current implementation needs to be improved.

In the following, we use the `addBribeERC20()` routine as an example. This function allows a user to add a bribe in the form of an ERC20 token to a specific distributor pool over a set number of epochs. However, there is an issue with the current code when checking the validity of `_startingEpoch` (line 308). Specifically, if `_startingEpoch < getCurrentEpoch()` or if `_startingEpoch == getCurrentEpoch()` but the current week has already entered the admin voting period, the execution of this function should revert, rather than continuing execution. Note that other routines, such as `batchHarvestVLLTPBribes()`, `moveBribe()`, `withdrawAllBribe()`, share the similar issue.

ListapieBribeManager::addBribeERC20()

```
290   function addBribeERC20(
291       uint16 _distributorId,
292       address _token,
293       uint256 _amount,
294       uint256 _startingEpoch,
295       uint256 _numOfEpochs
296   )
297       external
298       nonReentrant
299   {
300       if (_distributorId > listaVault.distributorId()) revert OutOfPoolIndex();
```

```
302        if (!listaVote.activeDistributors(_distributorId)) revert PoolNotActive();

304        if (poolInfos[_distributorId].rewarder == address(0)) revert InvalidPool();

306        if (!allowedReward[_token]) revert InvalidBribeToken();

308        if (_startingEpoch == getCurrentEpoch() && !isVotable()) {
309            revert EpochEnded();
310        }

312        if (_numOfEpochs == 0) revert InvalidEpoch();

314        IERC20(_token).safeTransferFrom(msg.sender, address(this), _amount);
315        // Calculate the base bribe per epoch and the remainder
316        uint256 baseBribe = _amount / _numOfEpochs;
317        uint256 remainder = _amount % _numOfEpochs;
318        // Distribute the bribe amount across epochs
319        for (uint256 epoch = _startingEpoch; epoch < _startingEpoch + _numOfEpochs;
                 epoch++) {
320            uint256 bribeForThisEpoch = baseBribe;
321            // Add the remainder to the first epoch
322            if (epoch == _startingEpoch) {
323                bribeForThisEpoch += remainder;
324            }
325            bribes[epoch][_distributorId][_token] += bribeForThisEpoch;
326        }

328        emit BribeAdded(_distributorId, _token, _amount, _startingEpoch,
               _numOfEpochs);
329   }
```

Remediation   Perform proper validity checks on the `epoch` used in the function mentioned above.

[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()

```
560  function _multiClaim(
561      address[] calldata _stakingTokens,
562      address _user,
563      address _receiver,
564      address[][] memory _rewardTokens,
565      bool _withLtp
566  ) internal nonReentrant {
567      uint256 length = _stakingTokens.length;
568      if (length != _rewardTokens.length) revert LengthMismatch();

570      uint256 defaultPoolAmount;

572      for (uint256 i = 0; i < length; ++i) {
573          address _stakingToken = _stakingTokens[i];
574          UserInfo storage user = userInfo[_stakingToken][_user];

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

579          if (_withLtp) {
580              if (_stakingToken == address(vlListapie)) {
581                  revert InvalidToken();
582              } else {
583                  defaultPoolAmount += claimableListapie;
584              }
585              user.unClaimedListapie = 0;
586          } else {
587              user.unClaimedListapie = claimableListapie;
588          }

590          user.rewardDebt =
591              (user.amount * tokenToPoolInfo[_stakingToken].accListapiePerShare) /
592              1e12;

594          _claimBaseRewarder(_stakingToken, _user, _receiver, _rewardTokens[i]);
595      }

597      if (!_withLtp) return;

599      _sendListapie(_user, _receiver, defaultPoolAmount);
600  }
```

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.

17

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

18