SUPREMACY

Smart Contract Security Audit Report

Prepared for Listapie

Prepared by Supremacy

January 22, 2025

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:

Version	Repository	Commit Hash
1	Listapie	`5e1497e3e6196df66e62bc29e9986f4ba7285b90`
2	Listapie	`73e33e8848ac6b18fc5ba5408a464a551c124528`
3	Listapie	`0cf32c1812a02b7d134d19197ce1f076476b8b0b`
4	Listapie	`50361da164ffac35541b2d0e175a8e18d570cfe7`
5	Listapie	`048191865ff82390d91e817eed6587cb7c93bad8`

1.3 Changelogs

Version	Date	Description
0.1	August 10, 2024	Initial Draft
1.0	August 27, 2024	Final Release
1.1	December 05, 2024	Post-Final Release #1
1.2	December 13, 2024	Post-Final Release #2
1.3	December 17, 2024	Post-Final Release #3
1.4	January 03, 2025	Post-Final Release #4
1.5	January 22, 2025	Post-Final Release #5

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.

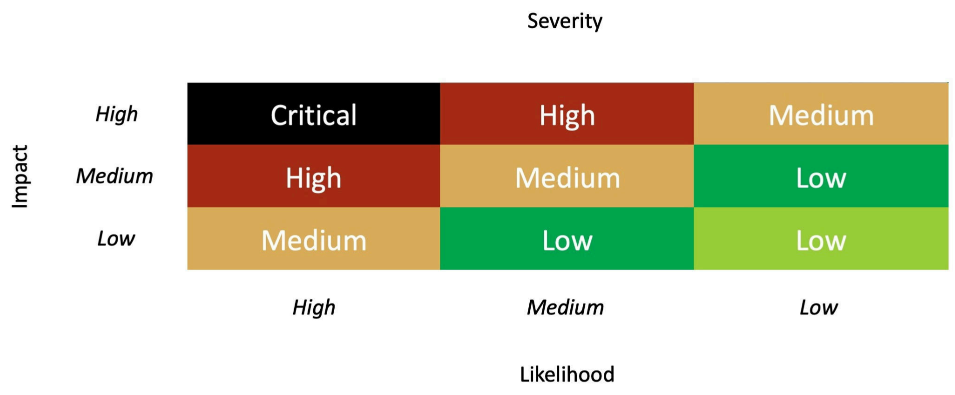
We are reachable at X (https://x.com/SupremacyHQ), or Email (contact@supremacy.email).

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.



Severity

Impact \ Likelihood	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

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	High	Permanent freezing of rewards	Fixed
2	Medium	Lack of `onlyActivePool()` check	Fixed
3	Medium	Missing pool deactivation mechanism	Fixed
4	Low	Lack of input validation in `registerPool()`	Fixed
5	Informational	Lack of address validation	Fixed
6	Informational	Lack of `__gap[50]` storage variable	Fixed
7	Informational	Hardcoded harvest time gap limit	Fixed
8	Informational	Lack of comments	Fixed
9	Informational	Follow Check-Effects-Interactions Pattern	Fixed
10	Informational	Redundant code	Fixed

2.1 High

1. Permanent freezing of rewards [High]

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

`Status: Fixed`

Description

The `batchHarvest()` function in the `ListapieBribeManager` contract external call `ListaStaking::batchClaimVotingRewards()` to claim rewards. The `batchClaimVotingRewards()` function gets the actual rewards via `uint256 rewardDifference = IERC20(rewards[j]).balanceOf(address(this)) - balanceBeforeClaim[j]` before and after `VotingIncentive::batchClaim()` claim the rewards. In the following, we show the `VotingIncentive::claim()` code that causes the incompatibility issue. This will allow malicious actor to call `getReward()` on behalf of `ListaStaking`, and basically prevent them to get the rewards.

```
    function batchHarvest(IVotingIncentive.ClaimParams[] memory claimParams) external nonReentrant onlyAllowedOperator {
        address[] memory rewarders = new address[](claimParams.length);

        for (uint256 i = 0; i < claimParams.length; i++) {
            uint16 distributorId = claimParams[i].distributorId;
            rewarders[i] = poolInfos[distributorId].rewarder;
        }

        listaStaking.batchClaimVotingRewards(claimParams, rewarders);
    }
```

ListapieBribeManager.sol

```
    function batchClaimVotingRewards(
        IVotingIncentive.ClaimParams[] memory _claimParams,
        address[] memory _rewarders
    )
        external
        nonReentrant
        whenNotPaused
        onlyBribeManager
    {

        for (uint256 i = 0; i < _claimParams.length; i++) {
            address[] memory rewards = _claimParams[i].assets;
            uint256[] memory balanceBeforeClaim = new uint256[](rewards.length);

            for (uint256 j = 0; j < rewards.length; j++) {
                if (rewards[j] == address(0)) {
                    balanceBeforeClaim[j] = address(this).balance;
                    rewards[j] = wBNB;
                } else {
                    balanceBeforeClaim[j] = IERC20(rewards[j]).balanceOf(address(this));
                }
            }

            IVotingIncentive.ClaimParams[] memory _toListaDAO = new IVotingIncentive.ClaimParams[](1);
            _toListaDAO[0] = _claimParams[i];
            IVotingIncentive(votingIncentive).batchClaim(_toListaDAO);

            for (uint256 j = 0; j < rewards.length; j++) {
                uint256 rewardDifference = IERC20(rewards[j]).balanceOf(address(this)) - balanceBeforeClaim[j];
                if (rewardDifference > 0) {
                    IERC20(rewards[j]).forceApprove(_rewarders[i], rewardDifference);
                    IBaseRewardPool(_rewarders[i]).queueNewRewards(rewardDifference, rewards[j]);

                    emit RewardProcessed(rewards[j], rewardDifference, _rewarders[i]);
                }
            }

            emit VotingRewardsClaimed(_claimParams[i].distributorId, _claimParams[i].week, rewards);
        }

    }
```

ListaStaking.sol

```
  /**
   * @dev Claim all incentives for a distributor for a week
   * @param _input array of ClaimParams to claim
   */
  function batchClaim(ClaimParams[] memory _input) external {
    address user = msg.sender;
    for (uint256 i = 0; i < _input.length; ++i) {
      ClaimParams memory _params = _input[i];
      address[] memory _assets = _params.assets;
      for (uint256 j = 0; j < _assets.length; ++j) {
        if (claimedIncentives[user][_params.distributorId][_params.week][_assets[j]]) continue;
        claim(user, _params.distributorId, _params.week, _assets[j]);
      }
    }
  }

  /**
   * @dev Claim incentives for a distributor for a week
   * @param _user address of the user
   * @param _distributorId id of the distributor
   * @param _week week number
   * @param _asset address of the asset
   */
  function claim(address _user, uint16 _distributorId, uint16 _week, address _asset) public nonReentrant whenNotPaused {
    require(_user != adminVoter, "Invalid voter");
    require(_week <= vault.getWeek(block.timestamp), "Invalid week");
    require(_distributorId > 0 && _distributorId <= vault.distributorId(), "Invalid distributorId");
    require(!claimedIncentives[_user][_distributorId][_week][_asset], "Already claimed");
    require(weeklyIncentives[_distributorId][_week][_asset] > 0, "No incentives");

    uint256 adminWeight = getRawWeight(adminVoter, _distributorId, _week);
    uint256 amountToClaim = calculateAmount(_user, _distributorId, _week, _asset, adminWeight);

    claimedIncentives[_user][_distributorId][_week][_asset] = true;
    if (_asset == address(0)) {
      (bool success, ) = payable(_user).call{ value: amountToClaim }("");
      require(success, "Transfer failed");
    } else {
      IERC20(_asset).safeTransfer(_user, amountToClaim);
    }

    emit IncentiveClaimed(_user, _distributorId, _week, _asset, amountToClaim);
  }
```

VotingIncentive.sol

Recommendation

Consider adding the ability to sweep tokens in the `batchClaimVotingRewards` function.

Feedback: Fixed in `7823b65`.

2.2 Medium

2. Lack of `onlyActivePool()` check [Medium]

```
Severity: Medium                  Likelihood: Medium                  Impact: Medium
Status: Fixed
```

Description

The `withdrawAndClaim()` function in the `V2LiquidityPoolPoolHelper` contract does not perform a pool activation check because it omits the `onlyActivePool` modifier. This is inconsistent with the `deposit` function, which requires the pool to be active. As a result, there may be unexpected effects.

```
    /* ============ External Functions ============ */
    function deposit(address _pool, uint256 _amount) external nonReentrant whenNotPaused onlyActivePool(_pool) {
        Pool memory poolInfo = pools[_pool];
        bool _harvest = false;
        if (poolInfo.lastHarvestTime + harvestTimeGap < block.timestamp) {
            _harvest = true;
            pools[_pool].lastHarvestTime = block.timestamp;
        }
        IListaStaking(listaStaking).depositV2LPFor(msg.sender, _pool, _amount, _harvest);
        //   mintable token
        IMintableERC20(poolInfo.receiptToken).mint(msg.sender, _amount);

        emit NewDeposit(msg.sender, _pool, _amount);
    }

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

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

V2LiquidityPoolHelper.sol

Recommendation

Consider adding `onlyActivePool(_pool)` check.

Feedback: Fixed in `3b36cfb`.

3. Missing pool deactivation mechanism [Medium]

```
Severity: Medium                Likelihood: Medium                Impact: Medium
Status: Fixed
```

Description

The contract allows pool registration but provides no method to deactivate pools. This restricts the owner's ability to manage and enforce access control over pools that may need to be closed due to security issues, liquidity concerns, or protocol updates. If a pool becomes compromised or deprecated, users can continue interacting with it, leading to potential loss of funds or exploit risks.

```
    /* ============ Admin Functions ============ */
    function registerPool(
        address _poolAddress, //  V2 pool
        uint256 _allocPoints, // Allocation points for V2 pools
        address _depositToken, // For V2, it's LP token
        uint256 _poolType, //
        string memory _name,
        string memory _symbol
    )
        external
        onlyOwner
    {
        if (pools[_poolAddress].isActive) revert PoolOccupied();
        if (_poolType != V2Type) {
            revert InvalidPoolType();
        }
        IERC20 newToken;
        address rewarder;
        newToken =
            IERC20(ListapieUitilLib.createReceipt(_poolAddress,
address(masterListapie), address(this), _name, _symbol));
```

```
        rewarder = masterListapie.createRewarder(address(newToken), address(rewardDistributor), 7 days);
        masterListapie.add(_allocPoints, _depositToken, address(newToken), address(rewarder));
        pools[_poolAddress] = Pool({
            poolAddress: _poolAddress,
            depositToken: _depositToken,
            rewarder: address(rewarder),
            receiptToken: address(newToken),
            lastHarvestTime: block.timestamp,
            poolType: _poolType,
            isActive: true
        });
        poolList.push(_poolAddress);

        emit PoolAdded(_poolAddress, address(rewarder), address(newToken));
    }
```

V2LiquidityPoolHelper.sol

Recommendation

Introduce a function that allows the owner to deactivate a pool by updating its `isActive` status. This ensures that the owner can dynamically manage the lifecycle of pools.

Feedback: Fixed in `c9ba6c2`.

2.3 Low

4. Lack of input validation in `registerPool()` [Low]

```
Severity: Low                    Likelihood: Low                    Impact: Low
Status: Fixed
```

Description

The `registerPool()` function does not validate critical input parameters such as `_poolAddress`, and `_depositToken`. This can lead to incorrect or malicious configurations.

```
    /* ============ Admin Functions ============ */
    function registerPool(
        address _poolAddress, //  V2 pool
        uint256 _allocPoints, // Allocation points for V2 pools
        address _depositToken, // For V2, it's LP token
        uint256 _poolType, //
        string memory _name,
        string memory _symbol
    )
        external
        onlyOwner
    {
        if (pools[_poolAddress].isActive) revert PoolOccupied();
        if (_poolType != V2Type) {
            revert InvalidPoolType();
        }
        IERC20 newToken;
        address rewarder;
        newToken =
            IERC20(ListapieUitilLib.createReceipt(_poolAddress,
address(masterListapie), address(this), _name, _symbol));
        rewarder = masterListapie.createRewarder(address(newToken),
address(rewardDistributor), 7 days);
        masterListapie.add(_allocPoints, _depositToken, address(newToken),
address(rewarder));
        pools[_poolAddress] = Pool({
            poolAddress: _poolAddress,
            depositToken: _depositToken,
            rewarder: address(rewarder),
            receiptToken: address(newToken),
            lastHarvestTime: block.timestamp,
            poolType: _poolType,
            isActive: true
        });
        poolList.push(_poolAddress);

        emit PoolAdded(_poolAddress, address(rewarder), address(newToken));
    }
```

V2LiquidityPoolHelper.sol

Recommendation

Consider adding zero address validation and `isContract()` validation.

Feedback: Fixed in `a7899c3`.

2.4 Informational

5. Lack of address validation [Informational]

`Status: Fixed`

Description

In the `VLListapie::__vlListapie_init_()`, `ListapieBribeManager::setFeeCollector()`, and `V2LiquidityPoolHelper::updateUsdtPoolOutputTokens()` function, lack of checking of zero address by multiple address parameters.

```
    function __vlListapie_init_(
        address _masterListapie,
        uint256 _maxSlots,
        address _listapie,
        uint256 _coolDownInSecs
    )
        public
        initializer
    {
        __Ownable_init();
        __Pausable_init();
        __ReentrancyGuard_init();
        __ERC20_init("Vote Locked ListaPie", "vlListapie");
        if (_maxSlots == 0) {
            revert MaxSlotShouldNotZero();
        }
        maxSlot = _maxSlots;
        masterListapie = _masterListapie;
        listapie = IERC20(_listapie);
        coolDownInSecs = _coolDownInSecs;
    }
```

VLListapie.sol

Recommendation

Consider adding zero address validation.

Feedback: Fixed in `c09da12` and `4619ed2`.

6. Lack of `__gap[50]` storage variable [Informational]

`Status: Fixed`

Description

To allow for new storage variables in future upgrades of `VLListapie` contract, consider adding the `__gap[50]` variable.

```
    function setHarvestTimeGap(uint256 _period) external onlyOwner {
        if (_period > 7 days) revert TimeGapTooMuch();
        harvestTimeGap = _period;
        emit HarvestTimeGapUpdated(_period);
    }
```

V2LiquidityPoolHelper.sol

Recommendation

It is considered a recommand practice for upgradeable contracts is to include a storage variable called `__gap`. This `__gap` storage variable will be used as a reserve for future upgrades. It allows new storage variables to be freely added in the future without affecting storage compatibility with existing deployments. The size of the `__gap` array is typically calculated such that the storage used by the contract always adds up to the same amount (usually 50 storage slots).

Feedback: Fixed in `29851c2`.

7. Hardcoded harvest time gap limit [Informational]

`Status: Fixed`

Description

The `setHarvestTimeGap()` function limits the maximum harvest interval to 7 days. This hardcoded value may not be flexible enough for future protocol updates.

```
    function setHarvestTimeGap(uint256 _period) external onlyOwner {
        if (_period > 7 days) revert TimeGapTooMuch();
        harvestTimeGap = _period;
        emit HarvestTimeGapUpdated(_period);
    }
```

V2LiquidityPoolHelper.sol

Recommendation

Consider adjusting time units to expectations.

Feedback: Fixed in `8efd5b5`.

8. 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 in `89e6acd`.

9. 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.

```
    function withdraw(
        uint256 _amount
    ) external whenNotPaused nonReentrant {
        if(_amount >   userDeposited[msg.sender]) revert
withdrawAmountExceedDepositBalance();
        if (_amount == 0) revert InvalidAmount();

        IERC20(Lista).safeTransfer(msg.sender, _amount);

        userDeposited[msg.sender] -= _amount;
        totalDeposited -= _amount;

        emit Withdraw(msg.sender, _amount);
    }
```

listaRush.sol

Recommendation: Revise the code logic accordingly.

```
    function withdraw(
        uint256 _amount
    ) external whenNotPaused nonReentrant {
        if(_amount >   userDeposited[msg.sender]) revert
withdrawAmountExceedDepositBalance();
        if (_amount == 0) revert InvalidAmount();

        userDeposited[msg.sender] -= _amount;
        totalDeposited -= _amount;

        IERC20(Lista).safeTransfer(msg.sender, _amount);

        emit Withdraw(msg.sender, _amount);
    }
```

listaRush.sol

Feedback: Fixed in `a1dd5e4`.

10. Redundant code [Informational]

Status: Fixed

Description

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

```
    /* ============ Events ============ */

    // Fee
    event AddVeListaFees(address _to, uint256 _value, bool _isForVeLista, bool
_isAddress);
    event AddListaFees(address _to, uint256 _value, bool _isAddress);
    event SetVeListaOrRevenueFee(address _to, uint256 _value, bool
_isForVeLista);
    event SetListaFee(address _to, uint256 _value);
    event RemoveVeListaOrRevenueFee(uint256 value, address to, bool _isAddress,
bool _isForVeLista);
    event RemoveLTPOrListaFee(uint256 value, address to, bool _isAddress);

    event RewardPaidTo(
        address _to,
        address _rewardToken,
        uint256 _feeAmount
    );

    event VeRewardPaidTo(
        address _to,
        address _rewardToken,
        uint256 _feeAmount
    );

    event RewardFeeDustTo(address _reward, address _to, uint256 _amount);

    event SmartListaConvertUpdated(address _OldSmartListaConvert, address
_smartListaConvert);
```

RewardDistributor.sol

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

Feedback: Fixed in `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.