



Security Audit

Report for Lista Token

Date: October 16, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction	1
1.1 About Target Contracts	1
1.2 Disclaimer	1
1.3 Procedure of Auditing	2
1.3.1 Software Security	2
1.3.2 DeFi Security	2
1.3.3 NFT Security	3
1.3.4 Additional Recommendation	3
1.4 Security Model	3
Chapter 2 Findings	5
2.1 DeFi Security	5
2.1.1 Lack of check on <code>_adminVotePeriod</code> in function <code>initialize()</code>	5
2.2 Additional Recommendation	6
2.2.1 Redundant code	6
2.2.2 Incorrect comment in function <code>_vote()</code>	9
2.2.3 Lack of check in function <code>adminTransfer()</code>	11
2.3 Notes	11
2.3.1 Potential centralization risk	11
2.3.2 Voting capability of admin without <code>veLista</code> balance verification	12
2.3.3 Blocking use of voting results due to set of distributor percentages	12
2.3.4 Admin can skip user voting stage or admin voting stage	13
2.3.5 Potential risk of voting results manipulation by abusing flashloan	14

Report Manifest

Item	Description
Client	Lista
Target	Lista Token

Version History

Version	Date	Description
1.0	October 16, 2024	First release

Signature

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at [Email](#), [Twitter](#) and [Medium](#).

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repository of Lista Token¹ of Lista. Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include `lista-token` folder contract only. Specifically, the files covered in this audit include:

```
1 ClisBNBLaunchPoolDistributor.sol
2 EmissionVoting.sol
3 ListaVault.sol
```

Listing 1.1: Audit Scope for this Report

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version ([Version 1](#)), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
Lista Token	Version 1	0fc2a7c1db08d5ad85cab1688dcb48557b24a134
	Version 2	990d359bc72dda9c6fa73ae5ddd86635cb6977fe

1.2 Disclaimer

This 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 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. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

¹<https://github.com/lista-dao/lista-token/tree/voting-and-clisbnb-launchpool-audit-1.0>

The scope of this audit is limited to the code mentioned in Section 1.1. 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.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

- * Reentrancy
- * DoS
- * Access control
- * Data handling and data flow
- * Exception handling
- * Untrusted external call and control flow
- * Initialization consistency
- * Events operation
- * Error-prone randomness
- * Improper use of the proxy system

1.3.2 DeFi Security

- * Semantic consistency
- * Functionality consistency
- * Permission management
- * Business logic
- * Token operation
- * Emergency mechanism
- * Oracle security
- * Whitelist and blacklist
- * Economic impact
- * Batch transfer

1.3.3 NFT Security

- * Duplicated item
- * Verification of the token receiver
- * Off-chain metadata security

1.3.4 Additional Recommendation

- * Gas optimization
- * Code quality and style



Note The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology ² and Common Weakness Enumeration ³. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact	<i>High</i>	High	Medium
	<i>Low</i>	Medium	Low
		<i>High</i>	<i>Low</i>
		Likelihood	

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.

²https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

³<https://cwe.mitre.org/>

- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **one** potential security issue. Besides, we have **three** recommendations and **five** notes.

- Low Risk: 1
- Recommendation: 3
- Note: 5

ID	Severity	Description	Category	Status
1	Low	Lack of check on <code>_adminVotePeriod</code> in function <code>initialize()</code>	DeFi Security	Fixed
2	-	Redundant code	Recommendation	Confirmed
3	-	Incorrect comment in function <code>_vote()</code>	Recommendation	Fixed
4	-	Lack of check in function <code>adminTransfer()</code>	Recommendation	Confirmed
5	-	Potential centralization risk	Note	-
6	-	Voting capability of admin without <code>veLista</code> balance verification	Note	-
7	-	Blocking use of voting results due to set of distributor percentages	Note	-
8	-	Admin can skip user voting stage or admin voting stage	Note	-
9	-	Potential risk of voting results manipulation by abusing flashloan	Note	-

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Lack of check on `_adminVotePeriod` in function `initialize()`

Severity Low

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In the [EmissionVoting.sol](#) contract, the function `initialize()` will set up and configure the key parameters. However, the function does not check the `ADMIN_VOTE_PERIOD`. In this case, if `ADMIN_VOTE_PERIOD` is not within the specified range, admins or users cannot vote in expected time.

```
73  function initialize(  
74      address _admin,  
75      address _adminVoter,  
76      address _veLista,  
77      address _vault,  
78      uint256 _adminVotePeriod  
79  ) public initializer {  
80      require(_admin != address(0), "admin is a zero address");
```



```
81     require(_adminVoter != address(0), "adminVoter is a zero address");
82     require(_veLista != address(0), "veLista is a zero address");
83     require(_vault != address(0), "vault is a zero address");
84
85
86     __AccessControl_init();
87
88
89     _setupRole(DEFAULT_ADMIN_ROLE, _admin);
90     _setupRole(ADMIN_VOTER, _adminVoter);
91     veLista = IVeLista(_veLista);
92     vault = IVault(_vault);
93     ADMIN_VOTE_PERIOD = _adminVotePeriod;
94 }
```

Listing 2.1: EmissionVoting.sol

Impact If `ADMIN_VOTE_PERIOD` is set incorrectly, it may prevent either users or admins from being able to vote.

Suggestion Add check to ensure `ADMIN_VOTE_PERIOD` falls within the specified range.

2.2 Additional Recommendation

2.2.1 Redundant code

Status Confirmed

Introduced by Version 1

Description In the function `revokeEpoch()`, there is no need to check `epoch.totalAmount` as `setEpochMerkleRoot()` already ensures that `epoch.totalAmount` is greater than 0. Similarly, in `collectUnclaimed()`, checking `epoch.totalAmount` is unnecessary, this is because `epoch.totalAmount` does not change, and verifying `epoch.unclaimedAmount` is greater than 0 is sufficient. Additionally, in the function `_vote()`, checking `weight` is redundant because the design allows elements in the `weights` array to be zero, and the `uint256` type ensures that elements are non-negative.

```
113     function setEpochMerkleRoot(uint64 _epochId, bytes32 _merkleRoot, address _token, uint256
        _startTime, uint256 _endTime, uint256 _totalAmount)
114     external onlyRole(DEFAULT_ADMIN_ROLE)
115     {
116         require(_epochId == nextEpochId, "Invalid epochId");
117         require(_merkleRoot != bytes32(0), "Invalid merkle root");
118         require(_startTime > block.timestamp, "Invalid start time");
119         require(_endTime > _startTime, "Invalid end time");
120         require(_totalAmount > 0, "Invalid total amount");
121
122
123         uint64 currentEpochId = nextEpochId++;
124         Epoch storage epoch = epochs[currentEpochId];
125         epoch.merkleRoot = _merkleRoot;
```

```
126     epoch.token = _token;
127     epoch.startTime = _startTime;
128     epoch.endTime = _endTime;
129     epoch.totalAmount = _totalAmount;
130     epoch.unclaimedAmount = _totalAmount;
131     totalUnclaimedAmount[_token] += _totalAmount;
132
133
134     emit UpdateEpoch(currentEpochId, epoch.merkleRoot, epoch.token, epoch.startTime, epoch.
        endTime, epoch.totalAmount);
135 }
```

Listing 2.2: ClisBNBLaunchPoolDistributor.sol

```
139 function revokeEpoch(uint64 _epochId) external onlyRole(DEFAULT_ADMIN_ROLE) {
140     Epoch storage epoch = epochs[_epochId];
141     require(epoch.startTime > block.timestamp, "Epoch already started");
142     require(epoch.totalAmount > 0, "Invalid epochId");
143
144
145     address token = epoch.token;
146     uint256 epochTotalAmount = epoch.totalAmount;
147     uint256 epochUnclaimedAmount = epoch.unclaimedAmount;
148     totalUnclaimedAmount[token] -= epochUnclaimedAmount;
149
150
151     delete epochs[_epochId];
152
153
154     emit RevokeEpoch(_epochId, token, epochTotalAmount, epochUnclaimedAmount);
155 }
```

Listing 2.3: ClisBNBLaunchPoolDistributor.sol

```
158 function collectUnclaimed(uint64 _epochId) external onlyRole(DEFAULT_ADMIN_ROLE) {
159     Epoch storage epoch = epochs[_epochId];
160     require(epoch.totalAmount > 0, "Invalid epochId");
161     require(epoch.unclaimedAmount > 0, "No unclaimed amount");
162     require(epoch.endTime < block.timestamp, "Epoch not ended");
163
164
165     address token = epoch.token;
166     uint256 epochUnclaimedAmount = epoch.unclaimedAmount;
167     totalUnclaimedAmount[token] -= epochUnclaimedAmount;
168     epoch.unclaimedAmount = 0;
169
170
171     _transferTo(msg.sender, token, epochUnclaimedAmount);
172     emit CollectUnclaimed(_epochId, token, epoch.totalAmount, epochUnclaimedAmount);
173 }
```

Listing 2.4: ClisBNBLaunchPoolDistributor.sol

```
204 function _vote(uint16[] calldata distributorIds, uint256[] calldata weights, bool
    needBalanceCheck) internal {
205
206
207     require(distributorIds.length == weights.length, "distributorIds and weights length
        mismatch");
208     require(distributorIds.length > 0, "distributorIds and weights should not be empty");
209
210
211     // get current veLista balance of user
212     uint256 userLatestWeight = veLista.balanceOf(msg.sender);
213     // only user needs to check balance
214     if (needBalanceCheck) {
215         require(userLatestWeight > 0, "veLista balance must be greater than 0");
216     }
217     // the next week user voting for
218     uint16 votingWeek = veLista.getCurrentWeek() + 1;
219     // get user all votes of this week
220     Vote[] storage userVotes = userVotedDistributors[msg.sender][votingWeek];
221     // save user old weight of this week
222     uint256 oldUserVotedWeight = userWeeklyVotedWeight[msg.sender][votingWeek];
223     uint256 newUserVotedWeight = oldUserVotedWeight;
224
225
226     // process each vote
227     for (uint256 i = 0 ; i < distributorIds.length; ++i) {
228         uint16 distributorId = distributorIds[i];
229         uint256 weight = weights[i];
230         require(!disabledDistributors[distributorId], "distributor is disabled");
231         require(weight >= 0, "weight should be equals to or greater than 0");
232         require(distributorId > 0 && distributorId <= vault.distributorId(), "distributor does
            not exists");
233
234
235         int256 idx = int256(userVotedDistributorIndex[msg.sender][votingWeek][distributorId]) -
            1;
236         bool voted = idx >= 0;
237
238
239         // first time vote and weight is not 0
240         if (!voted) {
241             userVotes.push(Vote(distributorId, weight));
242             userVotedDistributorIndex[msg.sender][votingWeek][distributorId] = userVotes.length
                ;
243             newUserVotedWeight += weight;
244             distributorWeeklyTotalWeight[distributorId][votingWeek] += weight;
245         } else {
246             // updates user's vote record of this distributor
247             distributorWeeklyTotalWeight[distributorId][votingWeek] =
248                 distributorWeeklyTotalWeight[distributorId][votingWeek] - userVotes[uint256(idx)]
                    .weight + weight;
249             newUserVotedWeight = newUserVotedWeight - userVotes[uint256(idx)].weight + weight;
```

```
250         userVotes[uint256(idx)].weight = weight;
251     }
252 }
253 // update user's consumed weight of this week
254 userWeeklyVotedWeight[msg.sender][votingWeek] =
255     userWeeklyVotedWeight[msg.sender][votingWeek] - oldUserVotedWeight + newUserVotedWeight
256     ;
257 // all user's weight of this week
258 weeklyTotalWeight[votingWeek] =
259     weeklyTotalWeight[votingWeek] - oldUserVotedWeight + newUserVotedWeight;
260
261 // check balance is enough to vote
262 if (needBalanceCheck) {
263     require(userLatestWeight >= userWeeklyVotedWeight[msg.sender][votingWeek], "veLista
264         balance is not enough to vote");
265 }
266
267 emit UserVoted(msg.sender, distributorIds, weights);
268 }
```

Listing 2.5: EmissionVoting.sol

Suggestion Remove this redundant check.

2.2.2 Incorrect comment in function `_vote()`

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In the internal function `_vote()`, the comment “first time vote and weight is not 0” is inconsistent with the implementation. The current implementation allows the user to vote when the weight is 0.

```
198 /**
199  * @dev Vote for the next week
200  * @param distributorIds distributor ids
201  * @param weights weights
202  * @param needBalanceCheck need to check veLista balance
203  */
204 function _vote(uint16[] calldata distributorIds, uint256[] calldata weights, bool
205     needBalanceCheck) internal {
206
207     require(distributorIds.length == weights.length, "distributorIds and weights length
208         mismatch");
209     require(distributorIds.length > 0, "distributorIds and weights should not be empty");
210
211     // get current veLista balance of user
212     uint256 userLatestWeight = veLista.balanceOf(msg.sender);
213     // only user needs to check balance
```

```
214     if (needBalanceCheck) {
215         require(userLatestWeight > 0, "veLista balance must be greater than 0");
216     }
217     // the next week user voting for
218     uint16 votingWeek = veLista.getCurrentWeek() + 1;
219     // get user all votes of this week
220     Vote[] storage userVotes = userVotedDistributors[msg.sender][votingWeek];
221     // save user old weight of this week
222     uint256 oldUserVotedWeight = userWeeklyVotedWeight[msg.sender][votingWeek];
223     uint256 newUserVotedWeight = oldUserVotedWeight;
224
225
226     // process each vote
227     for (uint256 i = 0 ; i < distributorIds.length; ++i) {
228         uint16 distributorId = distributorIds[i];
229         uint256 weight = weights[i];
230         require(!disabledDistributors[distributorId], "distributor is disabled");
231         require(weight >= 0, "weight should be equals to or greater than 0");
232         require(distributorId > 0 && distributorId <= vault.distributorId(), "distributor does
            not exists");
233
234
235         int256 idx = int256(userVotedDistributorIndex[msg.sender][votingWeek][distributorId]) -
            1;
236         bool voted = idx >= 0;
237
238
239         // first time vote and weight is not 0
240         if (!voted) {
241             userVotes.push(Vote(distributorId, weight));
242             userVotedDistributorIndex[msg.sender][votingWeek][distributorId] = userVotes.length
                ;
243             newUserVotedWeight += weight;
244             distributorWeeklyTotalWeight[distributorId][votingWeek] += weight;
245         } else {
246             // updates user's vote record of this distributor
247             distributorWeeklyTotalWeight[distributorId][votingWeek] =
248                 distributorWeeklyTotalWeight[distributorId][votingWeek] - userVotes[uint256(idx)
                    ].weight + weight;
249             newUserVotedWeight = newUserVotedWeight - userVotes[uint256(idx)].weight + weight;
250             userVotes[uint256(idx)].weight = weight;
251         }
252     }
253     // update user's consumed weight of this week
254     userWeeklyVotedWeight[msg.sender][votingWeek] =
255         userWeeklyVotedWeight[msg.sender][votingWeek] - oldUserVotedWeight + newUserVotedWeight
            ;
256     // all user's weight of this week
257     weeklyTotalWeight[votingWeek] =
258         weeklyTotalWeight[votingWeek] - oldUserVotedWeight + newUserVotedWeight;
259
260
261     // check balance is enough to vote
```

```
262     if (needBalanceCheck) {
263         require(userLatestWeight >= userWeeklyVotedWeight[msg.sender][votingWeek], "veLista
            balance is not enough to vote");
264     }
265
266
267     emit UserVoted(msg.sender, distributorIds, weights);
268 }
```

Listing 2.6: EmissionVoting.sol

Suggestion Revise the comment for consistency.

2.2.3 Lack of check in function `adminTransfer()`

Status Confirmed

Introduced by Version 1

Description In the contract `ClisBNBLaunchPoolDistributor`, the admin can withdraw the reward tokens using the function `adminTransfer()`. However, there is no proper check to ensure that the amount being withdrawn is limited to the amount exceeding the allocated distribution amount.

```
177 function adminTransfer(address _token, uint256 _amount) external onlyRole(DEFAULT_ADMIN_ROLE)
    {
178     require(_amount > 0, "Invalid amount");
179     _transferTo(msg.sender, _token, _amount);
180 }
```

Listing 2.7: ClisBNBLaunchPoolDistributor.sol

Suggestion Add relevant checks in the function `adminTransfer()`.

Feedback from the project This method will be used to withdraw tokens for some security reason. So no amount check is applied here.

2.3 Notes

2.3.1 Potential centralization risk

Introduced by Version 1

Description The protocol includes several privileged functions, such as `emergencyWithdraw()`, and `adminTransfer()`. If the owner's private key is lost or maliciously exploited, it could potentially cause losses to users.

Feedback from the project All privileged functions will be operated under a multi-signature account.

2.3.2 Voting capability of admin without `veLista` balance verification

Introduced by [Version 1](#)

Description The privileged admin can use the function `adminVote()` to assign weights for each distributor for the next week. However, the assigned voting weights can exceed their actual voting power (i.e., `veLista` balance), with no upper limit. This can potentially override the results of user voting.

2.3.3 Blocking use of voting results due to set of distributor percentages

Introduced by [Version 1](#)

Description The function `getDistributorWeeklyEmissions()` is used to allocate the weekly rewards. Specifically, if the percentages for each distributor for a certain week have been set with function `setWeeklyDistributorPercent()`, the rewards for that week will be allocated according to this configuration instead of the voting results. Meanwhile, the function `getDistributorWeeklyEmissions()` checks if the percentages for distributors have been set by checking `weeklyDistributorPercent[week][0]` is set to 1, which will be configured once `setWeeklyDistributorPercent()` is invoked. In this case, once the percentages for a distributor have been configured, it is not possible to revert back to use the voting results for reward distribution.

```

125  /**
126   * @dev set weekly distributor percent
127   * @param week week number
128   * @param ids distributor ids
129   * @param percent distributor percent
130   */
131  function setWeeklyDistributorPercent(uint16 week, uint16[] memory ids, uint256[] memory
      percent) onlyRole(MANAGER) external {
132      require(week > veLista.getCurrentWeek(), "week must be greater than current week");
133      require(ids.length > 0 && ids.length == percent.length, "ids and percent length mismatch");
134      uint256 totalPercent;
135
136
137      if (weeklyDistributorPercent[week][0] == 1) {
138          // this week has set, reset last distributor percent
139          for (uint16 i = 1; i <= distributorId; ++i) {
140              weeklyDistributorPercent[week][i] = 0;
141          }
142      }
143      for (uint16 i = 0; i < ids.length; ++i) {
144          require(idToDistributor[ids[i]] != address(0), "distributor not registered");
145          require(weeklyDistributorPercent[week][ids[i]] == 0, "distributor percent already set")
              ;
146          weeklyDistributorPercent[week][ids[i]] = percent[i];
147          totalPercent += percent[i];
148      }
149
150
151      // mark this week set flag
152      weeklyDistributorPercent[week][0] = 1;

```

```
153     require(totalPercent <= 1e18, "Total percent must be less than or equal to 1e18");
154 }
```

Listing 2.8: EmissionVoting.sol

```
255  /**
256   * @dev get distributor weekly emissions
257   * @param id distributor id
258   * @param week week number
259   * @return emissions
260   */
261  function getDistributorWeeklyEmissions(uint16 id, uint16 week) public view returns (uint256) {
262      // emission voting contract not set OR override voting result
263      if (emissionVoting == IEmissionVoting(address(0)) || weeklyDistributorPercent[week][0] ==
264          1) {
265          uint256 pct = weeklyDistributorPercent[week][id];
266          return Math.mulDiv(weeklyEmissions[week], pct, 1e18);
267      }
268      // no one votes
269      if (emissionVoting.getWeeklyTotalWeight(week) == 0) {
270          return 0;
271      }
272      // @dev emission = weeklyEmissions[week] * distributorWeight / totalWeight
273      return Math.mulDiv(
274          weeklyEmissions[week],
275          emissionVoting.getDistributorWeeklyTotalWeight(id, week),
276          emissionVoting.getWeeklyTotalWeight(week)
277      );
278  }
```

Listing 2.9: EmissionVoting.sol

Feedback from the project When `setWeeklyDistributorPercent()` is invoked, in such situation we will not consider user's voting result for a specific week, will use this function to config the distribution percentage instead.

2.3.4 Admin can skip user voting stage or admin voting stage

Introduced by Version 1

Description In the function `setAdminVotePeriod()`, the condition `require(_adminVotePeriod >= 0 && _adminVotePeriod <= WEEK)`, allows the admin to set the period to zero or one week, which is incorrect. If the period was set to `WEEK`, users will be unable to vote due to the check `require(block.timestamp < veList.startTime() + (veList.getCurrentWeek() + 1) * WEEK - ADMIN_VOTE_PERIOD)`. In this case, `block.timestamp` will always be equal to or greater than `veList.startTime() + veList.getCurrentWeek() * WEEK`. Meanwhile, if the period was set to zero, the admin voter will be unable to vote due to the check `require(block.timestamp >= veList.startTime() + (veList.getCurrentWeek() + 1) * WEEK - ADMIN_VOTE_PERIOD)`. The `block.timestamp` will always be less than `veList.startTime() + (veList.getCurrentWeek() + 1) * WEEK`.


```
165 function setAdminVotePeriod(uint256 _adminVotePeriod) external onlyRole(DEFAULT_ADMIN_ROLE) {
166     require(_adminVotePeriod >= 0 && _adminVotePeriod <= WEEK, "admin vote period should within
        0 to 1 week");
167     ADMIN_VOTE_PERIOD = _adminVotePeriod;
168     emit AdminVotePeriodChanged(_adminVotePeriod);
169 }
```

Listing 2.10: EmissionVoting.sol

```
104 function adminVote(uint16[] calldata distributorIds, uint256[] calldata weights) public
    whenNotPaused onlyRole(ADMIN_VOTER) {
105     require(
106         block.timestamp >= veLista.startTime() + (veLista.getCurrentWeek() + 1) * WEEK -
            ADMIN_VOTE_PERIOD,
107         "non admin voting period"
108     );
109     _vote(distributorIds, weights, false);
110 }
```

Listing 2.11: EmissionVoting.sol

```
117 function vote(uint16[] calldata distributorIds, uint256[] calldata weights) public
    whenNotPaused {
118     require(
119         block.timestamp < veLista.startTime() + (veLista.getCurrentWeek() + 1) * WEEK -
            ADMIN_VOTE_PERIOD,
120         "only admin voter can vote now"
121     );
122     _vote(distributorIds, weights, true);
123 }
```

Listing 2.12: EmissionVoting.sol

Feedback from the project `ADMIN_VOTE_PERIOD` can be zero which is an intentional design, we can disable admin or user to vote by adjusting `ADMIN_VOTE_PERIOD` with a zero or arbitrary positive number.

2.3.5 Potential risk of voting results manipulation by abusing flashloan

Introduced by [Version 1](#)

Description In the `EmissionVoting` contract, a user's voting power is based on their balance of `veLista` tokens. The `veLista` contract includes an `earlyClaim()` function, which can be exploited by a user utilizing a flashloan to artificially increase their voting power. Specifically, a malicious user could borrow a substantial amount of `Lista` tokens through a flashloan, lock them into the `veLista` contract to temporarily boost their voting power, and then use this inflated power to influence the outcome of votes in the `EmissionVoting` contract. After casting their vote, the user can utilize the `earlyClaim()` function to pay a penalty fee, unlock their `Lista` tokens, and subsequently repay the flashloan.

Feedback from the project In case someone leverage the flashloan function to put a large amount votes to a specific pool. we will use the function `adminVote()` votes the other pools to

against it. Also, only the `ADMIN_VOTER` role can call the function `adminVote`, we will ensure admin voter accounts are multi-sig.

