BLOCKSEC

Security Audit

Report for SlisBNBProvider and VotingIncentive

Date: November 26, 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 2
1.3.4 Additional Recommendation 3
1.4 Security Model 3

Chapter 2 Findings **4**

2.1 DeFi Security 4
2.1.1 Failure of reward claim due to incorrect check in function `batchClaim()` . 4
2.1.2 Lack of proper check for querying claimable amount 5
2.1.3 Potential unfair rewards distribution due to improperly updating adminVoter 7
2.2 Additional Recommendation 8
2.2.1 Lack of check in function `provide()` 8
2.2.2 Inconsistent check of system variable configuration 10
2.2.3 Lack of check when updating adminVoter 11
2.2.4 Lack of check to ensure the address is not zero 12
2.3 Note 13
2.3.1 Potential centralization risk 13
2.3.2 Instantly manipulable clisBNB balance of lpReserveAddress 13
2.3.3 Potential delayed clisBNB balance update due to untimely synchronization 13

Report Manifest

Item	Description
Client	Lista
Target	SlisBNBProvider and VotingIncentive

Version History

Version	Date	Description
1.0	November 26, 2024	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2024-11-27 03:09:18+0000

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

This audit focuses on the code repositories of the SlisBNBProvider [1] and VotingIncentive [2] of Lista.

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
SlisBNBProvider	`Version 1`	`303c52c3973b2e9ac831bb63c2680a4b4523f8b6`
	`Version 2`	`840dd77d9c83167cacc4e727b9546320b6f2254e`
VotingIncentive	`Version 1`	`ea0490d890bb9ffa7d9d1c3851a7c3f91411326d`
	`Version 2`	`2931a1ae9bfdfdbc320b92e10eb7220e884fcb56`

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.

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] https://github.com/lista-dao/lista-dao-contracts/blob/303c52c3973b2e9ac831bb63c2680a4b4523f8b6/contracts/ceros/provider/SlisBNBProvider.sol

[2] https://github.com/lista-dao/lista-token/blob/ea0490d890bb9ffa7d9d1c3851a7c3f91411326d/contracts/dao/VotingIncentive.sol

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 and Common Weakness Enumeration. 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 \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

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.
- **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 **three** potential security issues. Besides, we have **four** recommendations and **three** notes.

- Medium Risk: 1
- Low Risk: 2
- Recommendation: 4
- Note: 3

ID	Severity	Description	Category	Status
1	Medium	Failure of reward claim due to incorrect check in function `batchClaim()`	DeFi Security	Fixed
2	Low	Lack of proper check for querying claimable amount	DeFi Security	Fixed
3	Low	Potential unfair rewards distribution due to improperly updating adminVoter	DeFi Security	Confirmed
4	-	Lack of check in function `provide()`	Recommendation	Fixed
5	-	Inconsistent check of system variable configuration	Recommendation	Fixed
6	-	Lack of check when updating adminVoter	Recommendation	Fixed
7	-	Lack of check to ensure the address is not zero	Recommendation	Fixed
8	-	Potential centralization risk	Note	-
9	-	Instantly manipulable clisBNB balance of lpReserveAddress	Note	-
10	-	Potential delayed clisBNB balance update due to untimely synchronization	Note	-

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Failure of reward claim due to incorrect check in function `batchClaim()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `VotingIncentive` contract, the function `batchClaim()` allows users to batch claim rewards with the given `ClaimParams`. However, when checking whether the user has already claimed rewards, the condition only allows the function to proceed with `claim()` if the user has already claimed before, which is incorrect. This can result in users being unable to claim rewards, leading to a DoS (Denial of Service) scenario.

```
176 function batchClaim(ClaimParams[] memory _input) external {
177   address user = msg.sender;
```

```
178   for (uint256 i = 0; i < _input.length; ++i) {
179     ClaimParams memory _params = _input[i];
180     address[] memory _assets = _params.assets;
181     for (uint256 j = 0; j < _assets.length; ++j) {
182       if (!claimedIncentives[user][_params.distributorId][_params.week][_assets[j]]) continue;
183       claim(user, _params.distributorId, _params.week, _assets[j]);
184     }
185   }
186 }
```

Listing 2.1: contracts/dao/VotingIncentive.sol

```
194 function claim(address _user, uint16 _distributorId, uint16 _week, address _asset) public
        nonReentrant whenNotPaused {
195   require(_user != adminVoter, "Invalid voter");
196   require(_week <= vault.getWeek(block.timestamp), "Invalid week");
197   require(_distributorId > 0 && _distributorId <= vault.distributorId(), "Invalid distributorId"
          );
198   require(!claimedIncentives[_user][_distributorId][_week][_asset], "Already claimed");
199   uint256 adminWeight = getRawWeight(adminVoter, _distributorId, _week);
200
201   uint256 amountToClaim = calculateAmount(_user, _distributorId, _week, _asset, adminWeight);
202
203   claimedIncentives[_user][_distributorId][_week][_asset] = true;
204   if (_asset == address(0)) {
205     (bool success, ) = payable(_user).call{ value: amountToClaim }("");
206     require(success, "Transfer failed");
207   } else {
208     IERC20(_asset).safeTransfer(_user, amountToClaim);
209   }
210
211   emit IncentiveClaimed(_user, _distributorId, _week, _asset, amountToClaim);
212 }
```

Listing 2.2: contracts/dao/VotingIncentive.sol

Impact The user will not be able to claim the rewards via the function `batchClaim()`.

Suggestion Revise the logic in `batchClaim()` function to skip claimed incentives instead of unclaimed ones.

2.1.2 Lack of proper check for querying claimable amount

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The public view function `getClaimableAmount()` is used to query the claimable reward amount for a user with given parameters. However, the called function `calculateAmount()`, which computes the claimable amount, does not account for the situation where `poolWeight` and `_adminWeight` are equal. Since their difference is used as the denominator in the calculation, this results in a division by zero issue and causes the function to revert.

For example, if the user queries a week in the future, the `poolWeight` and `_adminWeight` may be the same. In such cases, as a view function, it should return a claimable amount of 0 instead of throwing an error.

```
348 function getClaimableAmount(
349   address _user,
350   ClaimParams[] memory _input
351 ) public view returns (ClaimableAmount[] memory claimableAmt) {
352   claimableAmt = new ClaimableAmount[](_input.length);
353   for (uint256 i = 0; i < _input.length; ++i) {
354     ClaimParams memory _params = _input[i];
355     address[] memory _assets = _params.assets;
356     Incentive[] memory _incentives = new Incentive[](_assets.length);
357
358     for (uint256 j = 0; j < _assets.length; ++j) {
359       uint256 amount = calculateAmount(
360         _user,
361         _params.distributorId,
362         _params.week,
363         _assets[j],
364         getRawWeight(adminVoter, _params.distributorId, _params.week)
365       );
366       _incentives[j] = Incentive({ asset: _assets[j], amount: amount });
367     }
368     claimableAmt[i] = ClaimableAmount({
369       distributorId: _params.distributorId,
370       week: _params.week,
371       incentives: _incentives
372     });
373   }
374 }
```

Listing 2.3: contracts/dao/VotingIncentive.sol

```
222 function calculateAmount(
223   address _user,
224   uint16 _distributorId,
225   uint16 _week,
226   address _asset,
227   uint256 _adminWeight
228 ) internal view returns (uint256 _amount) {
229   uint256 poolWeight = emissionVoting.getDistributorWeeklyTotalWeight(_distributorId, _week);
230   uint256 usrWeight = getRawWeight(_user, _distributorId, _week);
231
232   uint256 incentive = weeklyIncentives[_distributorId][_week][_asset];
233   // If admin has voted, adjust user weight by removing admin weight from pool
234   _amount = (usrWeight * incentive) / (poolWeight - _adminWeight);
235 }
```

Listing 2.4: contracts/dao/VotingIncentive.sol

Impact When users query the claimable amount, the function `getClaimableAmount()` throws an error in some cases where it should have returned 0.

Suggestion Add a check to ensure that when `poolWeight` and `_adminWeight` are equal, the query result is still 0.

2.1.3 Potential unfair rewards distribution due to improperly updating adminVoter

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the `VotingIncentive` contract, users can claim rewards through the function `claim()`. Specifically, the reward amount is calculated by invoking the function `calculateAmount()` based on the ratio of the user's voting weight to the total voting weight. Meanwhile, if there is voting weight associated with the `adminVoter` address for the corresponding week, this weight will be subtracted from the total voting weight during the calculation. However, if the `adminVoter` address changes after the voting phase ends, the queried `adminVoter` might not match the expected value, which is incorrect.

```
194 function claim(address _user, uint16 _distributorId, uint16 _week, address _asset) public
        nonReentrant whenNotPaused {
195   require(_user != adminVoter, "Invalid voter");
196   require(_week <= vault.getWeek(block.timestamp), "Invalid week");
197   require(_distributorId > 0 && _distributorId <= vault.distributorId(), "Invalid distributorId"
          );
198   require(!claimedIncentives[_user][_distributorId][_week][_asset], "Already claimed");
199   uint256 adminWeight = getRawWeight(adminVoter, _distributorId, _week);
200
201   uint256 amountToClaim = calculateAmount(_user, _distributorId, _week, _asset, adminWeight);
202
203   claimedIncentives[_user][_distributorId][_week][_asset] = true;
204   if (_asset == address(0)) {
205     (bool success, ) = payable(_user).call{ value: amountToClaim }("");
206     require(success, "Transfer failed");
207   } else {
208     IERC20(_asset).safeTransfer(_user, amountToClaim);
209   }
210
211   emit IncentiveClaimed(_user, _distributorId, _week, _asset, amountToClaim);
212 }
```

Listing 2.5: contracts/dao/VotingIncentive.sol

```
222 function calculateAmount(
223   address _user,
224   uint16 _distributorId,
225   uint16 _week,
226   address _asset,
227   uint256 _adminWeight
228 ) internal view returns (uint256 _amount) {
229   uint256 poolWeight = emissionVoting.getDistributorWeeklyTotalWeight(_distributorId, _week);
230   uint256 usrWeight = getRawWeight(_user, _distributorId, _week);
231
```

```
232   uint256 incentive = weeklyIncentives[_distributorId][_week][_asset];
233   // If admin has voted, adjust user weight by removing admin weight from pool
234   _amount = (usrWeight * incentive) / (poolWeight - _adminWeight);
235 }
```

Listing 2.6: contracts/dao/VotingIncentive.sol

```
270 function setAdminVoter(address _adminVoter) external onlyRole(DEFAULT_ADMIN_ROLE) {
271   require(_adminVoter != address(0) && _adminVoter != adminVoter, "Invalid adminVoter");
272   adminVoter = _adminVoter;
273
274   emit AdminVoterChanged(_adminVoter);
275 }
```

Listing 2.7: contracts/dao/VotingIncentive.sol

```
310 function getRawWeight(address _account, uint16 _distributorId, uint16 _week) public view returns
        (uint256 _weight) {
311   int256 index = int256(emissionVoting.userVotedDistributorIndex(_account, _week, _distributorId
        )) - 1;
312   if (index < 0) {
313     return 0; // account has not voted
314   }
315   EmissionVoting.Vote[] memory votes = emissionVoting.getUserVotedDistributors(_account, _week);
316   EmissionVoting.Vote memory vote = votes[uint256(index)];
317
318   require(vote.distributorId == _distributorId, "Invalid distributorId");
319   _weight = vote.weight;
320 }
```

Listing 2.8: contracts/dao/VotingIncentive.sol

Impact After the voting phase ends, the `adminWeight` may change, which could result in an unfair situation when users claim their rewards.

Suggestion Revise the logic to ensure that the vote weight of the `adminVoter` remains unchanged after the voting phase ends.

Feedback from the project This is a known issue, the `adminVoter` address will not be changed for both `EmissionVoting` and `VotingIncentives`.

2.2 Additional Recommendation

2.2.1 Lack of check in function `provide()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `BaseTokenProvider` contract, users can deposit assets through the function `provide()` and delegate the received shares to a specified address. However, the function `provide()` does not check whether users are attempting to delegate their shares to themselves. Note that function `delegateAllTo()` allows users to reclaim the delegated shares back

to their own address. Therefore, allowing users to delegate shares to themselves is essentially redundant and meaningless.

```
80  function provide(uint256 _amount, address _delegateTo)
81      external
82      virtual
83      whenNotPaused
84      nonReentrant
85      returns (uint256)
86  {
87      require(_amount > 0, "zero deposit amount");
88      require(_delegateTo != address(0), "delegateTo cannot be zero address");
89      require(
90          delegation[msg.sender].delegateTo == _delegateTo ||
91          delegation[msg.sender].amount == 0, // first time, clear old delegatee
92          "delegateTo is differ from the current one"
93      );
94
95      IERC20(token).safeTransferFrom(msg.sender, address(this), _amount);
96      // do sync before balance modified
97      _syncLp(msg.sender);
98      uint256 userPartLp = _provideCollateral(msg.sender, _delegateTo, _amount);
99
100     Delegation storage userDelegation = delegation[msg.sender];
101     userDelegation.delegateTo = _delegateTo;
102     userDelegation.amount += userPartLp;
103
104     emit Deposit(msg.sender, _amount, userPartLp);
105     return userPartLp;
106 }
```

Listing 2.9: contracts/ceros/provider/BaseTokenProvider.sol

```
134 function delegateAllTo(address _newDelegateTo)
135     external
136     virtual
137     whenNotPaused
138     nonReentrant
139 {
140     require(_newDelegateTo != address(0), "delegateTo cannot be zero address");
141     _syncLp(msg.sender);
142     // get user total deposit
143     uint256 userTotalLp = userLp[msg.sender];
144     require(userTotalLp > 0, "zero lp to delegate");
145
146     Delegation storage currentDelegation = delegation[msg.sender];
147     address currentDelegateTo = currentDelegation.delegateTo;
148
149     // Step 1. burn all tokens
150     if (currentDelegation.amount > 0) {
151         // burn delegatee's token
152         lpToken.burn(currentDelegateTo, currentDelegation.amount);
153         // burn self's token
154         if (userTotalLp > currentDelegation.amount) {
```

```
155             _safeBurnLp(msg.sender, userTotalLp - currentDelegation.amount);
156         }
157     } else {
158         _safeBurnLp(msg.sender, userTotalLp);
159     }
160
161     // Step 2. save new delegatee and mint all tokens to delegatee
162     if (_newDelegateTo == msg.sender) {
163         // mint all to self
164         lpToken.mint(msg.sender, userTotalLp);
165         // remove delegatee
166         delete delegation[msg.sender];
167     } else {
168         // mint all to new delegatee
169         lpToken.mint(_newDelegateTo, userTotalLp);
170         // save delegatee's info
171         currentDelegation.delegateTo = _newDelegateTo;
172         currentDelegation.amount = userTotalLp;
173     }
174
175     emit ChangeDelegateTo(msg.sender, currentDelegateTo, _newDelegateTo);
176 }
```

Listing 2.10: contracts/ceros/provider/BaseTokenProvider.sol

Suggestion Add check to ensure that the parameter `_delegateTo` in the function `provide()` is not equal to `msg.sender`.

2.2.2 Inconsistent check of system variable configuration

Status Fixed in `Version 2`

Introduced by `Version 1`

Description According to the implementation of the `initialize()` function, `_userLpRate` only validates the upper limit, allowing `_userLpRate` to be 0. However, in the function `changeUserLpRate()`, both upper and lower limits are validated, explicitly disallowing `_userLpRate` from being 0.

```
47  function initialize(
48      address _admin,
49      address _manager,
50      address _proxy,
51      address _pauser,
52      address _lpToken,
53      address _token,
54      address _daoAddress,
55      address _lpReserveAddress,
56      uint128 _exchangeRate,
57      uint128 _userLpRate
58  ) public initializer {
59      require(_admin != address(0), "admin is the zero address");
60      require(_manager != address(0), "manager is the zero address");
61      require(_proxy != address(0), "proxy is the zero address");
62      require(_pauser != address(0), "pauser is the zero address");
```

```
63        require(_lpToken != address(0), "lpToken is the zero address");
64        require(_token != address(0), "token is the zero address");
65        require(_daoAddress != address(0), "daoAddress is the zero address");
66        require(_lpReserveAddress != address(0), "lpReserveAddress is the zero address");
67        require(_exchangeRate > 0, "exchangeRate invalid");
68        require(_userLpRate <= 1e18, "too big rate number");
69
70        __Pausable_init();
71        __ReentrancyGuard_init();
72        _grantRole(DEFAULT_ADMIN_ROLE, _admin);
73        _grantRole(MANAGER, _manager);
74        _grantRole(PROXY, _proxy);
75        _grantRole(PAUSER, _pauser);
76
77        token = _token;
78        lpToken = ILpToken(_lpToken);
79        dao = IDao(_daoAddress);
80        lpReserveAddress = _lpReserveAddress;
81        exchangeRate = _exchangeRate;
82        userLpRate = _userLpRate;
83
84        IERC20(token).approve(_daoAddress, type(uint256).max);
85    }
```

Listing 2.11: contracts/ceros/provider/SlisBNBProvider.sol

```
233   function changeUserLpRate(uint128 _userLpRate) external onlyRole(MANAGER) {
234       require(_userLpRate > 0 && _userLpRate <= 1e18, "userLpRate invalid");
235
236       userLpRate = _userLpRate;
237       emit ChangeUserLpRate(userLpRate);
238   }
```

Listing 2.12: contracts/ceros/provider/SlisBNBProvider.sol

Suggestion Revise the logic to ensure the validation is consistent.

2.2.3 Lack of check when updating adminVoter

Status Fixed in `Version 2`

Introduced by `Version 1`

Description According to the implementation of the `initialize()` function, `adminVoter` is checked to ensure that it holds the `ADMIN_VOTER` role in contract `emissionVoting` at the same time. However, in the function `setAdminVoter()`, the updated `adminVoter` cannot guarantee that it holds the `ADMIN_VOTER` role.

```
90  function initialize(
91    address _vault,
92    address _emissionVoting,
93    address _adminVoter,
94    address _admin,
95    address _manager,
```

```
96    address _pauser
97  ) public initializer {
98    __AccessControl_init();
99    __Pausable_init();
100   __ReentrancyGuard_init();
101
102   vault = IVault(_vault);
103   emissionVoting = IEmissionVoting(_emissionVoting);
104   require(emissionVoting.hasRole(emissionVoting.ADMIN_VOTER(), _adminVoter), "Invalid adminVoter
          ");
105   adminVoter = _adminVoter;
106
107   _setupRole(DEFAULT_ADMIN_ROLE, _admin);
108   _setupRole(MANAGER, _manager);
109   _setupRole(PAUSER, _pauser);
110 }
```

Listing 2.13: contracts/dao/VotingIncentive.sol

```
270 function setAdminVoter(address _adminVoter) external onlyRole(DEFAULT_ADMIN_ROLE) {
271   require(_adminVoter != address(0) && _adminVoter != adminVoter, "Invalid adminVoter");
272   adminVoter = _adminVoter;
273
274   emit AdminVoterChanged(_adminVoter);
275 }
```

Listing 2.14: contracts/dao/VotingIncentive.sol

Suggestion Revise the logic to ensure the updated `adminVoter` is valid.

2.2.4 Lack of check to ensure the address is not zero

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `initialize()` function lacks zero address checks, which may lead to critical contract addresses being incorrectly initialized.

```
90  function initialize(
91    address _vault,
92    address _emissionVoting,
93    address _adminVoter,
94    address _admin,
95    address _manager,
96    address _pauser
97  ) public initializer {
98    __AccessControl_init();
99    __Pausable_init();
100   __ReentrancyGuard_init();
101
102   vault = IVault(_vault);
103   emissionVoting = IEmissionVoting(_emissionVoting);
104   require(emissionVoting.hasRole(emissionVoting.ADMIN_VOTER(), _adminVoter), "Invalid adminVoter
          ");
```

```
105     adminVoter = _adminVoter;
106
107     _setupRole(DEFAULT_ADMIN_ROLE, _admin);
108     _setupRole(MANAGER, _manager);
109     _setupRole(PAUSER, _pauser);
110   }
```

Listing 2.15: contracts/dao/VotingIncentive.sol

Suggestion Add checks to ensure the addresses are not zero during the configuration.

2.3 Note

2.3.1 Potential centralization risk

Introduced by Version 1

Description In the current implementation, several privileged roles are set to govern and regulate the system-wide operations (e.g., parameter setting, pause/unpause, and granting roles). Additionally, the function `changeExchangeRate()` allows the privileged `manager` role to arbitrarily modify the exchange rate, with changes taking effect immediately, directly impacting the value of all users' assets. The `admin` role also has the ability to upgrade all the implementation contracts. If the private keys of these privileged roles are lost or maliciously exploited, it could potentially lead to losses for users.

2.3.2 Instantly manipulable clisBNB balance of lpReserveAddress

Introduced by Version 1

Description In the `SlisBNBProvider` contract, the `lpReserveAddress` will receive a portion of the `clisBNB` minted by users as fees. However, its balance can be manipulated instantly. Specifically, a user can use the `provide()` function to set the `delegatee` as `lpReserveAddress` and mint a large amount of `clisBNB` directly to this address. Later, when needed, the user can invoke the `delegateAllTo()` function to burn the corresponding `clisBNB` from the `lpReserveAddress`. It's important to note that the `clisBNB` balance of the `lpReserveAddress` should not be used for calculations in other parts of the protocol.

Feedback from the project The `clisBNB` in certain types of addresses will be rewarded with some tokens periodically. So `clisBNB` in `lpReserveAddress` can be regarded as a kind of fee which leads some part of the rewards be issued to the `lpReserveAddress` owned by `LISTA`(which matches the certain type).

2.3.3 Potential delayed clisBNB balance update due to untimely synchronization

Introduced by Version 1

Description In the `SlisBNBProvider` contract, the function `syncUserLp()` dynamically adjusts a user's `clisBNB` (share) balance based on changes in their collateral balance recorded in the `dao` contract. Specifically, if the user's collateral balance in the `dao` contract changes, `syncUserLp()`

will mint or burn the corresponding amount of `clisBNB` and update the user's `userLp`. However, if the user interacts directly with the `dao` contract, their collateral balance might change without triggering the function `syncUserLp()`. This could lead to the user's `clisBNB` balance not being updated in a timely manner.

Feedback from the project Direct interaction with `DAO` from user will be forbidden after `SlisBNBProvider` released.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS