BEOSIN
Blockchain Security

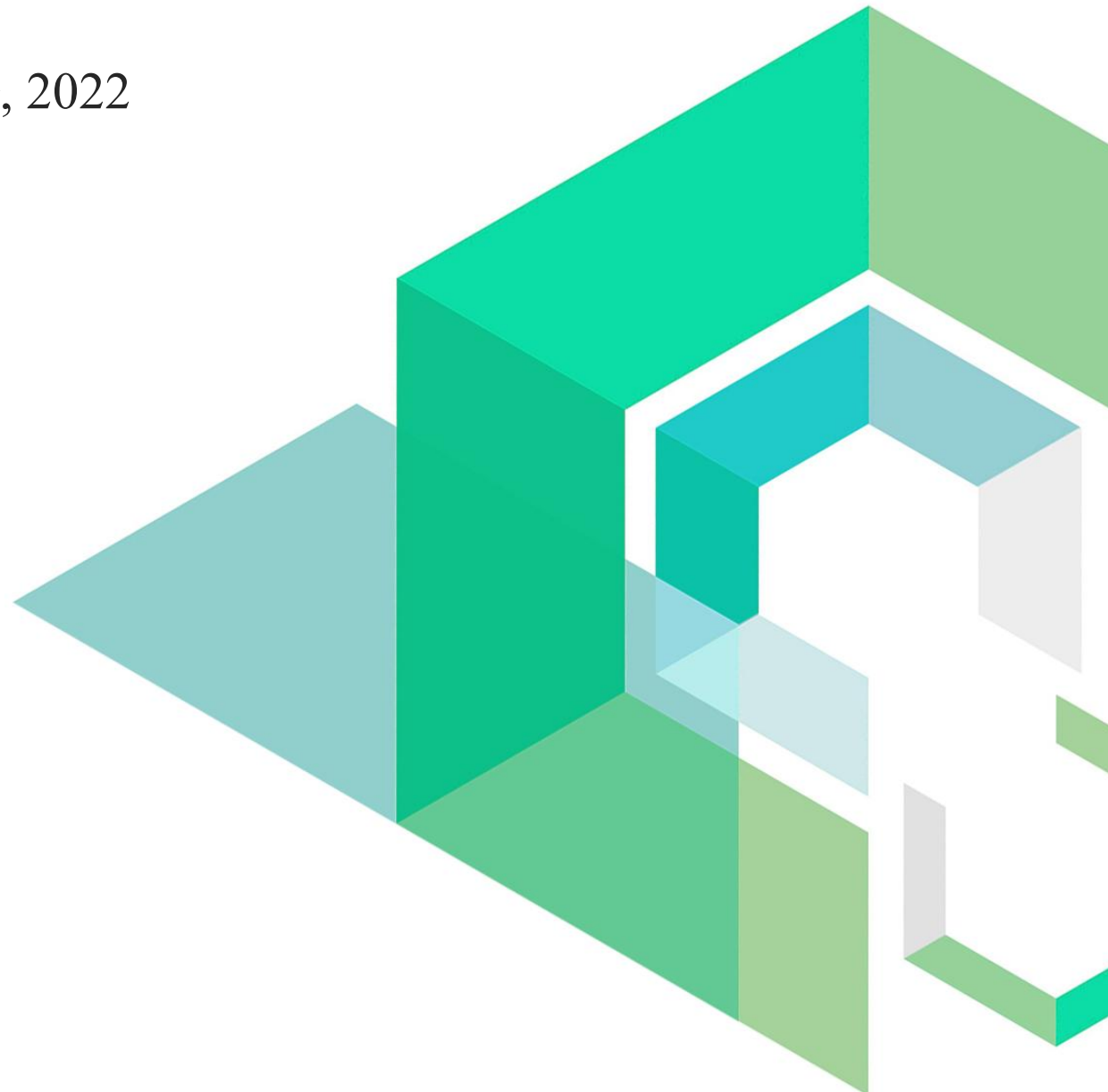
Roselle NFT and Nodes

Smart Contract Security Audit

V1.0

No. 202208101129

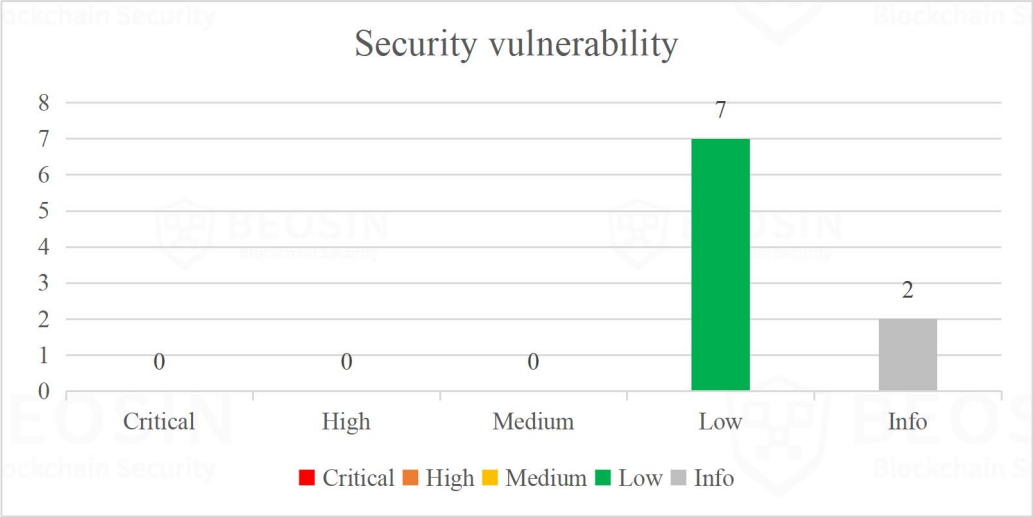
Aug 10th, 2022



Contents

Summary of Audit Results

After auditing, 7 Low-risk items and 2 Info items were identified in the Farm project. Specific audit details will be presented in the **Findings section**. Users should pay attention to the following aspects when interacting with this project:



*Notes:

- **Risk Description:**

1. Users cannot withdraw reward tokens when there are not enough reward tokens in the contract.

2. Too large owner permissions can restrict users from receiving reward tokens.

3. If use roselle tokens as staking tokens, it may result in the loss of users' staked roselle tokens. And because the stake amount and reward amount are included in lpSupply, the calculated accRosellePerShare will decrease.

4. The excess reward tokens in the contract cannot be withdrawn.

- **Project Description:**

1. Business overview

The project contains a BEP-20 token contract Roselle and two business contracts, Farm and Nodes. Roselle is a BEP-20 token issued on BNB Chain. The total supply of Roselle is 2.1 million, which cannot be minted and burned. The contract has a *back* function to withdraw the tokens stored in the Roselle contract.

In the Farm contract, the owner can add a designated pool to provide the stake mining function. The user can select a different pool and stake the LPtoken of the corresponding pool to the Farm contract to obtain roselle reward tokens. The owner and the DAO contract can modify the reward distribution ratio of each pool, and the owner can modify the state of the open variable, so the user cannot withdraw reward tokens.

In the node contract, the user can stake the specified 9999 fon tokens to register as a node, and then the user can cancel the node status and withdraw the staked 9999 fon tokens. When becoming a node, the user can modify the name and url of the node. Other users can stake fonos tokens to this node to improve the node's rank.

When the node's total fonos token stake exceeds the previous node's, the rank of this node's users can be moved forward. When a user withdraws stake fonos tokens from node users, if the total fonos token stake amount of the node is lower than that of the following node, the rank of this node will drop.

There are a total of 99 rank rankings. When the nodes of the rank are full, other nodes want to participate in the rank, they can only wait until the node user in the ranking cancels the node status, or the node's total fonos token stake amount is 0 and exits the rank. Can participate in rank.

It should be noted that there is no time limit for the user's fonos stake tokens to be withdrawn. If there is a situation in which the rewards are distributed according to the rank ranking, there may be a phenomenon of using the flash loan to brush the ranking.

1 Overview

1.1 Project Overview

Project Name	Roselle NFT and Nodes
Platform	BNB Chain
Contact Address	0xffe3639c0D667a056B9d18Fc505cf87342463DD4(ERC20.sol) 0x1B79549Bb3cAe5614f1A10D5E033F35C42d570BC(Farm.sol) 0x5932bce64A5354753853d4AEEb9eD6037E84FB38(Nodes.sol)

1.2 Audit Overview

Audit work duration：Aug 5, 2022 – Aug 10, 2022

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Technology Co. Ltd.

2 Findings

Index	Risk description	Severity level	Status
Farm-1	Reward issues of add and modify pool	Low	Fixed
Farm-2	Reward query questions	Low	Fixed
Farm-3	Reward token issuance issues	Low	Acknowledged
Farm-4	Centralization risk	Low	Acknowledged
Farm-5	Reward tokens cannot be staked	Low	Acknowledged
Farm-6	Excess reward tokens cannot be withdrawn	Low	Acknowledged
Farm-7	Missing event trigger	Info	Acknowledged
Nodes-1	Fake recharge risk	Low	Fixed
Nodes-2	Missing event trigger	Info	Acknowledged

Status Notes:

- Farm-3 is unfixed and will cause when there are not enough reward tokens in the contract, then the user cannot claim the reward tokens.
- Farm-4 is unfixed and will cause the owner can change the open variable to restrict users from withdrawing reward tokens.
- Farm-5 is unfixed and will cause when user stakes roselle tokens, other users will withdraw the roselle tokens as reward tokens.
- Farm-6 is unfixed and will cause the contract has no function to withdraw the excess tokens stored in the contract after the staking period ends.
- Farm-7 is unfixed and will not cause any issue.
- Nodes-2 is unfixed and will not cause any issue.

Finding Details:

[Farm-1] Reward issues of add and modify pool

Severity Level	Low
Type	Business Security
Lines	Farm.sol #L84-107, 109-126
Description	When the owner adds or modifies the pool, he can choose not to update the reward, which will reduce the allocation ratio of the pool where the original mortgaged user is located, thus reducing the original amount of reward.

```
84    function add(
85        uint256 _allocPoint,
86        IERC20 _lpToken,
87        bool _withUpdate
88    ) public onlyOwner {
89        require(!poolAdded[address(_lpToken)], "duplicate poll");
90        poolAdded[address(_lpToken)] = true;
91        if (_withUpdate) {
92            massUpdatePools();
93        }
94        uint256 lastRewardBlock = block.number > startBlock
95            ? block.number
96            : startBlock;
97        totalAllocPoint = totalAllocPoint.add(_allocPoint);
98        poolInfo.push(
99            PoolInfo({
100               lpToken: _lpToken,
101               allocPoint: _allocPoint,
102               lastRewardBlock: lastRewardBlock,
103               accRosellePerShare: 0
104           })
105       );
106   }
107
```

Figure 1 The source code of *add* function

```
109    function set(
110        uint256 _pid,
111        uint256 _allocPoint,
112        bool _withUpdate
113    ) public validatePoolByPid(_pid) {
114        require(
115            owner() == _msgSender() || daoaddr == _msgSender(),
116            "Ownable: caller is not the owner"
117        );
118        if (_withUpdate) {
119            massUpdatePools();
120        }
121        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
122            _allocPoint
123        );
124        poolInfo[_pid].allocPoint = _allocPoint;
125    }
126
```

Figure 2 The source code of *set* function

Recommendations	It is recommended to delete the if judgment and call the *massUpdatePools* function

when adding and modifying the pool to update the reward.

Status Fixed.

```
84    function add(
85        uint256 _allocPoint,
86        IERC20 _lpToken
87    ) public onlyOwner {
88        require(!poolAdded[address(_lpToken)], "duplicate poll");
89        poolAdded[address(_lpToken)] = true;
90        massUpdatePools();
91        uint256 lastRewardBlock = block.number > startBlock
92            ? block.number
93            : startBlock;
94        totalAllocPoint = totalAllocPoint.add(_allocPoint);
95        poolInfo.push(
96            PoolInfo({
97                lpToken: _lpToken,
98                allocPoint: _allocPoint,
99                lastRewardBlock: lastRewardBlock,
100               accRosellePerShare: 0
101           })
102       );
103   }
104
105   // Update the given pool's Roselle allocation point. Can only be called by the owner.
106   function set(
107       uint256 _pid,
108       uint256 _allocPoint
109   ) public validatePoolByPid(_pid) {
110       require(
111           owner() == _msgSender() || daoaddr == _msgSender(),
112           "Ownable: caller is not the owner"
113       );
114       massUpdatePools();
115       totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
116           _allocPoint
117       );
118       poolInfo[_pid].allocPoint = _allocPoint;
119   }
120
```

Figure 3 The source code of *add* and *set* functions(Fixed)

[Farm-2] Reward query questions

Severity Level	**Low**
Type	Business Security
Lines	Farm.sol #L139-166, L177-196
Description	When a user queries, the calculated roselleReward value is updated to 85%, while the actual the *update* function usage is 100%, resulting in inconsistent query and actual results.

```
139     function pendingRoselle(uint256 _pid, address _user)
140         external
141         view
142         validatePoolByPid(_pid)
143         returns (uint256)
144     {
145         PoolInfo storage pool = poolInfo[_pid];
146         UserInfo storage user = userInfo[_pid][_user];
147         uint256 accRosellePerShare = pool.accRosellePerShare;
148         uint256 lpSupply = pool.lpToken.balanceOf(address(this));
149         if (block.number > pool.lastRewardBlock && lpSupply != 0) {
150             uint256 multiplier = getMultiplier(
151                 pool.lastRewardBlock,
152                 block.number
153             );
154             uint256 roselleReward = multiplier
155                 .mul(roselllePerBlock)
156                 .mul(pool.allocPoint)
157                 .div(totalAllocPoint)
158                 .mul(85)
159                 .div(100);
160             accRosellePerShare = accRosellePerShare.add(
161                 roselleReward.mul(1e12).div(lpSupply)
162             );
163         }
164         return
165             user.amount.mul(accRosellePerShare).div(1e12).sub(user.rewardDebt);
166     }
```

Figure 4 The source code of *pendingRoselle* function

```
177     function updatePool(uint256 _pid) public validatePoolByPid(_pid) {
178         PoolInfo storage pool = poolInfo[_pid];
179         if (block.number <= pool.lastRewardBlock) {
180             return;
181         }
182         uint256 lpSupply = pool.lpToken.balanceOf(address(this));
183         if (lpSupply == 0) {
184             pool.lastRewardBlock = block.number;
185             return;
186         }
187         uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
188         uint256 roselleReward = multiplier
189             .mul(roselllePerBlock)
190             .mul(pool.allocPoint)
191             .div(totalAllocPoint);
192         pool.accRosellePerShare = pool.accRosellePerShare.add(
193             roselleReward.mul(1e12).div(lpSupply)
194         );
195         pool.lastRewardBlock = block.number;
196     }
```

Figure 5 The source code of *updatePool* function

Recommendations	It is recommended to unify the way of calculating roselleReward.
Status	Fixed.

```
133     function pendingRoselle(uint256 _pid, address _user)
134         external
135         view
136         validatePoolByPid(_pid)
137         returns (uint256)
138     {
139         PoolInfo storage pool = poolInfo[_pid];
140         UserInfo storage user = userInfo[_pid][_user];
141         uint256 accRosellePerShare = pool.accRosellePerShare;
142         uint256 lpSupply = pool.lpToken.balanceOf(address(this));
143         if (block.number > pool.lastRewardBlock && lpSupply != 0) {
144             uint256 multiplier = getMultiplier(
145                 pool.lastRewardBlock,
146                 block.number
147             );
148             uint256 roselleReward = multiplier
149                 .mul(rosellePerBlock)
150                 .mul(pool.allocPoint)
151                 .div(totalAllocPoint);
152             accRosellePerShare = accRosellePerShare.add(
153                 roselleReward.mul(1e12).div(lpSupply)
154             );
155         }
156         return
157             user.amount.mul(accRosellePerShare).div(1e12).sub(user.rewardDebt);
158     }
159
```

Figure 6 The source code of *pendingRoselle* function(Fixed)

[Farm-3] Reward token issuance issues

Severity Level	**Low**
Type	Business Security
Lines	Farm.sol #L285-291
Description	When the user receives the reward token, if there is not enough reward token stored in the contract in advance, the user will fail to receive the reward.

```
285     function takeToken(uint256 _pid) external validatePoolByPid(_pid) {
286         require(opened, "closed");
287         safeRoselleTransfer(msg.sender, userTokens[msg.sender][_pid]);
288         userTokens[msg.sender][_pid] = 0;
289     }
290 }
291
```

Figure 7 The source code of *takeToken* function

Recommendations	It is recommended to ensure that there should be enough reward tokens in the contract.
Status	Acknowledged.

[Farm-4] Centralization risk

Severity Level	**Low**
Type	Business Security
Lines	Farm.sol #L452-459
Description	The owner can set the open state through the *open* function to restrict users from withdrawing reward tokens.

```
280     function open() external onlyOwner {
281         require(!opened, "opened");
282         opened = true;
283     }
284
```

Figure 8 The source code of *open* function

```
285     function takeToken(uint256 _pid) external validatePoolByPid(_pid) {
286         require(opened, "closed");
287         safeRoselleTransfer(msg.sender, userTokens[msg.sender][_pid]);
288         userTokens[msg.sender][_pid] = 0;
289     }
290 }
291
```

Figure 9 The source code of *takeToken* function

Recommendations	It is recommended to use DAO contracts to manage this permission.
Status	Acknowledged.

[Farm-5] Reward tokens cannot be staked

Severity Level	**Low**
Type	Business Security
Lines	Farm.sol #L84-104
Description	When adding a pool, the owner needs to pay attention that the roselle reward token cannot be used as a stake token. If a user stakes roselle tokens, other users will withdraw the roselle tokens as reward tokens. And because the pledge amount and reward amount are included in lpSupply, the calculated accRosellePerShare will decrease.

```
84      function add(
85          uint256 _allocPoint,
86          IERC20 _lpToken
87      ) public onlyOwner {
88          require(!poolAdded[address(_lpToken)], "duplicate poll");
89          poolAdded[address(_lpToken)] = true;
90          massUpdatePools();
91          uint256 lastRewardBlock = block.number > startBlock
92              ? block.number
93              : startBlock;
94          totalAllocPoint = totalAllocPoint.add(_allocPoint);
95          poolInfo.push(
96              PoolInfo({
97                  lpToken: _lpToken,
98                  allocPoint: _allocPoint,
99                  lastRewardBlock: lastRewardBlock,
100                 accRosellePerShare: 0
101             })
102         );
103     }
104
```

Figure 10 The source code of *add* function

Recommendations	The proposed addition of the token cannot be the verification of the roselle address.
Status	Acknowledged.

[Farm-6] Excess reward tokens cannot be withdrawn

Severity Level	**Low**
Type	Business Security
Lines	Farm.sol #L277-282, 199-228, 230-253
Description	When the amount of reward tokens of the Farm contract is too large, the contract has no function to withdraw. Reward tokens in the contract can only be withdrawn through the *takenToken* function. After the end of the pledge period, if the user's withdrawal amount is less than the reward storage amount of the Farm contract, it will cause a loss of reward tokens.

```
277     function takeToken(uint256 _pid) external validatePoolByPid(_pid) {
278         require(opened, "closed");
279         safeRoselleTransfer(msg.sender, userTokens[msg.sender][_pid]);
280         userTokens[msg.sender][_pid] = 0;
281     }
282 }
```

Figure 11 The source code of *takeToken* function

```
199     function deposit(uint256 _pid, uint256 _amount)
200         public
201         validatePoolByPid(_pid)
202         nonReentrant
203     {
204         PoolInfo storage pool = poolInfo[_pid];
205         UserInfo storage user = userInfo[_pid][msg.sender];
206         updatePool(_pid);
207         if (user.amount > 0) {
208             uint256 pending = user
209                 .amount
210                 .mul(pool.accRosellePerShare)
211                 .div(1e12)
212                 .sub(user.rewardDebt);
213             if (pending > 0) {
214                 userTokens[msg.sender][_pid] += pending;
215             }
216         }
217         if (_amount > 0) {
218             pool.lpToken.safeTransferFrom(
219                 address(msg.sender),
220                 address(this),
221                 _amount
222             );
223             user.amount = user.amount.add(_amount);
224         }
225         user.rewardDebt = user.amount.mul(pool.accRosellePerShare).div(1e12);
226         emit Deposit(msg.sender, _pid, _amount);
227     }
228
```

Figure 12 The source code of *deposit* function

```
230     function withdraw(uint256 _pid, uint256 _amount)
231         public
232         validatePoolByPid(_pid)
233         nonReentrant
234     {
235         PoolInfo storage pool = poolInfo[_pid];
236         UserInfo storage user = userInfo[_pid][msg.sender];
237         require(user.amount >= _amount, "withdraw: not good");
238         updatePool(_pid);
239         uint256 pending = user
240             .amount
241             .mul(pool.accRosellePerShare)
242             .div(1e12)
243             .sub(user.rewardDebt);
244         if (pending > 0) {
245             userTokens[msg.sender][_pid] += pending;
246         }
247         if (_amount > 0) {
248             user.amount = user.amount.sub(_amount);
249             pool.lpToken.safeTransfer(address(msg.sender), _amount);
250         }
251         user.rewardDebt = user.amount.mul(pool.accRosellePerShare).div(1e12);
252         emit Withdraw(msg.sender, _pid, _amount);
253     }
```

Figure 13 The source code of *withdraw* function

Recommendations	It is recommended to set a global variable to record the total reward that the user has not received each time the user calls the *withdraw*, *deposit* and *takeToken* functions. After a period of time (such as a week) after the end of the staking period, if the balance of the contract reward token minus the total reward not received is greater than 0, the owner can claim this part of the reward.
Status	Acknowledged.

[Farm-7] Missing event trigger

Severity Level	Info
Type	Coding Conventions
Lines	Farm.sol #L280-284, 274-279, 285-290, 109-125, 84-107
Description	Events are not triggered in *dao*, *open*, *takeToken*, *setadd*, *set*, *add* functions.

```
280     function open() external onlyOwner {
281         require(!opened, "opened");
282         opened = true;
283     }
284
```

Figure 14 The source code of *open* function

```
274     // Update dao address by the previous dao.
275     function dao(address _daoaddr) public {
276         require(msg.sender == daoaddr, "dao: wut?");
277         daoaddr = _daoaddr;
278     }
279
```

Figure 15 The source code of *dao* function

```
285     function takeToken(uint256 _pid) external validatePoolByPid(_pid) {
286         require(opened, "closed");
287         safeRoselleTransfer(msg.sender, userTokens[msg.sender][_pid]);
288         userTokens[msg.sender][_pid] = 0;
289     }
290 }
```

Figure 16 The source code of *takeToken* function

```
109 function set(
110     uint256 _pid,
111     uint256 _allocPoint,
112     bool _withUpdate
113 ) public validatePoolByPid(_pid) {
114     require(
115         owner() == _msgSender() || daoaddr == _msgSender(),
116         "Ownable: caller is not the owner"
117     );
118     if (_withUpdate) {
119         massUpdatePools();
120     }
121     totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
122         _allocPoint
123     );
124     poolInfo[_pid].allocPoint = _allocPoint;
125 }
```

Figure 17 The source code of *set* function

```
84    function add(
85        uint256 _allocPoint,
86        IERC20 _lpToken,
87        bool _withUpdate
88    ) public onlyOwner {
89        require(!poolAdded[address(_lpToken)], "duplicate poll");
90        poolAdded[address(_lpToken)] = true;
91        if (_withUpdate) {
92            massUpdatePools();
93        }
94        uint256 lastRewardBlock = block.number > startBlock
95            ? block.number
96            : startBlock;
97        totalAllocPoint = totalAllocPoint.add(_allocPoint);
98        poolInfo.push(
99            PoolInfo({
100               lpToken: _lpToken,
101               allocPoint: _allocPoint,
102               lastRewardBlock: lastRewardBlock,
103               accRosellePerShare: 0
104           })
105       );
106   }
107
```

Figure 18 The source code of *add* function

Recommendations	It is recommended to add related events and trigger them in the corresponding function.
Status	Acknowledged.

[Nodes-1] Fake recharge risk

Severity Level	**Low**
Type	General Vulnerability
Lines	Nodes.sol #L94-112, 113-123, 179-201, 203-220
Description	When users stake and withdraw fon and fonos tokens through the contract, the *transfer* and *transferFrom* functions are used, and there is a risk of fake recharge.

```
94  function reg(string memory name, string memory url) public nonReentrant {
95      address owner = msg.sender;
96      require(!nodeAdded[owner], "duplicate node");
97      nodeAdded[owner] = true;
98      fon.transferFrom(owner, address(this), 9999 * 1e18);
99      uint256 index = nodeInfo.length;
100     nodeInfo.push(NodeInfo({
101         owner: owner,
102         name: name,
103         url: url,
104         isActive: true,
105         totalStaked: 0,
106         rank: MAX,
107         index: index
108     }));
109     nodeIndexFromOwner[owner] = index;
110     emit Reg(owner, name, url);
111 }
112
```

Figure 19 The source code of *reg* function

```
113     // unreg node
114     function unreg() public nonReentrant {
115         NodeInfo storage node = nodeInfo[getNodeIndexFromAddress()];
116         require(node.isActive, "not active");
117         require(node.owner == msg.sender, "only owner");
118         fon.transfer(msg.sender, 9999 * 1e18);
119         node.isActive = false;
120         rankDel(node.rank);
121         emit Close(msg.sender);
122     }
123
```

Figure 20 The source code of *unreg* function

```
179 function deposit(uint256 _nid, uint256 _amount) public validateNodeByPid(_nid) nonReentrant {
180     NodeInfo storage node = nodeInfo[_nid];
181     UserInfo storage user = userInfo[_nid][msg.sender];
182     if (_amount > 0) {
183         require(node.isActive, "node now is not active");
184         if(!userDeposited[msg.sender][_nid]){
185             userDeposited[msg.sender][_nid] = true;
186             userDepositedIndex[msg.sender].push(_nid);
187         }
188         fonos.transferFrom(address(msg.sender), address(this), _amount);
189         user.amount = user.amount.add(_amount);
190         if(node.totalStaked == 0){
191             require(ranking.length <= 99, "slot is full");
192             node.rank = ranking.length;
193             ranking.push(node.index);
194         }
195         node.totalStaked = node.totalStaked.add(_amount);
196         rankUp(node.rank);
197         emit Deposit(msg.sender, _nid, _amount);
198         emit CheckPoint(_nid, node.totalStaked);
199     }
200 }
201
```

Figure 21 The source code of *deposit* function

```
203     function withdraw(uint256 _nid, uint256 _amount) public validateNodeByPid(_nid) nonReentrant {
204         NodeInfo storage node = nodeInfo[_nid];
205         UserInfo storage user = userInfo[_nid][msg.sender];
206         require(user.amount >= _amount, "withdraw: not good");
207         if(_amount > 0) {
208             user.amount = user.amount.sub(_amount);
209             fonos.transfer(address(msg.sender), _amount);
210             node.totalStaked = node.totalStaked.sub(_amount);
211             if(node.totalStaked == 0){
212                 rankDel(node.rank);
213             }
214             else
215                 rankDown(node.rank);
216             emit Withdraw(msg.sender, _nid, _amount);
217             emit CheckPoint(_nid, node.totalStaked);
218         }
219     }
220
```

Figure 22 The source code of *withdraw* function

Recommendations	It is recommended to use the *safeTransfer* and *safeTransferFrom* functions for token transfers.

Status	Fixed

```
96      function reg(string memory name, string memory url) public nonReentrant {
97          address owner = msg.sender;
98          require(!nodeAdded[owner], "duplicate node");
99          nodeAdded[owner] = true;
100         fon.safeTransferFrom(owner, address(this), 9999 * 1e18);
101         uint256 index = nodeInfo.length;
102         nodeInfo.push(NodeInfo({
103             owner: owner,
104             name: name,
105             url: url,
106             isActive: true,
107             totalStaked: 0,
108             rank: MAX,
109             index: index
110         }));
111         nodeIndexFromOwner[owner] = index;
112         emit Reg(owner, name, url);
113     }
114
115     // unreg node
116     function unreg() public nonReentrant {
117         NodeInfo storage node = nodeInfo[getNodeIndexFromAddress()];
118         require(node.isActive, "not active");
119         require(node.owner == msg.sender, "only owner");
120         fon.safeTransfer(msg.sender, 9999 * 1e18);
121         node.isActive = false;
122         rankDel(node.rank);
123         emit Close(msg.sender);
124     }
```

Figure 23 The source code of *reg* and *unreg* function(Fixed)

```
186     function deposit(uint256 _nid, uint256 _amount) public validateNodeByPid(_nid) nonReentrant {
187         NodeInfo storage node = nodeInfo[_nid];
188         UserInfo storage user = userInfo[_nid][msg.sender];
189         if (_amount > 0) {
190             require(node.isActive, "node now is not active");
191             if(!userDeposited[msg.sender][_nid]){
192                 userDeposited[msg.sender][_nid] = true;
193                 userDepositedIndex[msg.sender].push(_nid);
194             }
195             fonos.safeTransferFrom(address(msg.sender), address(this), _amount);
196             user.amount = user.amount.add(_amount);
197             if(node.totalStaked == 0){
198                 require(ranking.length <= 99, "slot is full");
199                 node.rank = ranking.length;
200                 ranking.push(node.index);
201             }
202             node.totalStaked = node.totalStaked.add(_amount);
203             rankUp(node.rank);
204             emit Deposit(msg.sender, _nid, _amount);
205             emit CheckPoint(_nid, node.totalStaked);
206         }
207     }
208
209     // Withdraw FRT tokens from Node.
210     function withdraw(uint256 _nid, uint256 _amount) public validateNodeByPid(_nid) nonReentrant {
211         NodeInfo storage node = nodeInfo[_nid];
212         UserInfo storage user = userInfo[_nid][msg.sender];
213         require(user.amount >= _amount, "withdraw: not good");
214         if(_amount > 0) {
215             user.amount = user.amount.sub(_amount);
216             fonos.safeTransfer(address(msg.sender), _amount);
217             node.totalStaked = node.totalStaked.sub(_amount);
218             if(node.totalStaked == 0){
219                 rankDel(node.rank);
220             }
221             else
222                 rankDown(node.rank);
223             emit Withdraw(msg.sender, _nid, _amount);
224             emit CheckPoint(_nid, node.totalStaked);
225         }
226     }
227
```

Figure 24 The source code of *deposit* and *withdraw* function(Fixed)

[Nodes-2] Missing event trigger

Severity Level	Info
Type	Coding Conventions
Lines	Nodes.sol #L168-177
Description	The *rankDel* function missing event trigger.

```
168     function rankDel(uint256 index) internal{
169         if(index == MAX) return;
170         nodeInfo[ranking[index]].rank = MAX;
171         for(uint256 i = index; i < ranking.length - 1; ++ i){
172             ranking[i] = ranking[i + 1];
173             nodeInfo[ranking[i]].rank = i;
174         }
175         ranking.pop();
176     }
177
```

Figure 25 The source code of *rankDel* function

Recommendations	It is recommended to add related events and trigger them in the corresponding function.
Status	Acknowledged.

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact / Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in Blockchain.

3.4 About BEOSIN

Affiliated to BEOSIN Technology Pte. Ltd., BEOSIN is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions.BEOSIN has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, BEOSIN has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com