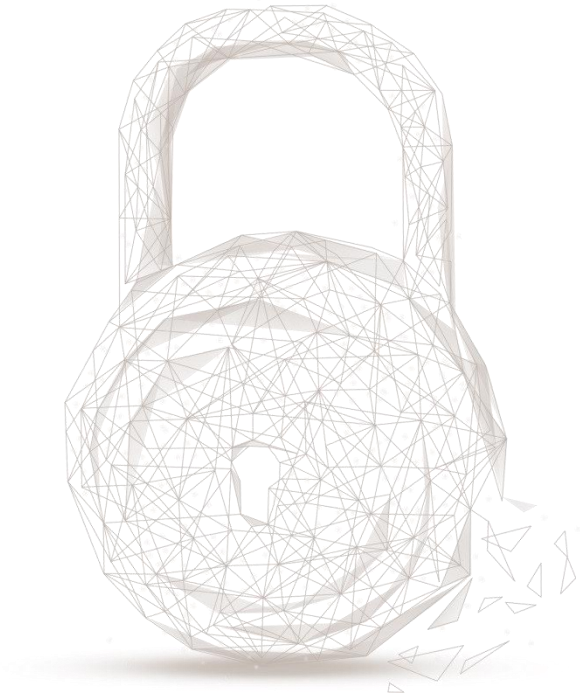
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202101221139

Report Query Name: POC

Smart Contract Info：

Smart Contract Name	Smart Contract Address	Smart Contract Address Link
POCBNBPool	Fill in after deployment	Fill in after deployment
POCDAIPool	Fill in after deployment	Fill in after deployment
POCDUCKPool	Fill in after deployment	Fill in after deployment
POCESDPool	Fill in after deployment	Fill in after deployment
POCFIREPool	Fill in after deployment	Fill in after deployment
POCFRAXPool	Fill in after deployment	Fill in after deployment
POCINJPool	Fill in after deployment	Fill in after deployment
POCPOLSPool	Fill in after deployment	Fill in after deployment
POCRAMPPool	Fill in after deployment	Fill in after deployment
POCUSDCPool	Fill in after deployment	Fill in after deployment
POCUSDTPool	Fill in after deployment	Fill in after deployment
POCUSDXPool	Fill in after deployment	Fill in after deployment
USDCPOCLPTokenSharePool	Fill in after deployment	Fill in after deployment
USDCPOSLPTokenSharePool	Fill in after deployment	Fill in after deployment

Start Date：2021.01.20

Completion Date：2021.01.22

Overall Result：Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass
		Access Control of Owner	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

Note: Audit results and suggestions in code comments

Disclaimer: This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contracts project POC, including Coding Standards, Security, and Business Logic. **The POC project passed all audit items. The overall result is Pass (Distinction).** The smart contract is able to function properly.

Audit Contents:

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing ETH.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.

- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract. In this project, the contract
- Result: Pass

2.10 Replay Attack

- Description: Check whether the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

In this project, fourteen "stake rewards" smart contracts were implemented based on the same code structure, namely POCBNBPool, POCDAIPool, POCDUCKPool, POCESDPool, POCFIREPool, POCFRAXPool, POCINJPool, POCPOLSPool, POCRAMPPool, POCUSDCPool, POCUSDTPool, and POCUSDXPool. The code logic of each smart contract implementation code is the same except for the name and the number of judgment newDeposits. Then there are two contracts, USDCPOCLPTokenSharePool and

USDCPOSLPTokenSharePool, which are somewhat different from the above logic. The following screenshots are based on POCBNBPool, USDCPOCLPTokenSharePool and USDCPOSLPTokenSharePool.

3.1 Business analysis of Contract POCBNBPool

(1) Stake initialization function

- Description: As shown in the figure below. The "stake-reward" mode of the contract needs to initialize the relevant parameters (reward ratio *rewardRate*, first update time *lastUpdateTime*, phase completion time *periodFinish*), call the *notifyRewardAmount* function through the specified reward distribution administrator address *rewardDistribution*, and enter the initial reward used to calculate the reward ratio value reward, initialize the stake and reward related parameters. This function can be called by the designated address *rewardDistribution* at any time to control the reward ratio (the reward ratio can also be modified before the stake starts). If the value is too small, the user's income will not match the expectation.

```
175     function notifyRewardAmount(uint256 reward)
176         external
177         override
178         onlyRewardDistribution
179         updateReward(address(0))
180     {
181         if (block.timestamp > starttime) {
182             if (block.timestamp >= periodFinish) {
183                 rewardRate = reward.div(DURATION);
184             } else {
185                 uint256 remaining = periodFinish.sub(block.timestamp);
186                 uint256 leftover = remaining.mul(rewardRate);
187                 rewardRate = reward.add(leftover).div(DURATION);
188             }
189             lastUpdateTime = block.timestamp;
190             periodFinish = block.timestamp.add(DURATION);
191             emit RewardAdded(reward);
192         } else {
193             rewardRate = reward.div(DURATION);
194             lastUpdateTime = starttime;
195             periodFinish = starttime.add(DURATION);
196             emit RewardAdded(reward);
197         }
198     }
199 }
```

Figure 1 source code of *notifyRewardAmountc*(POCBNBPool contract)

- Related functions: *notifyRewardAmount*
- Result: Pass

(2) Withdrawal of staked tokens

- Description: As shown in the figure below, the contract implements the *withdraw* function to withdraw the staked tokens. By calling the *safetransfer* function in the ERC20 contract, the contract address

transfers the specified amount of ERC20 tokens to the function caller (user) address, and update the deposits of the caller; this function restricts the user to call after the stake-reward mode is turned on (when the specified time is reached); each time the function is called to stake tokens, the reward-related data is updated through the modifier *updateReward*; and the modifier *checkStart* is used for each call to check whether the phase completion time is reached.

```
148     function withdraw(uint256 amount)
149         public
150         override
151         updateReward(msg.sender)
152         checkStart
153     {
154         require(amount > 0, 'POCBNBPool: Cannot withdraw 0');
155         deposits[msg.sender] = deposits[msg.sender].sub(amount);
156         super.withdraw(amount);
157         emit Withdrawn(msg.sender, amount);
158     }
```

Figure 2 source code of *withdraw*(POCBNBPool contract)

```
51      function withdraw(uint256 amount) public virtual {
52          _totalSupply = _totalSupply.sub(amount);
53          _balances[msg.sender] = _balances[msg.sender].sub(amount);
54          BNB.safeTransfer(msg.sender, amount);
55      }
56  }
```

Figure 3 source code of *withdraw*(BNBWrapper contract)

- Related functions: *withdraw*, *rewardPerToken*, *lastTimeRewardApplicable*
- Result: Pass

(3) Stake function

- Description: As shown in the figure below, the contract implements the stake function to stake ERC20 tokens. The user pre-approve the contract address. By calling the *safeTransferFrom* function in the ERC20 contract, the contract address transfers the specified amount of ERC20 tokens to the contract address on behalf of the user; this function limits the user to call this function only after the "stake-reward" mode is turned on (when the specified time is reached); each time the function is called to deposit tokens(note the maximum deposit amount limit here), the reward-related data is updated through the modifier *updateReward*; and the modifier *checkStart* is used for each call to check whether the phase completion time is reached.

```
131     function stake(uint256 amount)
132         public
133         override
134         updateReward(msg.sender)
135         checkStart
136     {
137         require(amount > 0, 'POCBNBPool: Cannot stake 0');
138         uint256 newDeposit = deposits[msg.sender].add(amount);
139         require(
140             newDeposit <= 20000e18,
141             'POCBNBPool: deposit amount exceeds maximum 20000'
142         );
143         deposits[msg.sender] = newDeposit;
144         super.stake(amount);
145         emit Staked(msg.sender, amount);
146     }
147
```

Figure 4 source code of *stake* (POCBNBPool contract)

```
94      modifier updateReward(address account) {
95          rewardPerTokenStored = rewardPerToken();
96          lastUpdateTime = lastTimeRewardApplicable();
97          if (account != address(0)) {
98              rewards[account] = earned(account);
99              userRewardPerTokenPaid[account] = rewardPerTokenStored;
100         }
101         _;
102     }
```

Figure 5 source code of *updateReward*(POCBNBPool contract)

```
89      modifier checkStart() {
90          require(block.timestamp >= starttime, 'POCBNBPool: not start');
91          _;
92      }
93
```

Figure 6 source code of *checkStart*(POCBNBPool contract)

```
45      function stake(uint256 amount) public virtual {
46          _totalSupply = _totalSupply.add(amount);
47          _balances[msg.sender] = _balances[msg.sender].add(amount);
48          BNB.safeTransferFrom(msg.sender, address(this), amount);
49      }
50
```

Figure 7 source code of *stake*(BNBWrapper contract)

- Related functions: *stake*, *rewardPerToken*, *lastTimeRewardApplicable*
- Result: Pass

(4) Get reward function

- Description: As shown in the figure below, the contract implements the *getReward* function to receive stake rewards. By calling the *safeTransfer* function in the BNBWrapper contract, the contract address transfers the specified number of ERC20 tokens (The user receives 91% of the reward and remaining reward is sent to the devAddr address) to the function caller (user) address and devAddr address; This function restricts the user to call only after the "stake-reward" mode is turned on (the specified time is reached); each time this function is called to stake tokens, the reward related data is updated through the modifier *updateReward*; and each call is through the modifier *checkStart* Check whether the phase completion time is reached.

```
165     function getReward() public updateReward(msg.sender) checkStart {
166         uint256 reward = earned(msg.sender);
167         if (reward > 0) {
168             rewards[msg.sender] = 0;
169             polkaCash.safeTransfer(msg.sender, reward.mul(91).div(100));
170             polkaCash.safeTransfer(devAddr, reward.mul(9).div(100));
171             emit RewardPaid(msg.sender, reward);
172         }
173     }
174
```

Figure 8 source code of *getReward*(POCBNBPool contract)

- Related functions: *getReward, earned*
- Result: Pass

(5) Exit function

- Description: As shown in the figure below, the contract implements the *exit* function for the caller to withdraw from the stake, call the *withdraw* function to extract all staked ERC20 tokens, call the *getReward* function to receive the caller's stake reward (The user receives 91% of the reward and remaining reward is sent to the devAddr address), and end the participation in the "stake-reward" mode. At this time, the user address cannot obtain new stake rewards because the amount of staked ERC20 tokens is empty.

```
159
160     function exit() external {
161         withdraw(balanceOf(msg.sender));
162         getReward();
163     }
164
```

Figure 9 source code of *exit*(POCBNBPool contract)

- Related functions: *exit, withdraw, getReward*

- Result: Pass

(6) Reward related data query function

- Description: As shown in the figure below, contract users can query the earliest time stamp between the current time stamp and the phase completion time by calling the *lastTimeRewardApplicable* function; calling the *rewardPerToken* function can query the stake rewards available for each stake token; calling the *earned* function can query the total stake rewards obtained by the specified address.

```
88
89      modifier checkStart() {
90          require(block.timestamp >= starttime, 'POCBNBPool: not start');
91          _;
92      }
93
94      modifier updateReward(address account) {
95          rewardPerTokenStored = rewardPerToken();
96          lastUpdateTime = lastTimeRewardApplicable();
97          if (account != address(0)) {
98              rewards[account] = earned(account);
99              userRewardPerTokenPaid[account] = rewardPerTokenStored;
100         }
101         _;
102     }
103
104     function lastTimeRewardApplicable() public view returns (uint256) {
105         return Math.min(block.timestamp, periodFinish);
106     }
107
108     function rewardPerToken() public view returns (uint256) {
109         if (totalSupply() == 0) {
110             return rewardPerTokenStored;
111         }
112         return
113             rewardPerTokenStored.add(
114                 lastTimeRewardApplicable()
115                     .sub(lastUpdateTime)
116                     .mul(rewardRate)
117                     .mul(1e18)
118                     .div(totalSupply())
119             );
120     }
121
122     function earned(address account) public view returns (uint256) {
123         return
124             balanceOf(account)
125                 .mul(rewardPerToken().sub(userRewardPerTokenPaid[account]))
126                 .div(1e18)
127                 .add(rewards[account]);
128     }
129
```

Figure 10 source code of *lastTimeRewardApplicable*, *rewardPerToken* and *earned*(POCBNBPool contract)

- Related functions: *lastTimeRewardApplicable*, *rewardPerToken*, *earned*, *balanceOf*, *totalSupply*

- Result: Pass

3.2 Business analysis of Contract USDCPOSLPTokenSharePool

(7) Stake function and withdraw function

- Description: As shown in the figure below, unlike the previous POCBNBPool contract, there is no limit to the amount of *stake* function tokens.

```
108     function stake(uint256 amount)
109         public
110         override
111         updateReward(msg.sender)
112         checkStart
113     {
114         require(amount > 0, 'USDCPOSLPTokenSharePool: Cannot stake 0');
115         super.stake(amount);
116         emit Staked(msg.sender, amount);
117     }
118
119     function withdraw(uint256 amount)
120         public
121         override
122         updateReward(msg.sender)
123         checkStart
124     {
125         require(amount > 0, 'USDCPOSLPTokenSharePool: Cannot withdraw 0');
126         super.withdraw(amount);
127         emit Withdrawn(msg.sender, amount);
128     }
129
```

Figure 11 source code of *stake* and *withdraw*(USDCPOSLPTokenSharePool contract)

- Related functions: *stake*, *withdraw*
- Result: Pass

3.3 Business analysis of Contract USDCPOCLPTokenSharePool

(8) Stake initialization function

- Description: As shown in the figure below, the difference from the POCBNBPool contract is that when the parameters are initialized, if block.timestamp does not reach starttime. the rewardrate will not be changed when inputing parameter reward, and will make the rewardrate a fixed value.

```
153
154     function notifyRewardAmount(uint256 reward)
155         external
156         override
157         onlyRewardDistribution
158         updateReward(address(0))
159     {
160         if (block.timestamp > starttime) {
161             if (block.timestamp >= periodFinish) {
162                 rewardRate = reward.div(DURATION);
163             } else {
164                 uint256 remaining = periodFinish.sub(block.timestamp);
165                 uint256 leftover = remaining.mul(rewardRate);
166                 rewardRate = reward.add(leftover).div(DURATION);
167             }
168             lastUpdateTime = block.timestamp;
169             periodFinish = block.timestamp.add(DURATION);
170             emit RewardAdded(reward);
171         } else {
172             rewardRate = initreward.div(DURATION);
173             lastUpdateTime = starttime;
174             periodFinish = starttime.add(DURATION);
175             emit RewardAdded(reward);
176         }
177     }
178 }
179
```

Figure 12 source code of *notifyRewardAmount*(USDCPOCLPTokenSharePool contract)

- Related functions: *notifyRewardAmount*
- Result: Pass

(9) Functions that have used checkhalve

- Description: As shown in the figure below, unlike the POCBNBPool contract, a new modifier *checkhalve* is added to determine whether the current time is greater than the end of the phase. If it is greater than the periodFinish, the rewardRate will be reduced and the periodFinish will be updated.

```
 99     function stake(uint256 amount)
100         public
101         override
102         updateReward(msg.sender)
103         checkhalve
104         checkStart
105     {
106         require(amount > 0, 'Cannot stake 0');
107         super.stake(amount);
108         emit Staked(msg.sender, amount);
109     }
110
```

Figure 13 source code of *stake*(USDCPOCLPTokenSharePool contract)

```
111     function withdraw(uint256 amount)
112         public
113         override
114         updateReward(msg.sender)
115         checkhalve
116         checkStart
117     {
118         require(amount > 0, 'Cannot withdraw 0');
119         super.withdraw(amount);
120         emit Withdrawn(msg.sender, amount);
121     }
122
```

Figure 14 source code of *withdraw*(USDCPOCLPTokenSharePool contract)

```
8   function getReward() public updateReward(msg.sender) checkhalve checkStart {
9       uint256 reward = earned(msg.sender);
0       if (reward > 0) {
1           rewards[msg.sender] = 0;
2           polkaShare.safeTransfer(msg.sender, reward.mul(91).div(100));
3           polkaShare.safeTransfer(devAddr, reward.mul(9).div(100));
4           emit RewardPaid(msg.sender, reward);
5       }
6   }
```

Figure 15 source code of *getReward*(USDCPOCLPTokenSharePool contract)

```
137
138     modifier checkhalve() {
139         if (block.timestamp >= periodFinish) {
140             initreward = initreward.mul(80).div(100);
141
142             rewardRate = initreward.div(DURATION);
143             periodFinish = block.timestamp.add(DURATION);
144             emit RewardAdded(initreward);
145         }
146         _;
147     }
```

Figure 16 source code of *checkhalve*(USDCPOCLPTokenSharePool contract)

- Related functions: *getReward, earned, withdraw, stake, safeTransfer*
- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts project POC. The problems found by the audit team during the audit process have been notified to the project party and fixed, the overall audit result of the POC project's smart contract is **Pass**.

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com