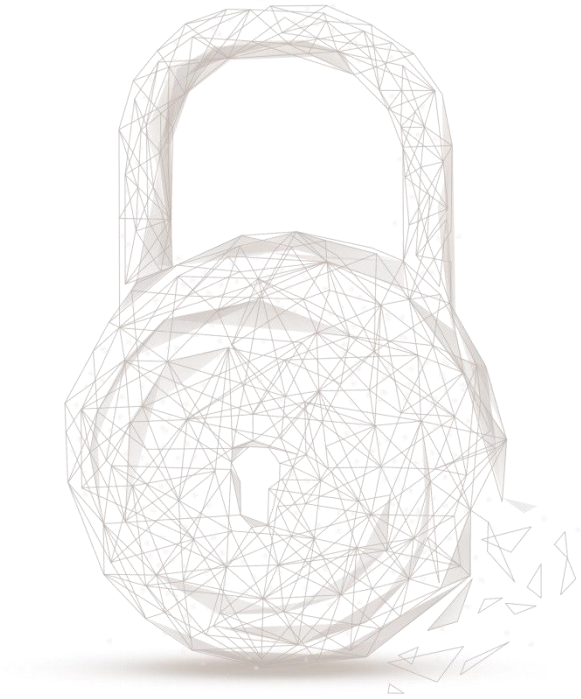
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number: 202107151412

Report Query Name: CakeAccruingStrategy

Smart Contract Address Link:

https://github.com/RAMP-DEFI/ramp-protocol/blob/development/contracts/strategies/bsc/CakeAccruingStrategy.sol

Commit Hash:

Start: b9570ebdee7802dad063cee2c5fd800b3c445a39

Final: 04b22f9385408897585f558369c6f5131bd95507

Start Date: 2021.06.11

Completion Date: 2021.07.15

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

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 contract CakeswapAccruingStrategy, including Coding Standards, Security, and Business Logic. **The CakeswapAccruingStrategy contract passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

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 BNB.
- 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.
- 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

This contract is a strategic contract of vault and bank contracts, used to specify assets to invest in pancake projects.

(1) onDeposit

- Description: This function is used to trigger when the vault contract deposits the specified asset. The vault contract will trigger this function to invest. This function will send the funds to masterChef contract for investment.

```
93      function onDeposit(
94          address _token,
95          address, // _account is not used in Accrueing strategies (no per-user ledger)
96          uint256 _amount
97      ) external override onlyVault {
98          // Checks
99          require(_token == address(cake), "Wrong token!");
100
101         cake.approve(address(masterChef), _amount);
102
103         // Reinvest _amount in Masterchef
104         masterChef.enterStaking(_amount);
105
106         // TODO We have now received Syrup. We could do something profitable with that too.
107
108         emit StrategyDeposit(_amount);
109     }
```

Figure 1 Source code of function *onDeposit*

- Related functions: *approve*, *enterStaking*
- Result: Pass

(2) onWithdraw

- Description: This function is used to trigger when the vault contract withdraws money. The vault contract specifies the recipient and the withdrawal amount. This contract will take out the corresponding amount of funds from the invested project and send it to the user.

```
112     function onWithdraw(
113         address,
114         address _account,
115         uint256 _amount
116     ) external override onlyVault {
117         // NOTE: We are NOT running "update" here, the Vault does this already.
118
119         // Remove _amount from MasterChef. Syrup is being burned, and Cake is received.
120         masterChef.leaveStaking(_amount);
121
122         // Calculate withdrawal fees
123         uint256 fee = _amount.mul(withdrawalFeePercentage).div(100);
124
125         // Calculate net amount paid to user
126         uint256 netAmount = _amount.sub(fee);
127
128         // Send _amount to user
129         cake.safeTransfer(_account, netAmount);
130
131         // Transfer fee to treasury
132         if (fee > 0) cake.safeTransfer(devAddress, fee);
133
134         emit StrategyWithdraw(_account, netAmount, fee);
135     }
```

Figure 2 Source code of function *onWithdraw*

- Related functions: *leaveStaking*, *safeTransfer*
- Result: Pass

(3) onLiquidate

- Description: This function is used to trigger when the bank contract liquidate, the vault contract will specify the amount of funds withdrawn, and the contract will withdraw the funds according to the specified amount and send them to the vault contract.

```
75      function onLiquidate(
76          address,
77          address,
78          uint256 _amount
79      ) external override onlyVault {
80
81          // Remove _amount from MasterChef. Syrup is being burned, and Cake is received.
82          masterChef.leaveStaking(_amount);
83
84          // Send _amount to vault
85          cake.safeTransfer(vault, _amount);
86
87          // Emit event for logging
88          emit Liquidated(_amount);
89
90      }
```

Figure 3 Source code of function *onLiquidate*

- Related functions: *leaveStaking*, *safeTransfer*
- Result: Pass

(4) emergencyWithdraw

- Description: This function is used to trigger when the vault contract is withdrawn in an emergency. This function will withdraw all the investment and send it to the vault contract.

```
138     function emergencyWithdraw(address _token, bool _abandonRewards) external override onlyVault returns (uint256 lastPoolAmount) {
139         uint256 withdrawAmount;
140         uint256 syrupSent;
141 
142         update(_token);
143 
144         // Get the amount of Cake staked
145         (lastPoolAmount,) = masterChef.userInfo(MASTERCHEF_CAKE_POOL_ID, address(this));
146 
147         if (_abandonRewards) {
148             // Remove _amount from MasterChef. Syrup is being burned, and Cake is received.
149             masterChef.emergencyWithdraw(MASTERCHEF_CAKE_POOL_ID);
150 
151         } else {
152 
153             // Get the amount of Syrup held in the strategy. This is the max of Cake we can withdraw.
154             uint256 syrupBalance = syrup.balanceOf(address(this));
155 
156             if (syrupBalance > lastPoolAmount) {
157                 // Weird, we have more Syrup than we need. Send the extra to devAddress
158                 syrupSent = syrupBalance.sub(lastPoolAmount);
159                 if (!_abandonRewards) syrup.transfer(devAddress, syrupSent);
160 
161             }
162             // Either not enough syrup, or just enough
163             withdrawAmount = lastPoolAmount;
164 
165             // Remove _amount from MasterChef. Syrup is being burned, and Cake is received.
166             masterChef.leaveStaking(withdrawAmount);
167 
168         }
169 
170         // We'll send all the Cake in the strategy
171         uint256 cakeSent = cake.balanceOf(address(this));
172 
173         lastPoolAmount = cakeSent;
174 
175         // Send all Cake owned by Strategy to the vault
176         cake.safeTransfer(vault, cakeSent);
177 
178         emit EmergencyWithdraw(address(cake), cakeSent, syrupSent);
179     }
```

Figure 4 Source code of *emergencyWithdraw*

- Related functions: *update*, *leaveStaking*, *emergencyWithdraw*, *safeTransfer*
- Result: Pass

(5) work

- Description: Operator can call this function at any time. This function will redeem the Cake in contract for reinvestment.

```
190     function work(address _token) external onlyOperator override {
191         _harvest();
192     }
```

Figure 5 Source code of *work*

```
215     function _harvest() internal {
216         // leavestaking with 0 amount will effectively claim pending Cake rewards and send them to caller
217         masterChef.leaveStaking(0);
218 
219         // Check if we received anything more than dust
220         uint256 cakeHarvested = cake.balanceOf(address(this));
221 
222         // Apply fees on rewards
223         uint256 fee = cakeHarvested.mul(devFeePercentage).div(100);
224 
225         uint256 reInvestedAmount = cakeHarvested.sub(fee);
226 
227         // Approve the amount to be staked
228         cake.approve(address(masterChef), reInvestedAmount);
229 
230         // Reinvest into MasterChef
231         masterChef.enterStaking(reInvestedAmount);
232 
233         // Transfer fee to treasury
234         if (fee > 0) cake.safeTransfer(devAddress, fee);
235     }
```

Figure 6 Source code of *_harvest*

- Related functions: *safeTransfer,enterStaking,, approve, leaveStaking, balanceOf*
- Result: Pass

(6) update

- Description: Anyone can call this function at any time. This function will redeem the Cake in contract for reinvestment

```
182     function update(address) public override {
183 
184         // For this strategy the work to be done is the same as the public work()
185         //  function, so we call it
186         _harvest();
187     }
```

Figure 7 Source code of *update*

- Related functions: *_harvest*
- Result: Pass

4. Conclusion

Beosin(Chengdu LianAn) conducted a detailed audit on the design and code implementation of the smart contract CakeAccruingStrategy. The contract CakeAccruingStrategy passed all audit items, the overall audit result is **Pass.**

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com