BEOSIN
Blockchain Security

Smart contract security audit report

Audit Number: 202107151413

Report Query Name: CakeLpStrategy

Smart Contract Address Link:

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

Commit Hash:

Start: c992cc602bea8ea2bc6c24169134f7538f53c716

Final: 190c8053b5950383737402d8abaf4c5749fc6b7c

Start Date: 2021.06.21

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

Disclaimer: This report is made in response to the project code. No description, expression or wording in this report shall be construed as an endorsement, affirmation or confirmation of the project. 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 contract CakeLpStrategy, including Coding Standards, Security, and Business Logic. **The CakeLpStrategy 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

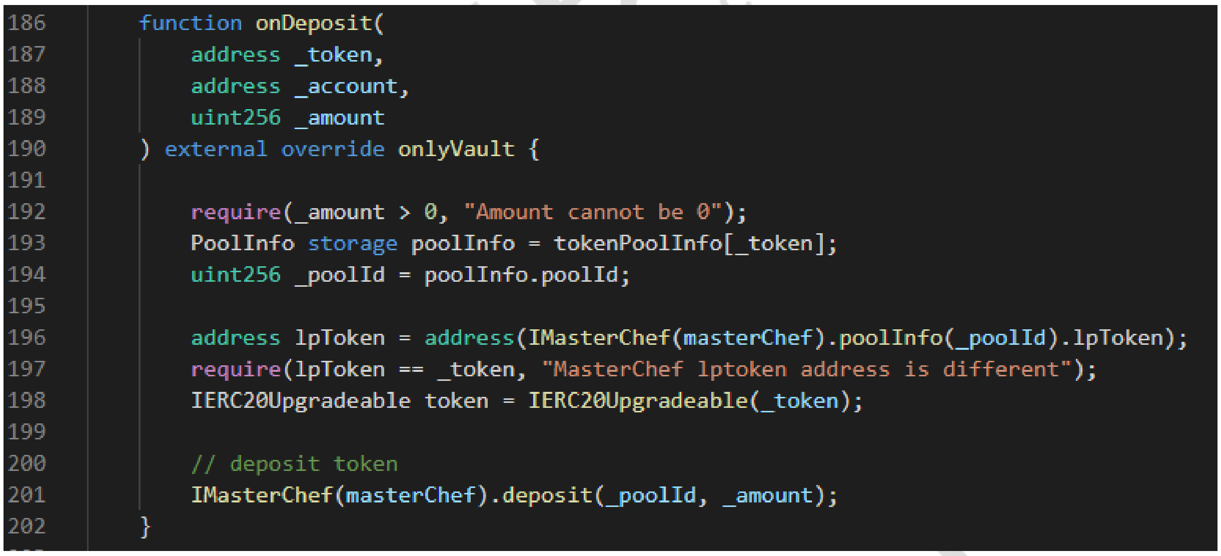
- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

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.



```
    function onDeposit(
        address _token,
        address _account,
        uint256 _amount
    ) external override onlyVault {

        require(_amount > 0, "Amount cannot be 0");
        PoolInfo storage poolInfo = tokenPoolInfo[_token];
        uint256 _poolId = poolInfo.poolId;

        address lpToken = address(IMasterChef(masterChef).poolInfo(_poolId).lpToken);
        require(lpToken == _token, "MasterChef lptoken address is different");
        IERC20Upgradeable token = IERC20Upgradeable(_token);

        // deposit token
        IMasterChef(masterChef).deposit(_poolId, _amount);
    }
```

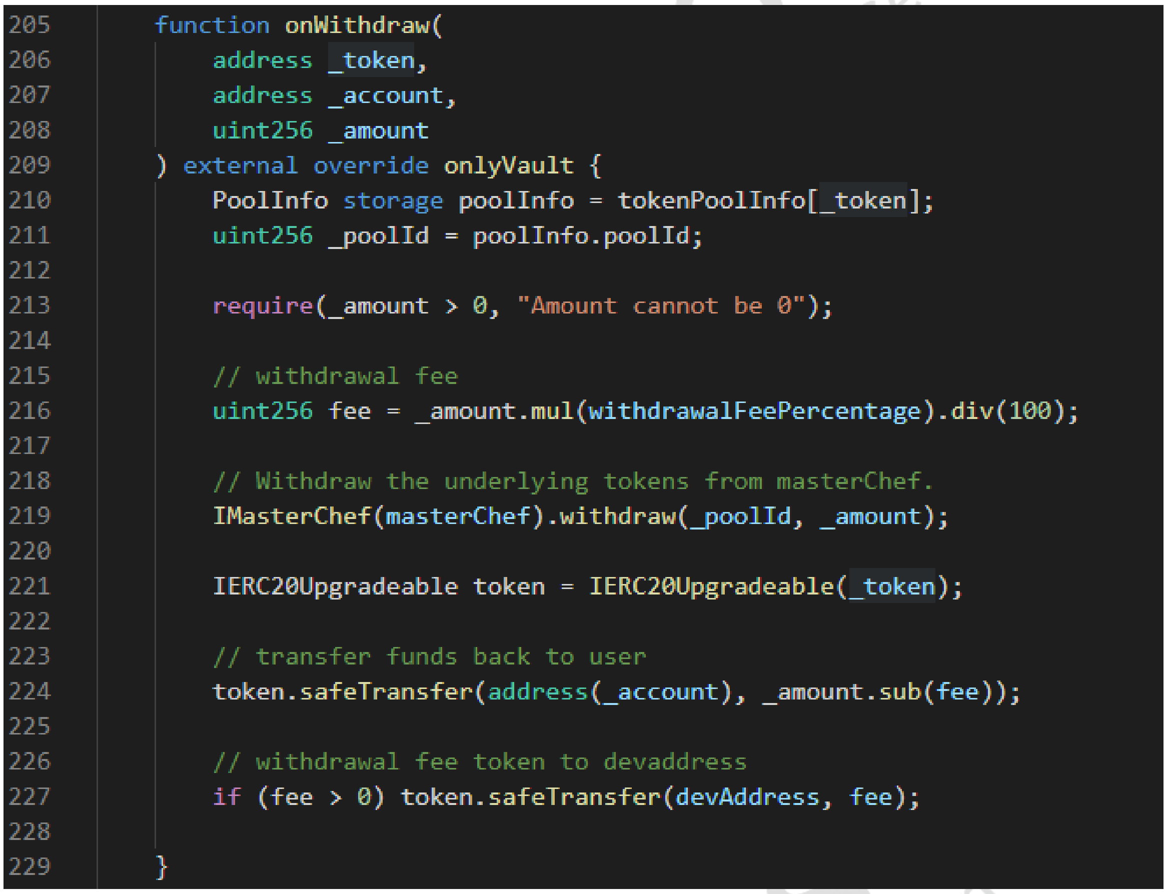
Figure 1 Source code of function *onDeposit*

● Related functions: *deposit*

● 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.



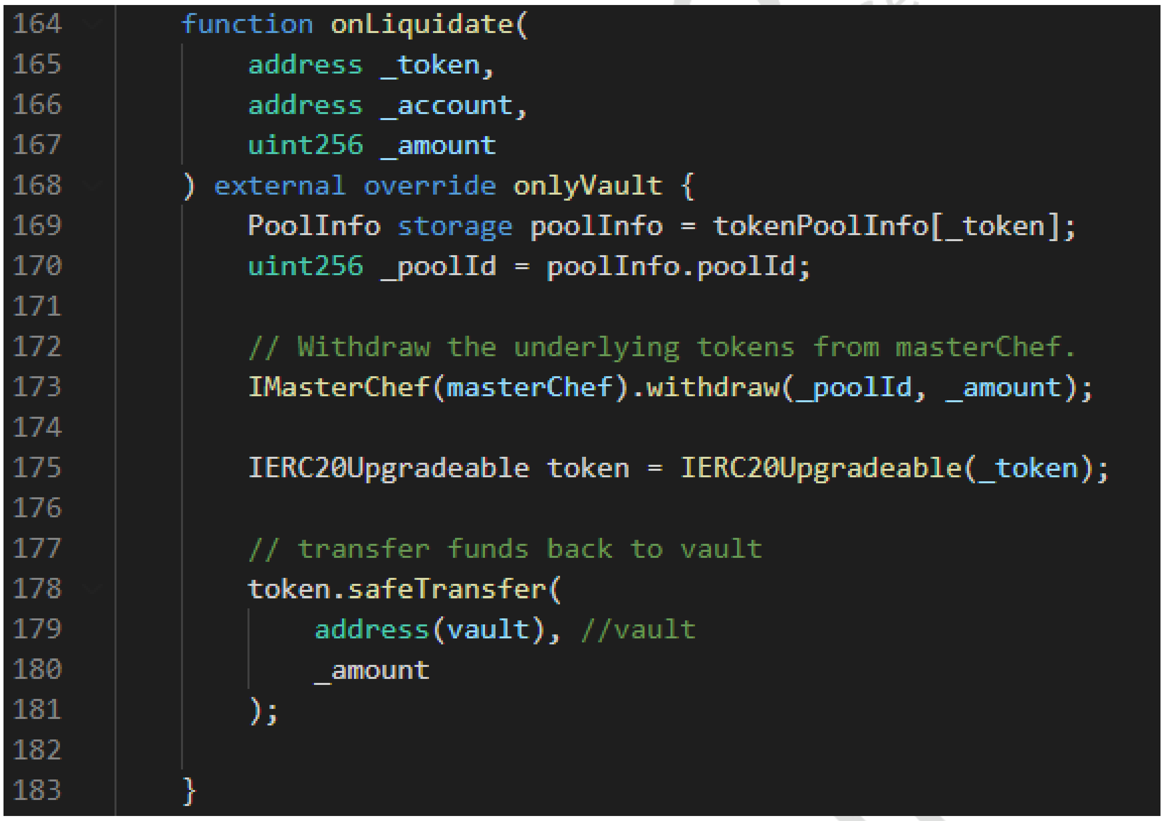
```
    function onWithdraw(
        address _token,
        address _account,
        uint256 _amount
    ) external override onlyVault {
        PoolInfo storage poolInfo = tokenPoolInfo[_token];
        uint256 _poolId = poolInfo.poolId;

        require(_amount > 0, "Amount cannot be 0");

        // withdrawal fee
        uint256 fee = _amount.mul(withdrawalFeePercentage).div(100);

        // Withdraw the underlying tokens from masterChef.
        IMasterChef(masterChef).withdraw(_poolId, _amount);

        IERC20Upgradeable token = IERC20Upgradeable(_token);

        // transfer funds back to user
        token.safeTransfer(address(_account), _amount.sub(fee));

        // withdrawal fee token to devaddress
        if (fee > 0) token.safeTransfer(devAddress, fee);

    }
```

Figure 2 Source code of function *onWithdraw*

- Related functions: *withdraw*, *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.



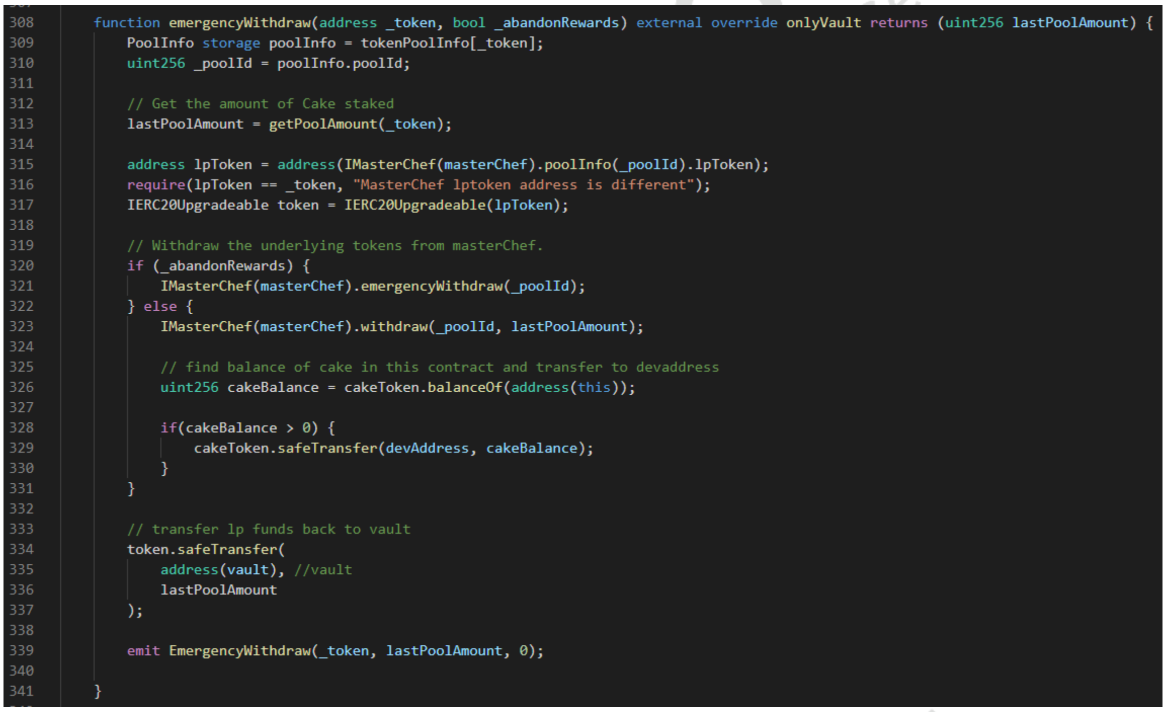
```solidity
    function onLiquidate(
        address _token,
        address _account,
        uint256 _amount
    ) external override onlyVault {
        PoolInfo storage poolInfo = tokenPoolInfo[_token];
        uint256 _poolId = poolInfo.poolId;

        // Withdraw the underlying tokens from masterChef.
        IMasterChef(masterChef).withdraw(_poolId, _amount);

        IERC20Upgradeable token = IERC20Upgradeable(_token);

        // transfer funds back to vault
        token.safeTransfer(
            address(vault), //vault
            _amount
        );

    }
```

Figure 3 Source code of function *onLiquidate*

- Related functions: *withdraw*, *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.



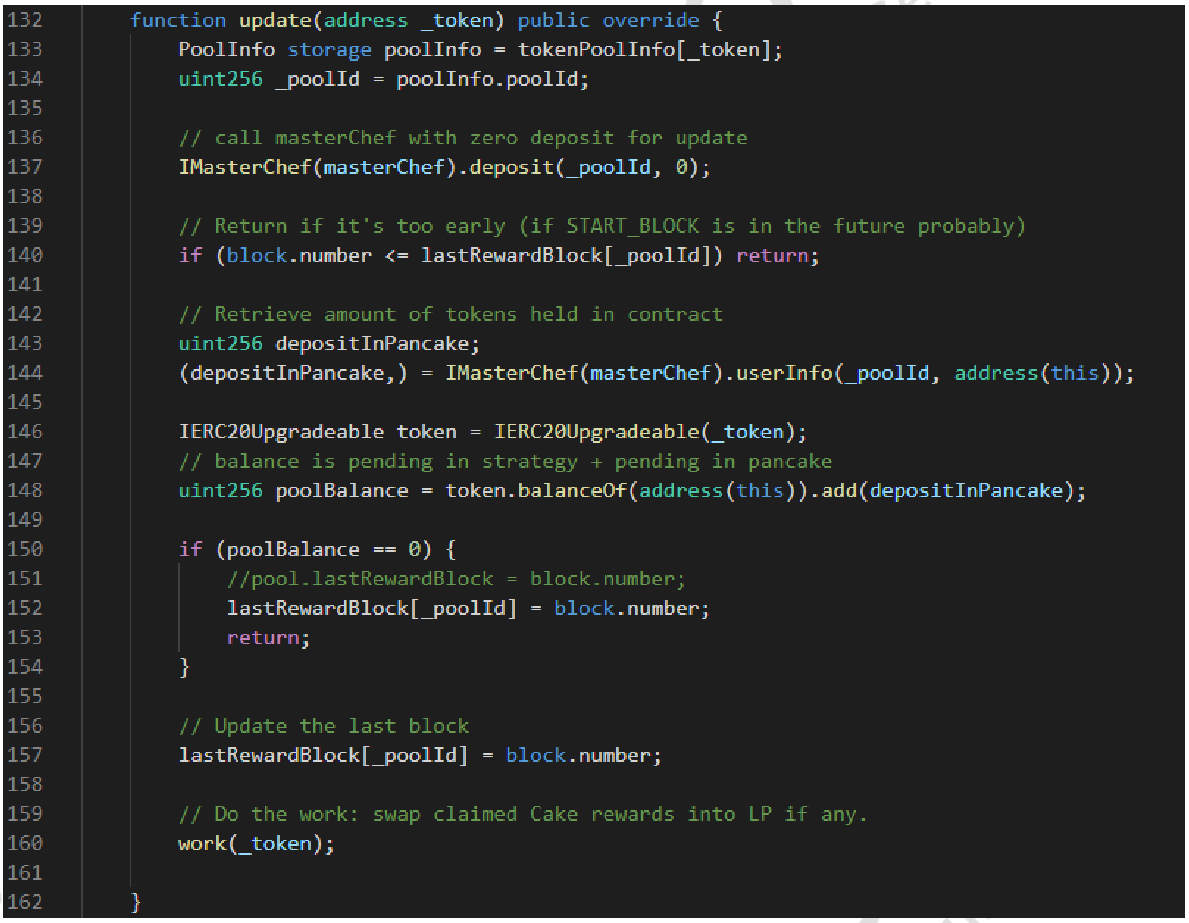
```solidity
    function emergencyWithdraw(address _token, bool _abandonRewards) external override onlyVault returns (uint256 lastPoolAmount) {
        PoolInfo storage poolInfo = tokenPoolInfo[_token];
        uint256 _poolId = poolInfo.poolId;

        // Get the amount of Cake staked
        lastPoolAmount = getPoolAmount(_token);

        address lpToken = address(IMasterChef(masterChef).poolInfo(_poolId).lpToken);
        require(lpToken == _token, "MasterChef lptoken address is different");
        IERC20Upgradeable token = IERC20Upgradeable(lpToken);

        // Withdraw the underlying tokens from masterChef.
        if (_abandonRewards) {
            IMasterChef(masterChef).emergencyWithdraw(_poolId);
        } else {
            IMasterChef(masterChef).withdraw(_poolId, lastPoolAmount);

            // find balance of cake in this contract and transfer to devaddress
            uint256 cakeBalance = cakeToken.balanceOf(address(this));

            if(cakeBalance > 0) {
                cakeToken.safeTransfer(devAddress, cakeBalance);
            }
        }

        // transfer lp funds back to vault
        token.safeTransfer(
            address(vault), //vault
            lastPoolAmount
        );

        emit EmergencyWithdraw(_token, lastPoolAmount, 0);
    }
```

Figure 4 Source code of *emergencyWithdraw*

- Related functions: *withdraw*, *emergencyWithdraw*, *safeTransfer*, *getPoolAmount*
- Result: Pass

(5) update

- Description: Anyone can call this function at any time, and this function will settle the investment reward to this contract.



```
    function update(address _token) public override {
        PoolInfo storage poolInfo = tokenPoolInfo[_token];
        uint256 _poolId = poolInfo.poolId;

        // call masterChef with zero deposit for update
        IMasterChef(masterChef).deposit(_poolId, 0);

        // Return if it's too early (if START_BLOCK is in the future probably)
        if (block.number <= lastRewardBlock[_poolId]) return;

        // Retrieve amount of tokens held in contract
        uint256 depositInPancake;
        (depositInPancake,) = IMasterChef(masterChef).userInfo(_poolId, address(this));

        IERC20Upgradeable token = IERC20Upgradeable(_token);
        // balance is pending in strategy + pending in pancake
        uint256 poolBalance = token.balanceOf(address(this)).add(depositInPancake);

        if (poolBalance == 0) {
            //pool.lastRewardBlock = block.number;
            lastRewardBlock[_poolId] = block.number;
            return;
        }

        // Update the last block
        lastRewardBlock[_poolId] = block.number;

        // Do the work: swap claimed Cake rewards into LP if any.
        work(_token);

    }
```

Figure 5 Source code of *update*

- Related functions: *deposit*, *work*
- Result: Pass

4. Conclusion

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

BEOSIN

Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com