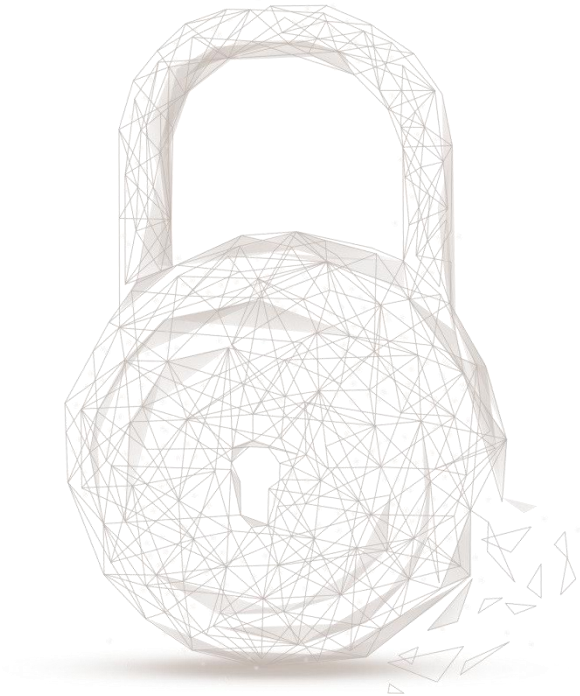
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number: 202110091630

Report Inquiry Name: JGN DeFi

Smart Contract Name:

JGNBNBRewards

Smart Contract Address:

0x3311354798880ef029405c52bb275d10a69a9cde

Smart Contract Address Link:

https://bscscan.com/address/0x3311354798880ef029405c52bb275d10a69a9cde#code

Start Date: 2021.10.09

Completion Date: 2021.10.09

Overall Result: Pass

Audit Team: Beosin 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 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 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 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 Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin. Due to the technical limitations of any organization, this report conducted by Beosin still has the possibility that the entire risk cannot be completely detected. Beosin disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin.

Audit Results Explained:

Beosin Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract JGNBNBRewards, including Coding Standards, Security, and Business Logic. **The JGNBNBRewards 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

3.1 Stake Initialization

- Description: The "stake-reward" mode of the contract needs to initialize the relevant parameters (*rewardRate, lastUpdateTime, periodFinish*), call the *notifyRewardAmount* function by the specified reward distribution manager address *rewardDistribution*, and enter the initial reward used to calculate the *rewardRate*, initialize the stake and reward related parameters. This function can be called by the specified address rewardDistribution at any time to control the reward rate and the key time judgment condition, even if the *rewardRate* is updated when the *checkhalve* modifier executes the logic, it can still be modified by entering the specified value reward in this function. If the value is too small, the user's reward will not match expectations.

```
    function notifyRewardAmount(uint256 reward)
        external
        onlyRewardDistribution
        updateReward(address(0))
    {
        if (block.timestamp >= periodFinish) {
            rewardRate = reward.div(DURATION);
        } else {
            uint256 remaining = periodFinish.sub(block.timestamp);
            uint256 leftover = remaining.mul(rewardRate);
            rewardRate = reward.add(leftover).div(DURATION);
        }
        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp.add(DURATION);
        emit RewardAdded(reward);
    }
}
```

Figure 1 source code of *notifyRewardAmount*

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

3.2 Stake LP tokens

- Description: The contract implements the *stake* function to stake the LP tokens. The user need to *approve* the contract address in advance. By calling the *transferFrom* function in the LP contract, the contract address transfers the specified amount of LP tokens to the contract address on behalf of the user; 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*.(It should be noted here that the rewardRate will be reduced by half in the second cycle, and there will be no rewards in the future)

```
// stake visibility is public as overriding LPTokenWrapper's stake() function
function stake(uint256 amount) public updateReward(msg.sender) checkhalve checkStart{
    require(amount > 0, "Cannot stake 0");
    super.stake(amount);
    emit Staked(msg.sender, amount);
}
```

Figure 2 source code of *stake* function(1/2)

```
function stake(uint256 amount) public {
    _totalSupply = _totalSupply.add(amount);
    _balances[msg.sender] = _balances[msg.sender].add(amount);
    y.safeTransferFrom(msg.sender, address(this), amount);
}
```

Figure 3 source code of *stake* function(2/2)

```
modifier checkhalve(){
    if (block.timestamp >= periodFinish) {
        if(halveTimes > 0){
            initreward = initreward.mul(50).div(100);
            rewardRate = initreward.div(DURATION);
            periodFinish = block.timestamp.add(DURATION);
            halveTimes = halveTimes.sub(1);
            emit RewardAdded(initreward);
        }
        else{
            rewardRate = 0;
        }
    }
    _;
}
```

Figure 4 source code of modifier checkhalve

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

3.3 Withdraw LP tokens

- Description: The contract implements the *withdraw* function to withdraw the LP tokens. By calling the *transfer* function in the token contract, the contract address transfers the specified amount of LP tokens to the user; 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*. (It should be noted here that the rewardRate will be reduced by half in the second cycle, and there will be no rewards in the future)

```
function withdraw(uint256 amount) public updateReward(msg.sender) checkhalve checkStart{
    require(amount > 0, "Cannot withdraw 0");
    super.withdraw(amount);
    emit Withdrawn(msg.sender, amount);
}
```

Figure 5 source code of *withdraw* function(1/2)

```
function withdraw(uint256 amount) public {
    _totalSupply = _totalSupply.sub(amount);
    _balances[msg.sender] = _balances[msg.sender].sub(amount);
    y.safeTransfer(msg.sender, amount);
}
```

Figure 6 source code of *withdraw* function(2/2)

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

3.4 Withdraw rewards (JGN)

- Description: The contract implements the *getReward* function to withdraw the rewards (JGN). By calling the *transfer* function in the JGN contract, the contract address transfers the specified amount (all rewards of caller) of JGN to the user; 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*. (It should be noted here that the rewardRate will be reduced by half in the second cycle, and there will be no rewards in the future)

```
function getReward() public updateReward(msg.sender) checkhalve checkStart{
    uint256 reward = earned(msg.sender);
    if (reward > 0) {
        rewards[msg.sender] = 0;
        jgn.safeTransfer(msg.sender, reward);
        emit RewardPaid(msg.sender, reward);
    }
}
```

Figure 7 source code of *getReward* function

- Related functions: *getReward, rewardPerToken, lastTimeRewardApplicable, earned, balanceOf*
- Result: Pass

3.5 Exit the stake participation

- Description: The contract implements the *exit* function to close the participation of "stake-reward" mode. Call the *withdraw* function to withdraw all stake JGN, call the *getReward* function to receive all

rewards. The user address cannot get new rewards because the balance of LP tokens already staked is empty.

```
function exit() external {
    withdraw(balanceOf(msg.sender));
    getReward();
}
```

Figure 8 source code of *exit* function

- Related functions: *exit, withdraw, getReward, rewardPerToken, lastTimeRewardApplicable, earned, balanceOf*
- Result: Pass

3.6 Reward related data query function

- Description: Contract users can query the earliest timestamp between the current timestamp and the *periodFinish* by calling the *lastTimeRewardApplicable* function; calling the *rewardPerToken* function can query the gettable rewards for each stake LP; calling the *earned* function can query the total claimable stake rewards of the specified address.
- Related functions: *lastTimeRewardApplicable, rewardPerToken, earned*
- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contract JGNBNBRewards contract 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