CyberCrime Shield

cybercrimeshield.org

Smart Contract Audit Report

BEE'N'BEE

https://bee-n-bee.io/

AUDIT TYPE: **PUBLIC**

YOU CAN CHECK THE VALIDITY USING THE QR CODE OR LINK:

https://cybercrimeshield.org/secure/bee-n-bee

Report ID: 291995

May 14, 2022

TABLE OF CONTENTS

SMART CONTRACT

https://bscscan.com/address/0xbb1ccd65fabadf606c7266e58488e2eff412a4f4#code

Mirror:

https://cybercrimeshield.org/secure/uploads/bee-n-bee.sol

CRC32: BDEF13D5

MD5: 1B7CD4CEAEA21159B1A7DD0961FE4641

SHA-1: 40BF6575DD65646EEB1A7C5A9B72E6346589CC4D

INTRODUCTION

Blockchain platforms, such as Nakamoto's Bitcoin, enable the trade of crypto-currencies between mutually mistrusting parties.

To eliminate the need for trust, Nakomoto designed a peer-to-peer network that enables its peers to agree on the trading transactions.

Smart contracts have shown to be applicable in many domains including financial industry, public sector and cross-industry.

The increased adoption of smart contracts demands strong security guarantees. Unfortunately, it is challenging to create smart contracts that are free of security bugs.

As a consequence, critical vulnerabilities in smart contracts are discovered and exploited every few months.

In turn, these exploits have led to losses reaching billions worth of USD in the past few years.

It is apparent that effective security checks for smart contracts are strictly needed.

Our company provides comprehensive, independent smart contract auditing.

We help stakeholders confirm the quality and security of their smart contracts using our standardized audit process.

The scope of this audit was to analyze and document the Bee-n-Bee contract.

This document is not financial advice, you perform all financial actions on your own responsibility.

AUDIT METHODOLOGY

1. Design Patterns

We inspect the structure of the smart contract, including both manual and automated analysis.

2. Static Analysis

The static analysis is performed using a series of automated tools, purposefully designed to test the security of the contract.

All the issues found by tools were manually checked (rejected or confirmed).

3. Manual Analysis

Contract reviewing to identify common vulnerabilities. Comparing of requirements and implementation. Reviewing of a smart contract for compliance with specified customer requirements. Checking for energy optimization and self-documentation. Running tests of the properties of the smart contract in test net.

ISSUES DISCOVERED

Issues are listed from most critical to least critical. Severity is determined by an assessment of the risk of exploitation or otherwise unsafe behavior.

Severity Levels

Critical - Funds may be allocated incorrectly, lost or otherwise result in a significant loss.
Medium - Affects the ability of the contract to operate.
Low - Minimal impact on operational ability.
Informational - No impact on the contract.

AUDIT SUMMARY

The summary result of the audit performed is presented in the table below

Findings list:

LEVEL	AMOUNT
Critical	0
Medium	0
Low	0
Informational	0

CONCLUSION

- Contract has high code readability
- Gas usage is optimal
- Contract is fully BSC completable
- No backdoors or overflows are present in the contract

SOURCE CODE

```
/**
 *Submitted for verification at BscScan.com on 2022-05-13
*/

// --------*--------
// |   BEE'N'BEE   |
// --------*--------
// The Bee'n'Bee (BNB)

// Website : https://bee-n-bee.io/
// Telegram : https://t.me/bee_n_bee

// SPDX-License-Identifier: MIT
pragma solidity 0.8.9;

```

```
16. contract BeeNBee {
17.     using SafeMath for uint256;
18. 
19.     /** base parameters **/
20.     uint256 public EGGS_TO_HIRE_1MINERS = 1440000;
21.     uint256 public REFERRAL = 70;
22.     uint256 public PERCENTS_DIVIDER = 1000;
23.     uint256 public TAX = 50;
24.     uint256 public MARKET_EGGS_DIVISOR = 2;
25. 
26.     uint256 public MIN_INVEST_LIMIT = 1 * 1e17; /** 0.1 BNB  **/
27.     uint256 public WALLET_DEPOSIT_LIMIT = 25 * 1e18; /** 25 BNB  **/
28. 
29.     uint256 public COMPOUND_BONUS = 20;
30.     uint256 public COMPOUND_BONUS_MAX_TIMES = 10;
31.     uint256 public COMPOUND_STEP = 12 * 60 * 60;
32. 
33.     uint256 public WITHDRAWAL_TAX = 800;
34.     uint256 public COMPOUND_FOR_NO_TAX_WITHDRAWAL = 10;
35. 
36.     uint256 public totalStaked;
37.     uint256 public totalDeposits;
38.     uint256 public totalCompound;
39.     uint256 public totalRefBonus;
40.     uint256 public totalWithdrawn;
41. 
42.     uint256 public marketEggs;
43.     uint256 PSN = 10000;
44.     uint256 PSNH = 5000;
45.     bool public contractStarted;
46.     bool public blacklistActive = true;
47.     mapping(address => bool) public Blacklisted;
48. 
49.     uint256 public CUTOFF_STEP = 48 * 60 * 60;
50.     uint256 public WITHDRAW_COOLDOWN = 4 * 60 * 60;
51. 
52.     /* addresses */
53.     address public owner;
54.     address payable private ceoAddr;
55. 
56.     struct User {
57.         uint256 initialDeposit;
58.         uint256 userDeposit;
59.         uint256 miners;
60.         uint256 claimedEggs;
61.         uint256 lastHatch;
```

```
62.         address referrer;
63.         uint256 referralsCount;
64.         uint256 referralEggRewards;
65.         uint256 totalWithdrawn;
66.         uint256 dailyCompoundBonus;
67.         uint256 farmerCompoundCount; //added to monitor farmer consecutive compound without cap
68.         uint256 lastWithdrawTime;
69.     }
70.
71.     mapping(address => User) public users;
72.
73.     constructor() {
74.         owner = msg.sender;
75.         ceoAddr = payable(msg.sender);
76.     }
77.
78.     function setblacklistActive(bool isActive) public{
79.         require(msg.sender == owner, "Admin use only.");
80.         blacklistActive = isActive;
81.     }
82.
83.     function blackListWallet(address Wallet, bool isBlacklisted) public{
84.         require(msg.sender == owner, "Admin use only.");
85.         Blacklisted[Wallet] = isBlacklisted;
86.     }
87.
88.     function blackMultipleWallets(address[] calldata Wallet, bool isBlacklisted) public{
89.         require(msg.sender == owner, "Admin use only.");
90.         for(uint256 i = 0; i < Wallet.length; i++) {
91.             Blacklisted[Wallet[i]] = isBlacklisted;
92.         }
93.     }
94.
95.     function checkIfBlacklisted(address Wallet) public view returns(bool blacklisted){
96.         require(msg.sender == owner, "Admin use only.");
97.         blacklisted = Blacklisted[Wallet];
98.     }
99.
100.          function startApiary(address addr) public payable{
101.              if (!contractStarted) {
102.                  if (msg.sender == owner) {
103.                      require(marketEggs == 0);
104.                      contractStarted = true;
```

```
105.                     marketEggs = 144000000000;
106.                     buildHives(addr);
107.                 } else revert("Contract not yet started.");
108.             }
109.         }
110.
111.         //fund contract with BNB before launch.
112.         function fundContract() external payable {}
113.
114.         function buildMoreHives(bool isCompound) public {
115.             User storage user = users[msg.sender];
116.             require(contractStarted, "Contract not yet Started.");
117.
118.             uint256 eggsUsed = getMyEggs();
119.             uint256 eggsForCompound = eggsUsed;
120.
121.             if(isCompound) {
122.                 uint256 dailyCompoundBonus = getDailyCompoundBonus(msg.sender,
    eggsForCompound);
123.                 eggsForCompound = eggsForCompound.add(dailyCompoundBonus);
124.                 uint256 eggsUsedValue = calculateEggSell(eggsForCompound);
125.                 user.userDeposit = user.userDeposit.add(eggsUsedValue);
126.                 totalCompound = totalCompound.add(eggsUsedValue);
127.             }
128.
129.             if(block.timestamp.sub(user.lastHatch) >= COMPOUND_STEP) {
130.                 if(user.dailyCompoundBonus < COMPOUND_BONUS_MAX_TIMES) {
131.                     user.dailyCompoundBonus = user.dailyCompoundBonus.add(1);
132.                 }
133.                 //add compoundCount for monitoring purposes.
134.                 user.farmerCompoundCount = user.farmerCompoundCount .add(1);
135.             }
136.
137.             user.miners =
    user.miners.add(eggsForCompound.div(EGGS_TO_HIRE_1MINERS));
138.             user.claimedEggs = 0;
139.             user.lastHatch = block.timestamp;
140.
141.             marketEggs = marketEggs.add(eggsUsed.div(MARKET_EGGS_DIVISOR));
142.         }
143.
144.         function sellHoney() public{
145.             require(contractStarted, "Contract not yet Started.");
146.
147.             if (blacklistActive) {
148.                 require(!Blacklisted[msg.sender], "Address is blacklisted.");
```

```
149.             }
150.
151.             User storage user = users[msg.sender];
152.             uint256 hasEggs = getMyEggs();
153.             uint256 eggValue = calculateEggSell(hasEggs);
154.
155.             /**
156.                 if user compound < to mandatory compound days**/
157.             if(user.dailyCompoundBonus < COMPOUND_FOR_NO_TAX_WITHDRAWAL){
158.                 //daily compound bonus count will not reset and eggValue will be
    deducted with 60% feedback tax.
159.                 eggValue =
    eggValue.sub(eggValue.mul(WITHDRAWAL_TAX).div(PERCENTS_DIVIDER));
160.             }else{
161.                 //set daily compound bonus count to 0 and eggValue will remain
    without deductions
162.                 user.dailyCompoundBonus = 0;
163.                 user.farmerCompoundCount = 0;
164.             }
165.
166.             user.lastWithdrawTime = block.timestamp;
167.             user.claimedEggs = 0;
168.             user.lastHatch = block.timestamp;
169.             marketEggs = marketEggs.add(hasEggs.div(MARKET_EGGS_DIVISOR));
170.
171.             if(getBalance() < eggValue) {
172.                 eggValue = getBalance();
173.             }
174.
175.             uint256 eggsPayout = eggValue.sub(payFees(eggValue));
176.             payable(address(msg.sender)).transfer(eggsPayout);
177.             user.totalWithdrawn = user.totalWithdrawn.add(eggsPayout);
178.             totalWithdrawn = totalWithdrawn.add(eggsPayout);
179.         }
180.
181.
182.         /** transfer amount of BNB **/
183.         function buildHives(address ref) public payable{
184.             require(contractStarted, "Contract not yet Started.");
185.             User storage user = users[msg.sender];
186.             require(msg.value >= MIN_INVEST_LIMIT, "Minimum investment not
    met.");
187.             require(user.initialDeposit.add(msg.value) <=
    WALLET_DEPOSIT_LIMIT, "Max deposit limit reached.");
188.             uint256 eggsBought = calculateEggBuy(msg.value,
    address(this).balance.sub(msg.value));
```

```
189.             user.userDeposit = user.userDeposit.add(msg.value);
190.             user.initialDeposit = user.initialDeposit.add(msg.value);
191.             user.claimedEggs = user.claimedEggs.add(eggsBought);
192.
193.             if (user.referrer == address(0)) {
194.                 if (ref != msg.sender) {
195.                     user.referrer = ref;
196.                 }
197.
198.                 address upline1 = user.referrer;
199.                 if (upline1 != address(0)) {
200.                     users[upline1].referralsCount =
    users[upline1].referralsCount.add(1);
201.                 }
202.             }
203.
204.             if (user.referrer != address(0)) {
205.                 address upline = user.referrer;
206.                 if (upline != address(0)) {
207.                     uint256 refRewards =
    msg.value.mul(REFERRAL).div(PERCENTS_DIVIDER);
208.                     payable(address(upline)).transfer(refRewards);
209.                     users[upline].referralEggRewards =
    users[upline].referralEggRewards.add(refRewards);
210.                     totalRefBonus = totalRefBonus.add(refRewards);
211.                 }
212.             }
213.
214.             uint256 eggsPayout = payFees(msg.value);
215.             totalStaked = totalStaked.add(msg.value.sub(eggsPayout));
216.             totalDeposits = totalDeposits.add(1);
217.             buildMoreHives(false);
218.         }
219.
220.         function payFees(uint256 eggValue) internal returns(uint256){
221.             uint256 tax = eggValue.mul(TAX).div(PERCENTS_DIVIDER);
222.             ceoAddr.transfer(tax);
223.             return tax;
224.         }
225.
226.         function getDailyCompoundBonus(address _adr, uint256 amount) public view
    returns(uint256){
227.             if(users[_adr].dailyCompoundBonus == 0) {
228.                 return 0;
229.             } else {
```

```
230.                 uint256 totalBonus =
    users[_adr].dailyCompoundBonus.mul(COMPOUND_BONUS);
231.                 uint256 result = amount.mul(totalBonus).div(PERCENTS_DIVIDER);
232.                 return result;
233.             }
234.         }
235.
236.         function getUserInfo(address _adr) public view returns(uint256
    _initialDeposit, uint256 _userDeposit, uint256 _miners,
237.             uint256 _claimedEggs, uint256 _lastHatch, address _referrer, uint256
    _referrals,
238.             uint256 _totalWithdrawn, uint256 _referralEggRewards, uint256
    _dailyCompoundBonus, uint256 _farmerCompoundCount, uint256 _lastWithdrawTime) {
239.             _initialDeposit = users[_adr].initialDeposit;
240.             _userDeposit = users[_adr].userDeposit;
241.             _miners = users[_adr].miners;
242.             _claimedEggs = users[_adr].claimedEggs;
243.             _lastHatch = users[_adr].lastHatch;
244.             _referrer = users[_adr].referrer;
245.             _referrals = users[_adr].referralsCount;
246.             _totalWithdrawn = users[_adr].totalWithdrawn;
247.             _referralEggRewards = users[_adr].referralEggRewards;
248.             _dailyCompoundBonus = users[_adr].dailyCompoundBonus;
249.             _farmerCompoundCount = users[_adr].farmerCompoundCount;
250.             _lastWithdrawTime = users[_adr].lastWithdrawTime;
251.         }
252.
253.         function getBalance() public view returns(uint256){
254.             return address(this).balance;
255.         }
256.
257.         function getTimeStamp() public view returns (uint256) {
258.             return block.timestamp;
259.         }
260.
261.         function getAvailableEarnings(address _adr) public view returns(uint256)
    {
262.             uint256 userEggs =
    users[_adr].claimedEggs.add(getEggsSinceLastHatch(_adr));
263.             return calculateEggSell(userEggs);
264.         }
265.
266.         function calculateTrade(uint256 rt,uint256 rs, uint256 bs) public view
    returns(uint256){
267.             return SafeMath.div(
268.                 SafeMath.mul(PSN, bs),
```

```
                    SafeMath.add(PSNH,
                    SafeMath.div(
                        SafeMath.add(
                            SafeMath.mul(PSN, rs),
                            SafeMath.mul(PSNH, rt)),
                        rt)));
        }

        function calculateEggSell(uint256 eggs) public view returns(uint256){
            return calculateTrade(eggs, marketEggs, getBalance());
        }

        function calculateEggBuy(uint256 eth,uint256 contractBalance) public
    view returns(uint256){
            return calculateTrade(eth, contractBalance, marketEggs);
        }

        function calculateEggBuySimple(uint256 eth) public view
    returns(uint256){
            return calculateEggBuy(eth, getBalance());
        }

        /** How many miners and eggs per day user will recieve based on BNB
    deposit **/
        function getEggsYield(uint256 amount) public view
    returns(uint256,uint256) {
            uint256 eggsAmount = calculateEggBuy(amount ,
    getBalance().add(amount).sub(amount));
            uint256 miners = eggsAmount.div(EGGS_TO_HIRE_1MINERS);
            uint256 day = 1 days;
            uint256 eggsPerDay = day.mul(miners);
            uint256 earningsPerDay = calculateEggSellForYield(eggsPerDay,
    amount);
            return(miners, earningsPerDay);
        }

        function calculateEggSellForYield(uint256 eggs,uint256 amount) public
    view returns(uint256){
            return calculateTrade(eggs,marketEggs, getBalance().add(amount));
        }

        function getSiteInfo() public view returns (uint256 _totalStaked,
    uint256 _totalDeposits, uint256 _totalCompound, uint256 _totalRefBonus) {
            return (totalStaked, totalDeposits, totalCompound, totalRefBonus);
        }

```

```
307.          function getMyMiners() public view returns(uint256){
308.              return users[msg.sender].miners;
309.          }
310.
311.          function getMyEggs() public view returns(uint256){
312.              return
    users[msg.sender].claimedEggs.add(getEggsSinceLastHatch(msg.sender));
313.          }
314.
315.          function getEggsSinceLastHatch(address adr) public view
    returns(uint256){
316.              uint256 secondsSinceLastHatch =
    block.timestamp.sub(users[adr].lastHatch);
317.              /** get min time. **/
318.              uint256 cutoffTime = min(secondsSinceLastHatch, CUTOFF_STEP);
319.              uint256 secondsPassed = min(EGGS_TO_HIRE_1MINERS, cutoffTime);
320.              return secondsPassed.mul(users[adr].miners);
321.          }
322.
323.          function min(uint256 a, uint256 b) private pure returns (uint256) {
324.              return a < b ? a : b;
325.          }
326.
327.          function CHANGE_OWNERSHIP(address value) external {
328.              require(msg.sender == owner, "Admin use only.");
329.              owner = value;
330.          }
331.
332.          /** percentage setters **/
333.
334.          // 2592000 - 3%, 2160000 - 4%, 1728000 - 5%, 1440000 - 6%, 1200000 - 7%
335.          // 1080000 - 8%, 959000 - 9%, 864000 - 10%, 720000 - 12%
336.
337.          function PRC_EGGS_TO_HIRE_1MINERS(uint256 value) external {
338.              require(msg.sender == owner, "Admin use only.");
339.              require(value >= 479520 && value <= 720000); /** min 3% max 12%**/
340.              EGGS_TO_HIRE_1MINERS = value;
341.          }
342.
343.          function PRC_TAX(uint256 value) external {
344.              require(msg.sender == owner, "Admin use only.");
345.              require(value <= 15);
346.              TAX = value;
347.          }
348.
349.          function PRC_REFERRAL(uint256 value) external {
```

```
350.             require(msg.sender == owner, "Admin use only.");
351.             require(value >= 10 && value <= 100);
352.             REFERRAL = value;
353.         }
354.
355.         function PRC_MARKET_EGGS_DIVISOR(uint256 value) external {
356.             require(msg.sender == owner, "Admin use only.");
357.             require(value <= 50);
358.             MARKET_EGGS_DIVISOR = value;
359.         }
360.
361.         function SET_WITHDRAWAL_TAX(uint256 value) external {
362.             require(msg.sender == owner, "Admin use only.");
363.             require(value <= 900);
364.             WITHDRAWAL_TAX = value;
365.         }
366.
367.         function BONUS_DAILY_COMPOUND(uint256 value) external {
368.             require(msg.sender == owner, "Admin use only.");
369.             require(value >= 10 && value <= 900);
370.             COMPOUND_BONUS = value;
371.         }
372.
373.         function BONUS_DAILY_COMPOUND_BONUS_MAX_TIMES(uint256 value) external {
374.             require(msg.sender == owner, "Admin use only.");
375.             require(value <= 30);
376.             COMPOUND_BONUS_MAX_TIMES = value;
377.         }
378.
379.         function BONUS_COMPOUND_STEP(uint256 value) external {
380.             require(msg.sender == owner, "Admin use only.");
381.             require(value <= 24);
382.             COMPOUND_STEP = value * 60 * 60;
383.         }
384.
385.         function SET_INVEST_MIN(uint256 value) external {
386.             require(msg.sender == owner, "Admin use only");
387.             MIN_INVEST_LIMIT = value * 1e17;
388.         }
389.
390.         function SET_CUTOFF_STEP(uint256 value) external {
391.             require(msg.sender == owner, "Admin use only");
392.             CUTOFF_STEP = value * 60 * 60;
393.         }
394.
395.         function SET_WITHDRAW_COOLDOWN(uint256 value) external {
```

```
396.             require(msg.sender == owner, "Admin use only");
397.             require(value <= 24);
398.             WITHDRAW_COOLDOWN = value * 60 * 60;
399.         }
400.
401.         function SET_WALLET_DEPOSIT_LIMIT(uint256 value) external {
402.             require(msg.sender == owner, "Admin use only");
403.             require(value >= 10);
404.             WALLET_DEPOSIT_LIMIT = value * 1 ether;
405.         }
406.
407.         function SET_COMPOUND_FOR_NO_TAX_WITHDRAWAL(uint256 value) external {
408.             require(msg.sender == owner, "Admin use only.");
409.             require(value <= 12);
410.             COMPOUND_FOR_NO_TAX_WITHDRAWAL = value;
411.         }
412.     }
413.
414.     library SafeMath {
415.
416.         function mul(uint256 a, uint256 b) internal pure returns (uint256) {
417.             if (a == 0) {
418.                 return 0;
419.             }
420.             uint256 c = a * b;
421.             assert(c / a == b);
422.             return c;
423.         }
424.
425.         function div(uint256 a, uint256 b) internal pure returns (uint256) {
426.             uint256 c = a / b;
427.             return c;
428.         }
429.
430.         function sub(uint256 a, uint256 b) internal pure returns (uint256) {
431.             assert(b <= a);
432.             return a - b;
433.         }
434.
435.         function add(uint256 a, uint256 b) internal pure returns (uint256) {
436.             uint256 c = a + b;
437.             assert(c >= a);
438.             return c;
439.         }
440.
441.         function mod(uint256 a, uint256 b) internal pure returns (uint256) {
```

```
442.                 require(b != 0);
443.                 return a % b;
444.             }
445.     }
```