CyberCrime Shield

cybercrimeshield.org

Smart Contract Audit Report

MyCakeFarm

https://www.mycakefarm.com

AUDIT TYPE: **PUBLIC**

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

https://cybercrimeshield.org/secure/mycakefarm

Report ID: 291546

October 22, 2021

TABLE OF CONTENTS

SMART CONTRACTS

https://cybercrimeshield.org/secure/uploads/mycakefarm.sol

CRC32: 5E6F006C

MD5: B12D3F4DBD3C6648B43E359423276B75

SHA-1: 635608FAEA32B567C0F6A95F190065E4C3D8DACF

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 MyCakeFarm 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

- Contracts have high code readability
- Gas usage is optimal
- Contracts are fully BSC completable
- No backdoors or overflows are present in the contracts

SOURCE CODE

```
1.// SPDX-License-Identifier: MIT
2.pragma solidity 0.8.9;
3.
4.interface IERC20 {
5.    function totalSupply() external view returns (uint256);
6.
7.    function balanceOf(address account) external view returns (uint256);
8.
9.    function transfer(address recipient, uint256 amount)
10.        external
11.        returns (bool);
12.
13.    function allowance(address owner, address spender)
14.        external
```

```
15.         view
16.         returns (uint256);
17.
18.     function approve(address spender, uint256 amount) external returns
    (bool);
19.
20.     function transferFrom(
21.         address sender,
22.         address recipient,
23.         uint256 amount
24.     ) external returns (bool);
25.
26.     event Transfer(address indexed from, address indexed to, uint256
    value);
27.     event Approval(
28.         address indexed owner,
29.         address indexed spender,
30.         uint256 value
31.     );
32. }
33.
34. contract CAKEFarmer {
35.     IERC20 public token_CAKE;
36.     using SafeMath for uint256;
37.     address erctoken =
    address(0x0E09FaBB73Bd3Ade0a17ECC321fD13a19e81cE82); //CAKE
38.     uint256 public constant INVEST_MIN_AMOUNT = 5e16; // 0.05 bnb //
    1000000000000000000 = 1 Cake
39.     uint256 public REFERRAL_PERCENT = 70;
40.     uint256 public constant PROJECT_FEE = 33; // each
41.     uint256 public constant PERCENT_STEP = 5;
42.     uint256 public constant PERCENTS_DIVIDER = 1000;
43.     uint256 public constant TIME_STEP = 1 days;
44.
45.     uint256 public ADDITIONAL_PERCENT_PLAN_1 = 0;
46.     uint256 public ADDITIONAL_PERCENT_PLAN_2 = 0;
47.     uint256 public ADDITIONAL_PERCENT_PLAN_3 = 0;
48.     uint256 public ADDITIONAL_PERCENT_PLAN_4 = 0;
49.
50.
51.     uint256 public totalInvested;
52.     uint256 public totalRefBonus;
53.
54.     struct Plan {
55.         uint256 time;
```

```
56.         uint256 percent;
57.     }
58.
59.     Plan[] internal plans;
60.
61.     struct Deposit {
62.         uint8 plan;
63.         uint256 amount;
64.         uint256 start;
65.     }
66.
67.     struct User {
68.         Deposit[] deposits;
69.         uint256 checkpoint;
70.         address referrer;
71.         uint256 levels;
72.         uint256 bonus;
73.         uint256 totalBonus;
74.         uint256 withdrawn;
75.     }
76.
77.     mapping(address => User) internal users;
78.
79.     bool public started;
80.
81.     address public ceoAddress;
82.     address public ceoAddress2;
83.     address public ceoAddress3;
84.     event Newbie(address user);
85.     event NewDeposit(address indexed user, uint8 plan, uint256 amount);
86.     event Withdrawn(address indexed user, uint256 amount);
87.     event RefBonus(
88.         address indexed referrer,
89.         address indexed referral,
90.         uint256 amount
91.     );
92.     event FeePayed(address indexed user, uint256 totalAmount);
93.
94.     constructor() {
95.         ceoAddress = msg.sender;
96.         ceoAddress2=address(0x815AF6c4E81a38556243a2c272F38198Cd99aC69);
97.         ceoAddress3=address(0x89c1A11AF34BF269cFc717fA45614938B3CCBEf5);
98.
99.         token_CAKE = IERC20(erctoken);
100.
```

```
101.             plans.push(Plan(10000, 50));
102.             plans.push(Plan(40, 70));
103.             plans.push(Plan(60, 65));
104.             plans.push(Plan(90, 60));
105.         }
106.
107.         function invest(
108.             address referrer,
109.             uint8 plan,
110.             uint256 amounterc
111.         ) public {
112.             if (!started) {
113.                 if (msg.sender == ceoAddress) {
114.                     started = true;
115.                 } else revert("Not started yet");
116.             }
117.
118.             require(amounterc >= INVEST_MIN_AMOUNT);
119.             require(plan < 4, "Invalid plan");
120.
121.             token_CAKE.transferFrom(address(msg.sender), address(this),
    amounterc);
122.
123.             // dev fee
124.             uint256 fee =
    amounterc.mul(PROJECT_FEE).div(PERCENTS_DIVIDER);
125.
126.             token_CAKE.transfer(ceoAddress, fee);
127.             token_CAKE.transfer(ceoAddress2, fee);
128.             token_CAKE.transfer(ceoAddress3, fee);
129.
130.             User storage user = users[msg.sender];
131.
132.             if (user.referrer == address(0)) {
133.                 if (users[referrer].deposits.length > 0 && referrer !=
    msg.sender) {
134.                     user.referrer = referrer;
135.                 }
136.
137.                 address upline1 = user.referrer;
138.                 if (upline1 != address(0)) {
139.                     users[upline1].levels = users[upline1].levels.add(1);
140.                 }
141.             }
142.
```

```
143.             if (user.referrer != address(0)) {
144.                 address upline = user.referrer;
145.                 if (upline != address(0)) {
146.                     uint256 amount = amounterc.mul(REFERRAL_PERCENT).div(
147.                         PERCENTS_DIVIDER
148.                     );
149.                     users[upline].bonus =
    users[upline].bonus.add(amount);
150.                     users[upline].totalBonus =
    users[upline].totalBonus.add(amount);
151.                     emit RefBonus(upline, msg.sender, amount);
152.                 }
153.             }
154.
155.             if (user.deposits.length == 0) {
156.                 user.checkpoint = block.timestamp;
157.                 emit Newbie(msg.sender);
158.             }
159.
160.             user.deposits.push(Deposit(plan, amounterc,
    block.timestamp));
161.
162.             totalInvested = totalInvested.add(amounterc);
163.
164.             emit NewDeposit(msg.sender, plan, amounterc);
165.             emit FeePayed(msg.sender, fee);
166.         }
167.
168.         function withdraw() public {
169.             User storage user = users[msg.sender];
170.
171.             uint256 totalAmount = getUserDividends(msg.sender);
172.
173.             uint256 referralBonus = getUserReferralBonus(msg.sender);
174.             if (referralBonus > 0) {
175.                 user.bonus = 0;
176.                 totalAmount = totalAmount.add(referralBonus);
177.             }
178.
179.             require(totalAmount > 0, "User has no dividends");
180.
181.             user.checkpoint = block.timestamp;
182.             user.withdrawn = user.withdrawn.add(totalAmount);
183.
184.             token_CAKE.transfer(msg.sender, totalAmount);
```

```
185.
186.             emit Withdrawn(msg.sender, totalAmount);
187.         }
188.
189.         function getContractBalance() public view returns (uint256) {
190.             return address(this).balance;
191.         }
192.
193.         function getPlanInfo(uint8 plan)
194.             public
195.             view
196.             returns (uint256 time, uint256 percent)
197.         {
198.             time = plans[plan].time;
199.             percent = plans[plan].percent;
200.         }
201.
202.         function getUserDividends(address userAddress)
203.             public
204.             view
205.             returns (uint256)
206.         {
207.             User storage user = users[userAddress];
208.
209.             uint256 totalAmount;
210.
211.             for (uint256 i = 0; i < user.deposits.length; i++) {
212.                 uint256 finish = user.deposits[i].start.add(
213.                     plans[user.deposits[i].plan].time.mul(1 days)
214.                 );
215.                 if (user.checkpoint < finish) {
216.                     uint256 share = user
217.                         .deposits[i]
218.                         .amount;
219.
220.                     uint256 percent =
    plans[user.deposits[i].plan].percent;
221.                     if(user.deposits[i].plan == 0){
222.                         percent =
    percent.add(ADDITIONAL_PERCENT_PLAN_1);                 }
223.                     else if(user.deposits[i].plan == 1){
224.                         percent = percent.add(ADDITIONAL_PERCENT_PLAN_2);
225.                     }else if(user.deposits[i].plan == 2){
226.                         percent = percent.add(ADDITIONAL_PERCENT_PLAN_3);
227.                     }else if(user.deposits[i].plan == 3){
```

```
                        percent = percent.add(ADDITIONAL_PERCENT_PLAN_4);
                    }

                    share = share.mul(percent).div(PERCENTS_DIVIDER);

                    uint256 from = user.deposits[i].start >
   user.checkpoint
                        ? user.deposits[i].start
                        : user.checkpoint;
                    uint256 to = finish < block.timestamp
                        ? finish
                        : block.timestamp;
                    if (from < to) {
                        totalAmount = totalAmount.add(
                            share.mul(to.sub(from)).div(TIME_STEP)
                        );
                    }
                }
            }

            return totalAmount;
        }

        function getUserTotalWithdrawn(address userAddress)
            public
            view
            returns (uint256)
        {
            return users[userAddress].withdrawn;
        }

        function getUserCheckpoint(address userAddress)
            public
            view
            returns (uint256)
        {
            return users[userAddress].checkpoint;
        }

        function getUserReferrer(address userAddress)
            public
            view
            returns (address)
        {
            return users[userAddress].referrer;
```

```
272.         }
273.
274.         function getUserTotalReferrals(address userAddress)
275.             public
276.             view
277.             returns (uint256)
278.         {
279.             return users[userAddress].levels;
280.         }
281.
282.         function getUserReferralBonus(address userAddress)
283.             public
284.             view
285.             returns (uint256)
286.         {
287.             return users[userAddress].bonus;
288.         }
289.
290.         function getUserReferralTotalBonus(address userAddress)
291.             public
292.             view
293.             returns (uint256)
294.         {
295.             return users[userAddress].totalBonus;
296.         }
297.
298.         function getUserReferralWithdrawn(address userAddress)
299.             public
300.             view
301.             returns (uint256)
302.         {
303.             return
    users[userAddress].totalBonus.sub(users[userAddress].bonus);
304.         }
305.
306.         function getUserAvailable(address userAddress)
307.             public
308.             view
309.             returns (uint256)
310.         {
311.             return
312.                 getUserReferralBonus(userAddress).add(
313.                     getUserDividends(userAddress)
314.                 );
315.         }
```

```
316.
317.         function getUserAmountOfDeposits(address userAddress)
318.             public
319.             view
320.             returns (uint256)
321.         {
322.             return users[userAddress].deposits.length;
323.         }
324.
325.         function getUserTotalDeposits(address userAddress)
326.             public
327.             view
328.             returns (uint256 amount)
329.         {
330.             for (uint256 i = 0; i < users[userAddress].deposits.length;
    i++) {
331.                 amount =
    amount.add(users[userAddress].deposits[i].amount);
332.             }
333.         }
334.
335.         function getUserPlanTotalAmount(address userAddress, uint8 plan)
336.             public
337.             view
338.             returns (
339.                 uint256 totalAmount
340.             )
341.         {
342.             User storage user = users[userAddress];
343.             uint256 amount = 0;
344.
345.             for( uint256 i = 0; i < user.deposits.length; i++){
346.                 if(user.deposits[i].plan == plan){
347.                     amount = user.deposits[i].amount;
348.                 }
349.             }
350.
351.             totalAmount = amount;
352.         }
353.
354.         function getUserDepositInfo(address userAddress, uint256 index)
355.             public
356.             view
357.             returns (
358.                 uint8 plan,
```

```
359.                 uint256 percent,
360.                 uint256 amount,
361.                 uint256 start,
362.                 uint256 finish
363.             )
364.         {
365.             User storage user = users[userAddress];
366.
367.             if( index < user.deposits.length )
368.             {
369.                 plan = user.deposits[index].plan;
370.                 percent = plans[plan].percent;
371.                 amount = user.deposits[index].amount;
372.                 start = user.deposits[index].start;
373.                 finish = user.deposits[index].start.add(
374.                     plans[user.deposits[index].plan].time.mul(1 days)
375.                 );
376.             }else{
377.                 plan = 0;
378.                 percent = 0;
379.                 amount = 0;
380.                 start = 0;
381.                 finish = 0;
382.             }
383.         }
384.
385.         function getUserDepositsCount(address userAddress)
386.             public
387.             view
388.             returns (uint256 length)
389.         {
390.             User storage user = users[userAddress];
391.
392.             return user.deposits.length;
393.         }
394.
395.         function getSiteInfo()
396.             public
397.             view
398.             returns (uint256 _totalInvested, uint256 _totalBonus)
399.         {
400.             return (totalInvested, totalRefBonus);
401.         }
402.
403.         function getUserInfo(address userAddress)
```

```
404.             public
405.             view
406.             returns (
407.                 uint256 totalDeposit,
408.                 uint256 totalWithdrawn,
409.                 uint256 totalReferrals
410.             )
411.         {
412.             return (
413.                 getUserTotalDeposits(userAddress),
414.                 getUserTotalWithdrawn(userAddress),
415.                 getUserTotalReferrals(userAddress)
416.             );
417.         }
418.
419.         function isContract(address addr) internal view returns (bool) {
420.             uint256 size;
421.             assembly {
422.                 size := extcodesize(addr)
423.             }
424.             return size > 0;
425.         }
426.
427.         function setAdditionalPercent_Plan1(uint256 value) external {
428.             require(msg.sender == ceoAddress);
429.             ADDITIONAL_PERCENT_PLAN_1 = value;
430.         }
431.         function setAdditionalPercent_Plan2(uint256 value) external {
432.             require(msg.sender == ceoAddress);
433.             ADDITIONAL_PERCENT_PLAN_2 = value;
434.         }
435.         function setAdditionalPercent_Plan3(uint256 value) external {
436.             require(msg.sender == ceoAddress);
437.             ADDITIONAL_PERCENT_PLAN_3 = value;
438.         }
439.         function setAdditionalPercent_Plan4(uint256 value) external {
440.             require(msg.sender == ceoAddress);
441.             ADDITIONAL_PERCENT_PLAN_4 = value;
442.         }
443.     }
444.
445.     library SafeMath {
446.         function add(uint256 a, uint256 b) internal pure returns
    (uint256) {
447.             uint256 c = a + b;
```

```
448.             require(c >= a, "SafeMath: addition overflow");
449.
450.             return c;
451.         }
452.
453.         function sub(uint256 a, uint256 b) internal pure returns
    (uint256) {
454.             require(b <= a, "SafeMath: subtraction overflow");
455.             uint256 c = a - b;
456.
457.             return c;
458.         }
459.
460.         function mul(uint256 a, uint256 b) internal pure returns
    (uint256) {
461.             if (a == 0) {
462.                 return 0;
463.             }
464.
465.             uint256 c = a * b;
466.             require(c / a == b, "SafeMath: multiplication overflow");
467.
468.             return c;
469.         }
470.
471.         function div(uint256 a, uint256 b) internal pure returns
    (uint256) {
472.             require(b > 0, "SafeMath: division by zero");
473.             uint256 c = a / b;
474.
475.             return c;
476.         }
477.     }
```