CyberCrime Shield

cybercrimeshield.org

Smart Contract Audit Report

The BUSD Crops Farmer

https://busd.cropsfarmer.online/

AUDIT TYPE: **PUBLIC**

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

https://cybercrimeshield.org/secure/busd-cropsfarmer

Report ID: 291982

Apr 19, 2022

TABLE OF CONTENTS

SMART CONTRACT....3
DESCRIPTION....4
INTRODUCTION....4
AUDIT METHODOLOGY....5
ISSUES DISCOVERED....6
AUDIT SUMMARY....6
FINDINGS....7
CONCLUSION....8
SOURCE CODE....9

SMART CONTRACT

https://bscscan.com/address/0x8be8881c641dc5a40845253ee3ed04955edfe96d#code

Mirror:

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

CRC32: 43A8F699

MD5: 7F019CFEAF1150DF624D114612814C61

SHA-1: A07B2796E5DC7E55C143B3AB2A0F11B82B0EBE03

DESCRIPTION

8% Daily ~ 2920% APR

8% Referral Bonus

5% Development/Marketing Fee

2.5% Hire Bonus

12 Hours Compound Timer

4 Hours Withdraw Cooldown

48 Hours Rewards Accumulation Cut-Off

5 Times Mandatory Compound Feature

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 BUSD Crops Farmer 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 2022-04-18
*/

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

interface IToken {
    function totalSupply() external view returns (uint256);

    function balanceOf(address account) external view returns (uint256);

    function transfer(address recipient, uint256 amount)
        external
        returns (bool);

```

```
17.     function allowance(address owner, address spender)
18.         external
19.         view
20.         returns (uint256);
21.
22.     function approve(address spender, uint256 amount) external returns
    (bool);
23.
24.     function transferFrom(
25.         address sender,
26.         address recipient,
27.         uint256 amount
28.     ) external returns (bool);
29.
30.     event Transfer(address indexed from, address indexed to, uint256
    value);
31.     event Approval(
32.         address indexed owner,
33.         address indexed spender,
34.         uint256 value
35.     );
36.}
37.
38.library SafeMath {
39.
40.   function mul(uint256 a, uint256 b) internal pure returns (uint256) {
41.     if (a == 0) {
42.       return 0;
43.     }
44.     uint256 c = a * b;
45.     assert(c / a == b);
46.     return c;
47.   }
48.
49.
50.   function div(uint256 a, uint256 b) internal pure returns (uint256) {
51.     uint256 c = a / b;
52.     return c;
53.   }
54.
55.   function sub(uint256 a, uint256 b) internal pure returns (uint256) {
56.     assert(b <= a);
57.     return a - b;
58.   }
59.
```

```
60.   function add(uint256 a, uint256 b) internal pure returns (uint256) {
61.     uint256 c = a + b;
62.     assert(c >= a);
63.     return c;
64.   }
65.
66.   function mod(uint256 a, uint256 b) internal pure returns (uint256) {
67.     require(b != 0);
68.     return a % b;
69.   }
70. }
71.
72. contract BusdCrops {
73.     using SafeMath for uint256;
74.
75.     IToken public token_BUSD;
76.     //address erctoken = 0x78867BbEeF44f2326bF8DDd1941a4439382EF2A7; /**
    BUSD Testnet **/
77.     address erctoken = 0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56; /**
    BUSD Mainnet **/
78.
79.     uint256 public EGGS_TO_HIRE_1MINERS = 1080000;
80.     uint256 public PERCENTS_DIVIDER = 1000;
81.     uint256 public REFERRAL = 80;
82.     uint256 public TAX = 10;
83.     uint256 public MARKET_EGGS_DIVISOR = 2; // 50%
84.     uint256 public MARKET_EGGS_DIVISOR_SELL = 1; // 100%
85.
86.     uint256 public MIN_INVEST_LIMIT = 10 * 1e18; /** 10 BUSD  **/
87.     uint256 public WALLET_DEPOSIT_LIMIT = 10000 * 1e18; /** 10000
    BUSD  **/
88.
89.     uint256 public COMPOUND_BONUS = 25; /** 2.5% **/
90.     uint256 public COMPOUND_BONUS_MAX_TIMES = 10; /** 10 times / 5 days.
    **/
91.     uint256 public COMPOUND_STEP = 12 * 60 * 60; /** every 12 hours. **/
92.
93.     uint256 public WITHDRAWAL_TAX = 600;
94.     uint256 public COMPOUND_FOR_NO_TAX_WITHDRAWAL = 5; // compound days,
    for no tax withdrawal.
95.
96.     uint256 public totalStaked;
97.     uint256 public totalDeposits;
98.     uint256 public totalCompound;
99.     uint256 public totalRefBonus;
```

```
100.        uint256 public totalWithdrawn;
101.
102.        uint256 public marketEggs;
103.        uint256 PSN = 10000;
104.        uint256 PSNH = 5000;
105.        bool public contractStarted;
106.
107.        uint256 public CUTOFF_STEP = 48 * 60 * 60; /** 48 hours  **/
108.        uint256 public WITHDRAW_COOLDOWN = 4 * 60 * 60; /** 4 hours  **/
109.
110.        address public owner;
111.        address public dev1;
112.        address public dev2;
113.        address public dev3;
114.        address public dev4;
115.        address public mkt;
116.
117.        struct User {
118.            uint256 initialDeposit;
119.            uint256 userDeposit;
120.            uint256 miners;
121.            uint256 claimedEggs;
122.            uint256 lastHatch;
123.            address referrer;
124.            uint256 referralsCount;
125.            uint256 referralEggRewards;
126.            uint256 totalWithdrawn;
127.            uint256 dailyCompoundBonus;
128.            uint256 lastWithdrawTime;
129.        }
130.
131.        mapping(address => User) public users;
132.
133.        constructor(address _dev1, address _dev2, address _dev3, address
    _dev4, address _mkt) {
134.            require(!isContract(_dev1) && !isContract(_dev2) &&
    !isContract(_dev3) && !isContract(_dev4) && !isContract(_mkt));
135.            owner = msg.sender;
136.            dev1 = _dev1;
137.            dev2 = _dev2;
138.            dev3 = _dev3;
139.            dev4 = _dev4;
140.            mkt = _mkt;
141.            token_BUSD = IToken(erctoken);
142.        }
```

```
143.
144.        function isContract(address addr) internal view returns (bool) {
145.            uint size;
146.            assembly { size := extcodesize(addr) }
147.            return size > 0;
148.        }
149.
150.        function hatchEggs(bool isCompound) public {
151.            User storage user = users[msg.sender];
152.            require(contractStarted, "Contract not yet Started.");
153.
154.            uint256 eggsUsed = getMyEggs();
155.            uint256 eggsForCompound = eggsUsed;
156.
157.            if(isCompound) {
158.                uint256 dailyCompoundBonus =
    getDailyCompoundBonus(msg.sender, eggsForCompound);
159.                eggsForCompound =
    eggsForCompound.add(dailyCompoundBonus);
160.                uint256 eggsUsedValue =
    calculateEggSell(eggsForCompound);
161.                user.userDeposit = user.userDeposit.add(eggsUsedValue);
162.                totalCompound = totalCompound.add(eggsUsedValue);
163.            }
164.
165.            if(block.timestamp.sub(user.lastHatch) >= COMPOUND_STEP) {
166.                if(user.dailyCompoundBonus < COMPOUND_BONUS_MAX_TIMES) {
167.                    user.dailyCompoundBonus =
    user.dailyCompoundBonus.add(1);
168.                }
169.            }
170.
171.            user.miners =
    user.miners.add(eggsForCompound.div(EGGS_TO_HIRE_1MINERS));
172.            user.claimedEggs = 0;
173.            user.lastHatch = block.timestamp;
174.
175.            marketEggs =
    marketEggs.add(eggsUsed.div(MARKET_EGGS_DIVISOR));
176.        }
177.
178.        function sellEggs() public{
179.            require(contractStarted);
180.            User storage user = users[msg.sender];
181.            uint256 hasEggs = getMyEggs();
```

```
182.             uint256 eggValue = calculateEggSell(hasEggs);
183.
184.             /**
185.                 if user compound < to mandatory compound days**/
186.             if(user.dailyCompoundBonus < COMPOUND_FOR_NO_TAX_WITHDRAWAL){
187.                 //daily compound bonus count will not reset and eggValue
    will be deducted with 60% feedback tax.
188.                 eggValue =
    eggValue.sub(eggValue.mul(WITHDRAWAL_TAX).div(PERCENTS_DIVIDER));
189.             }else{
190.                 //set daily compound bonus count to 0 and eggValue will
    remain without deductions
191.                  user.dailyCompoundBonus = 0;
192.             }
193.
194.             user.lastWithdrawTime = block.timestamp;
195.             user.claimedEggs = 0;
196.             user.lastHatch = block.timestamp;
197.             marketEggs =
    marketEggs.add(hasEggs.div(MARKET_EGGS_DIVISOR_SELL));
198.
199.             if(getBalance() < eggValue) {
200.                 eggValue = getBalance();
201.             }
202.
203.             uint256 eggsPayout = eggValue.sub(payFees(eggValue));
204.             token_BUSD.transfer(msg.sender, eggsPayout);
205.             user.totalWithdrawn = user.totalWithdrawn.add(eggsPayout);
206.             totalWithdrawn = totalWithdrawn.add(eggsPayout);
207.         }
208.
209.         function buyEggs(address ref, uint256 amount) public{
210.             require(contractStarted);
211.             User storage user = users[msg.sender];
212.             require(amount >= MIN_INVEST_LIMIT, "Mininum investment not
    met.");
213.             require(user.initialDeposit.add(amount) <=
    WALLET_DEPOSIT_LIMIT, "Max deposit limit reached.");
214.
215.             token_BUSD.transferFrom(address(msg.sender), address(this),
    amount);
216.             uint256 eggsBought = calculateEggBuy(amount,
    getBalance().sub(amount));
217.             user.userDeposit = user.userDeposit.add(amount);
218.             user.initialDeposit = user.initialDeposit.add(amount);
```

```
219.             user.claimedEggs = user.claimedEggs.add(eggsBought);
220.
221.             if (user.referrer == address(0)) {
222.                 if (ref != msg.sender) {
223.                     user.referrer = ref;
224.                 }
225.
226.                 address upline1 = user.referrer;
227.                 if (upline1 != address(0)) {
228.                     users[upline1].referralsCount =
    users[upline1].referralsCount.add(1);
229.                 }
230.             }
231.
232.             if (user.referrer != address(0)) {
233.                 address upline = user.referrer;
234.                 if (upline != address(0)) {
235.                     uint256 refRewards =
    amount.mul(REFERRAL).div(PERCENTS_DIVIDER);
236.                     token_BUSD.transfer(upline, refRewards);
237.                     users[upline].referralEggRewards =
    users[upline].referralEggRewards.add(refRewards);
238.                     totalRefBonus = totalRefBonus.add(refRewards);
239.                 }
240.             }
241.
242.             uint256 eggsPayout = payFees(amount);
243.             /** less the fee on total Staked to give more transparency of
    data. **/
244.             totalStaked = totalStaked.add(amount.sub(eggsPayout));
245.             totalDeposits = totalDeposits.add(1);
246.             hatchEggs(false);
247.         }
248.
249.         function payFees(uint256 eggValue) internal returns(uint256){
250.             uint256 tax = eggValue.mul(TAX).div(PERCENTS_DIVIDER);
251.             token_BUSD.transfer(dev1, tax);
252.             token_BUSD.transfer(dev2, tax);
253.             token_BUSD.transfer(dev3, tax);
254.             token_BUSD.transfer(dev4, tax);
255.             token_BUSD.transfer(mkt, tax);
256.             return tax.mul(5);
257.         }
258.
```

```
259.         function getDailyCompoundBonus(address _adr, uint256 amount)
   public view returns(uint256){
260.             if(users[_adr].dailyCompoundBonus == 0) {
261.                 return 0;
262.             } else {
263.                 uint256 totalBonus =
   users[_adr].dailyCompoundBonus.mul(COMPOUND_BONUS);
264.                 uint256 result =
   amount.mul(totalBonus).div(PERCENTS_DIVIDER);
265.                 return result;
266.             }
267.         }
268.
269.         function getUserInfo(address _adr) public view returns(uint256
   _initialDeposit, uint256 _userDeposit, uint256 _miners,
270.          uint256 _claimedEggs, uint256 _lastHatch, address _referrer,
   uint256 _referrals,
271.          uint256 _totalWithdrawn, uint256 _referralEggRewards, uint256
   _dailyCompoundBonus, uint256 _lastWithdrawTime) {
272.              _initialDeposit = users[_adr].initialDeposit;
273.              _userDeposit = users[_adr].userDeposit;
274.              _miners = users[_adr].miners;
275.              _claimedEggs = users[_adr].claimedEggs;
276.              _lastHatch = users[_adr].lastHatch;
277.              _referrer = users[_adr].referrer;
278.              _referrals = users[_adr].referralsCount;
279.              _totalWithdrawn = users[_adr].totalWithdrawn;
280.              _referralEggRewards = users[_adr].referralEggRewards;
281.              _dailyCompoundBonus = users[_adr].dailyCompoundBonus;
282.              _lastWithdrawTime = users[_adr].lastWithdrawTime;
283.         }
284.
285.         function initialize(uint256 amount) public{
286.             if (!contractStarted) {
287.                 if (msg.sender == owner) {
288.                     require(marketEggs == 0);
289.                     contractStarted = true;
290.                     marketEggs = 86400000000;
291.                     buyEggs(msg.sender, amount);
292.                 } else revert("Contract not yet started.");
293.             }
294.         }
295.
296.         function getBalance() public view returns (uint256) {
297.             return token_BUSD.balanceOf(address(this));
```

```
298.         }
299.
300.         function getTimeStamp() public view returns (uint256) {
301.             return block.timestamp;
302.         }
303.
304.         function getAvailableEarnings(address _adr) public view
    returns(uint256) {
305.             uint256 userEggs =
    users[_adr].claimedEggs.add(getEggsSinceLastHatch(_adr));
306.             return calculateEggSell(userEggs);
307.         }
308.
309.         function calculateTrade(uint256 rt,uint256 rs, uint256 bs) public
    view returns(uint256){
310.             return SafeMath.div(SafeMath.mul(PSN, bs), SafeMath.add(PSNH,
    SafeMath.div(SafeMath.add(SafeMath.mul(PSN, rs), SafeMath.mul(PSNH, rt)),
    rt)));
311.         }
312.
313.         function calculateEggSell(uint256 eggs) public view
    returns(uint256){
314.             return calculateTrade(eggs, marketEggs, getBalance());
315.         }
316.
317.         function calculateEggBuy(uint256 eth,uint256 contractBalance)
    public view returns(uint256){
318.             return calculateTrade(eth, contractBalance, marketEggs);
319.         }
320.
321.         function calculateEggBuySimple(uint256 eth) public view
    returns(uint256){
322.             return calculateEggBuy(eth, getBalance());
323.         }
324.
325.         function getEggsYield(uint256 amount) public view
    returns(uint256,uint256) {
326.             uint256 eggsAmount = calculateEggBuy(amount ,
    getBalance().add(amount).sub(amount));
327.             uint256 miners = eggsAmount.div(EGGS_TO_HIRE_1MINERS);
328.             uint256 day = 1 days;
329.             uint256 eggsPerDay = day.mul(miners);
330.             uint256 earningsPerDay = calculateEggSellForYield(eggsPerDay,
    amount);
331.             return(miners, earningsPerDay);
```

```
332.         }
333.
334.         function calculateEggSellForYield(uint256 eggs,uint256 amount)
   public view returns(uint256){
335.             return calculateTrade(eggs,marketEggs,
   getBalance().add(amount));
336.         }
337.
338.         function getSiteInfo() public view returns (uint256 _totalStaked,
   uint256 _totalDeposits, uint256 _totalCompound, uint256 _totalRefBonus) {
339.             return (totalStaked, totalDeposits, totalCompound,
   totalRefBonus);
340.         }
341.
342.         function getMyMiners() public view returns(uint256){
343.             return users[msg.sender].miners;
344.         }
345.
346.         function getMyEggs() public view returns(uint256){
347.             return
   users[msg.sender].claimedEggs.add(getEggsSinceLastHatch(msg.sender));
348.         }
349.
350.         function getEggsSinceLastHatch(address adr) public view
   returns(uint256){
351.             uint256 secondsSinceLastHatch =
   block.timestamp.sub(users[adr].lastHatch);
352.                                     /** get min time. **/
353.             uint256 cutoffTime = min(secondsSinceLastHatch, CUTOFF_STEP);
354.             uint256 secondsPassed = min(EGGS_TO_HIRE_1MINERS,
   cutoffTime);
355.             return secondsPassed.mul(users[adr].miners);
356.         }
357.
358.         function min(uint256 a, uint256 b) private pure returns (uint256)
   {
359.             return a < b ? a : b;
360.         }
361.
362.         /** wallet addresses setters **/
363.         function CHANGE_OWNERSHIP(address value) external {
364.             require(msg.sender == owner, "Admin use only.");
365.             owner = value;
366.         }
367.
```

```
368.         function CHANGE_DEV1(address value) external {
369.             require(msg.sender == dev1, "Admin use only.");
370.             dev1 = value;
371.         }
372.
373.         function CHANGE_DEV2(address value) external {
374.             require(msg.sender == dev2, "Admin use only.");
375.             dev2 = value;
376.         }
377.
378.         function CHANGE_DEV3(address value) external {
379.             require(msg.sender == dev3, "Admin use only.");
380.             dev3 = value;
381.         }
382.
383.         function CHANGE_MKT_WALLET(address value) external {
384.             require(msg.sender == mkt, "Admin use only.");
385.             mkt = value;
386.         }
387.
388.         /** percentage setters **/
389.
390.         // 2592000 - 3%, 2160000 - 4%, 1728000 - 5%, 1440000 - 6%,
    1200000 - 7%, 1080000 - 8%
391.         // 959000 - 9%, 864000 - 10%, 720000 - 12%, 575424 - 15%, 540000
    - 16%, 479520 - 18%
392.
393.         function PRC_EGGS_TO_HIRE_1MINERS(uint256 value) external {
394.             require(msg.sender == owner, "Admin use only.");
395.             require(value >= 479520 && value <= 2592000); /** min 3% max
    12%**/
396.             EGGS_TO_HIRE_1MINERS = value;
397.         }
398.
399.         function PRC_TAX(uint256 value) external {
400.             require(msg.sender == owner, "Admin use only.");
401.             require(value <= 100); /** 10% max **/
402.             TAX = value;
403.         }
404.
405.         function PRC_REFERRAL(uint256 value) external {
406.             require(msg.sender == owner, "Admin use only.");
407.             require(value >= 10 && value <= 100); /** 10% max **/
408.             REFERRAL = value;
409.         }
```

```
410.
411.         function PRC_MARKET_EGGS_DIVISOR(uint256 value) external {
412.             require(msg.sender == owner, "Admin use only.");
413.             require(value <= 50); /** 50 = 2% **/
414.             MARKET_EGGS_DIVISOR = value;
415.         }
416.
417.         /** withdrawal tax **/
418.         function SET_WITHDRAWAL_TAX(uint256 value) external {
419.             require(msg.sender == owner, "Admin use only.");
420.             require(value <= 800); /** Max Tax is 80% or lower **/
421.             WITHDRAWAL_TAX = value;
422.         }
423.
424.         function SET_COMPOUND_FOR_NO_TAX_WITHDRAWAL(uint256 value)
    external {
425.             require(msg.sender == owner, "Admin use only.");
426.             COMPOUND_FOR_NO_TAX_WITHDRAWAL = value;
427.         }
428.
429.         function BONUS_DAILY_COMPOUND(uint256 value) external {
430.             require(msg.sender == owner, "Admin use only.");
431.             require(value >= 10 && value <= 900);
432.             COMPOUND_BONUS = value;
433.         }
434.
435.         function BONUS_DAILY_COMPOUND_BONUS_MAX_TIMES(uint256 value)
    external {
436.             require(msg.sender == owner, "Admin use only.");
437.             require(value <= 30);
438.             COMPOUND_BONUS_MAX_TIMES = value;
439.         }
440.
441.         function BONUS_COMPOUND_STEP(uint256 value) external {
442.             require(msg.sender == owner, "Admin use only.");
443.             COMPOUND_STEP = value * 60 * 60;
444.         }
445.
446.         function SET_MIN_INVEST_LIMIT(uint256 value) external {
447.             require(msg.sender == owner, "Admin use only");
448.             MIN_INVEST_LIMIT = value * 1e18;
449.         }
450.
451.         function SET_CUTOFF_STEP(uint256 value) external {
452.             require(msg.sender == owner, "Admin use only");
```

```
453.             CUTOFF_STEP = value * 60 * 60;
454.         }
455.
456.         function SET_WITHDRAW_COOLDOWN(uint256 value) external {
457.             require(msg.sender == owner, "Admin use only");
458.             require(value <= 24);
459.             WITHDRAW_COOLDOWN = value * 60 * 60;
460.         }
461.
462.         function SET_WALLET_DEPOSIT_LIMIT(uint256 value) external {
463.             require(msg.sender == owner, "Admin use only");
464.             require(value >= 20);
465.             WALLET_DEPOSIT_LIMIT = value * 1e18;
466.         }
467.     }
468.
469.
470.     /**
471.     BUSD Crops Farmer
472.      Hire Farmers, Gather Crops, Harvest Crops and Sell for BUSD.
473.       8% daily daily Rate
474.       8% Referral Bonus, will go directly to referrer wallet.
475.       2.5% stacking compound bonus every 12 hrs, max of 5 days. (25%)
476.       48 hours cut off time.
477.       10 BUSD minimum investment.
478.       10,000 BUSD max deposits per wallet.
479.       60% feedback for withdrawals that will be done not after 5
   consecutive compounds.(12.5%)
480.       Withdrawals will reset daily compound count back to 0.
481.        *Tax will stay in the contract.
482.     */
```