CyberCrime Shield

cybercrimeshield.org

Smart Contract Audit Report

ATLANTIS

by AGE OF MINING

https://atlantis.ageofmining.com

AUDIT TYPE: **PUBLIC**

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

https://cybercrimeshield.org/secure/atlantis

Report ID: A0125936

September 10, 2022

TABLE OF CONTENTS

SMART CONTRACT

https://bscscan.com/address/0xe2bbfdd762563df80bd4ad0f86d4aeb15968f1c4#code

Mirror: https://cybercrimeshield.org/secure/uploads/atlantis.sol

v0.8.16+commit.07a7930e

License: MIT

CRC32: B864647D

MD5: 796C6D19E5140623800E8D73A416DDA4

SHA-1: 123D8691DFF2EC542A67D82EE73014D213EB6F88

INTRODUCTION

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

To eliminate the need for trust, Nakamoto 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.

Consequently, 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 Atlantis 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

- The contract has a clear structure and is easy to read
- Gas usage is optimal
- Contract is fully Ethereum & BSC completable
- No backdoors or overflows are present in the contract

SOURCE CODE

```
/**
 *Submitted for verification at BscScan.com on 2022-09-08
*/

/**
    Vaporeons will collect pearls for you which can later be exchanged with BUSD. Atlantis will let you mine with a stable 12% daily roi.

    Working Time - 24 hours
    Withdraw - Every 7 days
    Referral Bonus: 4%
    Daily ROI: 8-14%, start form 12%
**/

// SPDX-License-Identifier: MIT
pragma solidity 0.8.16;
```

```
16.
17.abstract contract Context {
18.
19.    function _msgSender() internal view virtual returns (address) {
20.        return msg.sender;
21.    }
22.
23.    function _msgData() internal view virtual returns (bytes calldata) {
24.        return msg.data;
25.    }
26.}
27.
28.abstract contract ReentrancyGuard {
29.    bool internal locked;
30.
31.    modifier noReentrant() {
32.        require(!locked, "No re-entrancy");
33.        locked = true;
34.        _;
35.        locked = false;
36.    }
37.}
38.
39.contract Atlantis is Context , ReentrancyGuard {
40.    using SafeMath for uint256;
41.    bool public contractStarted;
42.    uint256 public constant min = 50 ether;
43.    uint256 public constant max = 30000 ether;
44.    uint256 public roi = 12;
45.    uint256 public constant fee = 1;
46.    uint256 public constant ref_fee = 4;
47.    uint256 public withdrawDays = 7 days;
48.    uint256 public claimDays = 1 days;
49.    address private dev;
50.    address private dev1;
51.    address private partner1;
52.    IERC20 private BusdInterface;
53.    address public tokenAdress;
54.    bool public antibotActive = true;
55.    mapping(address => bool) public AntiBotlisted;
56.
57.
58.    constructor(address _dev,address _dev1,address _partner1) {
59.        tokenAdress = 0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56;
60.        BusdInterface = IERC20(tokenAdress);
```

```
61.         owner = msg.sender;
62.         dev = _dev;
63.         dev1 = _dev1;
64.         partner1 = _partner1;
65.     }
66.
67.     struct refferal_system {
68.         address ref_address;
69.         uint256 reward;
70.     }
71.
72.     struct refferal_withdraw {
73.         address ref_address;
74.         uint256 totalWithdraw;
75.     }
76.
77.     struct user_investment_details {
78.         address user_address;
79.         uint256 invested;
80.     }
81.
82.     struct weeklyWithdraw {
83.         address user_address;
84.         uint256 startTime;
85.         uint256 deadline;
86.     }
87.
88.     struct claimDaily {
89.         address user_address;
90.         uint256 startTime;
91.         uint256 deadline;
92.     }
93.
94.     struct userWithdrawal {
95.         address user_address;
96.         uint256 amount;
97.     }
98.
99.     struct userTotalWithdraw {
100.            address user_address;
101.            uint256 amount;
102.        }
103.
104.        struct userTotalRewards {
105.            address user_address;
```

```
106.             uint256 amount;
107.         }
108.
109.         mapping(address => refferal_system) public refferal;
110.         mapping(address => user_investment_details) public investments;
111.         mapping(address => weeklyWithdraw) public weekly;
112.         mapping(address => claimDaily) public claimTime;
113.         mapping(address => userWithdrawal) public approvedWithdrawal;
114.         mapping(address => userTotalWithdraw) public totalWithdraw;
115.         mapping(address => userTotalRewards) public totalRewards;
116.         mapping(address => refferal_withdraw) public refTotalWithdraw;
117.         mapping(address => bool) public isInvested;
118.         address public owner;
119.
120.         function setAntiBotActive(bool isActive) public{
121.             require(msg.sender == owner, "Admin use only.");
122.             antibotActive = isActive;
123.         }
124.
125.         function antiBotWallet(address Wallet, bool isAntiBotlisted) public{
126.             require(msg.sender == owner, "Admin use only.");
127.             AntiBotlisted[Wallet] = isAntiBotlisted;
128.         }
129.
130.         function antiBotMultipleWallets(address[] calldata Wallet, bool
    isAntiBotlisted) public{
131.             require(msg.sender == owner, "Admin use only.");
132.             for(uint256 i = 0; i < Wallet.length; i++) {
133.                 AntiBotlisted[Wallet[i]] = isAntiBotlisted;
134.             }
135.         }
136.         function checkIfAntiBotlisted(address Wallet) public view returns(bool
    antibotlisted) {
137.             require(msg.sender == owner, "Admin use only.");
138.             antibotlisted = AntiBotlisted[Wallet];
139.         }
140.
141.         function userReward(address _userAddress) public view returns(uint256) {
142.             uint256 totalTime =
    claimTime[_userAddress].deadline.sub(claimTime[_userAddress].startTime);
143.             uint256 value =
    DailyRoi(investments[_userAddress].invested).div(totalTime);
144.
145.             if(claimTime[_userAddress].deadline >= block.timestamp) {
```

```
146.                 uint256 earned =
    block.timestamp.sub(claimTime[_userAddress].startTime);
147.                 return earned.mul(value);
148.             }
149.             else {
150.                 return DailyRoi(investments[_userAddress].invested);
151.             }
152.         }
153.
154.         function deposit(address _ref, uint256 _amount) public noReentrant  {
155.             require(contractStarted, "Not Launched!");
156.             require(_amount >= min && _amount <= max, "User cannot deposit.
    Please check minimum/maximum deposit.");
157.
158.             //referral bonus
159.             if(_ref != address(0) && _ref != msg.sender){
160.                 uint256 ref_fee_add = refFee(_amount);
161.                 refferal[_ref] = refferal_system(_ref,
    ref_fee_add.add(refferal[_ref].reward));
162.             }
163.
164.             //claim existing dividends
165.             if(isInvested[msg.sender] && userReward(msg.sender) > 0){
166.                 uint256 rewards = userReward(msg.sender);
167.                 approvedWithdrawal[msg.sender] = userWithdrawal(msg.sender,
    approvedWithdrawal[msg.sender].amount.add(rewards));
168.                 totalRewards[msg.sender].amount =
    totalRewards[msg.sender].amount.add(rewards);
169.             }
170.
171.             //record investment
172.             investments[msg.sender] = user_investment_details(msg.sender,
    investments[msg.sender].invested.add(_amount));
173.
174.             //reset weekly withdraw time
175.             weekly[msg.sender] = weeklyWithdraw(msg.sender, block.timestamp,
    block.timestamp.add(withdrawDays));
176.
177.             //reset daily claim time
178.             claimTime[msg.sender] = claimDaily(msg.sender, block.timestamp,
    block.timestamp.add(claimDays));
179.
180.             //transfer amount
181.             payFeesAndGetRemaining(_amount, true);
182.
```

```
183.             //will be set to true if new investor
184.             isInvested[msg.sender] = true;
185.         }
186.
187.         function payFeesAndGetRemaining(uint256 _amount, bool fromDeposit)
    internal {
188.             uint256 taxFee = projectFee(_amount);
189.             uint256 totalAmount = _amount.sub(taxFee.mul(5));
190.             if(fromDeposit){
191.                 BusdInterface.transferFrom(msg.sender, dev, taxFee);
192.                 BusdInterface.transferFrom(msg.sender, dev1, taxFee);
193.                 BusdInterface.transferFrom(msg.sender, partner1, taxFee);
194.                 BusdInterface.transferFrom(msg.sender, address(this),
    totalAmount);
195.             }else{
196.                 BusdInterface.transfer(dev, taxFee);
197.                 BusdInterface.transfer(dev1, taxFee);
198.                 BusdInterface.transfer(partner1, taxFee);
199.                 BusdInterface.transfer(msg.sender, totalAmount);
200.             }
201.         }
202.
203.         function withdrawal() public noReentrant {
204.             require(contractStarted, "Not Launched!");
205.             if (antibotActive) {
206.                 require(!AntiBotlisted[msg.sender], "Address is blacklisted.");
207.             }
208.             require(weekly[msg.sender].deadline <= block.timestamp, "User can't
    withdraw.");
209.             require(totalWithdraw[msg.sender].amount <=
    investments[msg.sender].invested.mul(3), "User's total withdrawn is already 3x of
    his investment.");
210.             uint256 aval_withdraw = approvedWithdrawal[msg.sender].amount;
211.
212.             if(totalWithdraw[msg.sender].amount.add(aval_withdraw) >=
    investments[msg.sender].invested.mul(3)){
213.                 aval_withdraw =
    investments[msg.sender].invested.mul(3).sub(totalWithdraw[msg.sender].amount);
214.             }
215.
216.             //current reward / 2
217.             uint256 aval_withdraw2 = aval_withdraw.div(2);
218.
219.             //reset to 0
220.             approvedWithdrawal[msg.sender] = userWithdrawal(msg.sender, 0);
```

```
221.
222.             //50% re-invested.
223.             investments[msg.sender] = user_investment_details(msg.sender,
    investments[msg.sender].invested.add(aval_withdraw2));
224.
225.             //50% withdrawn
226.             totalWithdraw[msg.sender] = userTotalWithdraw(msg.sender,
    totalWithdraw[msg.sender].amount.add(aval_withdraw2));
227.
228.             //reset weekly withdraw time
229.             weekly[msg.sender] = weeklyWithdraw(msg.sender, block.timestamp,
    block.timestamp.add(withdrawDays));
230.
231.             //transfer amount
232.             payFeesAndGetRemaining(aval_withdraw2, false);
233.         }
234.
235.         function claimDailyRewards() public noReentrant{
236.             require(contractStarted, "Not Launched!");
237.             require(claimTime[msg.sender].deadline <= block.timestamp, "User
    can't claim yet.");
238.
239.             //update withdrawable amount
240.             approvedWithdrawal[msg.sender] = userWithdrawal(msg.sender,
    userReward(msg.sender).add(approvedWithdrawal[msg.sender].amount));
241.
242.             //update available rewards
243.             totalRewards[msg.sender].amount =
    totalRewards[msg.sender].amount.add(userReward(msg.sender));
244.
245.             //reset claim time
246.             claimTime[msg.sender] = claimDaily(msg.sender, block.timestamp,
    block.timestamp.add(claimDays));
247.         }
248.
249.         function Ref_Withdraw() external noReentrant {
250.             require(contractStarted, "Not Launched!");
251.             require(investments[msg.sender].invested > 0, "Need to invest!");
252.             uint256 value = refferal[msg.sender].reward.div(2);
253.             refferal[msg.sender] = refferal_system(msg.sender, 0);
254.
255.             //50% re-invested
256.             investments[msg.sender] = user_investment_details(msg.sender,
    investments[msg.sender].invested.add(value));
257.
```

```
258.             //50% referral bonus withdrawn
259.             refTotalWithdraw[msg.sender] = refferal_withdraw(msg.sender,
   refTotalWithdraw[msg.sender].totalWithdraw.add(value));
260.
261.             //transfer referral bonus
262.             BusdInterface.transfer(msg.sender, value);
263.         }
264.
265.         function updateDailyRoi(uint256 value) external {
266.             require(msg.sender == owner, "Admin use only.");
267.             require(value >= 8 && value <= 14 );
268.             roi = value;
269.         }
270.
271.         function Start(bool value) public {
272.             require(msg.sender == owner, "Admin use only.");
273.             contractStarted = value;
274.         }
275.
276.         function DailyRoi(uint256 _amount) public view returns(uint256) {
277.             return _amount.mul(roi).div(100);
278.         }
279.
280.         function refFee(uint256 _amount) public pure returns(uint256) {
281.             return _amount.mul(ref_fee).div(100);
282.         }
283.
284.         function projectFee(uint256 _amount) public pure returns(uint256) {
285.             return _amount.mul(fee).div(100);
286.         }
287.
288.         function getBalance() public view returns(uint256){
289.              return BusdInterface.balanceOf(address(this));
290.         }
291.
292.         function CHANGE_OWNERSHIP(address value) external {
293.             require(msg.sender == owner, "Admin use only.");
294.             owner = value;
295.         }
296.     }
297.
298.     interface IERC20 {
299.         function totalSupply() external view returns (uint256);
300.
301.         function balanceOf(address account) external view returns (uint256);
```

```
302.
303.         function transfer(address recipient, uint256 amount) external returns
    (bool);
304.
305.         function allowance(address owner, address spender) external view returns
    (uint256);
306.
307.         function approve(address spender, uint256 amount) external returns
    (bool);
308.
309.         function transferFrom(address sender, address recipient, uint256 amount)
    external returns (bool);
310.
311.         event Transfer(address indexed from, address indexed to, uint256 value);
312.
313.         event Approval(address indexed owner, address indexed spender, uint256
    value);
314.     }
315.
316.     library SafeMath {
317.
318.         function tryAdd(uint256 a, uint256 b) internal pure returns (bool,
    uint256) {
319.             unchecked {
320.                 uint256 c = a + b;
321.                 if (c < a) return (false, 0);
322.                 return (true, c);
323.             }
324.         }
325.
326.         function trySub(uint256 a, uint256 b) internal pure returns (bool,
    uint256) {
327.             unchecked {
328.                 if (b > a) return (false, 0);
329.                 return (true, a - b);
330.             }
331.         }
332.
333.         function tryMul(uint256 a, uint256 b) internal pure returns (bool,
    uint256) {
334.             unchecked {
335.                 if (a == 0) return (true, 0);
336.                 uint256 c = a * b;
337.                 if (c / a != b) return (false, 0);
338.                 return (true, c);
```

```
339.             }
340.         }
341.
342.         function tryDiv(uint256 a, uint256 b) internal pure returns (bool,
   uint256) {
343.             unchecked {
344.                 if (b == 0) return (false, 0);
345.                 return (true, a / b);
346.             }
347.         }
348.
349.         function tryMod(uint256 a, uint256 b) internal pure returns (bool,
   uint256) {
350.             unchecked {
351.                 if (b == 0) return (false, 0);
352.                 return (true, a % b);
353.             }
354.         }
355.
356.         function add(uint256 a, uint256 b) internal pure returns (uint256) {
357.             return a + b;
358.         }
359.
360.         function sub(uint256 a, uint256 b) internal pure returns (uint256) {
361.             return a - b;
362.         }
363.
364.         function mul(uint256 a, uint256 b) internal pure returns (uint256) {
365.             return a * b;
366.         }
367.
368.         function div(uint256 a, uint256 b) internal pure returns (uint256) {
369.             return a / b;
370.         }
371.
372.         function mod(uint256 a, uint256 b) internal pure returns (uint256) {
373.             return a % b;
374.         }
375.
376.         function sub(
377.             uint256 a,
378.             uint256 b,
379.             string memory errorMessage
380.         ) internal pure returns (uint256) {
381.             unchecked {
```

```
382.                 require(b <= a, errorMessage);
383.                 return a - b;
384.             }
385.         }
386.
387.         function div(
388.             uint256 a,
389.             uint256 b,
390.             string memory errorMessage
391.         ) internal pure returns (uint256) {
392.             unchecked {
393.                 require(b > 0, errorMessage);
394.                 return a / b;
395.             }
396.         }
397.
398.         function mod(
399.             uint256 a,
400.             uint256 b,
401.             string memory errorMessage
402.         ) internal pure returns (uint256) {
403.             unchecked {
404.                 require(b > 0, errorMessage);
405.                 return a % b;
406.             }
407.         }
408.     }
```