Smart Contract Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the BNB (BNB) on 2024.04.28. The following are the details and results of this smart contract security audit:

Token Name :

BNB (BNB)

The contract address :

https://etherscan.io/token/0xB8c77482e45F1F44dE1745F52C74426C631bDD52

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

NO.	Audit Items	Result
1	Replay Vulnerability	Passed
2	Denial of Service Vulnerability	Passed
3	Race Conditions Vulnerability	Passed
4	Authority Control Vulnerability Audit	Passed
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Passed
8	Uninitialized Storage Pointers Vulnerability	Passed
9	Arithmetic Accuracy Deviation Vulnerability	Passed
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed
12	Scoping and Declarations Audit	Passed
13	Safety Design Audit	Passed
14	Non-privacy/Non-dark Coin Audit	Passed

Audit Result : Passed

Audit Number : 0X002404280002

Audit Date : 2024.04.28 - 2024.04.28

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that does not contain the token vault section and dark coin functions. The total amount of contract tokens can be changed, users can burn their tokens through the burn function. The contract does not have the Overflow and the Race Conditions issue.

During the audit, we found the following information:

1. Users can freeze or unfreeze the locked tokens through the freeze and unfreeze functions and the frozen or the unfrozen tokens will be directly added or subtracted from the balance.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2017-07-06
*/
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
//SlowMist// It is recommended to use emit to declare event calls, and the compiler version needs to be greater than 0.4.21
pragma solidity ^0.4.8;

/**
 * Math operations with safety checks
 */
//SlowMist// SafeMath security module is used, which is a recommend approach
contract SafeMath {
  function safeMul(uint256 a, uint256 b) internal returns (uint256) {
    uint256 c = a * b;
    assert(a == 0 || c / a == b);
    return c;
  }

  function safeDiv(uint256 a, uint256 b) internal returns (uint256) {
    assert(b > 0);
    uint256 c = a / b;
    assert(a == b * c + a % b);
    return c;
  }

  function safeSub(uint256 a, uint256 b) internal returns (uint256) {
```

```
    assert(b <= a);
    return a - b;
  }

  function safeAdd(uint256 a, uint256 b) internal returns (uint256) {
    uint256 c = a + b;
    assert(c>=a && c>=b);
    return c;
  }

  function assert(bool assertion) internal {
    if (!assertion) {
      throw;
    }
  }
}
contract BNB is SafeMath{
    string public name;
    string public symbol;
    uint8 public decimals;
    uint256 public totalSupply;
        address public owner;

    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;
        mapping (address => uint256) public freezeOf;
    mapping (address => mapping (address => uint256)) public allowance;

    /* This generates a public event on the blockchain that will notify clients */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /* This notifies clients about the amount burnt */
    event Burn(address indexed from, uint256 value);

        /* This notifies clients about the amount frozen */
    event Freeze(address indexed from, uint256 value);

        /* This notifies clients about the amount unfrozen */
    event Unfreeze(address indexed from, uint256 value);

    /* Initializes contract with initial supply tokens to the creator of the contract
*/
    function BNB(
        uint256 initialSupply,
        string tokenName,
        uint8 decimalUnits,
        string tokenSymbol
        ) {
        balanceOf[msg.sender] = initialSupply;              // Give the creator all
```

```
initial tokens
        totalSupply = initialSupply;                        // Update total supply
        name = tokenName;                                   // Set the name for
display purposes
        symbol = tokenSymbol;                               // Set the symbol for
display purposes
        decimals = decimalUnits;                            // Amount of decimals for
display purposes
                owner = msg.sender;
    }

    /* Send coins */
    //SlowMist// It's recommended to add return value to return true or false
    function transfer(address _to, uint256 _value) {
        //SlowMist// This kind of check is very good, avoiding user mistake leading to
the loss of token during transfer
        if (_to == 0x0) throw;                               // Prevent transfer to
0x0 address. Use burn() instead
                if (_value <= 0) throw;
        if (balanceOf[msg.sender] < _value) throw;           // Check if the sender
has enough
        if (balanceOf[_to] + _value < balanceOf[_to]) throw; // Check for overflows
        balanceOf[msg.sender] = SafeMath.safeSub(balanceOf[msg.sender], _value);
// Subtract from the sender
        balanceOf[_to] = SafeMath.safeAdd(balanceOf[_to], _value);
// Add the same to the recipient
        Transfer(msg.sender, _to, _value);                   // Notify anyone
listening that this transfer took place
    }

    /* Allow another contract to spend some tokens in your behalf */
    function approve(address _spender, uint256 _value)
        returns (bool success) {
                if (_value <= 0) throw;
        allowance[msg.sender][_spender] = _value;
        //SlowMist// The return value conforms to the EIP20 specification
        return true;
    }


    /* A contract attempts to get the coins */
    function transferFrom(address _from, address _to, uint256 _value) returns (bool
success) {
        //SlowMist// This kind of check is very good, avoiding user mistake leading to
the loss of token during transfer
        if (_to == 0x0) throw;                               // Prevent transfer to
0x0 address. Use burn() instead
                if (_value <= 0) throw;
        if (balanceOf[_from] < _value) throw;                // Check if the sender
```

```
has enough
        if (balanceOf[_to] + _value < balanceOf[_to]) throw;  // Check for overflows
        if (_value > allowance[_from][msg.sender]) throw;     // Check allowance
        balanceOf[_from] = SafeMath.safeSub(balanceOf[_from], _value);
// Subtract from the sender
        balanceOf[_to] = SafeMath.safeAdd(balanceOf[_to], _value);
// Add the same to the recipient
        allowance[_from][msg.sender] = SafeMath.safeSub(allowance[_from][msg.sender],
_value);
        Transfer(_from, _to, _value);
        //SlowMist// The return value conforms to the EIP20 specification
        return true;
    }

    function burn(uint256 _value) returns (bool success) {
        if (balanceOf[msg.sender] < _value) throw;            // Check if the sender
has enough
                if (_value <= 0) throw;
        balanceOf[msg.sender] = SafeMath.safeSub(balanceOf[msg.sender], _value);
// Subtract from the sender
        totalSupply = SafeMath.safeSub(totalSupply,_value);
// Updates totalSupply
        Burn(msg.sender, _value);
        return true;
    }

        function freeze(uint256 _value) returns (bool success) {
        if (balanceOf[msg.sender] < _value) throw;            // Check if the sender
has enough
                if (_value <= 0) throw;
        balanceOf[msg.sender] = SafeMath.safeSub(balanceOf[msg.sender], _value);
// Subtract from the sender
        freezeOf[msg.sender] = SafeMath.safeAdd(freezeOf[msg.sender], _value);
// Updates totalSupply
        Freeze(msg.sender, _value);
        return true;
    }

        function unfreeze(uint256 _value) returns (bool success) {
        if (freezeOf[msg.sender] < _value) throw;            // Check if the sender
has enough
                if (_value <= 0) throw;
        freezeOf[msg.sender] = SafeMath.safeSub(freezeOf[msg.sender], _value);
// Subtract from the sender
                balanceOf[msg.sender] = SafeMath.safeAdd(balanceOf[msg.sender],
_value);
        Unfreeze(msg.sender, _value);
        return true;
    }
```

```
        // transfer balance to owner
        function withdrawEther(uint256 amount) {
                if(msg.sender != owner)throw;
                owner.transfer(amount);
        }

        // can accept ether
        function() payable {
    }
}
```

Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist