Smart Contract Security Audit Report

[2021]

The SlowMist Security Team received the BKC team's application for smart contract security audit of the BankCoin on 2021.05.31. The following are the details and results of this smart contract security audit:

Token Name :

BankCoin

The contract address :

https://github.com/BankCoin-BKC/BKC

Commit: 52951ad8fdef2b197fd5d36934fd5c5221359080

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	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

NO.	Audit Items	Result
12	Scoping and Declarations Audit	Passed
13	Safety Design Audit	Passed

Audit Result : Passed

Audit Number : 0x002106020001

Audit Date : 2021.05.31 - 2021.06.02

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that does not contain the tokenVault section. The total amount of tokens in the contract can be changed. SafeMath security module is not used. The contract does not have the Overflow and the Race Conditions issue.

The source code:

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
//SlowMist// Using the OpenZeppelin's SafeMath security Module, it is a recommend approach
pragma solidity ^0.4.26;

interface tokenRecipient { function receiveApproval(address _from, uint256 _value, address _token, bytes _extraData) external; }

contract BKC {

    string public name;
    string public symbol;
    uint8 public decimals = 18;

    uint256 public totalSupply;


    mapping (address => uint256) public balanceOf;
    mapping (address => mapping (address => uint256)) public allowance;

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Burn(address indexed from, uint256 value);
```

```
    event Approval(address indexed owner, address indexed spender, uint256 value);

    constructor (uint256 initialSupply, string tokenName, string tokenSymbol) public
{
        totalSupply = initialSupply * 10 ** uint256(decimals);  // Update total
supply with the decimal amount
        balanceOf[msg.sender] = totalSupply;                    // Give the creator
all initial tokens
        name = tokenName;                                       // Set the name for
display purposes
        symbol = tokenSymbol;                                   // Set the symbol for
display purposes
    }

    /**
      * @dev total number of tokens issued
    */
    function totalSupply() public view returns (uint256) {
        return totalSupply;
    }

    /**
      * @dev Gets the balance of the specified address.
      * @param _owner The address to query the the balance of.
      * @return An uint256 representing the amount owned by the passed address.
    */
    function _balanceOf(address _owner) public view returns (uint256 balance) {
        return balanceOf[_owner];
    }

    function _transfer(address _from, address _to, uint _value) internal {
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to the loss of token during transfer
        require(_to != 0x0);
        require(balanceOf[_from] >= _value);
        require(balanceOf[_to] + _value > balanceOf[_to]);
        uint previousBalances = balanceOf[_from] + balanceOf[_to];
        balanceOf[_from] -= _value;
        balanceOf[_to] += _value;
        emit Transfer(_from, _to, _value);
        assert(balanceOf[_from] + balanceOf[_to] == previousBalances);
    }

    /**
      * @dev transfer token for a specified address
```

```
      * @param _to The address to transfer to.
      * @param _value The amount to be transferred.
    */
    function transfer(address _to, uint256 _value) public {
        _transfer(msg.sender, _to, _value);
    }

    /**
       * @dev Transfer tokens from one address to another
       * @param _from address The address which you want to send tokens from
       * @param _to address The address which you want to transfer to
       * @param _value uint256 the amount of tokens to be transferred
    */
    function transferFrom(address _from, address _to, uint256 _value) public returns
(bool success) {
        require(_value <= allowance[_from][msg.sender]);     // Check allowance
        allowance[_from][msg.sender] -= _value;
        _transfer(_from, _to, _value);
        //SlowMist// The return value conforms to the EIP20 specification
        return true;
    }

    /**
       * @dev Approve the passed address to spend the specified amount of tokens on
behalf of msg.sender.
       * @param _spender The address which will spend the funds.
       * @param _value The amount of tokens to be spent.
    */
    function approve(address _spender, uint256 _value) public
        returns (bool success) {
        allowance[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);
        //SlowMist// The return value conforms to the EIP20 specification
        return true;
    }

    function approveAndCall(address _spender, uint256 _value, bytes _extraData)
        public
        returns (bool success) {
        tokenRecipient spender = tokenRecipient(_spender);
        if (approve(_spender, _value)) {
            spender.receiveApproval(msg.sender, _value, this, _extraData);
            return true;
        }
    }
```

```
    /**
       * @dev Function to check the amount of tokens that an owner allowed to a
spender.
       * @param _owner address The address which owns the funds.
       * @param _spender address The address which will spend the funds.
       * @return A uint256 specifying the amount of tokens still available for the
spender.
    */
    function _allowance(address _owner, address _spender) public view returns
(uint256) {
        return allowance[_owner][_spender];
    }

    function burn(uint256 _value) public returns (bool success) {
        require(balanceOf[msg.sender] >= _value);   // Check if the sender has enough
        balanceOf[msg.sender] -= _value;            // Subtract from the sender
        totalSupply -= _value;                      // Updates totalSupply
        emit Burn(msg.sender, _value);
        return true;
    }

    //SlowMist// Because burnFrom() and transferFrom() share the allowed amount of
approve(), if the agent be evil, there is the possibility of malicious burn
    function burnFrom(address _from, uint256 _value) public returns (bool success) {
        require(balanceOf[_from] >= _value);                // Check if the targeted
balance is enough
        require(_value <= allowance[_from][msg.sender]);    // Check allowance
        balanceOf[_from] -= _value;                         // Subtract from the
targeted balance
        allowance[_from][msg.sender] -= _value;             // Subtract from the
sender's allowance
        totalSupply -= _value;                              // Update totalSupply
        emit Burn(_from, _value);
        return true;
    }
}
```

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