Smart Contract Security Audit Report

The SlowMist Security Team received the BNA team's application for smart contract security audit of the Banana on January 06, 2020. The following are the details and results of this smart contract security audit:

Token name :

BNA

The Contract address :

0x20910e5b5f087f6439dfcb0dda4e27d1014ac2b8

Link address :

https://etherscan.io/address/0x20910e5b5f087f6439dfcb0dda4e27d1014ac2b8

The audit items and results :

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

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive auditing authority	Passed
4	Safety Design Audit	Zeppelin module safe use	Passed
		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed
6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Passed
8	“False Deposit” vulnerability Audit	-	Passed

9	Malicious Event Log Audit	-	Passed
10	Scoping and Declarations Audit	-	Passed
11	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
12	Uninitialized Storage Pointers Audit	-	Passed
13	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : **Passed**

Audit Number : 0X002001100001

Audit Date : January 10, 2020

Audit Team : SlowMist Security Team

(**Statement :** SlowMist only issues this report based on the fact that has occurred or existed before the report is issued, and bears the corresponding responsibility in this regard. For the facts occur or exist later after the report, SlowMist cannot judge the security status of its smart contract. SlowMist is not responsible for it. The security audit analysis and other contents of this report are based on the documents and materials provided by the information provider to SlowMist as of the date of this report (referred to as "the provided information"). SlowMist assumes that: there has been no information missing, tampered, deleted, or concealed. If the information provided has been missed, modified, deleted, concealed or reflected and is inconsistent with the actual situation, SlowMist will not bear any responsibility for the resulting loss and adverse effects. SlowMist will not bear any responsibility for the background or other circumstances of the project.)

Summary: This is a token contract that contain the tokenVault section. The total amount of contract tokens can be changed, users can burn their tokens through the burn function. Owner can freeze any user address through the freezeAccount function. Owner can lock any user address through the lockAccount function. OpenZeppelin's SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue. The comprehensive evaluation contract is no risk.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2019-03-21
*/
```

//SlowMist// The contract does not have the Overflow and the Race Conditions issue

```
pragma solidity >=0.4.22 <0.6.0;
```

```
/**
 * @title SafeMath
 * @dev Unsigned math operations with safety checks that revert on error
 */
```

//SlowMist// OpenZeppelin ' s SafeMath security Module is used, which is a recommend approach

```
library SafeMath {
    /**
    * @dev Multiplies two unsigned integers, reverts on overflow.
    */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
    * @dev Integer division of two unsigned integers truncating the quotient, reverts on division by zero.
    */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
    * @dev Subtracts two unsigned integers, reverts on overflow (i.e. if subtrahend is greater than minuend).
    */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
```

```
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Adds two unsigned integers, reverts on overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    /**
     * @dev Divides two unsigned integers and returns the remainder (unsigned integer modulo),
     * reverts when dividing by zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}


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

/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev The Ownable constructor sets the original `owner` of the contract to the sender
```

```
     * account.
     */
    constructor () internal {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

    /**
     * @return the address of the owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner());
        _;
    }

    /**
     * @return true if `msg.sender` is the owner of the contract.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }

    /**
     * @dev Allows the current owner to relinquish control of the contract.
     * @notice Renouncing to ownership will leave the contract without an owner.
     * It will not be possible to call the functions with the `onlyOwner`
     * modifier anymore.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Allows the current owner to transfer control of the contract to a newOwner.
```

```
     * @param newOwner The address to transfer ownership to.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0)); //SlowMist// This check is quite good in avoiding losing control of the contract caused by user mistakes
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}


/**
 * @title Pausable
 * @dev Base contract which allows children to implement an emergency stop mechanism.
 */
contract Pausable is Ownable{
    event Paused(address account);
    event Unpaused(address account);

    bool private _paused;

    constructor () internal {
        _paused = false;
    }

    /**
     * @return true if the contract is paused, false otherwise.
     */
    function paused() public view returns (bool) {
        return _paused;
    }
```

```
    /**
     * @dev Modifier to make a function callable only when the contract is not paused.
     */
    modifier whenNotPaused() {
        require(!_paused);
        _;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is paused.
     */
    modifier whenPaused() {
        require(_paused);
        _;
    }

    /**
     * @dev called by the owner to pause, triggers stopped state
     */
```

//SlowMist// Suspending all transactions upon major abnormalities is a recommended approach

```
    function pause() public onlyOwner whenNotPaused {
        _paused = true;
        emit Paused(msg.sender);
    }

    /**
     * @dev called by the owner to unpause, returns to normal state
     */
    function unpause() public onlyOwner whenPaused {
        _paused = false;
        emit Unpaused(msg.sender);
    }
}


contract ERC20Token{
    using SafeMath for uint256;

    // Public variables of the token
```

```
    string public name;
    string public symbol;
    uint8 public decimals = 18;
    // 18 decimals is the strongly suggested default, avoid changing it
    uint256 public totalSupply;

    // This creates an array with all balances
    mapping (address => uint256) public balanceOf;
    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 generates a public event on the blockchain that will notify clients
    event Approval(address indexed _owner, address indexed _spender, uint256 _value);

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

    /**
     * Constructor function
     *
     * Initializes contract with initial supply tokens to the creator of the contract
     */
    constructor(uint256 initialSupply,string memory tokenName,string memory 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
    }

    /**
     * Internal transfer, only can be called by this contract
     */
    function _transfer(address _from, address _to, uint _value) internal {
        // Prevent transfer to 0x0 address. Use burn() instead
        require(_to != address(0x0)); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer
        balanceOf[_from] = balanceOf[_from].sub(_value);
```

```
        balanceOf[_to] = balanceOf[_to].add(_value);

        emit Transfer(_from, _to, _value);
    }

    /**
     * Transfer tokens
     *
     * Send `_value` tokens to `_to` from your account
     *
     * @param _to The address of the recipient
     * @param _value the amount to send
     */
    function transfer(address _to, uint256 _value) public returns (bool success) {
        _transfer(msg.sender, _to, _value);

        return true; //SlowMist// The return value conforms to the EIP20 specification

    }

    /**
     * Transfer tokens from other address
     *
     * Send `_value` tokens to `_to` on behalf of `_from`
     *
     * @param _from The address of the sender
     * @param _to The address of the recipient
     * @param _value the amount to send
     */
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool success) {

        //SlowMist// require(_value <= allowance[_from][msg.sender]);

        //SlowMist// It is recommended to add to code above, can optimize Gas

        allowance[_from][msg.sender] = allowance[_from][msg.sender].sub(_value);
        _transfer(_from, _to, _value);

        return true; //SlowMist// The return value conforms to the EIP20 specification

    }

    /**
     * Set allowance for other address
     *
```

```
    * Allows `_spender` to spend no more than `_value` tokens on your behalf
    *
    * @param _spender The address authorized to spend
    * @param _value the max amount they can spend
    */
    function approve(address _spender, uint256 _value) public
        returns (bool success) {
        allowance[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);

        return true; //SlowMist// The return value conforms to the EIP20 specification

    }

    /**
    * Set allowance for other address and notify
    *
    * Allows `_spender` to spend no more than `_value` tokens on your behalf, and then ping the contract about it
    *
    * @param _spender The address authorized to spend
    * @param _value the max amount they can spend
    * @param _extraData some extra information to send to the approved contract
    */
    function approveAndCall(address _spender, uint256 _value, bytes memory _extraData)
        public
        returns (bool success) {
        tokenRecipient spender = tokenRecipient(_spender);
        if (approve(_spender, _value)) {
            spender.receiveApproval(msg.sender, _value, address(this), _extraData);
            return true;
        }
    }

    /**
    * Destroy tokens
    *
    * Remove `_value` tokens from the system irreversibly
    *
    * @param _value the amount of money to burn
    */
    function burn(uint256 _value) public returns (bool success) {
        balanceOf[msg.sender] = balanceOf[msg.sender].sub(_value);                    // Subtract from the sender
        totalSupply = totalSupply.sub(_value);                                        // Updates totalSupply
```

```
        emit Burn(msg.sender, _value);
        return true;
    }

    /**
     * Destroy tokens from other account
     *
     * Remove `_value` tokens from the system irreversibly on behalf of `_from`.
     *
     * @param _from the address of the sender
     * @param _value the amount of money to burn
     */
```

//SlowMist// Because burnFrom() and transferFrom() share the allowance 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) {
        balanceOf[_from] = balanceOf[_from].sub(_value);                            // Subtract from the targeted balance
        allowance[_from][msg.sender] = allowance[_from][msg.sender].sub(_value);    // Subtract from the sender's allowance
        totalSupply = totalSupply.sub(_value);                                      // Update totalSupply
        emit Burn(_from, _value);
        return true;
    }

}


contract BananaToken is ERC20Token, Ownable,Pausable{

    mapping (address => bool) public frozenAccount;

    mapping(address => uint256) public lockedAccount;

    event FreezeAccount(address account, bool frozen);

    event LockAccount(address account,uint256 unlockTime);

    constructor() ERC20Token(5000000000,"Banana","BNA") public {
    }
```

```
    /**
     * Freeze Account
     */
    function freezeAccount(address account) onlyOwner public {
        frozenAccount[account] = true;
        emit FreezeAccount(account, true);
    }


    /**
     * unFreeze Account
     */
    function unFreezeAccount(address account) onlyOwner public{
        frozenAccount[account] = false;
        emit FreezeAccount(account, false);
    }

    /**
     * lock Account, if account is locked, fund can only transfer in but not transfer out.
     * Can not unlockAccount, if need unlock account , pls call unlockAccount interface
      */
    function lockAccount(address account, uint256 unlockTime) onlyOwner public{
        require(unlockTime > now);
        lockedAccount[account] = unlockTime;
        emit LockAccount(account,unlockTime);
    }

   /**
     * unlock Account
      */
    function unlockAccount(address account) onlyOwner public{
        lockedAccount[account] = 0;
        emit LockAccount(account,0);
    }


    function changeName(string memory newName) public onlyOwner {
        name = newName;
    }

    function changeSymbol(string memory newSymbol) public onlyOwner{
```

```
        symbol = newSymbol;
    }

    /**
    * Internal transfer, only can be called by this contract
    */
    function _transfer(address _from, address _to, uint _value) internal whenNotPaused {
        // Prevent transfer to 0x0 address. Use burn() instead
        require(_to != address(0x0)); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer

        //if account is frozen, then fund can not be transfer in or out.
        require(!frozenAccount[_from]);
        require(!frozenAccount[_to]);

        //if account is locked, then fund can only transfer in but can not transfer out.
        require(!isAccountLocked(_from));

        balanceOf[_from] = balanceOf[_from].sub(_value);
        balanceOf[_to] = balanceOf[_to].add(_value);

        emit Transfer(_from, _to, _value);
    }


    function isAccountLocked(address account) public view returns (bool) {
        return lockedAccount[account] > now;
    }

    function isAccountFrozen(address account) public view returns (bool){
        return frozenAccount[account];
    }
}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account