Smart Contract Security Audit Report

The SlowMist Security Team received the BSI team's application for smart contract security audit of the BSI token on Feb. 26, 2021. The following are the details and results of this smart contract security audit:

Token name :

BSI

The Contract address :

0xEB50455805ebF8396d9177BBC4A371A376D00ECC

Link address :

https://etherscan.io/address/0xEB50455805ebF8396d9177BBC4A371A376D00ECC

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 authority audit	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 top-up” 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 : 0X002102280001

Audit Date : Feb. 28, 2021

Audit Team : SlowMist Security Team

Summary: This is a token contract that contains the tokenVault section. The total amount of contract tokens can be changed, owner can burn his own tokens through the burn function. SafeMath security module is used, which is a recommend approach. The contract does not have the Overflow and the Race Conditions issue. During the audit, we found the following information:

1. The owner can distribute and lock tokens to users through the distributeWithLockup function.

2. The owner can unlock the token distributed and locked by the owner to the user through the unlock function.

3. The owner can freeze the account of any user through freezeAccount.

4. The owner can withdraw any tokens in this contract through the claimToken function.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2021-02-25
*/
```

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

```
pragma solidity ^0.4.25;
```

//SlowMist// SafeMath security module is used, which is a recommend approach

```
library SafeMath {

    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }
        uint256 c = a * b;
        assert(c / a == b);
        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // assert(b > 0); // Solidity automatically throws when dividing by 0
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold
        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        assert(b <= a);
        return a - b;
    }

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



contract Ownable {
    address public owner;
```

```
    address public newOwner;

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

    constructor() public {
        owner = msg.sender;
        newOwner = address(0);
    }

    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }
    modifier onlyNewOwner() {
        require(msg.sender != address(0));
        require(msg.sender == newOwner);
        _;
    }

    function transferOwnership(address _newOwner) public onlyOwner {
        require(_newOwner != address(0)); //SlowMist// This check is quite good in avoiding losing control of
the contract caused by user mistakes
        newOwner = _newOwner;
    }

    function acceptOwnership() public onlyNewOwner returns(bool) {
        emit OwnershipTransferred(owner, newOwner);
        owner = newOwner;
    }
}

contract Pausable is Ownable {
  event Pause();
  event Unpause();

  bool public paused = false;

  modifier whenNotPaused() {
    require(!paused);
    _;
```

```
    }

    modifier whenPaused() {
        require(paused);
        _;
    }

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

    function pause() onlyOwner whenNotPaused public {
        paused = true;
        emit Pause();
    }

    function unpause() onlyOwner whenPaused public {
        paused = false;
        emit Unpause();
    }
}

contract ERC20 {
    function totalSupply() public view returns (uint256);
    function balanceOf(address who) public view returns (uint256);
    function allowance(address owner, address spender) public view returns (uint256);
    function transfer(address to, uint256 value) public returns (bool);
    function transferFrom(address from, address to, uint256 value) public returns (bool);
    function approve(address spender, uint256 value) public returns (bool);

    event Approval(address indexed owner, address indexed spender, uint256 value);
    event Transfer(address indexed from, address indexed to, uint256 value);
}


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


contract BSIToken is ERC20, Ownable, Pausable {

    using SafeMath for uint256;

    struct LockupInfo {
```

```
        uint256 releaseTime;
        uint256 termOfRound;
        uint256 unlockAmountPerRound;
        uint256 lockupBalance;
    }

    string public name;
    string public symbol;
    uint8 public decimals;
    uint256 internal initialSupply;
    uint256 internal totalSupply_;

    mapping(address => uint256) internal balances;
    mapping(address => bool) internal locks;
    mapping(address => bool) public frozen;
    mapping(address => mapping(address => uint256)) internal allowed;
    mapping(address => LockupInfo) internal lockupInfo;

    event Unlock(address indexed holder, uint256 value);
    event Lock(address indexed holder, uint256 value);
    event Burn(address indexed owner, uint256 value);
    event Mint(uint256 value);
    event Freeze(address indexed holder);
    event Unfreeze(address indexed holder);

    modifier notFrozen(address _holder) {
        require(!frozen[_holder]);
        _;
    }

    constructor() public {
        name = "BSI Token";
        symbol = "BSI";
        decimals = 18;
        initialSupply = 1000000000;
        totalSupply_ = initialSupply * 10 ** uint(decimals);
        balances[owner] = totalSupply_;
        emit Transfer(address(0), owner, totalSupply_);
    }

    function () public payable {
        revert();
```

```
    }

    function totalSupply() public view returns (uint256) {
        return totalSupply_;
    }

    function transfer(address _to, uint256 _value) public whenNotPaused notFrozen(msg.sender) returns (bool) {
        if (locks[msg.sender]) {
            autoUnlock(msg.sender);
        }

        require(_to != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake
leading to the loss of token during transfer

        require(_value <= balances[msg.sender]);



        // SafeMath.sub will throw if there is not enough balance.
        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        emit Transfer(msg.sender, _to, _value);

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

    }

    function balanceOf(address _holder) public view returns (uint balance) {
        return balances[_holder];
    }

    function lockupBalance(address _holder) public view returns (uint256 balance) {
        return lockupInfo[_holder].lockupBalance;
    }

    function transferFrom(address _from, address _to, uint256 _value) public whenNotPaused notFrozen(_from)returns
(bool) {
        if (locks[_from]) {
            autoUnlock(_from);
        }

        require(_to != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake
leading to the loss of token during transfer
```

```
        require(_value <= balances[_from]);
        require(_value <= allowed[_from][msg.sender]);



        balances[_from] = balances[_from].sub(_value);
        balances[_to] = balances[_to].add(_value);
        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
        emit Transfer(_from, _to, _value);

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

    }

    function approve(address _spender, uint256 _value) public whenNotPaused returns (bool) {
        allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);

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

    }

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

    function allowance(address _holder, address _spender) public view returns (uint256) {
        return allowed[_holder][_spender];
    }

    function lock(address _holder, uint256 _amount, uint256 _releaseStart, uint256 _termOfRound, uint256 _releaseRate)
internal onlyOwner returns (bool) {
        require(locks[_holder] == false);
        require(_releaseStart > now);
        require(_termOfRound > 0);
        require(_amount.mul(_releaseRate).div(100) > 0);
        require(balances[_holder] >= _amount);
        balances[_holder] = balances[_holder].sub(_amount);
        lockupInfo[_holder] = LockupInfo(_releaseStart, _termOfRound, _amount.mul(_releaseRate).div(100), _amount);
```

```
        locks[_holder] = true;

        emit Lock(_holder, _amount);

        return true;
    }

    //SlowMist// The owner can unlock the token distributed and locked by the owner to the user through the unlock function

    function unlock(address _holder) public onlyOwner returns (bool) {
        require(locks[_holder] == true);
        uint256 releaseAmount = lockupInfo[_holder].lockupBalance;

        delete lockupInfo[_holder];
        locks[_holder] = false;

        emit Unlock(_holder, releaseAmount);
        balances[_holder] = balances[_holder].add(releaseAmount);

        return true;
    }

    //SlowMist// The owner can freeze the account of any user through freezeAccount

    function freezeAccount(address _holder) public onlyOwner returns (bool) {
        require(!frozen[_holder]);
        frozen[_holder] = true;
        emit Freeze(_holder);
        return true;
    }

    //SlowMist// The owner can unfreeze the account of any user through unfreezeAccount

    function unfreezeAccount(address _holder) public onlyOwner returns (bool) {
        require(frozen[_holder]);
        frozen[_holder] = false;
        emit Unfreeze(_holder);
        return true;
    }

    function getNowTime() public view returns(uint256) {
      return now;
```

```
    }

    function showLockState(address _holder) public view returns (bool, uint256, uint256, uint256, uint256) {
        return (locks[_holder], lockupInfo[_holder].lockupBalance, lockupInfo[_holder].releaseTime,
lockupInfo[_holder].termOfRound, lockupInfo[_holder].unlockAmountPerRound);
    }

    function distribute(address _to, uint256 _value) public onlyOwner returns (bool) {
        require(_to != address(0));
        require(_value <= balances[owner]);

        balances[owner] = balances[owner].sub(_value);
        balances[_to] = balances[_to].add(_value);
        emit Transfer(owner, _to, _value);
        return true;
    }
```

//SlowMist// The owner can distribute and lock tokens to users through the distributeWithLockup function

```
    function distributeWithLockup(address _to, uint256 _value, uint256 _releaseStart, uint256 _termOfRound, uint256
_releaseRate) public onlyOwner returns (bool) {
        distribute(_to, _value);
        lock(_to, _value, _releaseStart, _termOfRound, _releaseRate);
        return true;
    }
```

//SlowMist// The owner can withdraw any tokens in this contract through the claimToken function

```
    function claimToken(ERC20 token, address _to, uint256 _value) public onlyOwner returns (bool) {
        token.transfer(_to, _value);
        return true;
    }

    function burn(uint256 _value) public onlyOwner returns (bool success) {
        require(_value <= balances[msg.sender]);
        address burner = msg.sender;
        balances[burner] = balances[burner].sub(_value);
        totalSupply_ = totalSupply_.sub(_value);
        emit Burn(burner, _value);
        return true;
```

```
    }

    function isContract(address addr) internal view returns (bool) {
        uint size;
        assembly{size := extcodesize(addr)}
        return size > 0;
    }

    function autoUnlock(address _holder) internal returns (bool) {
        if (lockupInfo[_holder].releaseTime <= now) {
            return releaseTimeLock(_holder);
        }
        return false;
    }

    function releaseTimeLock(address _holder) internal returns(bool) {
        require(locks[_holder]);
        uint256 releaseAmount = 0;
        // If lock status of holder is finished, delete lockup info.

        for( ; lockupInfo[_holder].releaseTime <= now ; )
        {
            if (lockupInfo[_holder].lockupBalance <= lockupInfo[_holder].unlockAmountPerRound) {
                releaseAmount = releaseAmount.add(lockupInfo[_holder].lockupBalance);
                delete lockupInfo[_holder];
                locks[_holder] = false;
                break;
            } else {
                releaseAmount = releaseAmount.add(lockupInfo[_holder].unlockAmountPerRound);
                lockupInfo[_holder].lockupBalance =
lockupInfo[_holder].lockupBalance.sub(lockupInfo[_holder].unlockAmountPerRound);

                lockupInfo[_holder].releaseTime = lockupInfo[_holder].releaseTime.add(lockupInfo[_holder].termOfRound);

            }
        }

        emit Unlock(_holder, releaseAmount);
        balances[_holder] = balances[_holder].add(releaseAmount);
        return true;
    }
}
```

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist