Smart Contract Security Audit Report

The SlowMist Security Team received the ShowGo team's application for smart contract security audit of the SHOW on June 05, 2020. The following are the details and results of this smart contract security audit:

Token name :

SHOW

The Contract address :

0xf7eee4440e02e32ff993911cd20552760f14ca52

Link address :

https://etherscan.io/address/0xf7eee4440e02e32ff993911cd20552760f14ca52

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 : 0X002006080001

Audit Date : June 08, 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 does not contain the tokenVault section. The total amount of contract tokens can be changed, owner can burn tokens of blacklisted users through destroyBlackFunds function. The owner can add any user to the blacklist through the addBlackList function. This is an upgradable contract. The owner can upgrade the contract at any time. It is suggested to set the owner to MultiSig contract to reduce the risk of being attacked. SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2020-06-04
*/
```

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

```
pragma solidity ^0.5.0;

/**
 * @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 public owner;

    /**
     * @dev The Ownable constructor sets the original `owner` of the contract to the sender
     * account.
     */
    constructor () public {
        owner = msg.sender;
    }

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

    /**
    * @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 {
        if (newOwner != address(0)) { //SlowMist// This check is quite good in avoiding losing control
of the contract caused by user mistakes
            owner = newOwner;
        }
    }

}

/**
```

```
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * See https://github.com/ethereum/EIPs/issues/179
 */
contract ERC20Basic {
    function totalSupply() public view returns (uint256);
    function balanceOf(address who) public view returns (uint256);
    function transfer(address to, uint256 value) public returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
}

/**
 * @title Basic token
 * @dev Basic version of StandardToken, with no allowances.
 */
contract BasicToken is ERC20Basic {
    using SafeMath for uint256;

    mapping(address => uint256) balances;

    uint256 totalSupply_;

    /**
    * @dev Total number of tokens in existence
    */
    function totalSupply() public view returns (uint256) {
        return totalSupply_;
    }

    /**
    * @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 returns (bool) {

        require(_to != address(0), "Address must not be zero."); //SlowMist// This kind of check is very good,
avoiding user mistake leading to the loss of token during transfer

        require(_value <= balances[msg.sender], "There is no 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
    }

    /**
    * @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) {
        return balances[_owner];
    }

}

/**
  * @title ERC20 interface
  * @dev see https://github.com/ethereum/EIPs/issues/20
  */
contract ERC20 is ERC20Basic {
    function allowance(address owner, address spender)
        public view returns (uint256);

    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
    );
}

/**
  * @title Standard ERC20 token
  *
  * @dev Implementation of the basic standard token.
  * https://github.com/ethereum/EIPs/issues/20
  * Based on code by FirstBlood: https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
```

```
 */
contract StandardToken is ERC20, BasicToken {

    mapping (address => mapping (address => uint256)) internal allowed;


    /**
    * @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)
    {

        require(_to != address(0), "Address must not be zero."); //SlowMist// This kind of check is very good,
avoiding user mistake leading to the loss of token during transfer

        require(_value <= balances[_from], "There is no enough balance.");
        require(_value <= allowed[_from][msg.sender], "There is no enough allowed balance.");

        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

    }


    /**
    * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
    * Beware that changing an allowance with this method brings the risk that someone may use both the old
    * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this
    * race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards:
    * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
    * @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) {
        allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);

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

    }

    /**
    * @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 allowed[_owner][_spender];
    }

    /**
    * @dev Increase the amount of tokens that an owner allowed to a spender.
    * approve should be called when allowed[_spender] == 0. To increment
    * allowed value is better to use this function to avoid 2 calls (and wait until
    * the first transaction is mined)
    * From MonolithDAO Token.sol
    * @param _spender The address which will spend the funds.
    * @param _addedValue The amount of tokens to increase the allowance by.
    */
    function increaseApproval(
        address _spender,
        uint256 _addedValue
    )
        public
        returns (bool)
    {
```

```
        allowed[msg.sender][_spender] = (
        allowed[msg.sender][_spender].add(_addedValue));
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }

    /**
    * @dev Decrease the amount of tokens that an owner allowed to a spender.
    * approve should be called when allowed[_spender] == 0. To decrement
    * allowed value is better to use this function to avoid 2 calls (and wait until
    * the first transaction is mined)
    * From MonolithDAO Token.sol
    * @param _spender The address which will spend the funds.
    * @param _subtractedValue The amount of tokens to decrease the allowance by.
    */
    function decreaseApproval(
        address _spender,
        uint256 _subtractedValue
    )
        public
        returns (bool)
    {
        uint256 oldValue = allowed[msg.sender][_spender];
        if (_subtractedValue > oldValue) {
            allowed[msg.sender][_spender] = 0;
        } else {
            allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
        }
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }

}

/**
  * @title Pausable
  * @dev Base contract which allows children to implement an emergency stop mechanism.
  */
contract Pausable is Ownable {
  event Pause();
  event Unpause();
```

```
  bool public paused = false;


  /**
   * @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() onlyOwner whenNotPaused public {
    paused = true;
    emit Pause();
  }

  /**
   * @dev called by the owner to unpause, returns to normal state
   */
  function unpause() onlyOwner whenPaused public {
    paused = false;
    emit Unpause();
  }
}

contract UpgradedStandardToken is StandardToken{
    // those methods are called by the legacy contract
```

```
    // and they must ensure msg.sender to be the contract address
    function transferByLegacy(address from, address to, uint256 value) public returns (bool);
    function transferFromByLegacy(address sender, address from, address spender, uint256 value) public returns (bool);
    function approveByLegacy(address from, address spender, uint256 value) public returns (bool);
}

contract BlackList is Ownable, BasicToken {

    /////// Getters to allow the same blacklist to be used also by other contracts (including upgraded Tether) ///////
    function getBlackListStatus(address _maker) external view returns (bool) {
        return isBlackListed[_maker];
    }

    function getOwner() external view returns (address) {
        return owner;
    }

    mapping (address => bool) public isBlackListed;

    function addBlackList (address _evilUser) public onlyOwner {
        isBlackListed[_evilUser] = true;
        emit AddedBlackList(_evilUser);
    }

    function removeBlackList (address _clearedUser) public onlyOwner {
        isBlackListed[_clearedUser] = false;
        emit RemovedBlackList(_clearedUser);
    }
```

//SlowMist// The owner can burn tokens of blacklisted users through destroyBlackFunds function

```
    function destroyBlackFunds (address _blackListedUser) public onlyOwner {
        require(isBlackListed[_blackListedUser]);
        uint256 dirtyFunds = balanceOf(_blackListedUser);
        balances[_blackListedUser] = 0;
        totalSupply_ -= dirtyFunds;
        emit DestroyedBlackFunds(_blackListedUser, dirtyFunds);
    }

    event DestroyedBlackFunds(address _blackListedUser, uint256 _balance);
```

```
    event AddedBlackList(address _user);

    event RemovedBlackList(address _user);

}

contract ShowGoToken is Pausable, StandardToken, BlackList {
    string public name = "ShowGo";
    string public symbol = "SHOW";
    uint8 public decimals = 18;
    uint256 public init_Supply = 10 * (10 ** 6) * (10 ** uint256(decimals));
    address public upgradedAddress;
    bool public deprecated;

    constructor() public {
        totalSupply_ = init_Supply;
        balances[msg.sender] = totalSupply_;
        deprecated = false;
    }

    // Forward ERC20 methods to upgraded contract if this one is deprecated
    function transfer(address _to, uint256 _value) public whenNotPaused returns (bool) {
        require(!isBlackListed[msg.sender]);
        require(!isBlackListed[_to]);
        if (deprecated) {
            return UpgradedStandardToken(upgradedAddress).transferByLegacy(msg.sender, _to, _value);
        } else {
            return super.transfer(_to, _value);
        }
    }

    // Forward ERC20 methods to upgraded contract if this one is deprecated
    function transferFrom(address _from, address _to, uint256 _value) public whenNotPaused returns (bool) {
        require(!isBlackListed[msg.sender]);
        require(!isBlackListed[_from]);
        require(!isBlackListed[_to]);
        if (deprecated) {
            return UpgradedStandardToken(upgradedAddress).transferFromByLegacy(msg.sender, _from, _to, _value);
        } else {
            return super.transferFrom(_from, _to, _value);
        }
    }
```

```
// Forward ERC20 methods to upgraded contract if this one is deprecated
function approve(address _spender, uint256 _value) public returns (bool) {
    require(!isBlackListed[msg.sender]);
    require(!isBlackListed[_spender]);
    if (deprecated) {
        return UpgradedStandardToken(upgradedAddress).approveByLegacy(msg.sender, _spender, _value);
    } else {
        return super.approve(_spender, _value);
    }
}

// Forward ERC20 methods to upgraded contract if this one is deprecated
function allowance(address _owner, address _spender) public view    returns (uint256) {
    if (deprecated) {
        return StandardToken(upgradedAddress).allowance(_owner, _spender);
    } else {
        return super.allowance(_owner, _spender);
    }
}

// Forward ERC20 methods to upgraded contract if this one is deprecated
function increaseApproval(address _spender, uint256 _addedValue) public returns (bool) {
    if (deprecated) {
        return StandardToken(upgradedAddress).increaseApproval(_spender, _addedValue);
    } else {
        return super.increaseApproval(_spender, _addedValue);
    }
}

// Forward ERC20 methods to upgraded contract if this one is deprecated
function decreaseApproval(address _spender, uint256 _subtractedValue) public returns (bool) {
    if (deprecated) {
        return StandardToken(upgradedAddress).increaseApproval(_spender, _subtractedValue);
    } else {
        return super.decreaseApproval(_spender, _subtractedValue);
    }
}

// deprecate current contract in favour of a new one
```

//SlowMist// This is an upgradeable contract, and a security audit of the upgradedAddress

contract is required before the contract is upgraded

```
    function deprecate(address _upgradedAddress) public onlyOwner {
        deprecated = true;
        upgradedAddress = _upgradedAddress;
        emit Deprecate(_upgradedAddress);
    }

    // Called when contract is deprecated
    event Deprecate(address newAddress);
}

/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
```

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

```
library SafeMath {

    /**
    * @dev Multiplies two numbers, throws on overflow.
    */
    function mul(uint256 a, uint256 b) internal pure returns (uint256 c) {
        // Gas optimization: this is cheaper than asserting '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;
        }

        c = a * b;
        assert(c / a == b); //SlowMist// It is recommended to replace "assert" with "require" to
```

optimize Gas

```
        return c;
    }

    /**
    * @dev Integer division of two numbers, truncating the quotient.
    */
```

```
    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 a / b;
    }

    /**
    * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than minuend).
    */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        assert(b <= a); //SlowMist// It is recommended to replace "assert" with "require" to optimize Gas
        return a - b;
    }

    /**
    * @dev Adds two numbers, throws on overflow.
    */
    function add(uint256 a, uint256 b) internal pure returns (uint256 c) {
        c = a + b;
        assert(c >= a); //SlowMist// It is recommended to replace "assert" with "require" to optimize Gas
        return c;
    }
}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account