Smart Contract Security Audit Report

The SlowMist Security Team received the BioPassport Coin team's application for smart contract security audit of the BioPassport Coin on Feb. 18, 2021. The following are the details and results of this smart contract security audit:

Token name :

BioPassport Coin

The project address :

https://github.com/biopassport/contract

commit: 688503a82e30a722929e3b2cc908cf1e4ec7fdbf

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 : 0X002102200003

Audit Date : Feb. 20, 2021

Audit Team : SlowMist Security Team

Summary: This is a token contract that does not contain the tokenVault section. The total amount of contract tokens can be changed, and the minter role can mint tokens unlimitedly through the mint function, but there is an upper limit to the total amount of tokens. 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 admin role can freeze the account balance of any user through the setReserve function.

2. The owner can arbitrarily set the address of the vault through the setVault function.

3. The vault role can arbitrarily set the address of the owner through the setOwner function.

4. The owner and vault role can arbitrarily set the address of the admin through the setAdmin function.

The source code:

BiotCoinSafeMath.sol:

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

pragma solidity >=0.5.0;

/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library BiotCoinSafeMath {
    function sub(uint8 a, uint8 b) internal pure returns (uint8) {

        assert(b <= a); //SlowMist// It is recommended to replace "assert" with "require" to optimize Gas

        return a - b;
    }

}
```

BiotCoin.sol:

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

pragma solidity >=0.5.0;

import "@openzeppelin/contracts/token/ERC20/ERC20Capped.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol";

/**
 * @title BiotCoin contract
 */
contract BiotCoin is ERC20Capped, ERC20Detailed {
    uint noOfTokens = 8_800_000_000; // 8.8B

    // Address of biot coin vault
    // The vault will have all the biot coin issued.
    address internal vault;

    // Address of biot coin owner
    // The owner can change admin and vault address.
    address internal owner;
```

```
// Address of biot coin admin
// The admin can change reserve. The reserve is the amount of token
// assigned to some address but not permitted to use.
address internal admin;

event OwnerChanged(address indexed previousOwner, address indexed newOwner);
event VaultChanged(address indexed previousVault, address indexed newVault);
event AdminChanged(address indexed previousAdmin, address indexed newAdmin);
event ReserveChanged(address indexed _address, uint amount);

/**
 * @dev reserved number of tokens per each address
 *
 * To limit token transaction for some period by the admin,
 * each address' balance cannot become lower than this amount
 *
 */
mapping(address => uint) public reserves;

/**
    * @dev modifier to limit access to the owner only
    */
modifier onlyOwner() {
      require(msg.sender == owner);
      _;
}

/**
    * @dev limit access to the vault only
    */
modifier onlyVault() {
      require(msg.sender == vault);
      _;
}

/**
    * @dev limit access to the admin only
    */
modifier onlyAdmin() {
      require(msg.sender == admin);
      _;
```

```
    }

    /**
     * @dev limit access to owner or vault
     */
    modifier onlyOwnerOrVault() {
        require(msg.sender == owner || msg.sender == vault);
        _;
    }

    /**
     * @dev initialize QRC20(ERC20)
     *
     * all token will deposit into the vault
     * later, the vault, owner will be multi sign contract to protect privileged operations
     *
     * @param _symbol token symbol
     * @param _name   token name
     * @param _owner  owner address
     * @param _admin  admin address
     * @param _vault  vault address
     *
     * Cap the mintable amount to 77.7B(77,700,000,000)
     *
     */
    constructor (string memory _symbol, string memory _name, address _owner,
        address _admin, address _vault) ERC20Detailed(_name, _symbol, 9)
ERC20Capped(77_700_000_000_000_000_000)
    public {
        require(bytes(_symbol).length > 0);
        require(bytes(_name).length > 0);

        owner = _owner;
        admin = _admin;
        vault = _vault;

        // mint coins to the vault
        _mint(vault, noOfTokens * (10 ** uint(decimals())));
    }

    /**
     * @dev change the amount of reserved token
```

```
     *
     * @param _address the target address whose token will be frozen for future use
     * @param _reserve   the amount of reserved token
     *
     */
```

//SlowMist// The admin role can freeze the account balance of any user through the setReserve function

```
    function setReserve(address _address, uint _reserve) public onlyAdmin {
        require(_reserve <= totalSupply());
        require(_address != address(0));

        reserves[_address] = _reserve;
        emit ReserveChanged(_address, _reserve);
    }

    /**
     * @dev transfer token from sender to other
     *            the result balance should be greater than or equal to the reserved token amount
     */
    function transfer(address _to, uint256 _value) public returns (bool) {
        // check the reserve
        require(balanceOf(msg.sender).sub(_value) >= reserveOf(msg.sender));
        return super.transfer(_to, _value);
    }

    /**
     * @dev change vault address
     *
     * @param _newVault new vault address
     */
```

//SlowMist// The owner can arbitrarily set the address of the vault through the setVault function

```
    function setVault(address _newVault) public onlyOwner {
        require(_newVault != address(0));
        require(_newVault != vault);

        address _oldVault = vault;

        // change vault address
        vault = _newVault;
```

```
        emit VaultChanged(_oldVault, _newVault);
    }

    /**
     * @dev change owner address
     * @param _newOwner new owner address
     */
```

//SlowMist// The vault role can arbitrarily set the address of the owner through the setOwner function

```
    function setOwner(address _newOwner) public onlyVault {
        require(_newOwner != address(0));
        require(_newOwner != owner);

        emit OwnerChanged(owner, _newOwner);
        owner = _newOwner;
    }

    /**
     * @dev change admin address
     * @param _newAdmin new admin address
     */
```

//SlowMist// The owner and vault role can arbitrarily set the address of the admin through the setAdmin function

```
    function setAdmin(address _newAdmin) public onlyOwnerOrVault {
        require(_newAdmin != address(0));
        require(_newAdmin != admin);

        emit AdminChanged(admin, _newAdmin);
        admin = _newAdmin;
    }

    /**
     * @dev Transfer tokens from one address to another
     *
     * The _from's biot balance should be larger than the reserved amount(reserves[_from]) plus _value.
     *
     *    NOTE: no one can tranfer from vault
     *
```

```
     */
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool) {
        require(_from != vault);
        require(_value <= balanceOf(_from).sub(reserves[_from]));
        return super.transferFrom(_from, _to, _value);
    }

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

    function getVault() public view returns (address) {
        return vault;
    }

    function getAdmin() public view returns (address) {
        return admin;
    }

    function getOneBiotCoin() public view returns (uint) {
        return (10 ** uint(decimals()));
    }

    /**
     * @dev get the amount of reserved token
     */
    function reserveOf(address _address) public view returns (uint _reserve) {
        return reserves[_address];
    }

    /**
     * @dev get the amount reserved token of the sender
     */
    function reserve() public view returns (uint _reserve) {
        return reserves[msg.sender];
    }
}
```

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist