SMART CONTRACT AUDIT REPORT

For

Big Coin (BIG)

Prepared By: SFI Team

Prepared for: Big Coin team

Prepared on: 6/12/2021

Table of Content

- # Disclaimer

  This is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against the team on the basis of what it says or doesn't say, or how team produced it, and it is important for you to conduct your own independent investigations before making any decisions. team go into more detail on this in the below disclaimer below – please make sure to read it in full.

  By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Saferico and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (SaferICO) owe no duty of care towards you or any other person, nor does Saferico make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Saferico hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Saferico hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Saferico, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

- # Overview of the audit

The project has 1 file. It contains approx 48 lines of Solidity code. Most of the functions and state variables are well commented on using the Nat spec documentation, but that does not create any vulnerability.

- # Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices automatically.

1. Unit tests passing.

2. Compilator warnings;

3. Race Conditions. Reentrancy. Cross-function Race Conditions. Pitfalls in Race Condition solutions;

4. Possible delays in data delivery;

5. Transaction-Ordering Dependence (front running);

6. Timestamp Dependence;

7. Integer Overflow and Underflow;

8. DoS with (unexpected) Revert;

9. DoS with Block Gas Limit

10. Call Depth Attack. Not relevant in modern ethereum network

11. Methods execution permissions;

12. Oracles calls;

13. Economy model. It’s important to forecast scenarios when a user is provided with additional economic motivation or faced with limitations. If application logic is based on incorrect economy model, the application will not function correctly and participants will incur financial losses. This type of issue is most often found in bonus rewards systems.
14. The impact of the exchange rate on the logic;
15. Private user data leaks.

- # Good things in smart contract

  Compiler version is static: -
  => In this file, you have put “pragma solidity 0.8.4;” which is a good way to define the compiler version.

```
pragma solidity 0.8.4;
```

Solidity < 0.8
Integers in Solidity overflow / underflow without any errors so no need for SafeMath in this contract

- **Most Write functions in the contract: -**

Big Coin is using approve function This function use to give allowance to send tokens

```
function approve(address spender, uint value) public returns(bool) {
        allowance[msg.sender][spender] = value;
        emit Approval(msg.sender, spender, value);
        return true;
    }
```

Big Coin is using transfer function This function use to send tokens from address to an other one

```
function transfer(address to, uint value) public returns(bool) {
        require(balanceOf(msg.sender) >= value, 'balance too low');
        balances[to] += value;
        balances[msg.sender] -= value;
        emit Transfer(msg.sender, to, value);
        return true;
    }
```

Big Coin is using transferFrom function This function use to send tokens from spender address to receiver address (which had given approve first)

```
function transferFrom(address from, address to, uint value) public returns(bool) {
        require(balanceOf(from) >= value, 'balance too low');
        require(allowance[from][msg.sender] >= value, 'allowance too low');
        balances[to] += value;
        balances[from] -= value;
        emit Transfer(from, to, value);
        return true;
    }
```

- **Most Read functions in the contract: -**
  - Big coin is using mapping and string to defined the token information (name, symbol, dec, total supply and check allowance for an address)

```
 mapping(address => uint) public balances;
mapping(address => mapping(address => uint)) public allowance;
uint public totalSupply = 21000000 * 10 ** 18;
string public name = "BIG COIN";
string public symbol = 'BIG';
uint public decimals = 18;
```

Big Coin is using balanceOf function to know any address balance.

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

- ## Critical vulnerabilities found in the contract

There not Critical severity vulnerabilities found

- ## High vulnerabilities found in the contract

There not High severity vulnerabilities found

- ## Medium vulnerabilities found in the contract

There not Medium severity vulnerabilities found

- ## Low vulnerabilities found in the contract

There not Low severity vulnerabilities found

- ## V. Low vulnerabilities found in the contract

There not V. Low severity vulnerabilities found

- ## Notes

#Gas Costs:

In detail the gas requirement of a function is higher than the block gas limit, it cannot be executed Please avoid loops in your functions or actions that modify large areas of storage
(This includes clearing or copying arrays in storage)

```
string public name = "BIG COIN"; string public symbol = 'BIG';
function transferFrom(address from, address to, uint value) public returns(bool) {
        require(balanceOf(from) >= value, 'balance too low');
        require(allowance[from][msg.sender] >= value, 'allowance too low');
        balances[to] += value;
        balances[from] -= value; emit Transfer(from, to, value); return true;}
```

#No license found:

SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code.

Testing proves:

1- Check for security

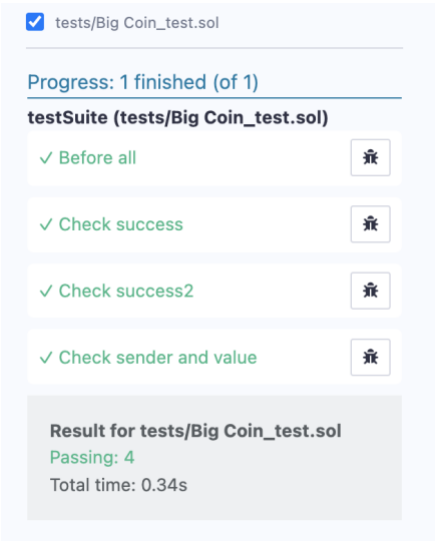
16f7a02fb9ccc355b7030c442716a00b1d2eda377a67210339e664e66bd1e675

File: Big Coin.sol | Language: solidity | Size: 1570 bytes | Date: 2021-12-06T11:50:31.115Z

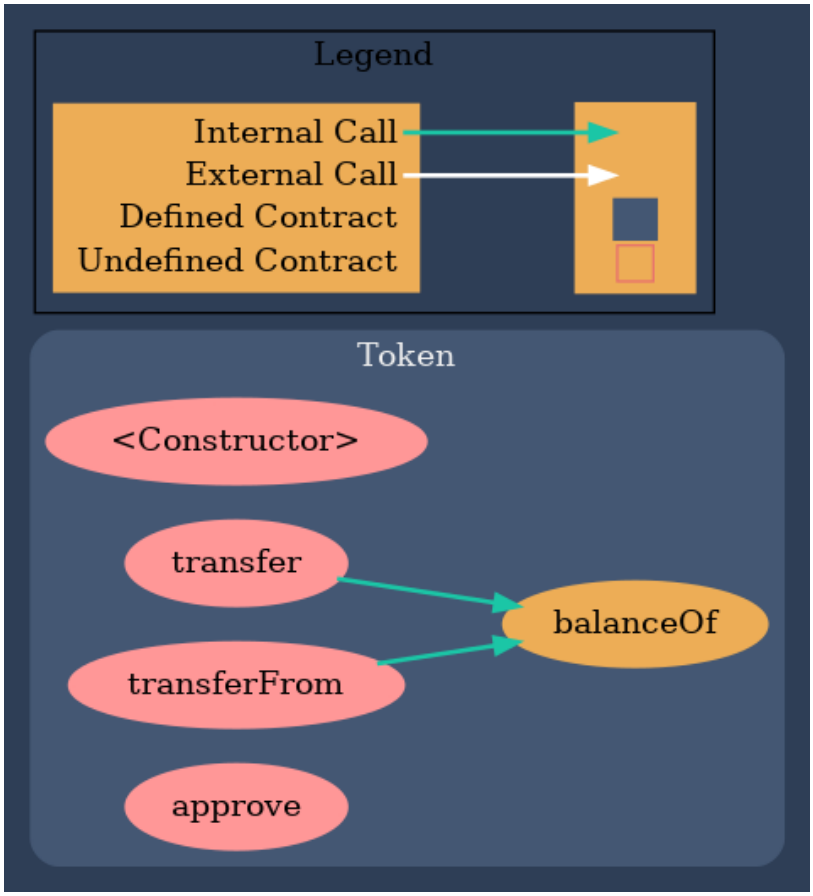
Critical	High	Medium	Low	Note
0	0	0	0	2

2- SOLIDITY STATIC ANALYSIS

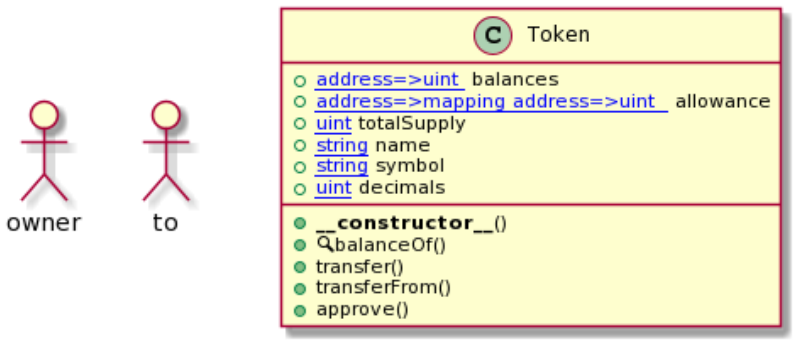
3- SOLIDITY UNIT TESTING



4- Call graph



Unified Modeling Language (UML)



Function Signature

```
70a08231  =>  balanceOf(address)
a9059cbb  =>  transfer(address,uint256)
23b872dd  =>  transferFrom(address,address,uint256)
095ea7b3  =>  approve(address,uint256)
```

- # Automatic general report

Files Description Table

File Name	SHA-1 Hash
/Users/macbook/Desktop/smart contracts/Big Coin.sol	7df433c337fa437ba1022be1079feff1a63e511d

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
Token	Implementation			
└	<Constructor>	Public ❗	🛑	NO❗
└	balanceOf	Public ❗		NO❗
└	transfer	Public ❗	🛑	NO❗
└	transferFrom	Public ❗	🛑	NO❗
└	approve	Public ❗	🛑	NO❗

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

- # Summary of the Audit

According to automatically test, the customer`s solidity smart contract is **Secured**.

The general overview is presented in the Project Information section and all issues found are located in the audit overview section.

The test found 0 critical, 0 high, 0 medium, 0 low, 0 Very low issues, and 2 notes.