SMART CONTRACT AUDIT REPORT
FOR
BFICOIN

(Order#FO7185670D6C5)

Prepared By: Carlos

Prepared For: profittclub240

Prepared on: 05/04/2021

OFFICE ADD: 777 BROCKTON AVENUE, LOS ANGELES, USA, 02351

AUDIT DETAILS:

- **Project Name:** BFI-COIN
- **Website Code:** https://bficoin.io/
- **Explorer:** **Bscscan**
- **Address:** 0xF0a0913BA2b173c1fbCa1c1b2FCFC0E3678f66a9
- **Languages**: Solidity (Smart contract),

Table of Content

• Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

• Overview of the audit

The project has 1 file. It contains approx 163 lines of Solidity code. All the functions and state variables are well commented using the natspec 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.

- ## Over and under flows

An overflow happens when the limit of the type variable uint256, 2 ** 256, is exceeded. What happens is that the value resets to zero instead of incrementing more. On the other hand, an underflow happens when you try to subtract 0 minus a number bigger than 0. For example, if you subtract 0 - 1 the result will be = 2 ** 256 instead of -1. This is quite dangerous.

This contract **does** check for overflows and underflows by using OpenZeppelin's SafeMath to mitigate this attack, but all the functions have strong validations, which prevented this attack.

- ## Short address attack

If the token contract has enough amount of tokens and the buy function doesn't check the length of the address of the sender, the Tron's virtual machine will just add zeros to the transaction until the address is complete.

Although this contract **is not vulnerable** to this attack, but there are some point where users can mess themselves due to this (Please see below). It is highly recommended to call functions after checking validity of the address.

- ## Visibility & Delegate call

It is also known as, The Parity Hack, which occurs while misuse of
Delegate call.

No such issues found in this smart contract and visibility also properly addressed. There are some places where there is no visibility defined. Smart Contract will assume "Public" visibility if there is no visibility defined. It is good practice to explicitly define the visibility, but again, the contract is not prone to any vulnerability due to this in this case.

- ## Reentrancy / TheDAO hack

Reentrancy occurs in this case: any interaction from a contract (A) with another contract (B) and any transfer of Tron hands over control to that contract (B).

This makes it possible for B to call back into A before this interaction is completed.

Use of "require" function in this smart contract mitigated this vulnerability.

- ## Forcing Tron to a contract

While implementing "selfdestruct" in smart contract, it sends all the tron to the target address. Now, if the target address is a contract address, then the fallback function of target contract does not get called. And thus Hacker can bypass the "Required" conditions. Here, the Smart Contract's balance has never been used as guard, which mitigated this vulnerability.

• Good things in smart contract

- ### SafeMath library:-
  - You are using SafeMath library it is a good thing. This protects you from underflow and overflow attacks.

```
library SafeMath {

  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a * b;
    assert(a == 0 || c / a == b);
```

- ### Good required condition in functions:-
  - Here you are checking that msg.sender address is not frozen, balance of msg.sender is bigger than _amount and _amount is bigger than 0.

```
    function transfer(address _to, uint256 _amount) public   returns (bool success)
        require(!frozen[msg.sender],'account is freez');
        require (balances[msg.sender]>=_amount&&_amount>0&&balances[_to]+_amount>ba
        balances[msg.sender]=balances[msg.sender].sub(_amount);
```

  - Here you are checking that _from address is not frozen, balance of _from address is bigger than _amount, msg.sender has enough allowance from _from address to transfer _amount, and _amount should be bigger than 0.

```
    function transferFrom(address _from,address _to,uint256 _amount) public   retu
        require(!frozen[_from],"From address is fronzen");
        require (balances[_from]>=_amount&&allowed[_from][msg.sender]>=_amount&&_a
        balances[_from]=balances[_from].sub(_amount);
```

- Here you are checking that address of msg.sender is not frozen, msg.sender balance should be bigger than _value.

```
97  function burn(uint256 _value) public returns (bool success) {
98      require(!frozen[msg.sender],"Account address is fronzen");
99        require(balances[msg.sender] >= _value);   // Check if the sender has enou
100       balances[msg.sender] =balances[msg.sender].sub( value);           // Subt
```

- Here you are checking that account address value is proper and valid address.

```
123  function _mint(address _account, uint256 _amount) external onlyOwner  {
124      require(_account != address(0), "ERC20: mint to the zero address");
125      balances[_account] = balances[_account].add(_amount);
126  }
```

•Critical vulnerabilities found in the contract

=> No Critial vulnerabilities found

• Medium vulnerabilities found in the contract

=> No Medium vulnerabilities found

• Low severity vulnerabilities found

○ 7.1: Short address attack:-

=> This is not a big issue in solidity, because of a new release of the solidity version. But it is good practice to check for the short address.

=> After updating the version of solidity it's not mandatory.

=> In some functions you are not checking the value of Address parameter here I am showing necessary functions.

Function: - transfer (_to')

```
55  function transfer(address _to, uint256 _amount) public   returns (bool success)
56      require(!frozen[msg.sender],'account is freez');
57      require (balances[msg.sender]>=_amount&& _amount>0&&balances[_to]+_amount>ba
```

- It's necessary to check the address value of "_to". Because here you are passing whatever variable comes in "_to" address from outside.

Function: - transferFrom (‘_from’, ‘_to’)

```
63    function transferFrom(address _from,address _to,uint256 _amount) public   retur
64        require(!frozen[_from],"From address is fronzen");
65        require (balances[_from]>=_amount&&allowed[_from][msg.sender]>=_amount&&_am
66        balances[_from]=balances[_from].sub(_amount);
```

- It's necessary to check the addresses value of "_from", "_to". Because here you are passing whatever variable comes in "_from", "_to" addresses from outside.

Function: - approve (‘_spender’)

```
73    function approve(address _spender, uint256 _amount) public   returns (bool succ
74        allowed[msg.sender][_spender]=_amount;
75        emit Approval(msg.sender, _spender, _amount);
```

- It's necessary to check the address value of "_spender". Because here you are passing whatever variable comes in "_spender" address from outside.

Function: - FreezeAcc, UnfreezeAcc (‘_target’)

```
83
84    function FreezeAcc(address target, bool freeze) onlyOwner public whenNotFrozen(t
85      freeze = true;
86      frozen[target]=freeze;
87      emit Freeze(target, true);

90
91    function UnfreezeAcc(address target, bool freeze) onlyOwner public whenFrozen(ta
92      freeze = false;
93      frozen[target]=freeze;
```

- It's necessary to check the address value of "_target". Because here you are passing whatever variable comes in "_target" address from outside.

Function: - changeOwner (‘_newOwner’)

```
128
129   function changeOwner(address payable _newOwner) public onlyOwner returns(bool)
130       balances[_newOwner] = balances[owner];
131       balances[owner] = 0;
```

- It's necessary to check the address value of "_newOwner". Because here you are passing whatever variable comes in "_newOwner" address from outside.

- ### 7.2: Compiler version is not fixed:-

  => In this file you have put “pragma solidity ^0.5.9;” which is not a good way to define compiler version.

  => Solidity source files indicate the versions of the compiler they can be compiled with. Pragma solidity >=^0.5.9; // bad: compiles 0.5.9 and above pragma solidity 0.5.9; //good: compiles 0.5.9 only

  => If you put(>=) symbol then you are able to get compiler version 0.5.9 and above. But if you don’t use(^/>=) symbol then you are able to use only 0.5.9 version. And if there are some changes come in the compiler and you use the old version then some issues may come at deploy time.

  => Use latest version of solidity.

• Summary of the Audit

Overall the code is well and performs well. There is no back

door to steal fund.

Please try to check the address and value of token externally before sending to the solidity code.

Our final recommendation would be to pay more attention to the visibility of the functions , hardcoded address and mapping since it’s quite important to define who’s supposed to executed the functions and to follow best practices regarding the use of assert, require etc. (which you are doing ;)).

Comments:

- Use case of smart contract is very well designed and Implemented. Overall, the code is clearly written, and demonstrates effective use of abstraction, separation of concerns, and modularity.
- Validations are done properly.

OFFICE ADD:

777 BROCKTON AVENUE, LOS ANGELES, USA, 02351