SMART CONTRACT AUDIT REPORT

For

DFEstaking

(Order #FO5B6F72C7C1)

Prepared By: Kishan Patel **Prepared For**: KING X

Prepared on: 06/12/2020

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 221 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.

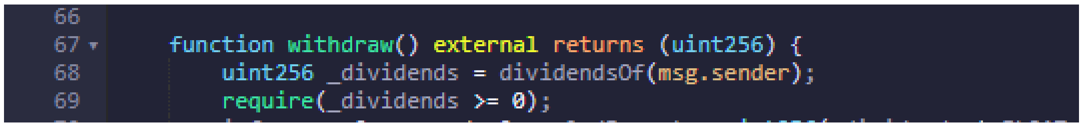
- ### Forcing Tron to a contract

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

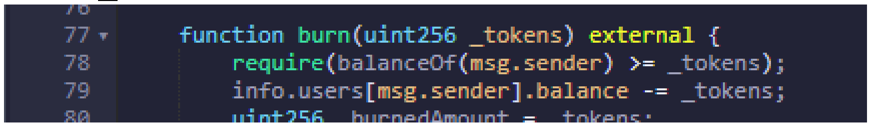
- ### Good required condition in functions:-

  - Here you are checking that withdraw function work when _dividends is more than 0.



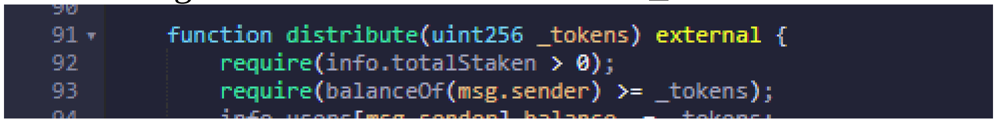
```
function withdraw() external returns (uint256) {
    uint256 _dividends = dividendsOf(msg.sender);
    require(_dividends >= 0);
```

  - Here you are checking that burn function work when msg.sender has more balance than _tokens.



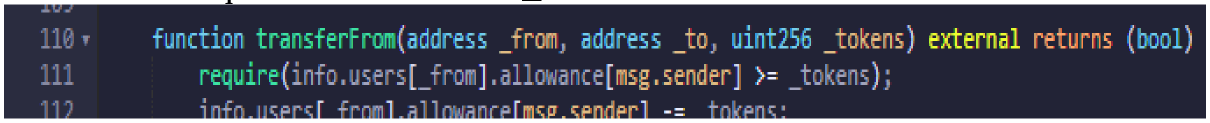
```
function burn(uint256 _tokens) external {
    require(balanceOf(msg.sender) >= _tokens);
    info.users[msg.sender].balance -= _tokens;
```

  - Here you are checking that distribute function work when totalStaken is bigger than 0 and msg.sender has more balance than _tokens.



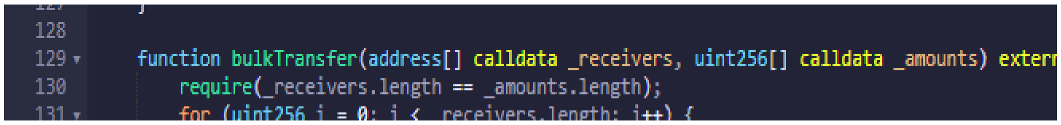
```
function distribute(uint256 _tokens) external {
    require(info.totalStaken > 0);
    require(balanceOf(msg.sender) >= _tokens);
```

  - Here you are checking that transferFrom function work when msg.sender has more or equal allowance than _tokens.



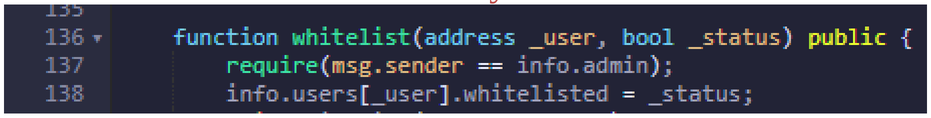
```
function transferFrom(address _from, address _to, uint256 _tokens) external returns (bool) {
    require(info.users[_from].allowance[msg.sender] >= _tokens);
    info.users[_from].allowance[msg.sender] -= _tokens;
```

  - Here you are checking that bulkTransfer function work when number of address and number of amounts are same.



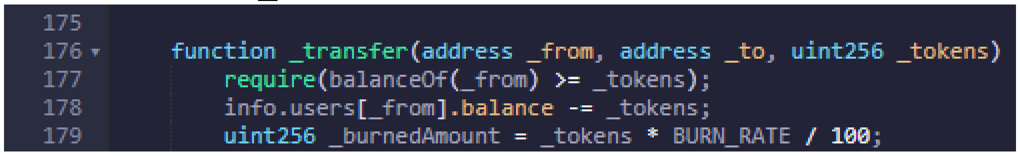
```
    }

    function bulkTransfer(address[] calldata _receivers, uint256[] calldata _amounts) extern
        require(_receivers.length == _amounts.length);
        for (uint256 i = 0; i < _receivers.length; i++) {
```

- Here you are checking that whitelist function work when msg.sender is admin of contract. You can use Ownable library for this.



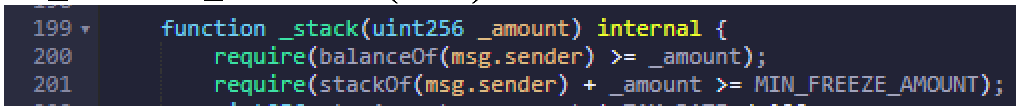
```
    function whitelist(address _user, bool _status) public {
        require(msg.sender == info.admin);
        info.users[_user].whitelisted = _status;
```

- Here you are checking that _transfer function work when _form address has balance more than _tokens.



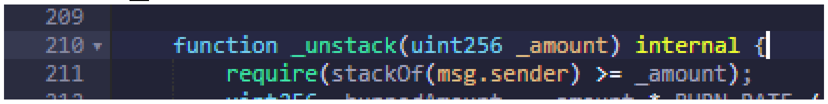
```

    function _transfer(address _from, address _to, uint256 _tokens)
        require(balanceOf(_from) >= _tokens);
        info.users[_from].balance -= _tokens;
        uint256 _burnedAmount = _tokens * BURN_RATE / 100;
```

- Here you are checking that _stack function work when msg.sender has more balance than _amount and msg.sender has more staked than MIN_FREEZE_AMOUNT(1e18).



```
    function _stack(uint256 _amount) internal {
        require(balanceOf(msg.sender) >= _amount);
        require(stackOf(msg.sender) + _amount >= MIN_FREEZE_AMOUNT);
```

- Here you are checking that _unstack function work when msg.sender has staked more value than _amount.



```

    function _unstack(uint256 _amount) internal {
        require(stackOf(msg.sender) >= _amount);
```

• Critical vulnerabilities found in the contract

=> No Critial vulnerabilities found

• Medium vulnerabilities found in the contract

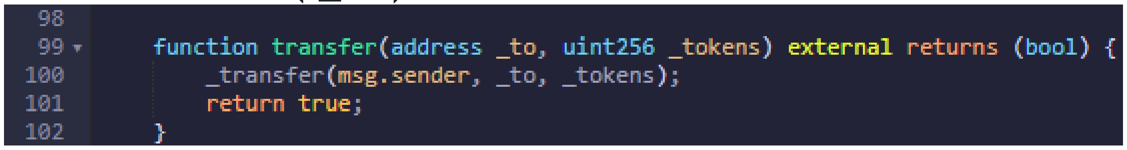
=> No Medium vulnerabilities found

- ## Low severity vulnerabilities found

○ 7.1: Short address attack:-

=> This is not big issue in solidity, because now a days is increased In the new solidity version. But it is good practice to Check for the short address.
=> After updating the version of solidity it's not mandatory.
=> In all functions you are not checking the value of Address parameter here I am showing only some functions.

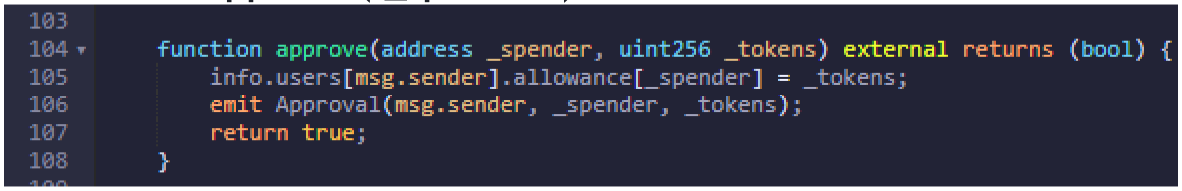
Function:- transfer ('_to')



```
    function transfer(address _to, uint256 _tokens) external returns (bool) {
        _transfer(msg.sender, _to, _tokens);
        return true;
    }
```

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

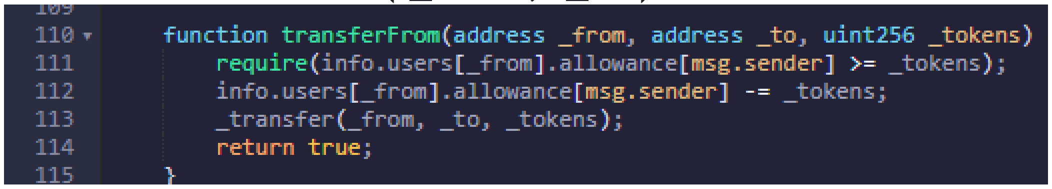
Function: - approve ('_spender')



```
    function approve(address _spender, uint256 _tokens) external returns (bool) {
        info.users[msg.sender].allowance[_spender] = _tokens;
        emit Approval(msg.sender, _spender, _tokens);
        return true;
    }
```

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

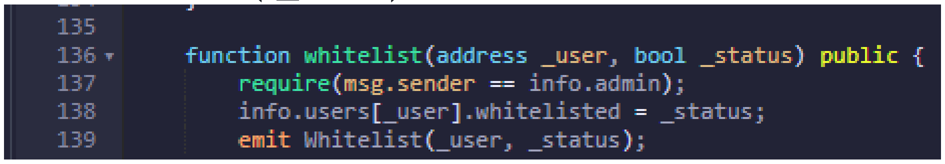
Function: - transferFrom ('_from', '_to')



```
    function transferFrom(address _from, address _to, uint256 _tokens)
        require(info.users[_from].allowance[msg.sender] >= _tokens);
        info.users[_from].allowance[msg.sender] -= _tokens;
        _transfer(_from, _to, _tokens);
        return true;
    }
```

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

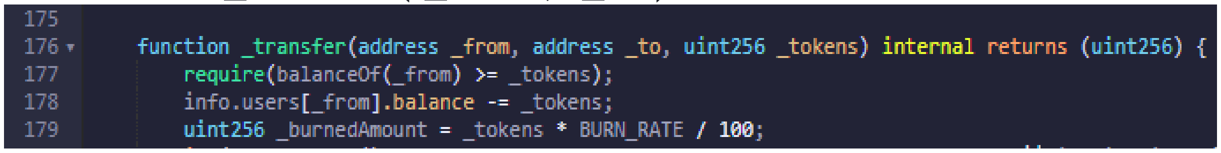
Function: - whitelist ('_user')



```
    function whitelist(address _user, bool _status) public {
        require(msg.sender == info.admin);
        info.users[_user].whitelisted = _status;
        emit Whitelist(_user, _status);
```

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

Function: - _transfer ('_from', '_to')



```
    function _transfer(address _from, address _to, uint256 _tokens) internal returns (uint256) {
        require(balanceOf(_from) >= _tokens);
        info.users[_from].balance -= _tokens;
        uint256 _burnedAmount = _tokens * BURN_RATE / 100;
```

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

○ 7.2: Compiler version is not fixed:-

=> In this file you have put "pragma solidity ^0.5.13;" 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 >=0.5.13; // bad: compiles 0.5.13 and above pragma solidity 0.5.13; //good: compiles 0.5.13 only

=> If you put(>=) symbol then you are able to get compiler version 0.5.13 and above. But if you don't use(^/>=) symbol then you are able to use only 0.5.13 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.

○ 7.3: SafeMath library Not used:-

=> I found that you have not used safemath library in your contract.
=> I would recommend that you can use safeMath.
=> This library is very necessary for your smart contract.
=> If will protect your contract from underflow and overflow attack.

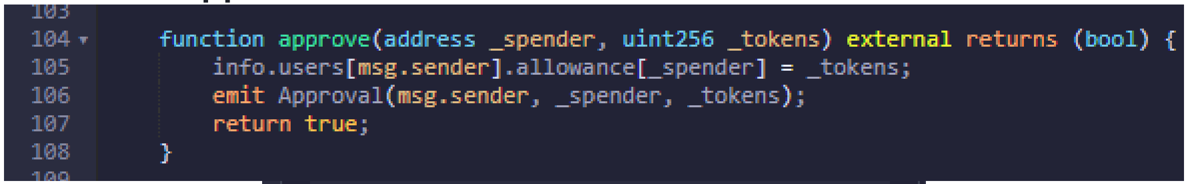
○ 7.4: Approve given more allowance:-

=> I have found that in your approve function user can give more allowance to user beyond their balance..

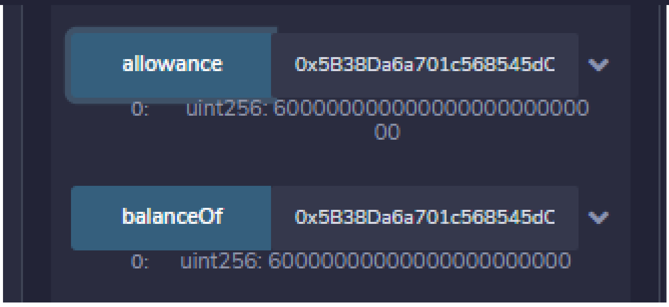
=> It is necessary to check that user can give allowance less or equal to their amount.

=> There is no validation about user balance.

Function: - approve



```
    function approve(address _spender, uint256 _tokens) external returns (bool) {
        info.users[msg.sender].allowance[_spender] = _tokens;
        emit Approval(msg.sender, _spender, _tokens);
        return true;
    }
```



```
allowance    0x5B38Da6a701c568545dC
0:    uint256: 6000000000000000000000000
                00

balanceOf    0x5B38Da6a701c568545dC
0:    uint256: 600000000000000000000000
```

- Here you can check you have more allowance than balance.

○ 7.5: Contract has Mint function? :-

=> As owner ask for to check that this contract has mint function or not.

=> So, I did not find mint function in this contract.

=> I only found burn function,

=> If you requested this functionality from developer then this contract is missing that functioanlity

• Summary of the Audit

Overall the code is well and performs well.

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 ;)).

- **Note:** Please focus on a version, use safeMath library,
  check Approve function and check addresses.