SMART CONTRACT AUDIT REPORT

For

Brainaut

Prepared By: Kishan Patel

Prepared For: SOUF

Prepared on: 17/03/2021

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 730 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.

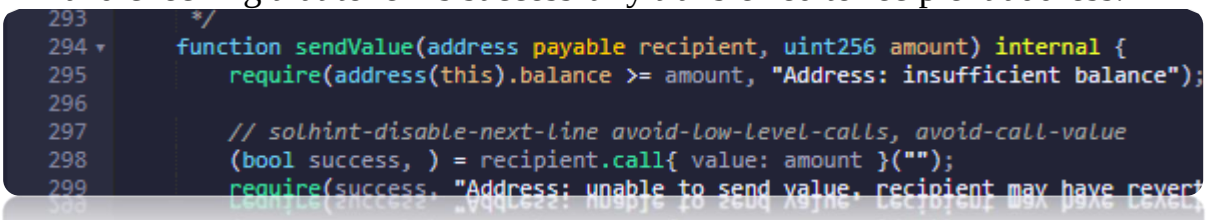
Use of "require" function in this smart contract mitigated this vulnerability.

- ## Forcing Tron to a contract

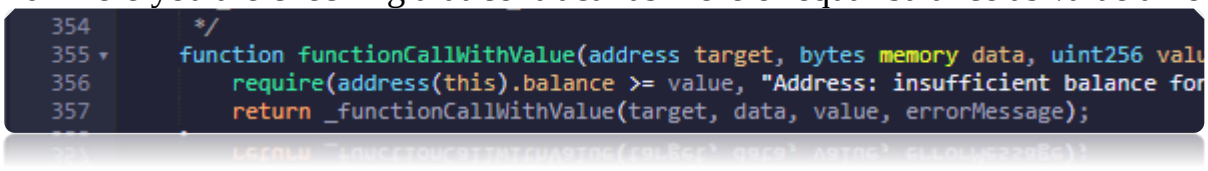
While implementing "selfdestruct" in smart contract, it sends all the tron to the target address. Now, if the target address is a contract address, then the fallback function of target contract does not get called. And thus Hacker can bypass the "Required" conditions. Here, the Smart Contract's balance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

• Good things in smart contract

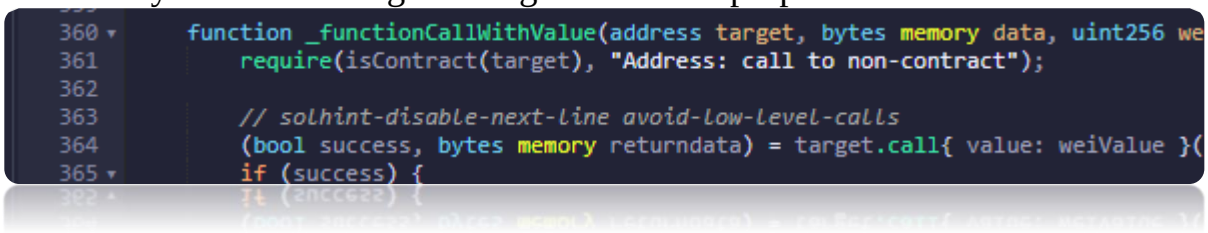
- ### Good required condition in functions:-
  - Here you are checking that balance of contract is bigger or equal to the amount and checking that token is successfully transferred to recipient address.



  - Here you are checking that contract has more or equal balance as value amount.



  - Here you are checking that target address is proper contract address.



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

```
440      */
441     function transferOwnership(address newOwner) public virtual onlyOwner {
442         require(newOwner != address(0), "Ownable: new owner is the zero address")
443         emit OwnershipTransferred(_owner, newOwner);
444         _owner = newOwner;
445     }
```

- Here you are checking that deliver address should not be called by the Excluded address.

```
544
545     function deliver(uint256 tAmount) public {
546         address sender = _msgSender();
547         require(!_isExcluded[sender], "Excluded addresses cannot call this functic
```

- Here you are checking that tAmount value should be less than or equal to the _tTotal amount (Total token value).

```
553
554     function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public
555         require(tAmount <= _tTotal, "Amount must be less than supply");
556         if (!deductTransferFee) {
557             (uint256 rAmount,,,,,) = _getValues(tAmount);
```

- Here you are checking that rAmount value should be less than or equal to the _rTotal amount (Total reflections value).

```
565     function tokenFromReflection(uint256 rAmount) public view returns(uint256) {
566         require(rAmount <= _rTotal, "Amount must be less than total reflections");
567         uint256 currentRate =  _getRate();
568         return rAmount.div(currentRate);
569     }
```

- Here you are checking that account address not already excluded from reward.

```
570
571     function excludeAccount(address account) external onlyOwner() {
572         require(account != 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, 'We can not
573         require(!_isExcluded[account], "Account is already excluded");
574         if(_rOwned[account] > 0) {
```

- Here you are checking that account address not already included for reward.

```
580
581     function includeAccount(address account) external onlyOwner() {
582         require(_isExcluded[account], "Account is already excluded");
583         for (uint256 i = 0; i < _excluded.length; i++) {
584             if (_excluded[i] == account) {
```

- Here you are checking that owner and spender address values are proper addresses.

```
593
594 ▾    function _approve(address owner, address spender, uint256 amount) private {
595          require(owner != address(0), "BEP20: approve from the zero address");
596          require(spender != address(0), "BEP20: approve to the zero address");
597
```

- Here you are checking that addresses values of sepnder and recipient are proper, amount should be bigger than 0.

```
601
602 ▾    function _transfer(address sender, address recipient, uint256 amount) private
603          require(sender != address(0), "BEP20: transfer from the zero address");
604          require(recipient != address(0), "BEP20: transfer to the zero address");
605          require(amount > 0, "Transfer amount must be greater than zero");
606
```

- **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:- isContract ('account')

```
266
267     function isContract(address account) internal view returns (bool) {
268         // According to EIP-1052, 0x0 is the value returned for not-yet created ac
269         // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470
270         // for accounts without code, i.e. `keccak256('')`
271         bytes32 codehash;
```

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

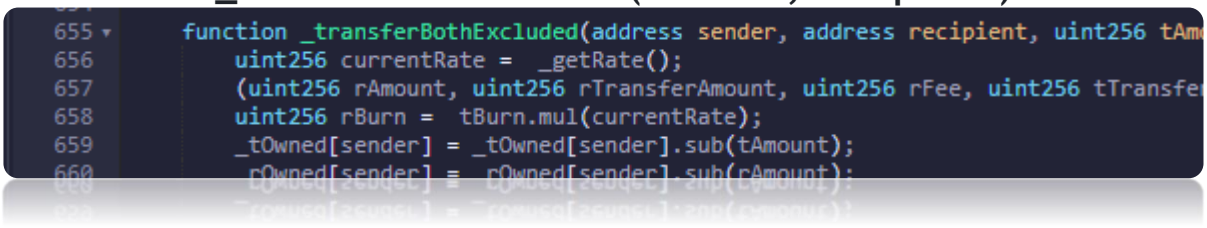
Function:- excludeFromReward, includeInReward ('account')

```
571     function excludeAccount(address account) external onlyOwner() {
572         require(account != 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, 'We can not
573         require(!_isExcluded[account], "Account is already excluded");
574         if(_rOwned[account] > 0) {
575             _tOwned[account] = tokenFromReflection(_rOwned[account]);
```

```
581     function includeAccount(address account) external onlyOwner() {
582         require(_isExcluded[account], "Account is already excluded");
583         for (uint256 i = 0; i < _excluded.length; i++) {
584             if (_excluded[i] == account) {
585                 _excluded[i] = _excluded[_excluded.length - 1];
586                 _tOwned[account] = 0;
```

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

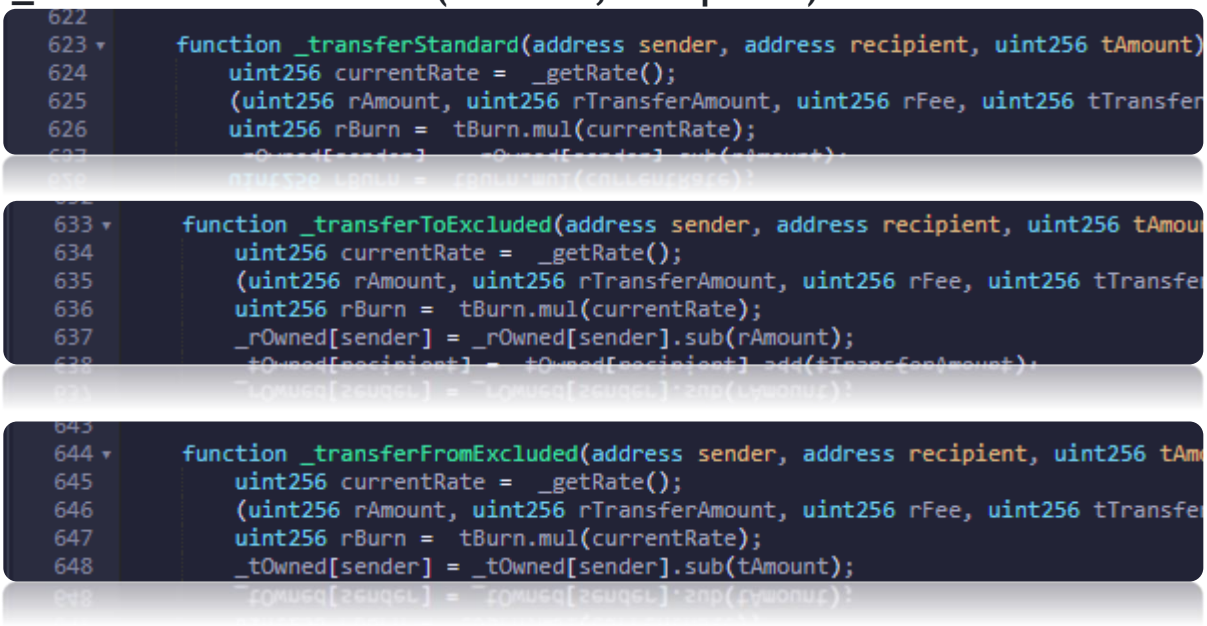
Function: - _transferBothExcluded (‘sender’, ‘recipient’)



```
function _transferBothExcluded(address sender, address recipient, uint256 tAm
    uint256 currentRate =  _getRate();
    (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransfe
    uint256 rBurn =  tBurn.mul(currentRate);
    _tOwned[sender] = _tOwned[sender].sub(tAmount);
    _rOwned[sender] = _rOwned[sender].sub(rAmount);
```

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

Function: - _transferStandard, _transferToExcluded, _transferFromExcluded (‘sender’, ‘recipient’)



```
function _transferStandard(address sender, address recipient, uint256 tAmount)
    uint256 currentRate =  _getRate();
    (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransfer
    uint256 rBurn =  tBurn.mul(currentRate);
```

```
function _transferToExcluded(address sender, address recipient, uint256 tAmou
    uint256 currentRate =  _getRate();
    (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransfe
    uint256 rBurn =  tBurn.mul(currentRate);
    _rOwned[sender] = _rOwned[sender].sub(rAmount);
```

```
function _transferFromExcluded(address sender, address recipient, uint256 tAm
    uint256 currentRate =  _getRate();
    (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransfe
    uint256 rBurn =  tBurn.mul(currentRate);
    _tOwned[sender] = _tOwned[sender].sub(tAmount);
```

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

7.2: Compiler version is not fixed:-

=> In this file you have put “pragma solidity ^0.6.0;” 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 >=^0.6.0; // bad: compiles 0.6.0 and above pragma solidity 0.6.0; //good: compiles 0.6.0 only

=> If you put(>=) symbol then you are able to get compiler version 0.6.0 and above. But if you don’t use(^/>=) symbol then you are able to use only 0.6.0 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.

7.3: 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

```
594    function _approve(address owner, address spender, uint256 amount) private {
595        require(owner != address(0), "BEP20: approve from the zero address");
596        require(spender != address(0), "BEP20: approve to the zero address");
597
598        _allowances[owner][spender] = amount;
```

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

• 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 ;)).

- **Good Point:** Address validation and value validation is done properly.
- **Suggestions:** Please add address validations at some place and also try to use static solidity version, check amount in approve function and I found some place you forgot use safeMath please do it if possible.