SMART CONTRACT AUDIT REPORT

For
Knightdoge (KNDG)

Prepared By: Kishan Patel

Prepared For: knightdoge (KNDG)

Prepared on: 23/11/2021

Table of Content

- Disclaimer
- Overview of the audit
- Attacks made to the contract
- Good things in smart contract
- Critical vulnerabilities found in the contract
- Medium vulnerabilities found in the contract
- Low severity vulnerabilities found in the contract
- Summary of the audit

• 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 1174 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 ethereum'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 ethereum 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 Ethereum to a contract

While implementing "selfdestruct" in smart contract, it sends all the ethereum 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

- ### Compiler version is fixed:-

  => In this file you have put "pragma solidity 0.6.12;" which is a good way to define compiler version.

  => Solidity source files indicate the versions of the compiler they can be compiled with. Pragma solidity ^0.6.12; // bad: compiles 0.6.12 and above pragma solidity 0.6.12; //good: compiles 0.6.12 only

  => If you put(>=) symbol then you are able to get compiler version 0.6.12 and above. But if you don't use(^/>=) symbol then you are able to use only 0.6.12 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.

  => Try to use latest version of solidity.

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

```
92
93  library SafeMath {
94      /**
95       * @dev Returns the addition of two unsigned integers, reverting on
96       * overflow.
97       *
```

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

```
296     function sendValue(address payable recipient, uint256 amount) internal {
297         require(address(this).balance >= amount, "Address: insufficient balance");
298 
299         // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
300         (bool success, ) = recipient.call{ value: amount }("");
301         require(success, "Address: unable to send value, recipient may have revert
302     }
```

  - Here you are checking that the contract has more or equal balance then value and target address is contract address.

```
356      */
357     function functionCallWithValue(address target, bytes memory data, uint256 valu
358         require(address(this).balance >= value, "Address: insufficient balance for
359         return _functionCallWithValue(target, data, value, errorMessage);
360     }
```

  - Here you are checking that the target address is a proper contract address or not.

```
361 
362     function _functionCallWithValue(address target, bytes memory data, uint256 wei
363         require(isContract(target), "Address: call to non-contract");
364 
```

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

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

  - Here you are checking that previous owner has no permission to unlock it and current time should be bigger than _lockTime. Make a condition that only owner of contract can call this function.

```
462 
463     //Unlocks the contract for owner when _lockTime is exceeds
464     function unlock() public virtual {
465         require(_previousOwner == msg.sender, "You don't have permission to unlock
466         require(now > _lockTime , "Contract is locked until 7 days");
467         emit OwnershipTransferred(_owner, _previousOwner);
```

- Here you are checking that this function is not called by the address which is excluded.

```
818 ▾   function deliver(uint256 tAmount) public {
819         address sender = _msgSender();
820         require(!_isExcluded[sender], "Excluded addresses cannot call this functio
821         (uint256 rAmount,,,,,) = _getValues(tAmount);
822         _rOwned[sender] = _rOwned[sender].sub(rAmount);
```

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

```
827 ▾   function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public v
828         require(tAmount <= _tTotal, "Amount must be less than supply");
829 ▾       if (!deductTransferFee) {
830             (uint256 rAmount,,,,,) = _getValues(tAmount);
831             return rAmount;
```

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

```
837
838 ▾   function tokenFromReflection(uint256 rAmount) public view returns(uint256) {
839         require(rAmount <= _rTotal, "Amount must be less than total reflections");
840         uint256 currentRate =  _getRate();
```

- Here you are checking that account address is not already excluded from a reward and account address is not pancake router address.

```
843
844 ▾   function excludeFromReward(address account) public onlyOwner() {
845         require(account != 0x05fF2B0DB69458A0750badebc4f9e13aDd608C7F, 'We can not
846         require(!_isExcluded[account], "Account is already excluded");
847 ▾       if(_rOwned[account] > 0) {
```

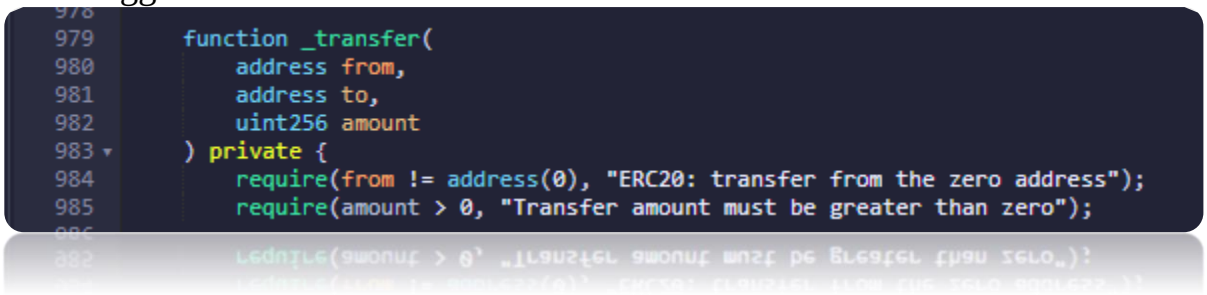
- Here you are checking that an account address is not already included for reward.

```
853
854 ▾   function includeInReward(address account) external onlyOwner() {
855         require(_isExcluded[account], "Account is already excluded");
856 ▾       for (uint256 i = 0; i < _excluded.length; i++) {
857 ▾           if (_excluded[i] == account) {
```

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

```
970
971 ▾   function _approve(address owner, address spender, uint256 amount) private {
972         require(owner != address(0), "ERC20: approve from the zero address");
973         require(spender != address(0), "ERC20: approve to the zero address");
974
975         _allowances[owner][spender] = amount;
```

- Here you are checking that address values of from is proper, an amount should be bigger than 0.



```
979     function _transfer(
980         address from,
981         address to,
982         uint256 amount
983     ) private {
984         require(from != address(0), "ERC20: transfer from the zero address");
985         require(amount > 0, "Transfer amount must be greater than zero");
```

- **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 only necessary functions.

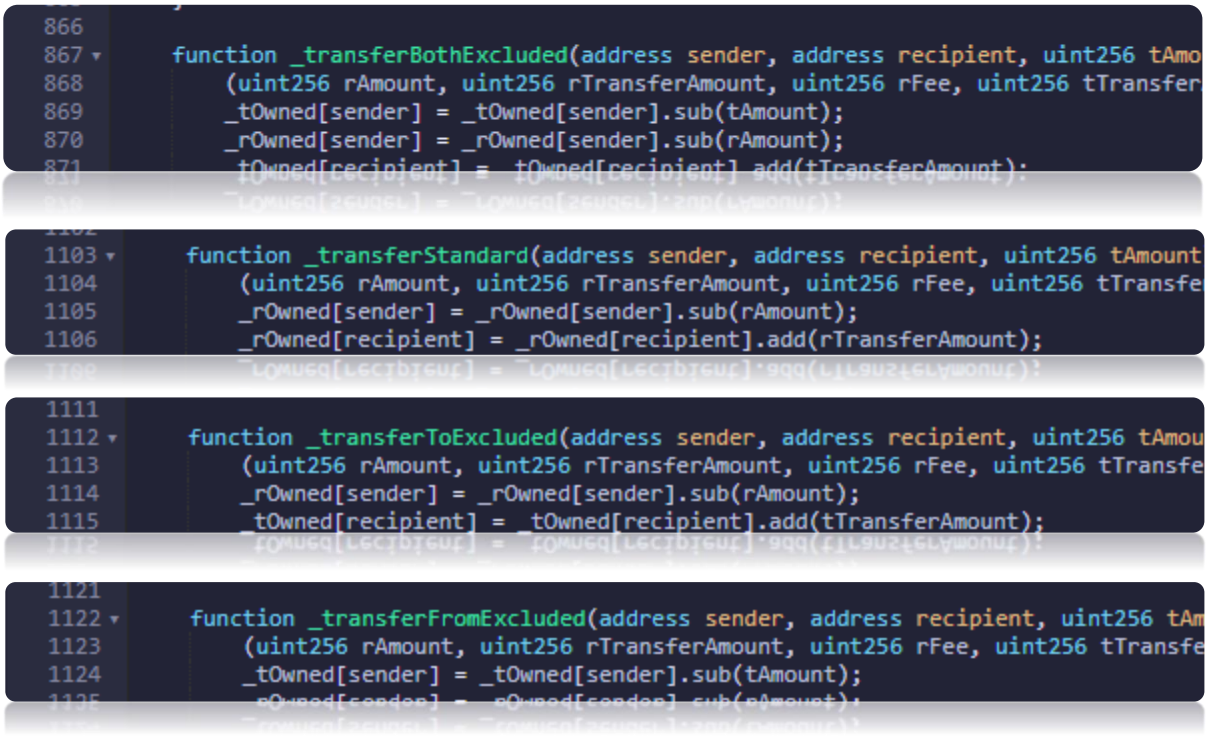
Function: - excludeFromReward, includeInReward ('account')

```
844     function excludeFromReward(address account) public onlyOwner() {
845         require(account != 0x05fF2B0DB69458A0750badebc4f9e13aDd608C7F, 'We can not
846         require(!_isExcluded[account], "Account is already excluded");
847         if(_rOwned[account] > 0) {
```

```
854     function includeInReward(address account) external onlyOwner() {
855         require(_isExcluded[account], "Account is already excluded");
856         for (uint256 i = 0; i < _excluded.length; i++) {
857             if (_excluded[i] == account) {
```

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

Function: - _transferStandard, _transferToExcluded, _transferFromExcluded, _transferBothExcluded ('sender', 'recipient')



```
866
867     function _transferBothExcluded(address sender, address recipient, uint256 tAmo
868         (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransfer
869         _tOwned[sender] = _tOwned[sender].sub(tAmount);
870         _rOwned[sender] = _rOwned[sender].sub(rAmount);
871         _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);

1103    function _transferStandard(address sender, address recipient, uint256 tAmount
1104        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransfe
1105        _rOwned[sender] = _rOwned[sender].sub(rAmount);
1106        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);

1111
1112    function _transferToExcluded(address sender, address recipient, uint256 tAmou
1113        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransfe
1114        _rOwned[sender] = _rOwned[sender].sub(rAmount);
1115        _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);

1121
1122    function _transferFromExcluded(address sender, address recipient, uint256 tAm
1123        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransfe
1124        _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: Suggestions to add validations In code:-

=> You have implemented required validation in contract.

=> There are some place where you can improve validation and security of your code.

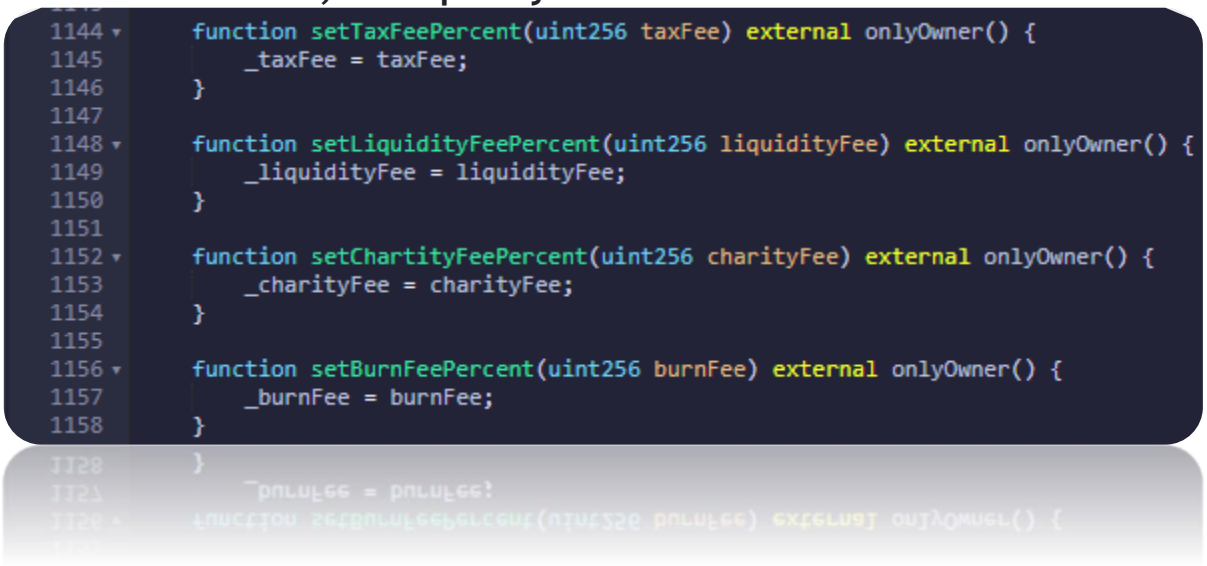
=> These are all just suggestion it is not bug.

Function: - excludeFromFee, includeInFee

- Here in excludeFromFee function you need to check that account address is not already excluded from fee if it is already excluded then this function should not run.

- Here in includeInFee function you need to check that account address is not already included for fee if it is already included then this function should not run.

- **Function: - setTaxFeePercent, setChartityFeePercent, setBurnFeePercent, setLiquidityFeePercent**



```
1144    function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
1145        _taxFee = taxFee;
1146    }
1147
1148    function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner() {
1149        _liquidityFee = liquidityFee;
1150    }
1151
1152    function setCharityFeePercent(uint256 charityFee) external onlyOwner() {
1153        _charityFee = charityFee;
1154    }
1155
1156    function setBurnFeePercent(uint256 burnFee) external onlyOwner() {
1157        _burnFee = burnFee;
1158    }
```

  - Here in all functions you can check that parameter value should be bigger than 0.

• Summary of the Audit

Overall, the code is written with all validation and all security

is implemented. Code is performs well and there is no way to

steal fund from this contract.

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:** Code performance and quality is good. Address validation and value validation is done properly.
- **Suggestions:** Please add address validation in shown functions, and try to include suggested validation in code.