

**SMART CONTRACT AUDIT REPORT**

**For**

**DolphinFinanceforNature**

**Prepared By:** Kishan Patel

**Prepared on:** 28/05/2021

**Prepared For:** Jesusnelson

# 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 1282 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**

- **SafeMath library:-**

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

```
100 library SafeMath {
101     /**
102      * @dev Returns the addition of two unsigned integers, with an overflow flag.
103      *
104      * Available since v3.4.
105      */
106     function add(uint256 a, uint256 b) internal pure returns (uint256, bool) {
107         uint256 c = a + b;
108         bool overflow = c < a;
109         return (c, overflow);
110     }
111 }
```

- **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.

```
374 function sendValue(address payable recipient, uint256 amount) internal {
375     require(address(this).balance >= amount, "Address: insufficient balance")
376
377     // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
378     (bool success, ) = recipient.call{ value: amount }("");
379     require(success, "Address: unable to send value, recipient may have reverted")
380 }
381
382 function _transfer(address sender, address recipient, uint256 amount) internal {
383     require(sender != address(0), "Address: zero address")
384     require(recipient != address(0), "Address: zero address")
385     require(sender != address(this), "Address: self transfer")
386     require(recipient != address(this), "Address: self transfer")
387     require(amount > 0, "Amount: zero")
388     require(address(this).balance >= amount, "Address: insufficient balance")
389     _doTransfer(sender, recipient, amount)
390 }
```

- Here you are checking that the contract has more or equal balance then value, target address is contract address or not.

```
435 function functionCallWithValue(address target, bytes memory data, uint256 value) public {
436     require(address(this).balance >= value, "Address: insufficient balance for call")
437     require(isContract(target), "Address: call to non-contract");
438     _doFunctionCall(target, data, value);
439 }
```

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

```
459 // ...
460 function functionStaticCall(address target, bytes memory data, string memory e
461     require(isContract(target), "Address: static call to non-contract");
462
463 // solhint-disable-next-line avoid-low-level-calls
```

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

```
484 function functionDelegateCall(address target, bytes memory data, string memory
485     require(isContract(target), "Address: delegate call to non-contract");
486
487 // solhint-disable-next-line avoid-low-level-calls
```

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

```
568 function transferOwnership(address newOwner) public virtual onlyOwner {
569     require(newOwner != address(0), "Ownable: new owner is the zero address");
570     emit OwnershipTransferred(_owner, newOwner);
571     _owner = newOwner;
572 }
```

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

*manual burn only - owner have to send tokens to burn address*

```
831 function _burn(address account, uint256 amount) public virtual onlyOwner{
832     require(account != address(0), "ERC20: burn from the zero address");
833
834     // _beforeTokenTransfer(account, address(0), amount);
835 }
```

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

```
923 function deliver(uint256 tAmount) public {
924     address sender = _msgSender();
925     require(!isExcluded[sender], "Excluded addresses cannot call this function");
926     (uint256 rAmount, uint256 rTotal, uint256 currentRate) = _getValues(tAmount);
927     tokenFromReflection(rAmount);
928 }
```

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

```
932 function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public view returns (uint256) {
933     require(tAmount <= _tTotal, "Amount must be less than supply");
934     if (!deductTransferFee) {
935         (uint256 rAmount, uint256 rTotal, uint256 currentRate) = _getValues(tAmount);
936         return rAmount;
937     }
938     (uint256 rAmount, uint256 rTotal, uint256 currentRate) = _getValues(tAmount);
939     return rAmount;
940 }
```

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

```
942 function tokenFromReflection(uint256 rAmount) public view returns (uint256) {
943     require(rAmount <= _rTotal, "Amount must be less than total reflections");
944     uint256 currentRate = _getRate();
945     return rAmount div(currentRate);
946 }
```

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

```
949 ▾ function excludeFromReward(address account) public onlyOwner() {
950     // require(account != 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, 'We can
951     require(!_isExcluded[account], "Account is already excluded");
952     if( _excluded[account] > 0) {
```

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

```
958
959 ▾ function includeInReward(address account) external onlyOwner() {
960     require(_isExcluded[account], "Account is already included");
961     for (uint256 i = 0; i < _excluded.length; i++) {
```

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

```
1118
1119 ▾ function _approve(address owner, address spender, uint256 amount) private {
1120     require(owner != address(0), "ERC20: approve from the zero address");
1121     require(spender != address(0), "ERC20: approve to the zero address");
```

- Here you are checking that addresses values of from and to are proper, an amount should be bigger than 0 and less than \_maxTxAmount (Maximum amount to transfer token).

```
1127 function _transfer(
1128     address from,
1129     address to,
1130     uint256 amount
1131 ) private {
1132     require(from != address(0), "ERC20: transfer from the zero address");
1133     require(to != address(0), "ERC20: transfer to the zero address");
1134     require(amount > 0, "Transfer amount must be greater than zero");
1135     if(from != owner() && to != owner())
1136         require(amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount");
1137
1138     _balances[from] -= amount;
1139     require(_balances[from] >= 0, "ERC20: transfer from the zero address");
1140     _balances[to] += amount;
```

## • Critical vulnerabilities found in the contract

=> No Critical 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: - isContract ('account')**

```
347 ▾ function isContract(address account) internal view returns (bool) {  
348     // This method relies on extcodesize, which returns 0 for contracts in  
349     // construction, since the code is only stored at the end of the  
350     // constructor execution.  
351  
352     uint256 size;  
353     assembly {  
354         size := extcodesize(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: - excludeFromReward, includeInReward ('account')**

```
948  
949 ▾ function excludeFromReward(address account) public onlyOwner() {  
950     // require(account != 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, 'We can  
951     require(!_isExcluded[account], "Account is already excluded");  
952     if( _owned[account] > 0) {
```

```
958  
959 ▾ function includeInReward(address account) external onlyOwner() {  
960     require(_isExcluded[account], "Account is already included");  
961     for (uint256 i = 0; i < _excluded.length; i++) {  
962         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: - \_transferBothExcluded ('sender', 'recipient')**

```
970  
971 ▾ function _transferBothExcluded(address sender, address recipient, uint256  
972     (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransfe  
973     _tOwned[sender] = _tOwned[sender].sub(tAmount);  
974     _tOwned[recipient] = _tOwned[recipient].add(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')

```
1249
1250 ▾ function _transferStandard(address sender, address recipient, uint256 tAmount
1251 (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransfe
1252 _rOwned[sender] = _rOwned[sender].sub(rAmount);
1253 _rOwned[recipient] = _rOwned[recipient].add(rAmount);

1255
1256 ▾ function _transferToExcluded(address sender, address recipient, uint256 tAmou
1261 (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransfe
1262 _rOwned[sender] = _rOwned[sender].sub(rAmount);
1263 _rOwned[recipient] = _rOwned[recipient].add(rAmount);

1265
1266 ▾ function _transferFromExcluded(address sender, address recipient, uint256 tAn
1271 (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransfe
1272 _tOwned[sender] = _tOwned[sender].sub(tAmount);
1273 _rOwned[sender] = _rOwned[sender].sub(rAmount);
1274 _rOwned[recipient] = _rOwned[recipient].add(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.

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

=> In this file you have put “pragma solidity ^0.8.3;” 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 >=0.8.3; // bad: compiles 0.8.3 and above`  
`pragma solidity 0.8.3; //good: compiles 0.8.3 only`

=> If you put(>=) symbol then you are able to get compiler version 0.8.3 and above. But if you don't use(^/>=) symbol then you are able to use only 0.8.3 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.



### ○ 7.3: Approve given more allowance:-

=> I have found that in approve function user can give more allowance to a 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. So it is good to check that a user not set approval wrongly.

#### ✚ Function: - \_approve

```
1119 function _approve(address owner, address spender, uint256 amount) private {  
1120     require(owner != address(0), "ERC20: approve from the zero address");  
1121     require(spender != address(0), "ERC20: approve to the zero address");  
1122  
1123     _allowances[owner][spender] = amount;  
1124     emit Approval(owner, spender, amount);  
1125 }
```

- Here you can check that balance of owner should be bigger or equal to amount value.

## • 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 ;) ).

- **Good Point:** Latest solidity version is used, code performance is good. Address validation and value validation is done properly.
- **Suggestions:** Please add address validations at some place and also try to use the static version of solidity, check amount in approve function, and check burn functionality.