# SMART CONTRACT AUDIT REPORT

# For

# WaterDrop(Order #FO2BD77C64C4)

**Prepared By**: Kishan Patel

**Prepared For**: WaterDrop Token

**Prepared on**: 22/02/2021

# 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 146 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 eth 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 amount will be less than balance of fundsWallet.

```
121     function() payable{
122         totalEthInWei = totalEthInWei + msg.value;
123         uint256 amount = msg.value * unitsOneEthCanBuy;
124         require(balances[fundsWallet] >= amount);
125 
126         balances[fundsWallet] = balances[fundsWallet] - 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. I am showing here only some important functions.

### Function:- transfer ('to')

```
47    function transfer(address _to, uint256 _value) returns (bool success) {
48        //Default assumes totalSupply can't be over max (2^256 - 1).
49        //If your token leaves out totalSupply and can issue more tokens as time g
50        //Replace the if with this one instead.
51        //if (balances[msg.sender] >= _value && balances[_to] + _value > balances[
52        if (balances[msg.sender] >= _value && _value > 0) {
```

- 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')

```
60    function transferFrom(address _from, address _to, uint256 _value) returns (bool
61        //same as above. Replace this line with the following if you want to protec
62        //if (balances[_from] >= _value && allowed[_from][msg.sender] >= _value &&
63        if (balances[_from] >= _value && allowed[_from][msg.sender] >= _value && _
64            balances[_to] += _value;
65            balances[_from] -= _value;
```

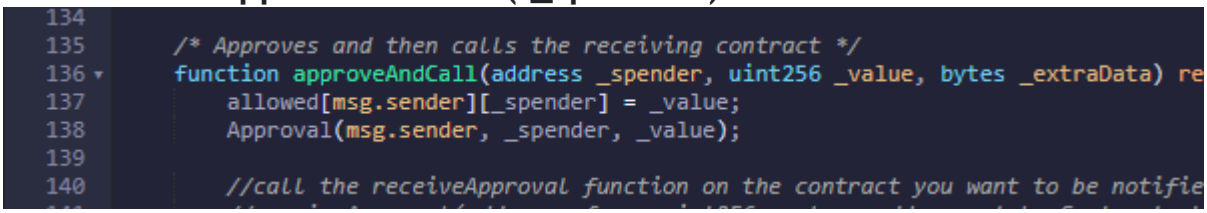
- It's necessary to check the addresses value of "_from","_to". Because here you are passing whatever variable come in "_from", "_to" addresses from outside.

### Function: - approve ('_spender')

```
76    function approve(address _spender, uint256 _value) returns (bool success) {
77        allowed[msg.sender][_spender] = _value;
78        Approval(msg.sender, _spender, _value);
79        return true;
```

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

- **Function: - approveAndCall (‘_spender’)**




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

## 7.2: Unchecked return value or response:-

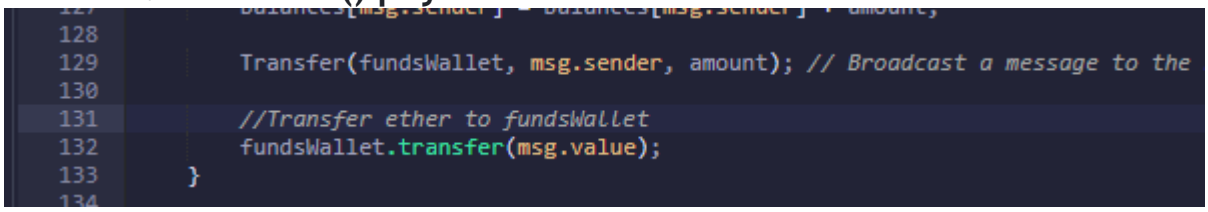
=> I have found that you are transferring fund to address using a transfer method.

=> It is always good to check the return value or response from a function call.

=> You are checking in ever place but at one place you forgot to check.

=> I suggest you to check that for more security.

- **Function: - function() payable**




  - Here you are calling transfer method 1 time. It is good to check that the transfer is successfully done or not.

## 7.3: SafeMath library:-

=> SafeMath library is good to protect your contract from underflow and overflow attacks.

=> You are not using safeMath library in contract.

=> I recommended you to implement safeMath library so you can make your contract safer for underflow and overflow attack.

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

  => In this file you have put “pragma solidity ^0.4.22;” 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 ^0.4.22; // bad: compiles 0.4.22 and above pragma solidity 0.4.22; //good: compiles 0.4.22 only

  => If you put(>=) symbol then you are able to get compiler version 0.4.22 and above. But if you don’t use (^/>=) symbol then you are able to use only 0.4.22 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.

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 latest and static verions, check the response of transfer method at one place, use SafeMath library, and check addresses.