SMART CONTRACT AUDIT REPORT

For

Staking (DogDeFi)

Prepared By: Kishan Patel

Prepared For: DogDeFiCoin Team

Prepared on: 24/10/2020

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 502 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

- ### Compiler version is fixed:-

=> In this file you have put "pragma solidity 0.6.11;" 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 >=0.6.11; // bad: compiles 0.6.11 and above pragma solidity 0.6.11; //good: compiles 0.6.11 only

=> If you put(>=) symbol then you are able to get compiler version 0.6.11 and above. But if you don't use (^/>=) symbol then you are able to use only 0.6.11 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.

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

```
12   */
13 library SafeMath {
14   function mul(uint256 a, uint256 b) internal pure returns (u
15     uint256 c = a * b;
16     assert(a == 0 || c / a == b);
17     return c;
18   }
```

- **Good required condition in functions:-**
  - Here you are checking that index is less than storage length.

```
163     */
164     function _at(Set storage set, uint256 index) private view returns (bytes32) {
165         require(set._values.length > index, "EnumerableSet: index out of bounds");
166         return set._values[index];
167     }
168
```

  - Here you are checking that newOwner address is proper also new owner is set after transaction is successful.

```
312     */
313   function transferOwnership(address newOwner) onlyOwner public {
314     require(newOwner != address(0));
315     emit OwnershipTransferred(owner, newOwner);
316     owner = newOwner;
317   }
```

  - Here you are checking that token is transferred to account address or not. If not then it throw error.

```
370
371     function updateAccount(address account) private {
372         uint pendingDivs = getPendingDivs(account);
373         if (pendingDivs > 0) {
374             require(Token(trustedRewardTokenAddress).
375             totalEarnedTokens[account] = totalEarnedT
```

  - Here you are checking that amountToStake is bigger than 0 and transferFrom method from token called successfully and if it failed then this method will throw error.

```
414     function stake(uint amountToStake) public {
415         require(amountToStake > 0, "Cannot stake 0 Tokens");
416         require(Token(trustedStakeTokenAddress).transferFrom(msg.sender, a
417
418         updateAccount(msg.sender);
```

  - Here you are checking that amountToWithdraw is bigger than 0 and user has sufficient balance to withdraw and also transfer method of token is called successfully.

```
428     function unstake(uint amountToWithdraw) public {
429         require(amountToWithdraw > 0, "Cannot unstake 0
430         require(depositedTokens[msg.sender] >= amountTo
431
432         updateAccount(msg.sender);
433
434         require(Token(trustedStakeTokenAddress).transfe
```

- Here you are checking that amountToWithdraw is bigger than 0 and user has sufficient balance to withdraw and also transfer method of token is called successfully.

```
444     // pending earnings will be lost / set to 0 if used emergency
445     function emergencyUnstake(uint amountToWithdraw) public {
446         require(amountToWithdraw > 0, "Cannot unstake 0 Tokens");
447         require(depositedTokens[msg.sender] >= amountToWithdraw,
448
449         // set pending earnings to 0 here
450         lastClaimedTime[msg.sender] = now;
451
452         require(Token(trustedStakeTokenAddress).transfer(msg.send
453
```

- Here you are checking that admin is not able to transfer Stake tokens and, Admin is able to transfer token after adminClaimableTime(396 days).

```
494     // Admin can transfer out reward tokens from this address once admi
495     function transferAnyERC20Tokens(address _tokenAddr, address _to, ui
496         require(_tokenAddr != trustedStakeTokenAddress, "Admin cannot t
497
498         require((_tokenAddr != trustedRewardTokenAddress) || (now > adm
499
```

• 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:- updateAccount (‘account’)**

```
371     function updateAccount(address account) private {
372         uint pendingDivs = getPendingDivs(account);
373         if (pendingDivs > 0) {
374             require(Token(trustedRewardTokenAddress).t
375             totalEarnedTokens[account] = totalEarnedTo
```

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

- **Function: - getPendingDivs (‘_holder’)**

```
382     function getPendingDivs(address _holder) public v
383         if (!holders.contains(_holder)) return 0;
384         if (depositedTokens[_holder] == 0) return 0;
385
386         uint timeDiff;
387         uint _now = now;
```

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

- **Function: - transferAnyERC20Tokens (‘_tokenAddr’, ‘_to’)**

```
495     function transferAnyERC20Tokens(address _tokenAddr, address _to,
496         require(_tokenAddr != trustedStakeTokenAddress, "Admin canno
497
498         require((_tokenAddr != trustedRewardTokenAddress) || (now >
499
500         Token(_tokenAddr).transfer(_to, _amount);
501     }
```

  - It's necessary to check the addresses value of "_tokenAddr", "_to". Because here you are passing whatever variable comes in "_tokenAddr", "_to" addresses 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: - transferAnyERC20Tokens**

```
495     function transferAnyERC20Tokens(address _tokenAddr, add
496         require(_tokenAddr != trustedStakeTokenAddress, "Ad
497
498         require((_tokenAddr != trustedRewardTokenAddress) |
499
500         Token(_tokenAddr).transfer(_to, _amount);
501     }
```

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

• 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, check the response of transfer method at one place, and check addresses.
- I have seen that a developer is using now method so, I like to tell you that write smart contracts with the notion that block values are not precise, and the use of them can lead to unexpected effects.