SMART CONTRACT AUDIT REPORT

For

Diamonds Are Forever

Prepared By: Kishan Patel

Prepared For: Diamond Token Services

Prepared on: 01/12/2021

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 115 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.

```
28  library SafeMath {
29    function mul(uint256 a, uint256 b) internal constant returns (uin
30      uint256 c = a * b;
31      assert(a == 0 || c / a == b);
```

  - **Good required condition in functions:-**
    - Here you are checking that _value should be not 0. I guess you don't need second condition (allowed[msg.sender][_spender] == 0)

```
85  function approve(address _spender, uint256 _value) returns (bool)
86    require((_value == 0) || (allowed[msg.sender][_spender] == 0));
87
```

- **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: Compiler version is not fixed:-

    => In this file you have put “pragma solidity ^0.4.11;” 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 >=0.4.11; // bad: compiles 0.4.11 and above pragma solidity 0.4.11; //good: compiles 0.4.11 only

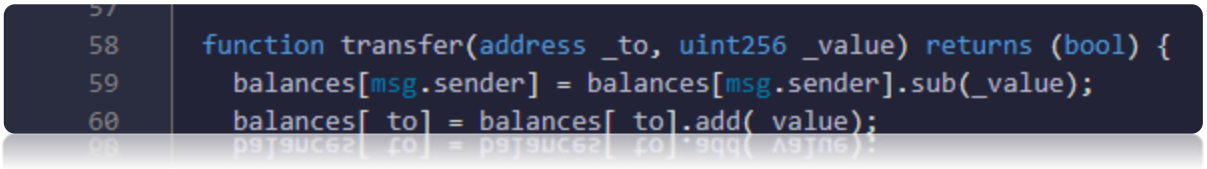
    => If you put(>=) symbol then you are able to get compiler version 0.4.11 and above. But if you don’t use(^/>=) symbol then you are able to use only 0.4.11 version. And if there are some changes come in the compiler and you use the old version then some issues may come at deploy time.

    => Try to use latest version of solidity.

  - ## 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: - transfer

    - Here in this function you need to check that _to address is valid and proper.

    - You need to check that _value amount should be bigger than 0 and smaller than balance of msg.sender address.

    - You need to give visibility level of this function.

Function: - transferFrom

```
75  function transferFrom(address _from, address _to, uint256 _value)
76    var _allowance = allowed[_from][msg.sender];
77
78    balances[_to] = balances[_to].add(_value);
```

- Here in this function you need to check that _from and _to addresses are valid and proper.
- You need to check that _value amount should be bigger than 0 and smaller than balance of from address.
- You need to check that from msg.sender to from address has more allowance than _value amount.
- You need to give visibility level of this function.

Function: - approve

```
85  function approve(address _spender, uint256 _value) returns (bool)
86    require((_value == 0) || (allowed[msg.sender][_spender] == 0));
87
88    allowed[msg.sender][_spender] = _value;
```

- Here in this function you need to check that _spender address is valid and proper.
- You need to check that _value amount should be bigger than 0 and smaller than balance of msg.sender address.
- You need to give visibility level of this function.

- # Summary of the Audit

Overall, the code is written without much validation and all security is not implemented. Code is performs well with functionality it has implemented but there is no much validation done.

Please try to check the address and value of token externally before sending to the solidity code.

- **Good Point:** Code performance and quality is good.
- **Suggestions:** Please try to use latest and static version of solidity, add suggested code validations.