SMART CONTRACT AUDIT REPORT

For

Beeuda

Prepared By: Kishan Patel

Prepared For: Beeuda

Prepared on: 10/07/2021

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 190 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.

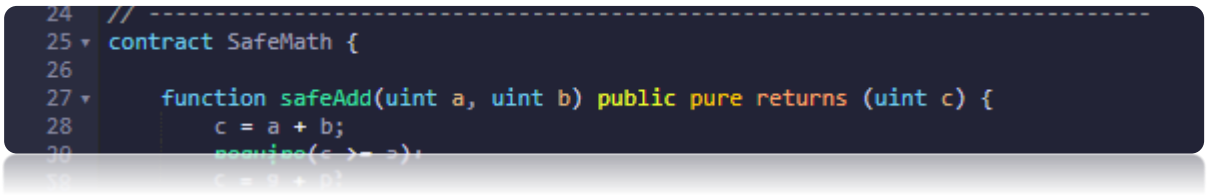
Use of "require" function in this smart contract mitigated this vulnerability.

- ## 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.



• 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:-

=> As you are using very old version of solidity it is problematic to not check address value in function. I suggest you to add address value validation.

=> After updating the version of solidity it's not mandatory.

=> In all functions you are not checking the value of Address parameter so please check address in all functions.

Function: - transfer('to')

```
120     // ------------------------------------------------------------------------
121     function transfer(address to, uint tokens) public returns (bool success) {
122         balances[msg.sender] = safeSub(balances[msg.sender], tokens);
123         balances[to] = safeAdd(balances[to], tokens);
124         emit Transfer(msg.sender, to, tokens);
```

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

Function: - approve ('spender')

```
136     // ------------------------------------------------------------------------
137     function approve(address spender, uint tokens) public returns (bool success) {
138         allowed[msg.sender][spender] = tokens;
139         emit Approval(msg.sender, spender, tokens);
140         return true;
```

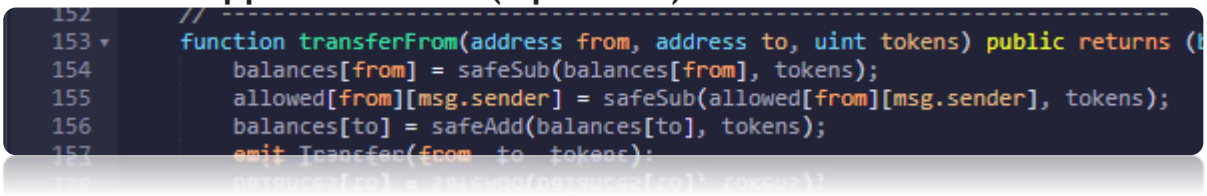
- It's necessary to check the address value of "spender". Because here you are passing whatever variable comes in "spender" address from outside.

Function: - transferFrom ('from', 'to')

```
152     // ------------------------------------------------------------------------
153     function transferFrom(address from, address to, uint tokens) public returns (
154         balances[from] = safeSub(balances[from], tokens);
155         allowed[from][msg.sender] = safeSub(allowed[from][msg.sender], tokens);
156         balances[to] = safeAdd(balances[to], tokens);
```

- It's necessary to check the addresses value of "from", "to". Because here you are passing whatever variable comes in "from", and "to" addresses 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: Compiler version is not fixed:-

=> In this file you have put "pragma solidity ^0.4.24;" 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 >=0.4.24; // bad: compiles 0.4.24 and above pragma solidity 0.4.24; //good: compiles 0.4.24 only

=> If you put(>=) symbol then you are able to get compiler version 0.4.24 and above. But if you don't use(^/>=) symbol then you are able to use only 0.4.24 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.

7.3: Approve can give 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

```
137 ▾   function approve(address spender, uint tokens) public returns (bool success) {
138         allowed[msg.sender][spender] = tokens;
139         emit Approval(msg.sender, spender, tokens);
140         return true;
141     }
```

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

Function: - approveAndCall

```
176 ▾   function approveAndCall(address spender, uint tokens, bytes data) public retur
177         allowed[msg.sender][spender] = tokens;
178         emit Approval(msg.sender, spender, tokens);
179         ApproveAndCallFallBack(spender).receiveApproval(msg.sender, tokens, this,
180         return true;
181     }
```

- Here you can check that balance of spender 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:** Code performance and quality is good.
- **Suggestions:** Please add address validations. use the latest and static version of solidity, and check user balance in approve function.