GDO Infotech Private Limited
Amreli – Surat – Poland – UK – Canada
Web: www.gdo.co.in **Email**: contact@gdo.co.in
Ph: +91 9712144344

SMART CONTRACT AUDIT REPORT

For

Ju_lia (Order #FO8178FB49A08)

Prepared By: Yogesh Padsala

Prepared For: ju_lia

Prepared on: 13/04/2018

Table of Content

1. 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.

2. Overview of the audit and nice features

The project has only one file, the MainToken.sol, which contains 577 lines of Solidity code. All the functions and state variables are well commented using the natspec documentation for the functions which is good to understand quickly how everything is supposed to work.

Nice Features:

The contract provides a good suite of functionality that will be useful for the entire contract.

It uses SafeMath (https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/math/SafeMath.sol) library to check for overflows and underflows which is a pretty good practice.

3. 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.

3.1: 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. However this contract checks for overflows and underflows by using OpenZeppelin's SafeMath and there is no instance of direct arithmetic operations.

3.2: 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.

This contract isn’t vulnerable to this attack since it doesn't have any Buy function but also it **does NOTHING to prevent** the *short address attack* during **ICO** or in an **exchange** (it will just depend if the ICO contract or exchange server checks the length of data, if they don't, short address attacks would drain out this coin from the exchange).

3.3: Visibility & Delegatecall*

It is also known as, The Parity Hack, which occurs while misuse of Delegatecall.

No such issues found in the smart contract and visibility also properly addressed. There are some places (for example line 104 and 106) 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.

3.4: Reentrancy / TheDAO hack*

Reentrancy occurs in this case: any interaction from a contract (A) with another contract (B) and any transfer of Ether 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.

3.5: Forcing ether to a contract*

While implementing “selfdestruct” in smart contract, it sends all the ether 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.

4. Critical vulnerabilities found in the contract

4.1: As per ERC20 standard transfer, transferFrom and approve methods returns boolean indicating whether the said operation is successfully completed or not. This is critical as it makes the token non ERC20 compliant.

4.2: MainToken.sol- In line 565 and 279. The contract contains unknown address. This address might be used for some malicious activity.

4.3: MainToken.sol - In line 503 To 505 , 418 to 420 , 499 to 501 and 414 to 416. Functions transfer and transferFrom of ERC-20 Token should throw in special cases.

1) transfer should throw if the _from account balance does not have enough tokens to spend

2) transferFrom should throw unless the _from account has deliberately authorized the sender of the message via some mechanism.

5. Medium vulnerabilities found in the contract

5.1: MainToken.sol - In line 248 to 255 , 181 to 185 , 507 to 509 and 422 to 424. function approve might lead to vulnerabilities of ERC-20. Only use the approve function of the ERC-20 standard to change allowed amount to 0 or from 0 (wait till transaction is mined and approved).

5.2: You're specifying a pragma version with the caret symbol (^) up front which tells the compiler to use any version of solidity bigger than 0.4.21 .

This is not a good practice since there could be major changes between versions that would make your code unstable. That's why I recommend to set a fixed version without the caret like 0.4.21.

5.3: I would recommend you to use latest versions of solidity compiler instead of older versions as latest version contains many critical bug fixes. Using compiler version 0.4.21 is recommended.

6. Low severity vulnerabilities found

6.1: MainToken.sol - In line 166 , 277 , 164 , 165 , 208 and 276. On these lines the problem of Reentrancy. Any interaction from a contract (A) with another contract (B) and any transfer of Ether hands over control to that contract (B). This makes it possible for B to call back into A before this interaction is completed.

To avoid re-entrancy, you can use the Checks-Effects-Interactions pattern as outlined further below:

```
        pragma solidity ^0.4.11;

        contract Fund {

        /// Mapping of ether shares of the contract.

        mapping(address => uint) shares;

        /// Withdraw your share.

        function withdraw() {

                var share = shares[msg.sender];

                shares[msg.sender] = 0;

                msg.sender.transfer(share);

        }

}
```

Note that re-entrancy is not only an effect of Ether transfer but of any function call on another contract. Furthermore, you also have to take multi-contract situations into account. A called contract could modify the state of another contract you depend on.

6.2: MainToken.sol - In line 104 and 106.On these lines you are not mentioned visibility level. The default function visibility level in Solidity is public. Explicitly define function visibility to prevent confusion.

```
mapping(address => uint256) public balances;

uint256 public totalSupply_;
```

6.3: MainToken.sol - Your transfer and transferFrom are accept wei value which is not good because token value is always digital so please try to focus on it.

7. Good to have

7.1: While using approve and transferFrom functions, please keep notes of

https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729

7.2: It will be good to have if in approve you check the sender has the balance before approving and has not approved others to use that balance.

Ex- Alice has 1000 tokens and she has already approved bob to spend 500 tokens on behalf of her now she wants to allow Sameep to spend 600 tokens on her behalf. So if we will look Alice has given capacity to spend 1100 tokens on her behalf whereas she only owns 1000 tokens. So when she was allowing sameep to spend 600 on her behalf the transaction should be reverted. This is good to have.

8. Summary of the Audit

Overall the code is well commented.

Our final recommendation would be to pay more attention to the visibility of the functions 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 ;).

We also recommend not to put commented code. Because of commented code, certain portion of the contract might be useless and hence the contract might be not ready for the deployment. And this is not safe to use, since the commented code deals with the payable methods and may result in the loss of funds.

Please add one condition in transfer and transferFrom function to check the value / amount of the transaction initially. Because even if the value of transaction is 0 (zero), then also these functions execute and that causes waste of gas.

Please also try to verify the sender and receiver address. We did many transactions with short address and it was gone through.