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

SMART CONTRACT AUDIT REPORT

For

FLOh (Order # FO711C80EA5C5)

Prepared By: Yogesh Padsala

Prepared For: Signup1

Prepared on: 18/06/2018

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

The project has 1 file FLOh.sol. It contains approx 904 lines of Solidity code. All the functions and state variables are well commented using the natspec documentation.

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.

This contract **does** check for overflows and underflows by using OpenZeppelin's SafeMath to mitigate this attack, but it still have some issues (discussed below in detail)

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 DApp to check the length of data. If they don't, then 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 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.

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: Underflow & Overflow attack:

=>In the contract, some functions accept negative value:

=>Function name: setMinContribEth , parameter : _int

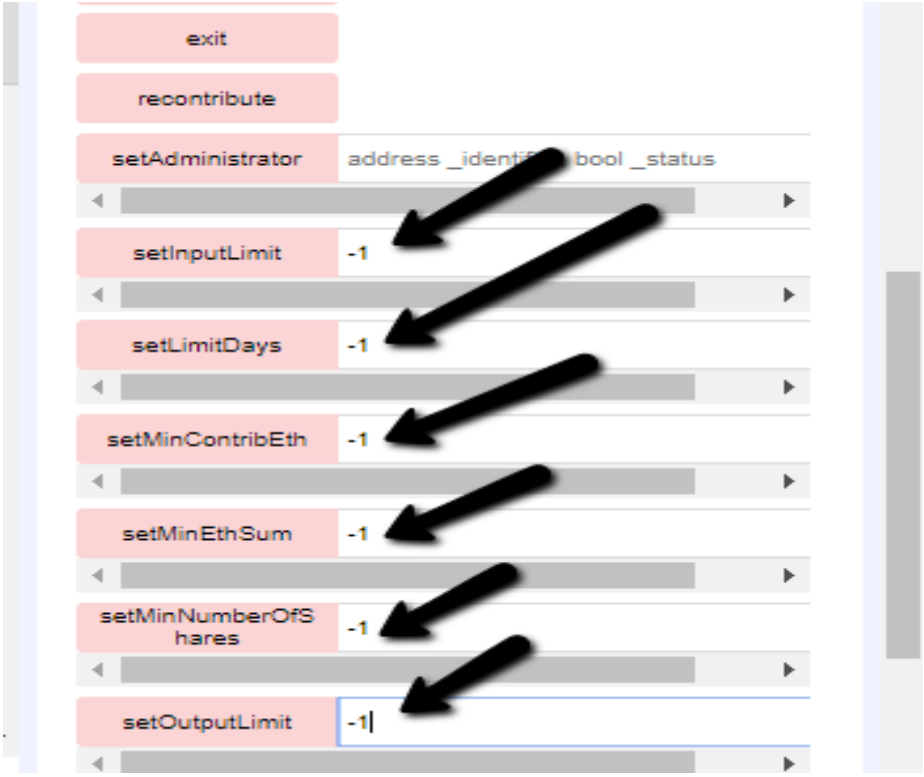
=>Function name: setMinNumberOfShares , parameter : _int

=>Function name: setMinEthSum , parameter : _int

=>Function name: setInputLimit , parameter : _int

=>Function name: setOutputLimit , parameter : _int

=>Function name: setLimitDays, parameter : _int



Transaction hash:

1) **Function :** setInputLimit

https://rinkeby.etherscan.io/tx/0xe3abd7964aa1cb10f3a2b0bc45dca0ec55240a6b0909af5365d79b0f3ffa8433

2) **Function :** setLimitDays

https://rinkeby.etherscan.io/tx/0x3472e46ab6e4c3014965f338d2b84ea132910c344c3f6de3716dc0564322b5b6

3) **Function :** setMinContribEth
https://rinkeby.etherscan.io/tx/0x50ee64198aea864533666e46822e6b068e8ac5dd137c4ec233340c5ef4eda2d1

4) **Function :** setMinEthSum

https://rinkeby.etherscan.io/tx/0x2624ccef30157bf744925f024629868e517c64b35b444e5793893a1da240ee0e

5) **Function :** setMinNumberOfShares
https://rinkeby.etherscan.io/tx/0xce48119adf17b0c437ffc8807b33b44606a1e59168693884b08b8c5c949c0fe8

6) **Function :** setOutputLimit

https://rinkeby.etherscan.io/tx/0xabcdad592bd6480148a626483c2a4696a500fdcecec2a65f3e4d575ba4ab85e5

Solution:-

- You have to check this variable value outside before sending to smart contract.
- The second way to stop this is that you have to do validation with default high value. So for example, all the input value must not be greater than certain fix amount. You can define any bigger number just for this validation.

5. Medium vulnerabilities found in the contract

No such vulnerabilities found.

6. Low severity vulnerabilities found

6.1: Implicit visibility level

=> This is not a big issue in solidity. But if you not define any visibility level, then it automatically takes public.

=> However, it is good practice to define visibility in variable and functions. Line number #167, #168, #171, #172, #175, #176, #177, #178, #179.

```
166
167     //Input FLOw control
168     mapping(address => uint256) lastContribute;
169     mapping(address => uint256) lastContributeDate;
170
171     //Output FLOw control
172     mapping(address => uint256) lastConversion;
173     mapping(address => uint256) lastConversionDate;
174
175     //FLOh level
176     mapping(address => address) sharer; //Who is your sharer
177     mapping(address => uint256) contributed; //How much have you contributed
178     mapping(address => uint256) acquired; //How much eth collected by shares
179     mapping(address => bool) counterFlag; //Already counted flag
180     mapping(address => uint8) numberOfShares; //How many valid shares have a sharer
181     mapping(address => uint8) public userLevel; //Level of a sharer
182
```

Solution:-

You have to add just public key word before variable name.

E.g

mapping(address => uint256) public lastContribute;

6.2: Defines a return type but never explicitly returns a value

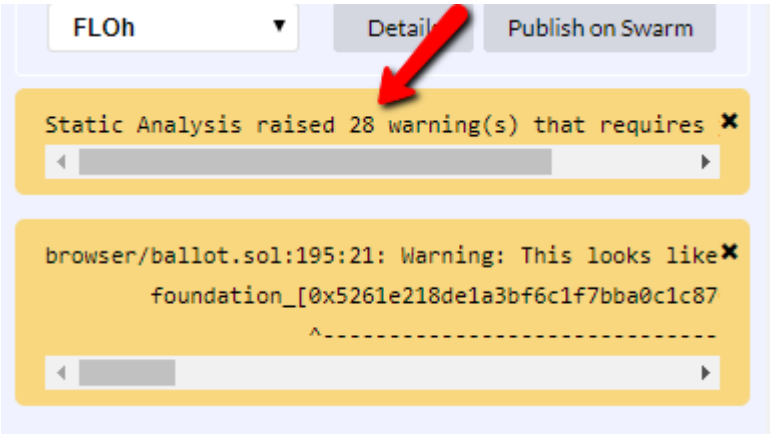
```
202     /**
203      * Converts all incoming ethereum to tokens for the caller, and passes down the sharer addy (if any)
204      */
205     function contribute(address _sharedBy)
206         public
207         payable
208         returns(uint256)
209     {
210         contributeTokens(msg.value, _sharedBy);
211     }
212
213     /**
```

=> The contribute function at line number #205 defines return type, but it does not return any value. **Solution**: Just remove that return type if not intended.

7. Summary of the Audit

Overall the code is well commented. And validations were performed at highest level at most of the places.

The compiler also displayed 28 warnings:



Now, we checked those warnings are due to their static analysis, which includes like gas errors and all. So, it is important to supply correct gas values while calling various functions.

Those warnings can be safely ignored as should be taken care while calling the smart contract functions.

We tried changing value of variable, *onlyFoundation* to *true* as well as *false*. When the value of *onlyFoundation* variable is *true*, then foundation can acquire tokens. Admin can set that variable to false by calling, *disableInitialStage()* function. When that variable is *false*, then it allows *transfer()* function, which allows donations from callers.

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 ;)).

Also, try to check the address and amount of token externally before sending to the solidity code.