Global Digital Opportunities

GDO Infotech Private Limited
Amreli – Surat, Gujarat, INDIA - 365410
Web: www.gdo.co.in **Email**: contact@gdo.co.in
Ph: +91 9712144344

SMART CONTRACT AUDIT REPORT

For

Fair Diamond Mining (Order # FO711C80)

Prepared By: Yogesh Padsala

Prepared For: FairDiamondMining

Prepared on: 22/08/2018

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 FairDiamondMining.sol. It contains approx 430 lines of Solidity code. All the functions and state variables are well commented using the natspec documentation, but that does not create any vulnerability.

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.

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.

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.

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. Good things in smart contract

4.1 transferOwnership Function:-

```
46
47      function transferOwnership(address newOwner) onlyOwner public {
48          require(newOwner != address(0));
49          emit OwnershipTransferred(owner, newOwner);
50          owner = newOwner;
51      }
```

- You are checking the value of newOwner before making the change, which is a good validation.

4.2 transfer Function:-

```
133      */
134     function transfer(address _to, uint _value, bytes _data) whenNotPaused external returns
135         // Standard function transfer similar to ERC20 transfer with no _data .
136         // Added due to backwards compatibility reasons .
137         require(balances[msg.sender] >= _value && _value > 0);
138         if(isContract(_to)){
```

- You are checking the value of balance of user before going further, which is a good thing.

4.3 transferToAddress Function:-

```
178 }
179 // function that is called when transaction target is an address
180   function transferToAddress(address _to, uint _value, bytes _data) private whenNotPaused returns (bool succes
181       require(_to != address(0));
182       require(balances[msg.sender] >= _value && _value > 0);
```

- Before making transfer, you are checking validity of address and also the balance of user should be higher than value, which is a good thing.

4.4 transferFrom Function:-

```
217     }
218     function transferFrom(address _from, address _to, uint256 _value) onlyPayloadSize(
219         require(_to != address(0));
220         require(_value <= balances[_from]);
221         require(_value <= allowed[_from][msg.sender]);
222
```

- Here you are checking the balance of from, allowance of sender and validity of address parameter, which is a good thing.

4.5 Approve Function:-

```
234     /// @return Whether the approval was successful or not
235     function approve(address _spender, uint256 _value) whenNotPaused public returns (bool success) {
236         require((balances[msg.sender] >= _value) && ((_value == 0) || (allowed[msg.sender][_spender] == 0)));
237         allowed[msg.sender][_spender] = _value;
238         emit Approval(msg.sender, _spender, _value);
239         return true;
240     }
```

- You are checking balance of user as well as the allowance before approving the transaction.

4.6 increaseApproval Function:-

```
247     }
248   function increaseApproval(address _spender, uint _addedValue) whenNotPaused public returns (bool success) {
249     require(balances[msg.sender] >= allowed[msg.sender][_spender].add(_addedValue), "Callers balance not enough");
250     allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
251     emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
252     return true;
```

- Here, it checks the balance of user must be higher than allowance in order to proceed, which is a good thing.

4.7 decreaseApproval Function:-

```
255   function decreaseApproval(address _spender, uint _subtractedValue) whenNotPaused p
256     uint oldValue = allowed[msg.sender][_spender];
257     require((_subtractedValue != 0) && (oldValue > _subtractedValue) , "The amount t
```

- Here first check the value of older allowance is bigger than new value then proceeding further, which is a good thing.

5. Critical vulnerabilities found in the contract

=> No critical vulnerabilities found

6. Medium vulnerabilities found in the contract

=> No Medium vulnerabilities found

7. Low severity vulnerabilities found

7.1: Compiler version not fixed

=> In this file you have put "pragma solidity ^0.4.24;" which is not 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 w 0.4.24 and above

pragma solidity 0.4.24; // good : compiles w 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 is some changes come in compiler and you use old version then some issue may come at deploy time.

7.2 Short address Issue

=> This is not a big issue in the solidity, because nowadays security is increased in new solidity version. But it is good practice to check for the short addresses.

=> In some functions you are not checking value of address parameter

- **function:- approve (‘_spender’’)**

```
233     /// @param _value The amount of wei to be approved for transfer
234     /// @return Whether the approval was successful or not
235     function approve(address _spender, uint256 _value) whenNotPaused public retu
236         require((balances[msg.sender] >= _value) && ((_value == 0) || (allowed[r
237         allowed[msg.sender][_spender] = _value;
238         emit Approval(msg.sender, _spender, _value);
239         return true;
```

  - **Solution:-**
- require(_spender != address(0));
- require(_to != address(0));

- **Function:- disApprove (‘_spender’)**

```
241     function disApprove(address _spender) whenNotPaused public retur
242     {
243         allowed[msg.sender][_spender] = 0;
244         assert(allowed[msg.sender][_spender] == 0);
```

  - **Solution:-**

    require(_spender != address(0));

- **Function:- increaseApproval (‘_spender’)**

```
248   function increaseApproval(address _spender, uint _addedValue) whenNotPaused public returns (boo
249     require(balances[msg.sender] >= allowed[msg.sender][_spender].add(_addedValue), "Callers balan
250     allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
```

  - **Solution:-**

    require(_spender != address(0));

- **function:- decreaseApproval (‘_spender’)**

```
255   function decreaseApproval(address _spender, uint _subtractedValue) whenNotPaused public returns (bool su
256     uint oldValue = allowed[msg.sender][_spender];
257     require((_subtractedValue != 0) && (oldValue > _subtractedValue) , "The amount to be decreased is inco
258     allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
```

  - **Solution:-**

    require(_spender != address(0));

- **function:- burnFrom (‘_from)**

```
255   function decreaseApproval(address _spender, uint _subtractedValue) whenNotPaused public returns
256     uint oldValue = allowed[msg.sender][_spender];
257     require((_subtractedValue != 0) && (oldValue > _subtractedValue) , "The amount to be decreased
258     allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
```

  - **Solution:-**

    require(_from!= address(0));

- **function:- mint (‘_to’)**

```
333   function mint(address _to, uint256 _amount) onlyOwner canMint public returns (bool)
334       bytes memory empty;
335       uint256 availableMinedSupply;
336       availableMinedSupply =  (_totalMinedSupply.sub(_totalBurnedTokens)).add(_amount
337     require(_CAP >= availableMinedSupply , "All tokens minted, Cap reached");
```

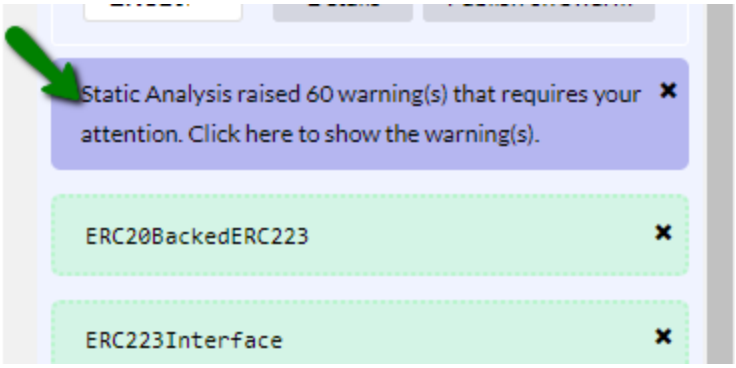
  - **Solution:-**

    require(_to != address(0));

8. Summary of the Audit

Overall the code is well commented, and performs good data validations.

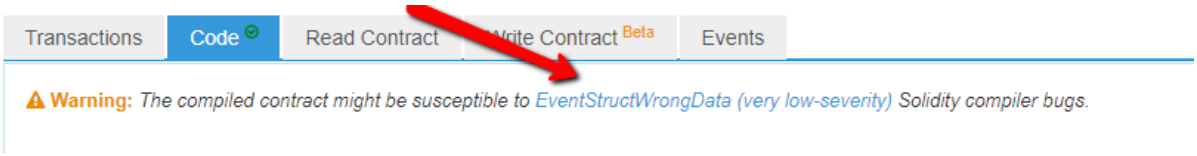
The compiler also displayed 60 warnings:



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 also noticed a warning in Etherscan, which we checked it is a solidity compiler bug which should be fixed in upcoming solidity version.



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:** You are using constant for display value but it's synonym of view. So it's better to use view instead of constant.