Global Digital Opportunities

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

ImmiCoin (Order # FO711C80EA5C5)

Prepared By: Yogesh Padsala **Prepared For**: ImmiCoin

Prepared on: 29/06/2018

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 4 files it contains approx 2200 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. However it has some concerns, which are discussed below.

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 contracts

4.1: File ImmiCoin.sol

4.1.1 _transferOwnership Function:-

```
 * @param newOwner The address to transfer ownership to.
 */
function transferOwnership(address newOwner) public onlyOwner {
    require(newOwner != address(0));
    pendingOwner = newOwner;
}

/**
```

- Here you are checking that the value of "_newOwner" is valid address or not, which a good thing.

4.1.2 **decreaseApproval Function:-**

```
function decreaseApproval(address _spender, uint _subtractedValue) public returns (bool) {
    uint oldValue = allowed[msg.sender][_spender];
    if (_subtractedValue > oldValue) {
        allowed[msg.sender][_spender] = 0;
    } else {
        allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
    }
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
}
}
```

- In this function, you are checking the old allowance value, and also comparing it with latest value, which is a good thing.

4.1.3 **transferFrom function:-**

```
function transferFrom(address _from, address _to, uint256 _value) public returns (bool) {
    require(_to != address(0));
    require(_value <= balances[_from]);
    require(_value <= allowed[_from][msg.sender]);

    balances[_from] = balances[_from].sub(_value);
    balances[_to] = balances[_to].add(_value);
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
    emit Transfer(_from, _to, _value);
    return true;
}
```

- In this function you are checking allowance and balance of sender before sending the amount which is a good thing.

4.1.4 **transfer Function:-**

```
*/
function transfer(address _to, uint256 _value) public returns (bool) {
    require(_to != address(0));
    require(_value <= balances[msg.sender]);

    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    emit Transfer(msg.sender, _to, _value);
    return true;
}

/**
```

- In this function, you are checking balance of sender before sending the amount, which is good.

4.1.5 addSupport function:-

```
*/
function addSupport(address _who) public onlyOwner {
    require(_who != address(0));
    require(_who != owner);
    require(!supportList[_who]);
    supportList[_who] = true;
    emit SupportAdded(_who);
}
```

- You did good validation in addSupport function.

5. Critical vulnerabilities found in the contract

5.1: File - ImmiCoin.sol

=> No critical vulnerabilities found

5.2: File - ImmiCoinCrowdsaleOpenSale.sol

5.2.1 Syntax error

```
598
599
600
601 contract ImmiCoinCrowdsaleOpenSale is WhitelistedCrowdsale, AllowanceCrowdsale, StateCrowdsale, Excl
602
603     function ImmiCoinCrowdsalePrivateRound(
604         uint256 _exchangeRate,
605         address _wallet,
606         ERC20 _token,
607         address _tokenWallet,
608         uint256 _cap
609     )
610     public
611     Crowdsale(_exchangeRate, _wallet, _token)
612     AllowanceCrowdsale(_tokenWallet)
613     CappedCrowdsale(_cap)
614     {
615     }
616
```

```
342      */
343     function _getTokenAmount(uint256 _weiAmount) internal view returns (
344         // How many token units a buyer gets per cent
345         uint256 rate;
346         rate = (1e6).div(_getCurrentPrice());
347         uint256 usdAmount = weiToUsdCents(_weiAmount);
348         return usdAmount.mul(rate);
349     }
350
351     /**
```

=>We found syntax error at line number #346,#611,#612,#613.

5.3: File - ImmiCoinCrowdsalePre-sale.sol

5.3.1 Syntax error

```
342         // How many token units a buyer gets per cent
343         uint256 rate;
344         rate = (1e6).div(_getCurrentPrice());
345         uint256 usdAmount = weiToUsdCents(_weiAmount);
346         return usdAmount.mul(rate);
347     }
348
349     /**
```

=>We found syntax error at line number #344.

5.4 ImmiCoinCrowdsalePrivateRound.sol

=> No critical vulnerabilities found

6. Medium vulnerabilities found in the contract

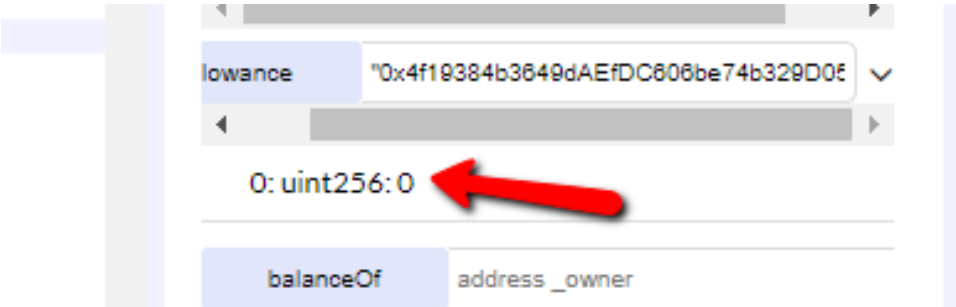
6.1: File - ImmiCoin.sol

6.1.1: Underflow & Overflow attack:

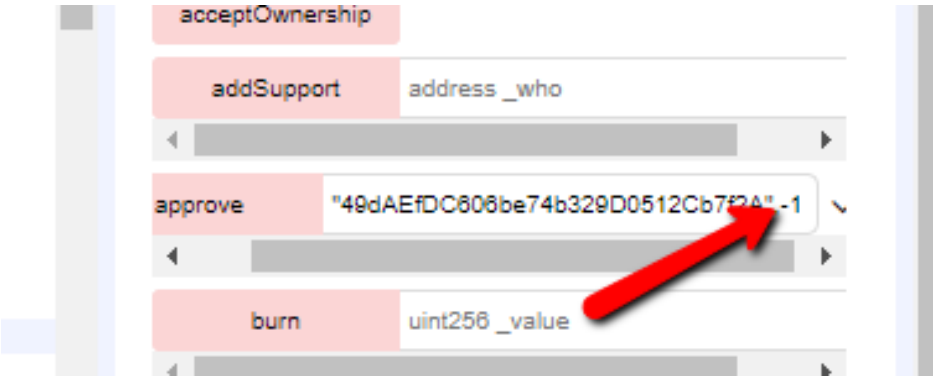
=>In your contract some functions accept negative value.

=>Function name: - approve, increaseApproval, decreaseApproval

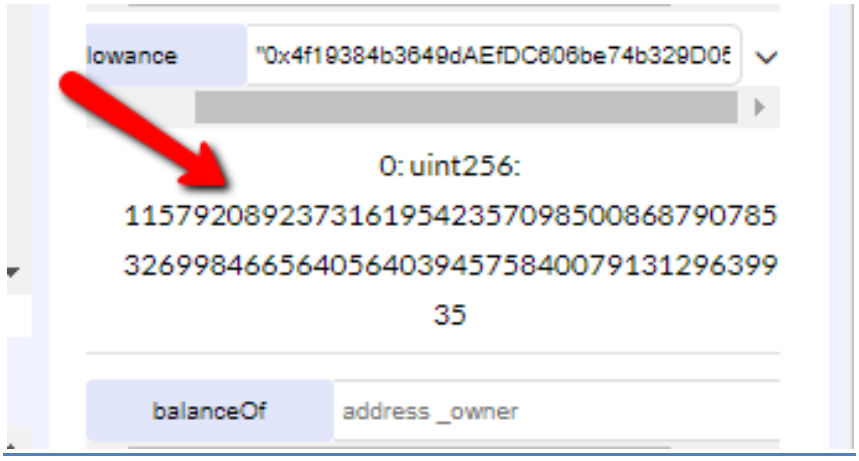
- **Approve**
  - Allowance value in starting.

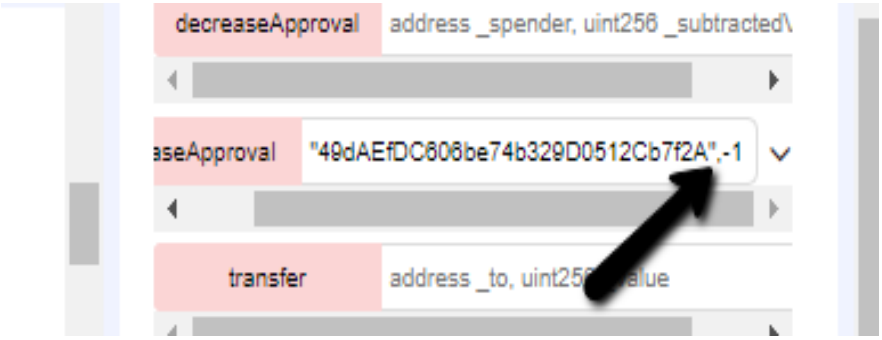


  - Now calling approve function with negative value.

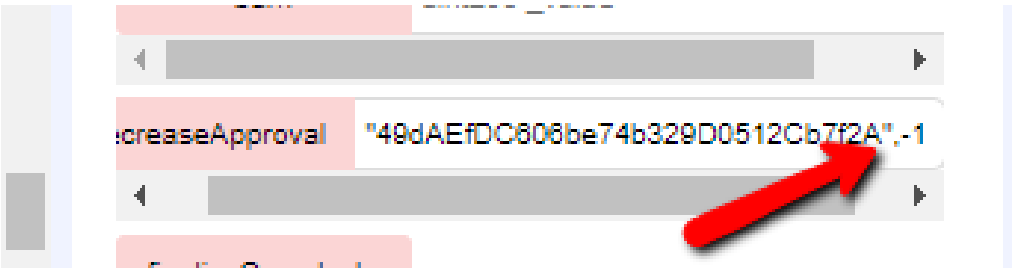


  - Transaction Hash:- https://rinkeby.etherscan.io/tx/0xc37bfdf6244dd729734d5be0052a15c963fb1a8f1335ef788ac2ddc5977333f4.
  - Allowance after negative approves.



- **increaseApproval**
  - 

    - Transaction Hash:-
      - https://rinkeby.etherscan.io/tx/0x93997bbf8621769a031b1228086e184bd7b1fa7f315495a87ecbff829231d72e.

- **decreaseApproval**
  - 

    - Transaction Hash:-
      - https://rinkeby.etherscan.io/tx/0xb38892d60b3de4189f19e83beec0e051b61c0cd6f5ab3787158675aa39319e99.

Solution:-

```
247     */
248     function approve(address _spender, uint256 _value) public returns (bool) {
249         allowed[msg.sender][_spender] = _value;
250         emit Approval(msg.sender, _spender, _value);
251         return true;
252     }
253
254     /**
255      * @dev Function to check the amount of tokens that an owner allowed to a spender.
```

- In approve, increaseApproval and decreaseApproval functions you have to put one condition.

  require(_value <= balances[msg.sender]);

- By this way, user only approves the amount which he has in the balance.

6.1.2: Short address attack

=>In your contract, some functions do not check the value of address variable.

=>Function name: - transferFrom("From"), approve("_spender").

=>Function name: - decreaseApproval ("_spender"), increaseApproval ("_spender").

```
225     */
226     function transferFrom(address _from, address _to, uint256 _value) public returns (bool) {
227         require(_to != address(0));
228         require(_value <= balances[_from]);
229         require(_value <= allowed[_from][msg.sender]);
230
231         balances[_from] = balances[_from].sub(_value);
232         balances[_to] = balances[_to].add(_value);
233         allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
234         emit Transfer(_from, _to, _value);
235         return true;
236     }
237
238     /**
239      * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender
```

- You are not checking the value of "from" variable.
- Anyone can request these function with short address.

Solution:-

- Add only one line in these functions.

- **require(address parameter != address(0));**

6.1.3: Compiler version not fixed

```
pragma solidity ^0.4.21;


/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {

    /**
    * @dev Multiplies two numbers, throws on overflow.
```

=> In this file you have put "pragma solidity ^0.4.21;" 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 ^0.4.21; // bad: compiles w 0.4.21 and above

pragma solidity 0.4.21; // good : compiles w 0.4.21 only

=> If you put (^) symbol then you are able to get compiler version 0.4.21 and above. But if you don't use (^) symbol then you are able to use only 0.4.21 version. And if there is some changes come in compiler and you use old version then some issue may come at deploy time.

=> And try to use latest version of solidity compiler (0.4.24).

6.2: File - ImmiCoinCrowdsaleOpenSale.sol

6.2.1 Compiler version not fixed

```
pragma solidity ^0.4.21;


/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
```

=> As said earlier, please remove the caret symbol (^) and try to use latest version of solidity compiler (0.4.24).

6.3: File - ImmiCoinCrowdsalePre-sale.sol

6.3.1 Compiler version not fixed

```
pragma solidity ^0.4.21;


/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
```

=> Please remove the caret symbol (^) and try to use latest version of solidity compiler (0.4.24).

6.4: ImmiCoinCrowdsalePrivateRound.sol

6.4.1 Compiler version not fixed

```
pragma solidity ^0.4.21;


/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
```

=> Please remove the caret symbol (^) and try to use latest version of solidity compiler (0.4.24).

7. Low severity vulnerabilities found

7.1: File - ImmiCoin.sol

7.1.1 Implicit visibility level

=> This is not a big issue in the solidity. Because if you do not put any visibility, then it will automatically take "public". But it is good practice to specify visibility at every variables and functions.

```
170
171     mapping(address => uint256) balances;
172
173     uint256 totalSupply_;
174
```

Solution:-

1) For # 171.
    a. mapping(address => uint256) public balances;
2) For # 173.
    a. uint256 public totalSupply_;

7.2: File - ImmiCoinCrowdsaleOpenSale.sol

=> No low vulnerabilities found

7.3: File - ImmiCoinCrowdsalePre-sale.sol

7.3.1 Implicit visibility level

```
570
571     uint256 public constant enlargedCap = 6000000000; //60000000
572
573     bool increasedCap = false;
574
575     /**
```

=> Please put public visibility at Line # 573.

bool public increasedCap = false;

7.4: ImmiCoinCrowdsalePrivateRound.sol

7.4.1 Implicit visibility level

```
547     uint256 public constant enlargedCap = 6000000000; //600000
548
549     bool increasedCap = false;
550
551     /**
552      * @dev Constructor, takes maximum amount of cents accepte
```

=> Please put public visibility at Line # 549.

bool public increasedCap = false;

8. Summary of the Audit

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

In the file, ImmiCoinCrowdsalePre-sale.sol, at line number: #438, you can not stop negative value.

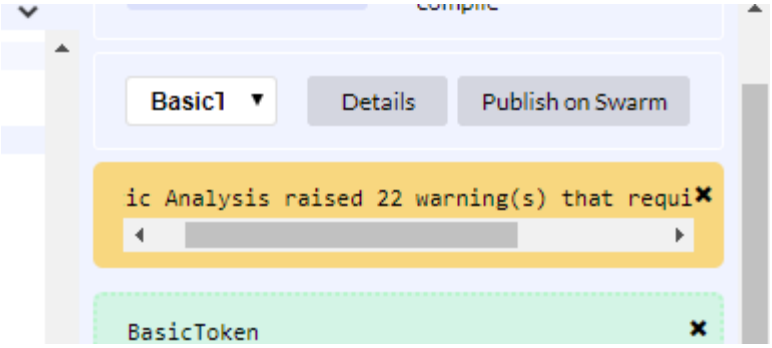
```
435      * @param _exchangeRate USD amout in cents for 1 Ether
436      */
437     function setExchangeRate(uint _exchangeRate) public onlyExchangeRateAgent {
438         require(_exchangeRate > 0);
439
440         exchangeRate = _exchangeRate;
441         emit ExchangeRateChanged(_exchangeRate);
442     }
443
444     /**
```

=> Because all the negative values, coming from user input, will be converted into positive big numbers, so that will pass this require condition

=>Now, this is not vulnerability, as long as that is not affecting your logic in anyway. And you have to take care when you call this function. The same thing at the function updateTotalData at line number #299

```
297      */
298     function updateTotalData (uint256 _usdCents) public onlyOwner {
299         require(_usdCents > 0);
300
301         totalUsdRaised = totalUsdRaised.add(_usdCents);
302     }
303
```

The compiler also displayed 22 warnings, in the file: ImmiCoin.sol



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.

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 ;)).