Contract Audit Report

Prepared for: Ajay Chandhok

Prepared by: Roman Golovay @ ICO Experts Agency

Classification

Defect Severity

• Minor - A defect that does not have a material impact on the contract execution and is likely to be subjective.
• Moderate - A defect that could impact the desired outcome of the contract execution in a specific scenario.
• Major - A defect that impacts the desired outcome of the contract execution or introduces a weakness that may be exploited.
• Critical - A defect that presents a significant security vulnerability or failure of the contract across a range of scenarios.

Audit Summary

Current state of audited smart contract is recognized to be so that it cannot be used for ICO purposes until the following issues are not eliminated:

1. Smart contract does not include about half of required logic (technical requirements).

2. Smart contract includes purchase logic which is not fully encapsulated from the owner`s will. So contributors cannot be sure in getting tokens they paid for.

3. Smart contract has major and critical issues which cannot be ignored.

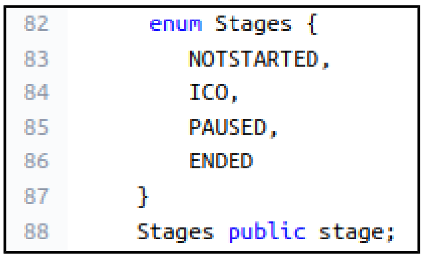
Findings

MINOR

1. Opportunity to simplify

address(0) and *address(this)* can be simplified as *0* and *this.*

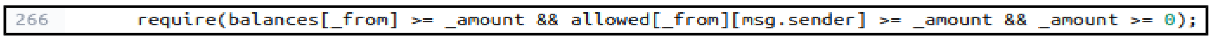
2. Opportunity to reduce GAS usage



```
    enum Stages {
        NOTSTARTED,
        ICO,
        PAUSED,
        ENDED
    }
    Stages public stage;
```

It is an uncommon practice to use enums for stages, but to date it works properly.

3. Redundant verification

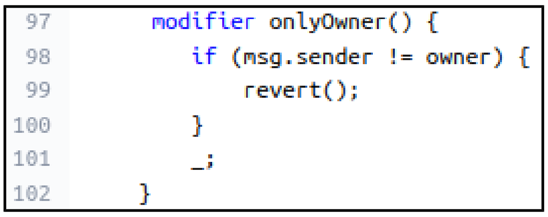


```
        require(balances[_from] >= _amount && allowed[_from][msg.sender] >= _amount && _amount >= 0);
```

require(***balances[_from] >= _amount && allowed[_from][msg.sender] >= _amount &&*** _amount >= 0)

Underlined code is redundant, since it is checked using the SafeMath library.

4. Opportunity to simplify



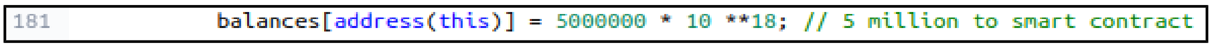
```
    modifier onlyOwner() {
        if (msg.sender != owner) {
            revert();
        }
        _;
    }
```

can be simplified as:

```
modifier onlyOwner() {
     assert (msg.sender != owner)
     _;
 }
```

in other analogous cases are the same.

5. Opportunity to simplify



```
            balances[address(this)] = 5000000 * 10 **18; // 5 million to smart contract
```

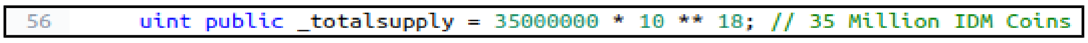
can be simplified as:

```
balances[this] = 5000000 ether;
```

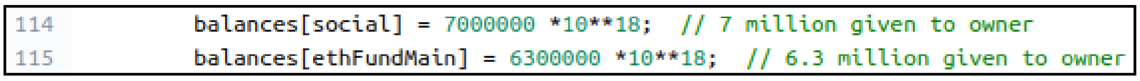
in other analogous cases are the same.

MODERATE

1. Imbalance of total supply and distribution:



```
    uint public _totalsupply = 35000000 * 10 ** 18; // 35 Million IDM Coins
```



```
        balances[social] = 7000000 *10**18;   // 7 million given to owner
        balances[ethFundMain] = 6300000 *10**18;   // 6.3 million given to owner
```

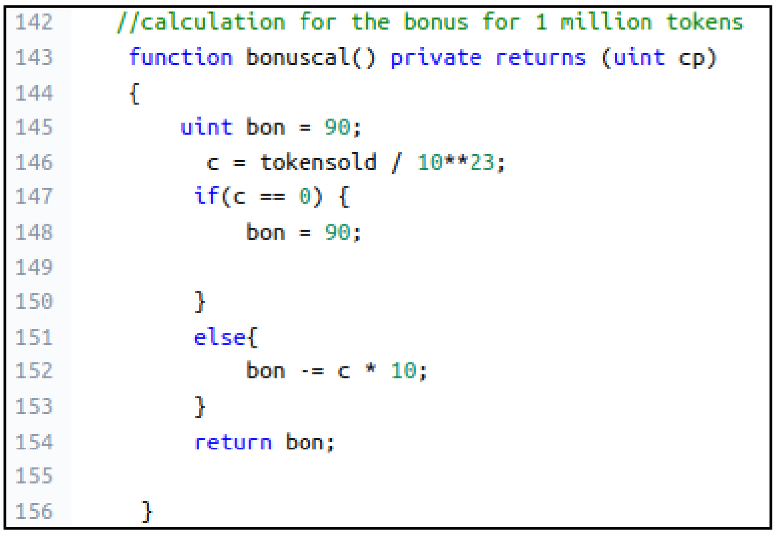
Total supply is 35 000 000 Tokens, but only 13 300 000 Tokens are distributed.

2. Redundant variables and assignments:

line 143: `cp` is unused;

line 146: `c` is global variable, should be local. Also `c` is redundant;

line 148: repeated assignment;



```
142    //calculation for the bonus for 1 million tokens
143     function bonuscal() private returns (uint cp)
144     {
145         uint bon = 90;
146           c = tokensold / 10**23;
147          if(c == 0) {
148              bon = 90;
149
150          }
151          else{
152              bon -= c * 10;
153          }
154          return bon;
155
156      }
```

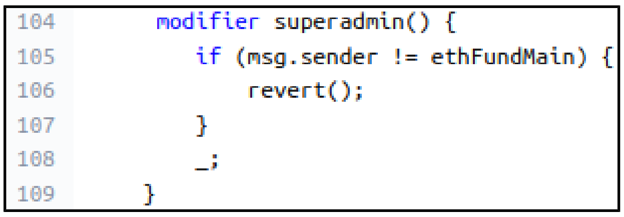
can be simplified:

```
function bonuscal() private returns (uint) {
    uint buffer = tokensold / 10**23;
    if(buffer == 0) {
      return 90;
    }
    return (90 - (buffer * 10));
}
```

or:

```
function bonuscal() private returns (uint) {
    if(tokensold / 10**23 == 0)
    return 90;
    return (90 - (tokensold / 10**22));
}
```

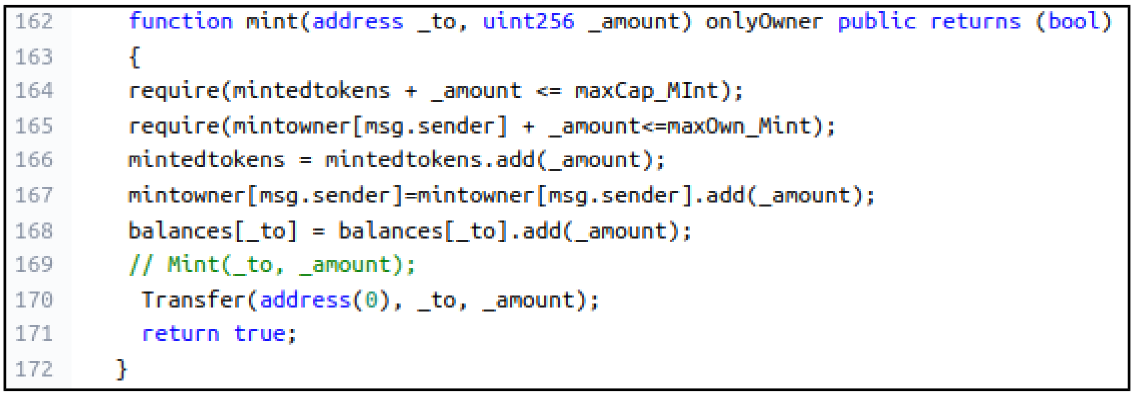
3. Inconsistency with the requirements



```
    modifier superadmin() {
        if (msg.sender != ethFundMain) {
            revert();
        }
        _;
    }
```

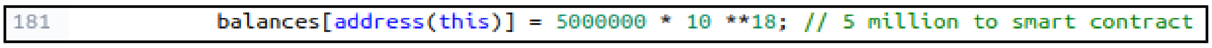
There's no superadmin in technical task.

4. It is hard to check the correctness of this code due to absence of requirements for how it should work within provided documentation



```
    function mint(address _to, uint256 _amount) onlyOwner public returns (bool)
    {
    require(mintedtokens + _amount <= maxCap_MInt);
    require(mintowner[msg.sender] + _amount<=maxOwn_Mint);
    mintedtokens = mintedtokens.add(_amount);
    mintowner[msg.sender]=mintowner[msg.sender].add(_amount);
    balances[_to] = balances[_to].add(_amount);
    // Mint(_to, _amount);
     Transfer(address(0), _to, _amount);
     return true;
   }
```

5. Inconsistency with the requirements

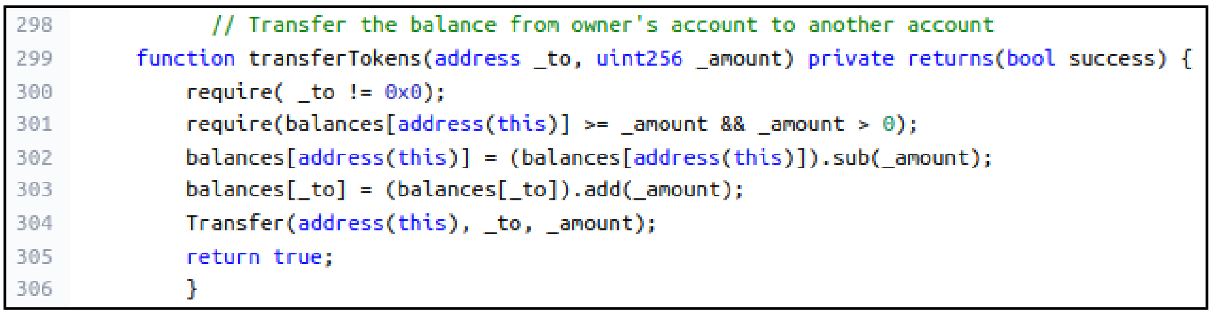


```
            balances[address(this)] = 5000000 * 10 **18; // 5 million to smart contract
```

Condition is not presented in technical task.

MAJOR

1. Unused function



```
        // Transfer the balance from owner's account to another account
    function transferTokens(address _to, uint256 _amount) private returns(bool success) {
        require( _to != 0x0);
        require(balances[address(this)] >= _amount && _amount > 0);
        balances[address(this)] = (balances[address(this)]).sub(_amount);
        balances[_to] = (balances[_to]).add(_amount);
        Transfer(address(this), _to, _amount);
        return true;
        }
```

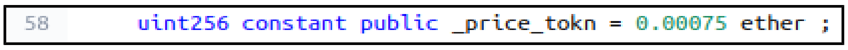
Function is defined as Private, but it is never called.

2. Inconsistency with the requirements

Minimum Cap: ___100,000 IDM

Minimum Cap doesn’t implemented.

3. Inconsistency with the requirements



```
    uint256 constant public _price_tokn = 0.00075 ether ;
```

Token Price in ETH: ___THE EQUIVALENT OF 75 CENTS

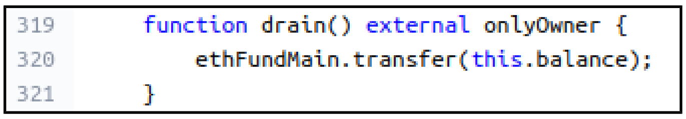
75 cents are not equal to 0.00075 ether. It is recommended to use Oraclize to be able to update currency rate.

4. Inconsistency with the requirements

First Duration: ______starting MARCH 5

This requirement is not implemented in the smart contract.
All functions must be manually called by the owner.

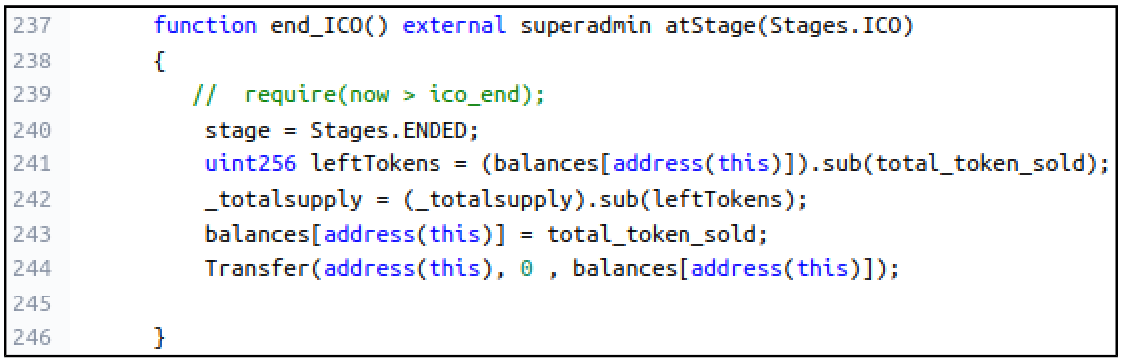
5. Logic flow conflict



```
    function drain() external onlyOwner {
        ethFundMain.transfer(this.balance);
    }
```

This function can break all contract functionality because Owner can transfer all Ether to himself at any time.

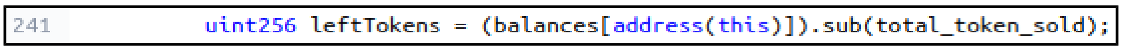
6. Logic flow: exception ready



```
    function end_ICO() external superadmin atStage(Stages.ICO)
    {
      //  require(now > ico_end);
       stage = Stages.ENDED;
       uint256 leftTokens = (balances[address(this)]).sub(total_token_sold);
       _totalsupply = (_totalsupply).sub(leftTokens);
       balances[address(this)] = total_token_sold;
       Transfer(address(this), 0 , balances[address(this)]);

    }
```

It may work incorrectly in a large number of cases and may call exception:

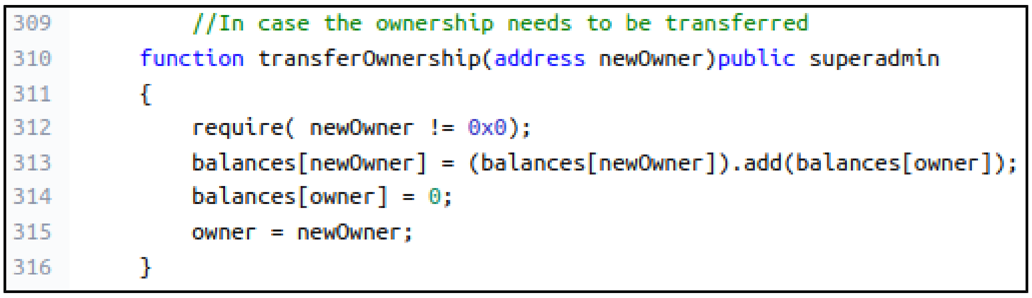


```
       uint256 leftTokens = (balances[address(this)]).sub(total_token_sold);
```

i.e. Having 10 tokens on balance, customer will transfer 6 of them to another account. Then his balance will be equal 4 tokens and according to described logic leftTokens = 4 - 6 will invoke the exception.

CRITICAL

1. Significant security vulnerability or failure of the contract across a range of scenarios



```
        //In case the ownership needs to be transferred
    function transferOwnership(address newOwner)public superadmin
    {
        require( newOwner != 0x0);
        balances[newOwner] = (balances[newOwner]).add(balances[owner]);
        balances[owner] = 0;
        owner = newOwner;
    }
```

Superadmin is able to take away balances of any contributor. Use contract Ownable for identify owner.