
# SMART CONTRACT AUDIT REPORT

## For

## MOT Token (Order # 20June2020 )

**Prepared By**: Chandan Kumar

**Prepared For**: MarketOrders Ltd.

**Prepared on**: 05/07/2020

https://MarketOrders.io

audit@etherauthority.io

# 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 one main and five importable smart contract files:

- MOT.sol (Main)
- SafeMath.sol
- Ownable.sol
- Initializable.sol
- BlackListable.sol
- Pausable.sol

Main contract (MOT.sol) contains approx **587** lines of Solidity code. Other contracts all common standard popular contract codes, for the purpose of Safety and administrative controls, and one of them is initialize, which could be simply a constructor in the main contract, but separate initializable contract is also OK, and worked well in our test run. All the functions and state variables are well commented, logical approach of coding is very neat and clean, and taken care of required security measures, but it had some simple compiling errors specially the fractional part of the assigned digit to the variable on some places, which was the reason of failing with any compiler version. so it was modified to compile successfully, and deployed by the auditor to attempt advance test for audit.

After modification the contract compiled successfully with 0.5.10 solidity version in truffle, as well as in remix IDE. To compile in remix IDE all 6 contract files are merged into one, and the same is deployed on Rinkeby Test network successfully to test further.

https://rinkeby.etherscan.io/address/0x3a19404fa6157846l3f5d72f5401a34d4d513087#code

Changes:

- All files merged into one for better readability and compact compiled output.
- Specified compiler version with which it is compiled to test is 0.5.10.
- Some language/syntax error removed before compile.
- Linked Openzepline source was missing in the given archive, which are downloaded from the original source and merged in source. This was needed to deploy contract in remix IDE.

The audit was performed by two senior solidity auditors at EtherAuthority. The team has extensive work experience in developing and auditing the smart contracts.

This audit procedure also included the use of automated software to further scan of the code to identify potential issues:

For example:

https://tool.smartdec.net/scan/be5fb190ac2a4b9ebe6022e8cde532ee

https://mythx.io tool provided as remix.ethereum.org plug-in.

Some Points are as follows:

1. In line 881 for _airDrop internal functions will always return false ( because return value is no where assigned) , but function part will work well, because this _airDrop is nowhere called under require condition or this return is not assigned anywhere.
2. Costly Loop : All loops are either limited by a number 150, or by the limited length of defined variable, so this is not a problem, the contract will work well.
3. If private key of owner is lost, the contract will fall in critical zone, and will be out of control.
4. Private variable will not be the guarantee of secret or hidden.
5. airDropSingle is public which anyone can call, if it is part of plan then OK.
6. saleMode, and birthDate , visibility not specified.

**NOTE :** Decimal removed from contract to pass in compile, so its numerical value should be checked and modified with proper value as per plan of project, before moving to production.

## Quick Stats:

| Main Category | Subcategory | Result |
|---|---|---|
| Contract Programming | Solidity version not specified | Passed |
| | Solidity version is old | Passed |
| | Integer overflow/underflow | Passed |
| | Function input parameters lack of check | Passed |
| | Function input parameters check bypass | Passed |
| | Function access control lacks management | Passed |
| | Critical operation lacks event log | Passed |
| | Human/contract checks bypass | Passed |
| | Random number generation/use vulnerability | N/A |
| | Fallback function misuse | Passed |
| | Race condition | Passed |
| | Logical vulnerability | Passed |
| | Other programming issues | Moderated |
| Code Specification | Visibility not explicitly declared | Moderated |
| | Var. storage location not explicitly declared | Passed |
| | Use keywords/functions to be deprecated | Passed |

| | | |
|---|---|---|
| | Other code specification issues | Passed |
| Gas Optimization | Assert() misuse | Passed |
| | High consumption ‘for/while’ loop | Passed |
| | High consumption ‘storage’ storage | Passed |
| | “Out of Gas” Attack | Passed |
| Business Risk | The maximum limit for mintage not set | N/A |
| | “Short Address” Attack | Passed |
| | “Double Spend” Attack | Passed |

**Overall Audit Result: PASSED**

# 3. Attacks tested on the contract

In order to check for the security of the contract, we tested several attacks on the code. Some of those are as below:

## 3.1: Over and under flows

SafeMath library is used in the contract, which prevented the possibility of overflow and underflow attacks.

## 3.2: Short address attack

Although this contract **is not vulnerable** to this attack, it is highly recommended to call functions after checking the validity of the address from the outside client.

## 3.3: Visibility & Delegatecall

Delegatecall is not used in the contract thus it does not have this vulnerability. And visibility is also used properly.

## 3.4: Reentrancy / TheDAO hack

Use of "require" function and Checks-Effects-Interactions pattern in this smart contract mitigated this vulnerability.

## 3.5: Forcing ether to a contract

Here, the Smart Contract's balance has never been used as guard, which mitigated this vulnerability

## 3.6: Denial Of Service (DoS)

There **is No** any process consuming loops in the contracts which can be used for DoS attacks. and thus this contract is not prone to DoS.

# 4. Good things in the smart contract

### 4.1 Checks-Effects-Interactions pattern

While transferring tokens, this contract does all the process first and then transfers them. The same while doing other process too. This is very good practice which prevents malicious possibility. For example: transfer() function.

### 4.2 Functions input parameters passed

The functions in this contract verifies the validity of the input parameters, and this validation cannot be by-passed in anyway.

### 4.3 Conditions validations

```
function transfer(address _to, uint256 _value) public returns (bool success) {
    require(!internalWallet[msg.sender],"invalid sender its internal wallet");
    //no need to check for input validations, as that is ruled by SafeMath
    _transfer(msg.sender, _to, _value);

    return true;
}
```

Although the validation of input parameters are not done to prevent overflow and underflow of integers, but the use of SafeMath library is effective here and which has prevented this problem. All Good!

# 5. Critical vulnerabilities found in the contract

Critical issues that could damage heavily the integrity of the contract. Some bug that would allow attackers to steal ether is a critical issue.

**=> No such critical vulnerabilities found. Good Job Team!**

# 6. Medium severity vulnerabilities found

Those vulnerabilities that could damage the contract but with some kind of limitations. Like a bug allowing people to modify a random variable.

**=> No such vulnerabilities found.**

# 7. Low severity vulnerabilities found

Those do not damage the contract, but better to resolve and make code clean.

### 7.1: init function misuse

If owner calls init function by mistake more than once, then the entire schedule will be changed. We understand the owner will never do this. But just a rare case of human error.

# 8. Very low severity vulnerabilities found

The presence of these things does not make any negative effect. But just to clean up the code.

### 8.1: No explicit visibility

Visibility is not specified at some storage variable like birthDate, saleMode etc. Please note that this is not a big issue. But it's suggested to explicitly define visibility to avoid confusion.

### 8.2: Simplify constructor

Although it does not raise any problems, the way constructor function was defined via initialize, can be replaced with simple constructor.

### 8.3: Unused variable

This reserved space for future update, in initializable contract. This is not used anywhere.

```
  // Reserved storage space to allow for layout changes in the future.
  uint256[50] private ______gap;
}
```

Resolution:

Just need to re-consider the above reserved space. If it has no purpose should be removed.

### 8.4: Non-initialized return value in contract

_ airDrop () function doesn't initialize return value. As a result, default value will be returned, which will be false always.

We checked. It does not have any damaging effect. But just to make the code clean, it's better to return true if the function call was a success.

## 9. Gas Optimization Discussion

**=> The Contract is most optimum for the gas cost. There is loop, but that has limited iterations so that is good.**

## 10. Discussions and improvements

### 10.1 approve() of ERC20 Standard

To prevent attack vectors regarding approve() like the one described here: https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit , clients SHOULD make sure to create user interfaces in such a way that they set the allowance first to 0 before setting it to another value for the same spender. THOUGH the contract itself shouldn't enforce it, to allow backwards compatibility with contracts deployed before

### 10.2: Compiler version should be considered while deployment

The contract has defined range of solidity compiler. There is no problem with that, but it will allow to deploy code in any compiler version in that range.

So, it is good practice to deploy the contract having latest solidity version. The solidity version at a time of audit is: 0.5.14

# 11. Summary of the Audit

Overall, the code is ERC20 token implementation. The compiler shows a couple of warnings, as below:


Now, we checked that the warnings in purple division, are due to their static analysis, which includes like gas estimations 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.

It is also encouraged to run bug bounty program and let community help to further polish the code to perfection.