

AUDIT REPORT



Style Guide Violations

Not necessary to resolve but increase readability and enforces style guide standards accepted by the global community. An example image from the contracts has been added for each violation but similar violations can be cited in many contracts.

1. Usage of obsolete pragma versions of solidity instead of the latest stable one (0.4.18).

```
pragma solidity ^0.4.11;

import '../token/DataCentre.sol';
import '../Pausable.sol';
```

2. It is a standard practice to use the same pragma version throughout the contracts.

```
pragma solidity ^0.4.11;

import '../token/DataCentre.sol';
import '../Pausable.sol';
```

```
pragma solidity 0.4.18;

import "../ownership/Ownable.sol";
```

3. Pragma should be locked before deployment in every contract. [\(ref\)](#)
(For example, 0.4.18 is locked whereas ^0.4.18 is not).

4. While declaring a function there should be a space between ")" and "{".

```
public canMint(false, _owner){
```

5. Visibility of all functions should be specified, failing which it is defaulted to 'public'. [\(ref\)](#)

```
function totalSupply() constant returns (uint256);
function balanceOf(address _owner) constant returns (uint256);
function allowance(address _owner, address _spender) constant returns (uint256);
```

6. Top level declarations in solidity should have two blank lines before them. [\(ref\)](#)

```

1  pragma solidity ^0.4.11;
2
3  import './SimpleControl.sol';
4  import './token/Token.sol';
5
6  contract CrowdsaleControl is SimpleControl {
7      using SafeMath for uint;

```

7. The default value for a bool variable is false so there is no need to initialize it. [\(ref\)](#)

```

// not necessary to store in data centre
bool public mintingFinished = false;

```

8. There should be a space between if and (). [\(ref\)](#)

```

if(totalSupply < 150000000e18) {
    return 125;
}
else if(totalSupply < 450000000e18) {
    return 120;
}
else if(totalSupply < 1200000000e18) {
    return 110;
}
else if(totalSupply < 5700000000e18) {
    return 105;
}
else if(totalSupply < 11700000000e18) {
    return 103;
}
else {
    return 100;
}

```

9. Two spaces have been used for indentation in almost all contracts, instead four spaces are recommended per indentation level. [\(ref\)](#)

```

contract WhiteList is Ownable {
    mapping (address => bool) internal whiteListMap;

    function isWhiteListed(address investor) constant returns (bool) {
        return whiteListMap[investor];
    }

    function addWhiteListed(address whiteListAddress) public onlyOwner {
        require(whiteListMap[whiteListAddress] == false);
        whiteListMap[whiteListAddress] = true;
    }

    function removeWhiteListed(address whiteListAddress) public onlyOwner {
        require(whiteListMap[whiteListAddress] == true);
        whiteListMap[whiteListAddress] = false;
    }
}

```

10. Line length should be restricted to 120 characters.

```
function transferFrom(address _owner, address _from, address _to, uint256 _amount, bytes _data) public canMint(false, _owner) {  
    return super.transferFrom(_owner, _from, _to, _amount, _data);  
}
```

11. Throw is deprecated in favour of revert(), require and assert(). It will be removed in future.

```
modifier transactionExists(uint transactionId) {  
    if (transactions[transactionId].destination == 0)  
        throw;  
    _;  
}
```

12. String Literals must be quoted with "double quotes" only. Including imports. [\(ref\)](#)

```
constant returns (uint256) {  
    return _getBalance('TRI', _owner);  
}  
  
returns (uint256) {  
    return _getValue('TRI', 'totalSupply');  
}  
  
(address _spender) constant returns (uint256) {  
    return _getConstraint('TRI', _owner, _spender);  
}  
  
(uint256 _newTotalSupply) internal {  
    return _setValue('TRI', 'totalSupply', _newTotalSupply);  
}  
  
(uint256 _newValue) internal {  
    return _setBalance('TRI', _owner, _newValue);  
}  
  
(uint256 _newValue) internal {  
    return _setConstraint('TRI', _owner, _spender, _newValue);  
}
```

13. Function order is incorrect at multiple instances within the contracts, the correct order is as follows: [\(ref\)](#)

- Constructor
- Fallback function (if exists)
- External
- Public

- Internal
- Private

```

/// @return Confirmation status.
function isConfirmed(uint transactionId)
    public
    constant
    returns (bool)
{
    uint count = 0;
    for (uint i=0; i<owners.length; i++) {
        if (confirmations[transactionId][owner
            count += 1;
        if (count == required)
            return true;
    }
}

/*
 * Internal functions
 */
/// @dev Adds a new transaction to the transac
/// @param destination Transaction target add
/// @param value Transaction ether value.
/// @param data Transaction data payload.
/// @return Returns transaction ID.
function addTransaction(address destination, u
    internal

```

14. There should be a single space before a function. ([ref](#))

```

function mint(address _to, uint256 _amount) when
    setTotalSupply(totalSupply().add(_amount));
    setBalanceOf(_to, balanceOf(_to).add(_amount));
    Token(satellite).mint(_to, _amount);
    return true;
}

```

15. Function with wrong modifiers present within contracts, currently solidity does not enforce modifier types, but it is a wrong practise. For instance the following function is marked constant, though it is

updating a state variable totalSupply.

16. Confusing parameter and function names are present within contracts, same names can shadow a previous declaration, as in the example shown below, isAdmin is both a return type and a function name:

```
function inflateTotalSupply(uint256 newSupply) public constant returns (uint256) {
    totalSupply = newSupply;
    return super.bonusFactor();
}
```

17. Unused parameters are present inside various functions, like in the following example _cap parameter was never used.

```
function FinalizableCrowdsaleImpl (
    uint256 _startTime,
    uint256 _endTime,
    uint256 _rate,
    address _wallet,
    address controller,
    uint256 _cap
)
public
Crowdsale(_startTime, _endTime, _rate, _wallet, controller)
FinalizableCrowdsale()
{
}
```

Automated Testing:

Mythril:

Mythril displays the following two concerns for the audited contracts:

1. A possible Integer underflow exists in function transferFrom.

```
===== Integer Underflow =====
Type: Warning
Contract: MAIN
Function name: transferFrom(address,address,uint256,bytes)
PC address: 2060
A possible integer underflow exists in the function transferFrom(address,address,uint256,bytes).
The SUB instruction at address 2060 may result in a value < 0.
```

```

++++ Debugging info ++++
(32) - (calldata_MAIN_4 + calldata_MAIN_100 +
32 +
164 +
96 +
32*UDiv(calldata_MAIN_4 + calldata_MAIN_100 + 31, 32) +
32).]

```

2. A possible Integer underflow exists in function transfer.

```

===== Integer Underflow =====
Type: Warning
Contract: MAIN
Function name: transfer(address,uint256,bytes)
PC address: 2424
A possible integer underflow exists in the function transfer(address,uint256,bytes).
The SUB instruction at address 2424 may result in a value < 0.
-----
++++ Debugging info ++++
(32) - (calldata_MAIN_4 + calldata_MAIN_68 +
32 +
32 +
32 +
32 +
32 +
4 +
96 +
32*UDiv(calldata_MAIN_4 + calldata_MAIN_68 + 31, 32) +
32).]

```


Securify:



1. Force.sol:


Security Report






Transaction Reordering


Transactions May Affect Ether Receiver




[info](#)


Transactions May Affects Ether Amount




[info](#)




Recursive Calls


Gas-dependent Reentrancy


[info](#)


Reentrancy With Constant Gas

[info](#)


Reentrant Method Call

[info](#)

 techracers

USA



Insecure Coding Patterns

- 🔒 **Unchecked Transaction Data Length** [info](#)
- ✅ **Unhandled Exception** 👍🗨 [info](#)
- ✅ **Use of Origin Instruction** 👍🗨 [info](#)
- 🔒 **Missing Input Validation** [info](#)

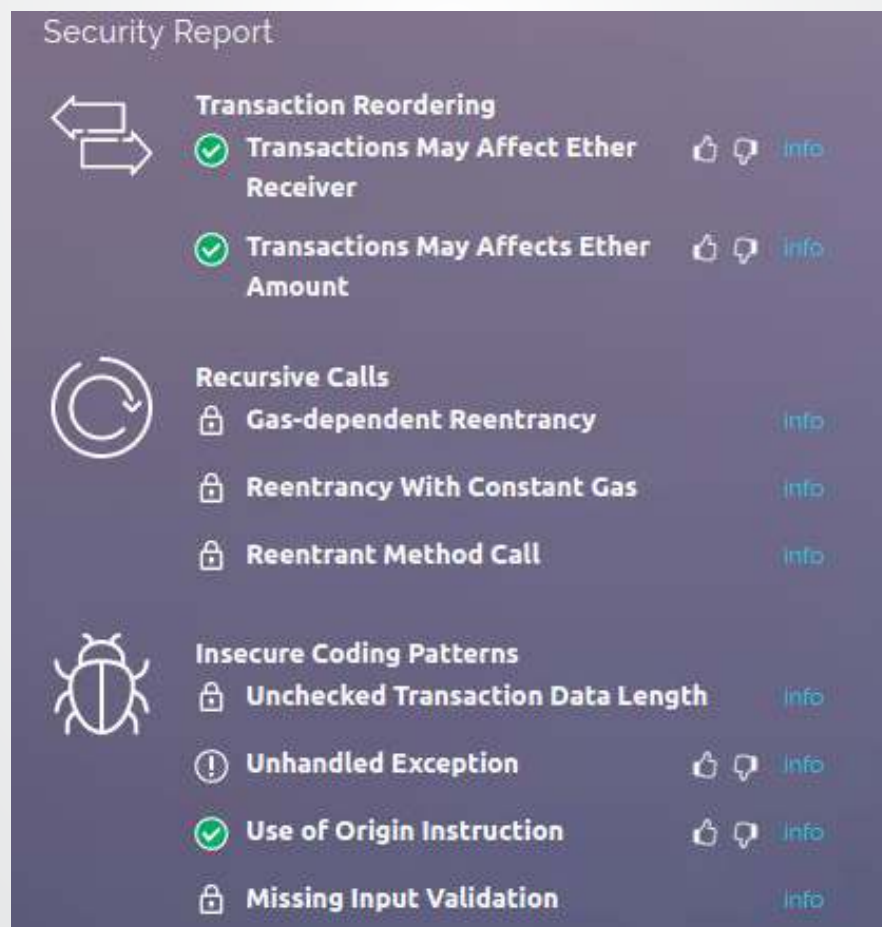
Unexpected Ether Flows

- 🔒 **Locked Ether** [info](#)

Use of Untrusted Inputs in Security Operations

- ✅ **Use of Untrusted Input** 👍🗨 [info](#)

2. TriController.sol:



Security Report

Transaction Reordering

- ✅ **Transactions May Affect Ether Receiver** 👍🗨 [info](#)
- ✅ **Transactions May Affects Ether Amount** 👍🗨 [info](#)

Recursive Calls

- 🔒 **Gas-dependent Reentrancy** [info](#)
- 🔒 **Reentrancy With Constant Gas** [info](#)
- 🔒 **Reentrant Method Call** [info](#)

Insecure Coding Patterns

- 🔒 **Unchecked Transaction Data Length** [info](#)
- ⚠️ **Unhandled Exception** 👍🗨 [info](#)
- ✅ **Use of Origin Instruction** 👍🗨 [info](#)
- 🔒 **Missing Input Validation** [info](#)



Unexpected Ether Flows
 Locked Ether [info](#)

Use of Untrusted Inputs in Security Operations
 Use of Untrusted Input [info](#)

3. TriForceNetworkCrowdsale.sol:

Security Report

Transaction Reordering

- Transactions May Affect Ether Receiver [info](#)
 Matched lines: [L1660](#)
- Transactions May Affects Ether Amount [info](#)
 Matched lines: [L1660](#)

Recursive Calls

- Gas-dependent Reentrancy [info](#)
- Reentrancy With Constant Gas [info](#)
- Reentrant Method Call [info](#)

Insecure Coding Patterns

- Unchecked Transaction Data Length [info](#)
- Unhandled Exception [info](#)
- Use of Origin Instruction [info](#)
- Missing Input Validation [info](#)

Unexpected Ether Flows
 Locked Ether [info](#)

Use of Untrusted Inputs in Security Operations
 Use of Untrusted Input [info](#)

Oyente:

1. WhiteList.sol:

```
INFO:root:Contract /home/rails/work/audit/TriForceNetworkCrowdsale.sol:WhiteList:
INFO:symExec:Running, please wait...
INFO:symExec:  ===== Results =====
INFO:symExec:    EVM code coverage:      99.8%
INFO:symExec:    Callstack bug:          False
INFO:symExec:    Money concurrency bug:   False
INFO:symExec:    Time dependency bug:    False
INFO:symExec:    Reentrancy bug:         False
INFO:symExec:    Assertion failure:      False
```

2. TriForceNetworkCrowdsale.sol:

```
INFO:root:Contract /home/rails/work/audit/TriForceNetworkCrowdsale.sol:TriForceNetworkCrowdsale:
INFO:symExec:Running, please wait...
INFO:symExec:  ===== Results =====
INFO:symExec:    EVM code coverage:      79.3%
INFO:symExec:    Callstack bug:          False
INFO:symExec:    Money concurrency bug:   False
INFO:symExec:    Time dependency bug:    False
INFO:symExec:    Reentrancy bug:         False
INFO:symExec:    Assertion failure:      False
```

3. RefundVault.sol:

```
INFO:root:Contract /home/rails/work/audit/TriForceNetworkCrowdsale.sol:RefundVault:
INFO:symExec:Running, please wait...
INFO:symExec:  ===== Results =====
INFO:symExec:    EVM code coverage:      98.7%
INFO:symExec:    Callstack bug:          False
INFO:symExec:    Money concurrency bug:   True
Flow 1:
/home/rails/work/audit/TriForceNetworkCrowdsale.sol:RefundVault:305:13
wallet.call.gas(2000).value(this.balance)()
^
Flow 2:
/home/rails/work/audit/TriForceNetworkCrowdsale.sol:RefundVault:318:5
investor.transfer(depositedValue)
^
INFO:symExec:    Time dependency bug:    False
INFO:symExec:    Reentrancy bug:         False
INFO:symExec:    Assertion failure:      False
```

4. Force.sol:

```
INFO:root:Contract /home/rails/work/audit/Force.sol:Force:
INFO:symExec:Running, please wait...
INFO:symExec:  ===== Results =====
INFO:symExec:    EVM code coverage:      93.1%
INFO:symExec:    Callstack bug:          False
INFO:symExec:    Money concurrency bug:   False
INFO:symExec:    Time dependency bug:    False
INFO:symExec:    Reentrancy bug:         False
INFO:symExec:    Assertion failure:      False
```


5. Token.sol:

```
INFO:root:Contract /home/rails/work/audit/TriController.sol:Token:
INFO:symExec:Running, please wait...
INFO:symExec:  ===== Results =====
INFO:symExec:    EVM code coverage:      92.1%
INFO:symExec:    Callstack bug:             False
INFO:symExec:    Money concurrency bug:     False
INFO:symExec:    Time dependency bug:      False
INFO:symExec:    Reentrancy bug:           False
INFO:symExec:    Assertion failure:         False
```

6. SimpleControl.sol:

```
INFO:root:Contract /home/rails/work/audit/TriController.sol:SimpleControl:
INFO:symExec:Running, please wait...
INFO:symExec:  ===== Results =====
INFO:symExec:    EVM code coverage:      58.8%
INFO:symExec:    Callstack bug:             False
INFO:symExec:    Money concurrency bug:     False
INFO:symExec:    Time dependency bug:      False
INFO:symExec:    Reentrancy bug:           False
INFO:symExec:    Assertion failure:         False
```

7. DataManager.sol:

```
INFO:root:Contract /home/rails/work/audit/TriController.sol:DataManager:
INFO:symExec:Running, please wait...
INFO:symExec:  ===== Results =====
INFO:symExec:    EVM code coverage:      63.1%
INFO:symExec:    Callstack bug:             False
INFO:symExec:    Money concurrency bug:     False
INFO:symExec:    Time dependency bug:      False
INFO:symExec:    Reentrancy bug:           False
INFO:symExec:    Assertion failure:         False
```

8. DataCentre.sol:

```
INFO:root:Contract /home/rails/work/audit/TriController.sol:DataCentre:
INFO:symExec:Running, please wait...
INFO:symExec:  ===== Results =====
INFO:symExec:    EVM code coverage:      99.9%
INFO:symExec:    Callstack bug:             False
INFO:symExec:    Money concurrency bug:     False
INFO:symExec:    Time dependency bug:      False
INFO:symExec:    Reentrancy bug:           False
INFO:symExec:    Assertion failure:         False
```

9. CrowdsaleControl.sol:

```
INFO:root:Contract /home/rails/work/audit/TriController.sol:CrowdsaleControl:
INFO:symExec:Running, please wait...
INFO:symExec:  ===== Results =====
INFO:symExec:    EVM code coverage:      48.2%
INFO:symExec:    Callstack bug:          False
INFO:symExec:    Money concurrency bug:  False
INFO:symExec:    Time dependency bug:    False
INFO:symExec:    Reentrancy bug:         False
INFO:symExec:    Assertion failure:      False
```

10. Controller.sol:

```
INFO:root:Contract /home/rails/work/audit/TriController.sol:Controller:
INFO:symExec:Running, please wait...
INFO:symExec:  ===== Results =====
INFO:symExec:    EVM code coverage:      44.7%
INFO:symExec:    Callstack bug:          False
INFO:symExec:    Money concurrency bug:  False
INFO:symExec:    Time dependency bug:    False
INFO:symExec:    Reentrancy bug:         False
INFO:symExec:    Assertion failure:      False
```

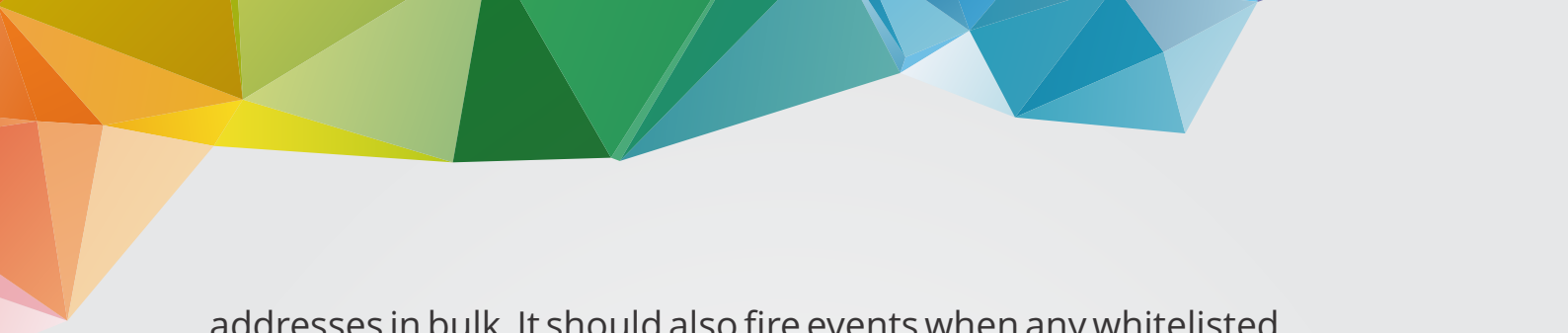
11. TriController.sol:

```
INFO:symExec:    Assertion failure:      False
INFO:root:Contract /home/rails/work/audit/TriController.sol:TriController:
INFO:symExec:Running, please wait...
INFO:symExec:  ===== Results =====
INFO:symExec:    EVM code coverage:      42.7%
INFO:symExec:    Callstack bug:          False
INFO:symExec:    Money concurrency bug:  False
INFO:symExec:    Time dependency bug:    False
INFO:symExec:    Reentrancy bug:         False
INFO:symExec:    Assertion failure:      False
```

Manual Testing:

1. TriForceNetworkCrowdsale.sol

- There is no need for WhitelistedCrowdsale contract to inherit Crowdsale contract. Everything will work fine even without it.
- In whitelist contract which is being used in this crowdsale 2 more functions can be added which will add or remove the whitelisted



addresses in bulk. It should also fire events when any whitelisted address is added or removed so that, later if we want to find the list of all addresses who have been whitelisted then we cannot get this list from a mapping.

- c. An interface of whitelist contract can be included which only has one function isWhitelisted(), this will slightly reduce gas at the time of deployment of TriForceNetworkCrowdsale.sol.
- d. In refundvault contract being used in this crowdsale we can fire an event when deposit function is called.
- c. Selfdestruct functionality can be added in the refund vault contract so that all the ethers in the vault can be transferred to a safe address in case of any bug.

2. TriController.sol

- a. Token contract is being used in this TriController contract. We need to call mint() and mintToggle() functions of Token contract deployed separately. To achieve this we need not to import all the contracts we can create a interface of functions that we are going to use. This will slightly reduce the gas consumed at time of deployment.

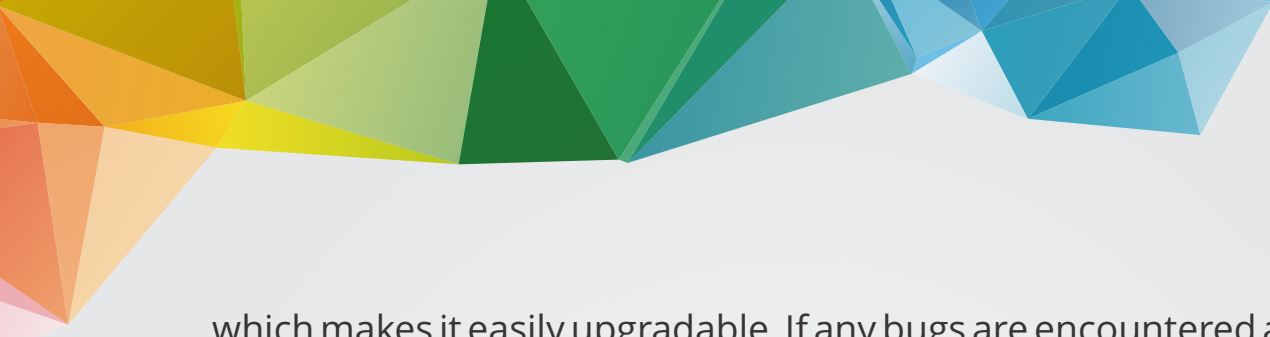
3. Force.sol

Everything looks good. All ERC20 and ERC223 functions working fine

Summary:

All above points are suggestions to optimize the code.

Overall the code looks very good and it is designed in a very good way



which makes it easily upgradable. If any bugs are encountered all the functions can be paused for everyone (except for admins) and then the controller contract be killed and a new controller can be set.