OpenANX Token Peer Review

Author/ Reviewer: Darryl Morris aka o0ragman0o
Date: 21 June 2017

Client: OpenANX, Bok Consulting

Web Presence: https://www.openanx.org/en/
Reference: Technical White paper
Github Repository: https://github.com/openanx/OpenANXToken

Disclosures

The reviewing author has an established non financial, community relationship with Bokky PooBah of Bok Consulting.
The author has no existing financial interests or relationships with the OpenANX Foundation.
The author has no established relationships with anyone in the OpenANX Foundation.

Review Request:

Request to community from *Bok Consulting* through *Ethereum Australia* Telegram channel 20 June 2017:

> *Hi All, if possible, I need some eyes checking my code for the openANX crowdsale starting Thursday, 22-Jun-17 13:00:00 UTC.*
>
> *Code at https://github.com/openanx/OpenANXToken .*
>
> *Bug bounty details at https://medium.com/@OAX_Foundation/openanx-bug-bounty-program-ccc6e981fd6a .*
>
> *Thx*

Scope

This is an, 'eyes over' code review only of the contracts' source code. No static or dynamic testing has been done by the reviewing author.

The contracts reviewed here are those pertaining to the crowd sale token at commit `0d8cb22a3008100c112f46279fcf1d0bc81e9747` in the Github repository.

These six contracts manifest the OAX token as described in chapter 5 of the OpenANX White Paper and consist of:

SafeMath.sol
Owned.sol
ERC20Interface.sol
OpenANXTokenConfig.sol
OpenANXToken.sol
LockedTokens.sol

Overview

All contracts under review have been written for the OpenANX Foundation.

OpenANX is a not for profit foundation with the intent of developing an open source decentralized assets exchange framework which will be able to integrate off chain asset bridges with KYC/AML, recourse and dispute resolution mechanisms where required for any particular asset.

The foundation will be governed by a membership. Membership can be acquired through the redemption (destruction) of OAX tokens for governance rights. This indicates that OAX purchased during the ICO and tradable in the secondary market act as an option to join the OpenANX governing body at a future date.

The governance contracts are yet to be developed and the contracts reviewed here pertain only to the OAX ‘Open Token Sale’ to be launched on 22 June 2012 (ref WP 4.2).

First Impressions

Having a consultant like Bokky PooBar as an advisor and developer in a project carries a great social and technical capital. Bokky is a highly respected developer, well known for his programing experience in asset exchange related industries and strong ‘community positive’ work well demonstrated by his extensive in depth posts to the Ethereum Stack Exchange website.

The Solidity code as written for OpenANX is of excellent readability and demonstrates its author/s have a deep understanding of and experience with the Solidity language and paradigms required for writing EVM smart contracts.

The OpenANX whitepaper demonstrates that the team behind it have a deep understanding of the issues and problems pertaining to both centralised and decentralised exchanges.

As this is a code review and not a project review, the author has not read too deeply into the whitepaper. However, it is concerning that the governance model and membership benefits does not appear to be properly described.

Contracts

SafeMath.sol

SafeMath.sol

```
pragma solidity ^0.4.11;

// ----------------------------------------------------------------------------
// OAX 'openANX Token' crowdfunding contract
//
// Refer to http://openanx.org/ for further information.
//
// Enjoy. (c) openANX and BokkyPooBah / Bok Consulting Pty Ltd 2017.
// The MIT Licence.
// ----------------------------------------------------------------------------


// ----------------------------------------------------------------------------
// Safe maths, borrowed from OpenZeppelin
// ----------------------------------------------------------------------------
library SafeMath {

    // ----------------------------------------------------------------------------
    // Add a number to another number, checking for overflows
    // ----------------------------------------------------------------------------
    function add(uint a, uint b) internal returns (uint) {
        uint c = a + b;
        assert(c >= a && c >= b);
        return c;
    }

    // ----------------------------------------------------------------------------
    // Subtract a number from another number, checking for underflows
    // ----------------------------------------------------------------------------
    function sub(uint a, uint b) internal returns (uint) {
        assert(b <= a);
        return a - b;
    }
}
```

This is a library implementation of the well known and trusted Safe Math addition and subtraction algorithms. These functions are used to protect against overflow and underflow which may happen in finite magnitude arithmetic typical of computers. The class of error protected against are the inherent modulo operations upon unsigned integers, for example of an 8 bit number (uint8):

```
1 + 255 == 256 mod 256 == 0 != 256
1 - 2 == -1 mod 256 == 255 != -1
```

Overflow In the `add(uint a, uint b)` function is caught by `assert(c >= a && c >= b);` which is a known and trusted test. It is worth noting that in such overflows in `c` will always be less `a` or `b` and that only comparison against one operand is required in the form of `assert(c >= a)`. However as such a single comparison has lead to readability confusion it is probably better to leave it as is.

The choice to implement these functions as a library allows for use of the `using for` keywords. This pattern allows the attaching of methods against primitive types.

Library functions which are modified as internal, such as the ones in this contract have their bytecode compiled inline with the contract bytecode and thereby act as if to be an inline function of the the contract itself. This is in contrast to external or public library functions which are deployed 'contract like' and linked to by a utilising contract through the `DELEGATECALL` opcode.

Linking to simple external library functions can create more bytecode than if those functions were internal. The SafeMaths functions here are simple enough that they warrant internal usage as it is the more efficient option.

Owned.sol

Owned.sol

```
pragma solidity ^0.4.11;

// ----------------------------------------------------------------------------
// OAX 'openANX Token' crowdfunding contract - Owned contracts
//
// Refer to http://openanx.org/ for further information.
//
// Enjoy. (c) openANX and BokkyPooBah / Bok Consulting Pty Ltd 2017.
// The MIT Licence.
// ----------------------------------------------------------------------------

// ----------------------------------------------------------------------------
// Owned contract
// ----------------------------------------------------------------------------
contract Owned {

    // ------------------------------------------------------------------------
    // Current owner, and proposed new owner
    // ------------------------------------------------------------------------
    address public owner;
    address public newOwner;

    // ------------------------------------------------------------------------
    // Constructor - assign creator as the owner
    // ------------------------------------------------------------------------
    function Owned() {
        owner = msg.sender;
    }
```

```
    // ----------------------------------------------------------------------
    // Modifier to mark that a function can only be executed by the owner
    // ----------------------------------------------------------------------
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }


    // ----------------------------------------------------------------------
    // Owner can initiate transfer of contract to a new owner
    // ----------------------------------------------------------------------
    function transferOwnership(address _newOwner) onlyOwner {
        newOwner = _newOwner;
    }


    // ----------------------------------------------------------------------
    // New owner has to accept transfer of contract
    // ----------------------------------------------------------------------
    function acceptOwnership() {
        require(msg.sender == newOwner);
        OwnershipTransferred(owner, newOwner);
        owner = newOwner;
    }
    event OwnershipTransferred(address indexed _from, address indexed _to);
}
```

`Owned.sol` is a contract to award deriving contracts the property of ownership by a single address. This is a common, well known and simple practice. Owned contracts typically utilise the `onlyOwner` modifier to permission access to certain functions.
Of note in this implementation is the requirement of the potential owner to accept ownership before ownership is transfer. This protects against potential loss of control due to an incorrect transfer address.

Well implemented. No issues found.

ERC20Interface.sol

ERC20Interface.sol

```
pragma solidity ^0.4.11;

// ----------------------------------------------------------------------
// OAX 'openANX Token' crowdfunding contract - ERC20 Token Interface
//
// Refer to http://openanx.org/ for further information.
//
```

```
// Enjoy. (c) openANX and BokkyPooBah / Bok Consulting Pty Ltd 2017.
// The MIT Licence.
// ----------------------------------------------------------------------------


// ----------------------------------------------------------------------------
// ERC Token Standard #20 Interface
// https://github.com/ethereum/EIPs/issues/20
// ----------------------------------------------------------------------------
contract ERC20Interface {
    uint public totalSupply;
    function balanceOf(address _owner) constant returns (uint balance);
    function transfer(address _to, uint _value) returns (bool success);
    function transferFrom(address _from, address _to, uint _value)
        returns (bool success);
    function approve(address _spender, uint _value) returns (bool success);
    function allowance(address _owner, address _spender) constant
        returns (uint remaining);
    event Transfer(address indexed _from, address indexed _to, uint _value);
    event Approval(address indexed _owner, address indexed _spender,
        uint _value);
}
```

This is a correct implementation of the well known ERC20 transferable token standard interface.

No issues found.

OpenANXTokenConfig.sol

OpenANXTokenConfig.sol

```
pragma solidity ^0.4.11;

// ----------------------------------------------------------------------------
// OAX 'openANX Token' crowdfunding contract - Configuration
//
// Refer to http://openanx.org/ for further information.
//
// Enjoy. (c) openANX and BokkyPooBah / Bok Consulting Pty Ltd 2017.
// The MIT Licence.
// ----------------------------------------------------------------------------

// ----------------------------------------------------------------------------
// openANX crowdsale token smart contract - configuration parameters
// ----------------------------------------------------------------------------
contract OpenANXTokenConfig {

    // ------------------------------------------------------------------------
    // Token symbol(), name() and decimals()
    // ------------------------------------------------------------------------
    string public constant SYMBOL = "OAX";
```

```
    string public constant NAME = "openANX Token";
    uint8 public constant DECIMALS = 18;


    // ------------------------------------------------------------------------
    // Decimal factor for multiplications from OAX unit to OAX natural unit
    // ------------------------------------------------------------------------
    uint public constant DECIMALSFACTOR = 10**uint(DECIMALS);

    // ------------------------------------------------------------------------
    // Tranche 1 soft cap and hard cap, and total tokens
    // ------------------------------------------------------------------------
    uint public constant TOKENS_SOFT_CAP = 13000000 * DECIMALSFACTOR;
    uint public constant TOKENS_HARD_CAP = 30000000 * DECIMALSFACTOR;
    uint public constant TOKENS_TOTAL = 100000000 * DECIMALSFACTOR;

    // ------------------------------------------------------------------------
    // Tranche 1 crowdsale start date and end date
    // Do not use the `now` function here
    // Start - Thursday, 22-Jun-17 13:00:00 UTC / 1pm GMT 22 June 2017
    // End - Saturday, 22-Jul-17 13:00:00 UTC / 1pm GMT 22 July 2017
    // ------------------------------------------------------------------------
    uint public constant START_DATE = 1498136400;
    uint public constant END_DATE = 1500728400;

    // ------------------------------------------------------------------------
    // 1 year and 2 year dates for locked tokens
    // Do not use the `now` function here
    // ------------------------------------------------------------------------
    uint public constant LOCKED_1Y_DATE = START_DATE + 365 days;
    uint public constant LOCKED_2Y_DATE = START_DATE + 2 * 365 days;

    // ------------------------------------------------------------------------
    // Individual transaction contribution min and max amounts
    // Set to 0 to switch off, or `x ether`
    // ------------------------------------------------------------------------
    uint public CONTRIBUTIONS_MIN = 0 ether;
    uint public CONTRIBUTIONS_MAX = 0 ether;
}
```

`OpenANXTokenConfig.sol` is a contract abstracting the configuration data supplied to the token sale contract. It consists of constant modified arithmetic factors and configuration data defining the behaviour of the `OpenANXToken` contract. All data is public modified which creates getter functions of the name of the constant's identifier.

`START_DATE` and `END_DATE` have both been verified as consistent with the date/time provided in the comments using Javascript `Date()` class.

Bug:

```
    uint public CONTRIBUTIONS_MIN = 0 ether;
    uint public CONTRIBUTIONS_MAX = 0 ether;
```

These two constants are missing the `constant` modifier which is against the convention and intent of the code. While this should not present any inconsistency in the logic or interface of the contracts, it must be considered a bug in that it unintentionally and unnecessarily consumes state slots and increases gas cost whenever these variable are called.

To further explain, a `private constant` value is compiled inline with the code. A `public constant` is compiled into a getter function with a minimal addition to the gas cost of a jump/return and the associated stack manipulation that it requires. Reading a state variable (as in the case of this bug) costs 300 gas in order to penalise heavy IO usage.

This bug should to be addressed prior to launch.

LockedTokens.sol

LockedTokens.sol

```
pragma solidity ^0.4.11;

// ----------------------------------------------------------------------------
// OAX 'openANX Token' crowdfunding contract - locked tokens
//
// Refer to http://openanx.org/ for further information.
//
// Enjoy. (c) openANX and BokkyPooBah / Bok Consulting Pty Ltd 2017.
// The MIT Licence.
// ----------------------------------------------------------------------------

import "./ERC20Interface.sol";
import "./SafeMath.sol";
import "./OpenANXTokenConfig.sol";


// ----------------------------------------------------------------------------
// Contract that holds the 1Y and 2Y locked token information
// ----------------------------------------------------------------------------
contract LockedTokens is OpenANXTokenConfig {
    using SafeMath for uint;

    // ------------------------------------------------------------------------
    // 1y and 2y locked totals, not including unsold tranche1 and all tranche2
    // tokens
    // ------------------------------------------------------------------------
    uint public constant TOKENS_LOCKED_1Y_TOTAL = 14000000 * DECIMALSFACTOR;
    uint public constant TOKENS_LOCKED_2Y_TOTAL = 26000000 * DECIMALSFACTOR;

    // ------------------------------------------------------------------------
    // Tokens locked for 1 year for sale 2 in the following account
    // ------------------------------------------------------------------------
    address public TRANCHE2_ACCOUNT = 0xBbBB34FA53A801b5F298744490a1596438bbBe50;

    // ------------------------------------------------------------------------
```

```
    // Current totalSupply of 1y and 2y locked tokens
    // ------------------------------------------------------------------------
    uint public totalSupplyLocked1Y;
    uint public totalSupplyLocked2Y;

    // ------------------------------------------------------------------------
    // Locked tokens mapping
    // ------------------------------------------------------------------------
    mapping (address => uint) public balancesLocked1Y;
    mapping (address => uint) public balancesLocked2Y;

    // ------------------------------------------------------------------------
    // Address of openANX crowdsale token contract
    // ------------------------------------------------------------------------
    ERC20Interface public tokenContract;


    // ------------------------------------------------------------------------
    // Constructor - called by crowdsale token contract
    // ------------------------------------------------------------------------
    function LockedTokens(address _tokenContract) {
        tokenContract = ERC20Interface(_tokenContract);

        // --- 1y locked tokens ---
        // Advisors
        add1Y(0xaBBa43E7594E3B76afB157989e93c6621497FD4b, 2000000 * DECIMALSFACTOR);
        // Directors
        add1Y(0xacCa534c9f62Ab495bd986e002DdF0f054caAE4f, 2000000 * DECIMALSFACTOR);
        // Early backers
        add1Y(0xAddA9B762A00FF12711113bfDc36958B73d7F915, 2000000 * DECIMALSFACTOR);
        // Developers
        add1Y(0xaeEa63B5479B50F79583ec49DACdcf86DDEff392, 8000000 * DECIMALSFACTOR);
        // Confirm 1Y totals
        assert(totalSupplyLocked1Y == TOKENS_LOCKED_1Y_TOTAL);

        // --- 2y locked tokens ---
        // Foundation
        add2Y(0xAAAA9De1E6C564446EBCA0fd102D8Bd92093c756, 20000000 * DECIMALSFACTOR);
        // Advisors
        add2Y(0xaBBa43E7594E3B76afB157989e93c6621497FD4b, 2000000 * DECIMALSFACTOR);
        // Directors
        add2Y(0xacCa534c9f62Ab495bd986e002DdF0f054caAE4f, 2000000 * DECIMALSFACTOR);
        // Early backers
        add2Y(0xAddA9B762A00FF12711113bfDc36958B73d7F915, 2000000 * DECIMALSFACTOR);
        // Confirm 2Y totals
        assert(totalSupplyLocked2Y == TOKENS_LOCKED_2Y_TOTAL);
    }


    // ------------------------------------------------------------------------
    // Add remaining tokens to locked 1y balances
    // ------------------------------------------------------------------------
    function addRemainingTokens() {
        // Only the crowdsale contract can call this function
```

```
        require(msg.sender == address(tokenContract));
        // Total tokens to be created
        uint remainingTokens = TOKENS_TOTAL;
        // Minus precommitments and public crowdsale tokens
        remainingTokens = remainingTokens.sub(tokenContract.totalSupply());
        // Minus 1y locked tokens
        remainingTokens = remainingTokens.sub(totalSupplyLocked1Y);
        // Minus 2y locked tokens
        remainingTokens = remainingTokens.sub(totalSupplyLocked2Y);
        // Unsold tranche1 and tranche2 tokens to be locked for 1y
        add1Y(TRANCHE2_ACCOUNT, remainingTokens);
    }


    // ------------------------------------------------------------------------
    // Add to 1y locked balances and totalSupply
    // ------------------------------------------------------------------------
    function add1Y(address account, uint value) private {
        balancesLocked1Y[account] = balancesLocked1Y[account].add(value);
        totalSupplyLocked1Y = totalSupplyLocked1Y.add(value);
    }


    // ------------------------------------------------------------------------
    // Add to 2y locked balances and totalSupply
    // ------------------------------------------------------------------------
    function add2Y(address account, uint value) private {
        balancesLocked2Y[account] = balancesLocked2Y[account].add(value);
        totalSupplyLocked2Y = totalSupplyLocked2Y.add(value);
    }


    // ------------------------------------------------------------------------
    // 1y locked balances for an account
    // ------------------------------------------------------------------------
    function balanceOfLocked1Y(address account) constant returns (uint balance) {
        return balancesLocked1Y[account];
    }


    // ------------------------------------------------------------------------
    // 2y locked balances for an account
    // ------------------------------------------------------------------------
    function balanceOfLocked2Y(address account) constant returns (uint balance) {
        return balancesLocked2Y[account];
    }


    // ------------------------------------------------------------------------
    // 1y and 2y locked balances for an account
    // ------------------------------------------------------------------------
    function balanceOfLocked(address account) constant returns (uint balance) {
        return balancesLocked1Y[account].add(balancesLocked2Y[account]);
    }
```

```
    // ------------------------------------------------------------------------
    // 1y and 2y locked total supply
    // ------------------------------------------------------------------------
    function totalSupplyLocked() constant returns (uint) {
        return totalSupplyLocked1Y + totalSupplyLocked2Y;
    }


    // ------------------------------------------------------------------------
    // An account can unlock their 1y locked tokens 1y after token launch date
    // ------------------------------------------------------------------------
    function unlock1Y() {
        require(now >= LOCKED_1Y_DATE);
        uint amount = balancesLocked1Y[msg.sender];
        require(amount > 0);
        balancesLocked1Y[msg.sender] = 0;
        totalSupplyLocked1Y = totalSupplyLocked1Y.sub(amount);
        if (!tokenContract.transfer(msg.sender, amount)) throw;
    }


    // ------------------------------------------------------------------------
    // An account can unlock their 2y locked tokens 2y after token launch date
    // ------------------------------------------------------------------------
    function unlock2Y() {
        require(now >= LOCKED_2Y_DATE);
        uint amount = balancesLocked2Y[msg.sender];
        require(amount > 0);
        balancesLocked2Y[msg.sender] = 0;
        totalSupplyLocked2Y = totalSupplyLocked2Y.sub(amount);
        if (!tokenContract.transfer(msg.sender, amount)) throw;
    }
}
```

`LockedTokens.sol` holds a distribution of preassigned tokens to various parties which can be released after the 1st and 2nd year from token launch.

Tokens are assigned in the constructor using multiple calls to private functions `add1y()` and `add2y()`. There is an inefficiency noted here given the multiple read/writes of the state variables `totalSupplyLocked1Y` and `totalSupplyLocked2Y` and the fact that both `add1y()` and `add2y()` remain as artifacts committed to the blockchain without further use.

For the sacrifice of perhaps readability, all the distribution could have been done in the constructor without repetitious state writes by caching running totals to memory before writing and without committing unnecessary bytecode to the blockchain as constructor bytecode is not committed.

Bug:
`address public TRANCHE2_ACCOUNT = 0xBbBB34FA53A801b5F298744490a1596438bbBe50;`

As described earlier for `OpenANXTokenConfig.sol` there is a similar minor bug in not modifying the `TRANCHE2_ACCOUNT` as `constant`. Again this leads to unnecessary state consumption. However upon further inspection and advice from the contract author, the Solidity compiler appear to be treating address literals as runtime variable:

```
ballot.sol:4:48: Warning: Initial value for constant variable has to be compile-time constant. This
    address constant public TRANCHE2_ACCOUNT = 0xBbBB34FA53A801b5F298744490a1596438bbBe50;
                                               ^----------------------------------------^
```

This appears to be counter intuitive behaviour on the part of the compiler itself and advice will be sought from the Ethereum Foundation developers as to the nature of the warning.

The contract compiles with the constant modifier but the contract author has elected to write code which does not trigger compiler errors and warning.

OpenANXToken.sol

[OpenANXToken.sol](#) ERCToken

```
pragma solidity ^0.4.11;

// ----------------------------------------------------------------------------
// OAX 'openANX Token' crowdfunding contract
//
// Refer to http://openanx.org/ for further information.
//
// Enjoy. (c) openANX and BokkyPooBah / Bok Consulting Pty Ltd 2017.
// The MIT Licence.
// ----------------------------------------------------------------------------

import "./ERC20Interface.sol";
import "./Owned.sol";
import "./SafeMath.sol";
import "./OpenANXTokenConfig.sol";
import "./LockedTokens.sol";


// ----------------------------------------------------------------------------
// ERC20 Token, with the addition of symbol, name and decimals
// ----------------------------------------------------------------------------
contract ERC20Token is ERC20Interface, Owned {
    using SafeMath for uint;

    // ------------------------------------------------------------------------
    // symbol(), name() and decimals()
    // ------------------------------------------------------------------------
    string public symbol;
    string public name;
    uint8 public decimals;

    // ------------------------------------------------------------------------
```

```
    // Balances for each account
    // ------------------------------------------------------------------------
    mapping(address => uint) balances;

    // ------------------------------------------------------------------------
    // Owner of account approves the transfer of an amount to another account
    // ------------------------------------------------------------------------
    mapping(address => mapping (address => uint)) allowed;


    // ------------------------------------------------------------------------
    // Constructor
    // ------------------------------------------------------------------------
    function ERC20Token(
        string _symbol,
        string _name,
        uint8 _decimals,
        uint _totalSupply
    ) Owned() {
        symbol = _symbol;
        name = _name;
        decimals = _decimals;
        totalSupply = _totalSupply;
        balances[owner] = _totalSupply;
    }


    // ------------------------------------------------------------------------
    // Get the account balance of another account with address _owner
    // ------------------------------------------------------------------------
    function balanceOf(address _owner) constant returns (uint balance) {
        return balances[_owner];
    }


    // ------------------------------------------------------------------------
    // Transfer the balance from owner's account to another account
    // ------------------------------------------------------------------------
    function transfer(address _to, uint _amount) returns (bool success) {
        if (balances[msg.sender] >= _amount             // User has balance
            && _amount > 0                              // Non-zero transfer
            && balances[_to] + _amount > balances[_to]  // Overflow check
        ) {
            balances[msg.sender] = balances[msg.sender].sub(_amount);
            balances[_to] = balances[_to].add(_amount);
            Transfer(msg.sender, _to, _amount);
            return true;
        } else {
            return false;
        }
    }


    // ------------------------------------------------------------------------
```

```
    // Allow _spender to withdraw from your account, multiple times, up to the
    // _value amount. If this function is called again it overwrites the
    // current allowance with _value.
    // ------------------------------------------------------------------------
    function approve(
        address _spender,
        uint _amount
    ) returns (bool success) {
        allowed[msg.sender][_spender] = _amount;
        Approval(msg.sender, _spender, _amount);
        return true;
    }


    // ------------------------------------------------------------------------
    // Spender of tokens transfer an amount of tokens from the token owner's
    // balance to another account. The owner of the tokens must already
    // have approve(...)-d this transfer
    // ------------------------------------------------------------------------
    function transferFrom(
        address _from,
        address _to,
        uint _amount
    ) returns (bool success) {
        if (balances[_from] >= _amount                  // From a/c has balance
            && allowed[_from][msg.sender] >= _amount    // Transfer approved
            && _amount > 0                              // Non-zero transfer
            && balances[_to] + _amount > balances[_to]  // Overflow check
        ) {
            balances[_from] = balances[_from].sub(_amount);
            allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_amount);
            balances[_to] = balances[_to].add(_amount);
            Transfer(_from, _to, _amount);
            return true;
        } else {
            return false;
        }
    }


    // ------------------------------------------------------------------------
    // Returns the amount of tokens approved by the owner that can be
    // transferred to the spender's account
    // ------------------------------------------------------------------------
    function allowance(
        address _owner,
        address _spender
    ) constant returns (uint remaining) {
        return allowed[_owner][_spender];
    }
}
```

This is an implementation of the the standard ERC20 transferable token which contains an excellent use case demonstration of the the `using for` paradigm which was discussed for the SafeMath.sol library.

While there are no obvious bugs in this contract, it raises a number of points for discussion regarding some weaknesses in the ERC20 standard which can lead to differing behaviour of compliant ERC20 token implementations. As of this writing, these points are under active debate.

A primary question surrounds whether ERC20 should throw on a failed transfer or simply return `false`. A throw is guaranteed to revert upstream side effects but, under Homestead, consumes the entire gas sent with the transaction. While safe in that regard, there is an assumption that a calling contract is not attempting to batch a number of transfers or other operations in the one transaction (though such a practice is undesirable design in a contract). In that instance all prior work in the transaction is defaulted.

In this implementation, that prior work would not automatically be lost, but forces (perhaps rightly) the caller to recognise and manage its own exceptions. Previously, an ERC20 caller could assume that a failed transfer would throw, thus leading now to a requirement to have what should be an unnecessary knowledge of any particular ERC20 implementation being dealt with.

In order to avoid throws and return `false` instead, this contract has vied for `if/then` logic blocks instead of `require()`. This seems like a step backwards as far as Solidity readability is concerned.

An additional point in active debate is whether a 0 value transfer should be treated as an error. This contract treats such as an error but does not throw.

No bugs found.

OpenANXToken.sol OpenANXToken

```
// ----------------------------------------------------------------------------
// openANX crowdsale token smart contract
// ----------------------------------------------------------------------------
contract OpenANXToken is ERC20Token, OpenANXTokenConfig {

    // ------------------------------------------------------------------------
    // Has the crowdsale been finalised?
    // ------------------------------------------------------------------------
    bool public finalised = false;

    // ------------------------------------------------------------------------
    // Number of tokens per 1,000 ETH
    // This can be adjusted as the ETH/USD rate changes
    //
    // Indicative rate of ETH per token of 0.00290923 at 8 June 2017
```

```
    //
    // This is the same as 1 / 0.00290923 = 343.733565238912015 OAX per ETH
    //
    // tokensPerEther  = 343.733565238912015
    // tokensPerKEther = 343,733.565238912015
    // tokensPerKEther = 343,734 rounded to an uint, six significant figures
    // ------------------------------------------------------------------------
    uint public tokensPerKEther = 343734;

    // ------------------------------------------------------------------------
    // Locked Tokens - holds the 1y and 2y locked tokens information
    // ------------------------------------------------------------------------
    LockedTokens public lockedTokens;

    // ------------------------------------------------------------------------
    // Wallet receiving the raised funds
    // ------------------------------------------------------------------------
    address public wallet;

    // ------------------------------------------------------------------------
    // Crowdsale participant's accounts need to be KYC verified KYC before
    // the participant can move their tokens
    // ------------------------------------------------------------------------
    mapping(address => bool) public kycRequired;


    // ------------------------------------------------------------------------
    // Constructor
    // ------------------------------------------------------------------------
    function OpenANXToken(address _wallet)
        ERC20Token(SYMBOL, NAME, DECIMALS, 0)
    {
        wallet = _wallet;
        lockedTokens = new LockedTokens(this);
        require(address(lockedTokens) != 0x0);
    }

    // ------------------------------------------------------------------------
    // openANX can change the crowdsale wallet address
    // Can be set at any time before or during the crowdsale
    // Not relevant after the crowdsale is finalised as no more contributions
    // are accepted
    // ------------------------------------------------------------------------
    function setWallet(address _wallet) onlyOwner {
        wallet = _wallet;
        WalletUpdated(wallet);
    }
    event WalletUpdated(address newWallet);


    // ------------------------------------------------------------------------
    // openANX can set number of tokens per 1,000 ETH
    // Can only be set before the start of the crowdsale
    // ------------------------------------------------------------------------
```

```
    function setTokensPerKEther(uint _tokensPerKEther) onlyOwner {
        require(now < START_DATE);
        require(_tokensPerKEther > 0);
        tokensPerKEther = _tokensPerKEther;
        TokensPerKEtherUpdated(tokensPerKEther);
    }
    event TokensPerKEtherUpdated(uint tokensPerKEther);


    // ------------------------------------------------------------------------
    // Accept ethers to buy tokens during the crowdsale
    // ------------------------------------------------------------------------
    function () payable {
        proxyPayment(msg.sender);
    }


    // ------------------------------------------------------------------------
    // Accept ethers from one account for tokens to be created for another
    // account. Can be used by exchanges to purchase tokens on behalf of
    // it's user
    // ------------------------------------------------------------------------
    function proxyPayment(address participant) payable {
        // No contributions after the crowdsale is finalised
        require(!finalised);

        // No contributions before the start of the crowdsale
        require(now >= START_DATE);
        // No contributions after the end of the crowdsale
        require(now <= END_DATE);

        // No contributions below the minimum (can be 0 ETH)
        require(msg.value >= CONTRIBUTIONS_MIN);
        // No contributions above a maximum (if maximum is set to non-0)
        require(CONTRIBUTIONS_MAX == 0 || msg.value < CONTRIBUTIONS_MAX);

        // Calculate number of tokens for contributed ETH
        // `18` is the ETH decimals
        // `- decimals` is the token decimals
        // `+ 3` for the tokens per 1,000 ETH factor
        uint tokens = msg.value * tokensPerKEther / 10**uint(18 - decimals + 3);

        // Check if the hard cap will be exceeded
        require(totalSupply + tokens <= TOKENS_HARD_CAP);

        // Add tokens purchased to account's balance and total supply
        balances[participant] = balances[participant].add(tokens);
        totalSupply = totalSupply.add(tokens);

        // Log the tokens purchased
        Transfer(0x0, participant, tokens);
        TokensBought(participant, msg.value, this.balance, tokens,
             totalSupply, tokensPerKEther);
```

```
        // KYC verification required before participant can transfer the tokens
        kycRequired[participant] = true;

        // Transfer the contributed ethers to the crowdsale wallet
        if (!wallet.send(msg.value)) throw;
    }
    event TokensBought(address indexed buyer, uint ethers,
        uint newEtherBalance, uint tokens, uint newTotalSupply,
        uint tokensPerKEther);


    // -----------------------------------------------------------------------
    // openANX to finalise the crowdsale - to adding the locked tokens to
    // this contract and the total supply
    // -----------------------------------------------------------------------
    function finalise() onlyOwner {
        // Can only finalise if raised > soft cap or after the end date
        require(totalSupply >= TOKENS_SOFT_CAP || now > END_DATE);

        // Can only finalise once
        require(!finalised);

        // Calculate and add remaining tokens to locked balances
        lockedTokens.addRemainingTokens();

        // Allocate locked and premined tokens
        balances[address(lockedTokens)] = balances[address(lockedTokens)].
            add(lockedTokens.totalSupplyLocked());
        totalSupply = totalSupply.add(lockedTokens.totalSupplyLocked());

        // Can only finalise once
        finalised = true;
    }


    // -----------------------------------------------------------------------
    // openANX to add precommitment funding token balance before the crowdsale
    // commences
    // -----------------------------------------------------------------------
    function addPrecommitment(address participant, uint balance) onlyOwner {
        require(now < START_DATE);
        require(balance > 0);
        balances[participant] = balances[participant].add(balance);
        totalSupply = totalSupply.add(balance);
        Transfer(0x0, participant, balance);
    }
    event PrecommitmentAdded(address indexed participant, uint balance);


    // -----------------------------------------------------------------------
    // Transfer the balance from owner's account to another account, with KYC
    // verification check for the crowdsale participant's first transfer
    // -----------------------------------------------------------------------
    function transfer(address _to, uint _amount) returns (bool success) {
```

```
        // Cannot transfer before crowdsale ends
        require(finalised);
        // Cannot transfer if KYC verification is required
        require(!kycRequired[msg.sender]);
        // Standard transfer
        return super.transfer(_to, _amount);
    }


    // ------------------------------------------------------------------------
    // Spender of tokens transfer an amount of tokens from the token owner's
    // balance to another account, with KYC verification check for the
    // crowdsale participant's first transfer
    // ------------------------------------------------------------------------
    function transferFrom(address _from, address _to, uint _amount)
        returns (bool success)
    {
        // Cannot transfer before crowdsale ends
        require(finalised);
        // Cannot transfer if KYC verification is required
        require(!kycRequired[_from]);
        // Standard transferFrom
        return super.transferFrom(_from, _to, _amount);
    }


    // ------------------------------------------------------------------------
    // openANX to KYC verify the participant's account
    // ------------------------------------------------------------------------
    function kycVerify(address participant) onlyOwner {
        kycRequired[participant] = false;
        KycVerified(participant);
    }
    event KycVerified(address indexed participant);


    // ------------------------------------------------------------------------
    // Any account can burn _from's tokens as long as the _from account has
    // approved the _amount to be burnt using
    //   approve(0x0, _amount)
    // ------------------------------------------------------------------------
    function burnFrom(
        address _from,
        uint _amount
    ) returns (bool success) {
        if (balances[_from] >= _amount                  // From a/c has balance
            && allowed[_from][0x0] >= _amount           // Transfer approved
            && _amount > 0                              // Non-zero transfer
            && balances[0x0] + _amount > balances[0x0]  // Overflow check
        ) {
            balances[_from] = balances[_from].sub(_amount);
            allowed[_from][0x0] = allowed[_from][0x0].sub(_amount);
            balances[0x0] = balances[0x0].add(_amount);
            totalSupply = totalSupply.sub(_amount);
```

```
            Transfer(_from, 0x0, _amount);
            return true;
        } else {
            return false;
        }
    }


    // ------------------------------------------------------------------------
    // 1y locked balances for an account
    // ------------------------------------------------------------------------
    function balanceOfLocked1Y(address account) constant returns (uint balance) {
        return lockedTokens.balanceOfLocked1Y(account);
    }


    // ------------------------------------------------------------------------
    // 2y locked balances for an account
    // ------------------------------------------------------------------------
    function balanceOfLocked2Y(address account) constant returns (uint balance) {
        return lockedTokens.balanceOfLocked2Y(account);
    }


    // ------------------------------------------------------------------------
    // 1y and 2y locked balances for an account
    // ------------------------------------------------------------------------
    function balanceOfLocked(address account) constant returns (uint balance) {
        return lockedTokens.balanceOfLocked(account);
    }


    // ------------------------------------------------------------------------
    // 1y locked total supply
    // ------------------------------------------------------------------------
    function totalSupplyLocked1Y() constant returns (uint) {
        if (finalised) {
            return lockedTokens.totalSupplyLocked1Y();
        } else {
            return 0;
        }
    }


    // ------------------------------------------------------------------------
    // 2y locked total supply
    // ------------------------------------------------------------------------
    function totalSupplyLocked2Y() constant returns (uint) {
        if (finalised) {
            return lockedTokens.totalSupplyLocked2Y();
        } else {
            return 0;
        }
    }
```

```
    // ------------------------------------------------------------------------
    // 1y and 2y locked total supply
    // ------------------------------------------------------------------------
    function totalSupplyLocked() constant returns (uint) {
        if (finalised) {
            return lockedTokens.totalSupplyLocked();
        } else {
            return 0;
        }
    }


    // ------------------------------------------------------------------------
    // Unlocked total supply
    // ------------------------------------------------------------------------
    function totalSupplyUnlocked() constant returns (uint) {
        if (finalised && totalSupply >= lockedTokens.totalSupplyLocked()) {
            return totalSupply.sub(lockedTokens.totalSupplyLocked());
        } else {
            return 0;
        }
    }


    // ------------------------------------------------------------------------
    // openANX can transfer out any accidentally sent ERC20 tokens
    // ------------------------------------------------------------------------
    function transferAnyERC20Token(address tokenAddress, uint amount)
      onlyOwner returns (bool success)
    {
        return ERC20Interface(tokenAddress).transfer(owner, amount);
    }
}
```

This is the main OAX token contract. It derives from `ERC20Token`, `OpenANXTokenConfig` and creates an instance of the `LockedTokens` contract during construction. It manages the sale period and calculates token balances during the sale. As an ERC20 token, it overloads the standard ERC20 contract with its own `transfer()` and `transferFrom()` in order to apply additional KYC permissions.

There is a lot going on in this contract. It involves overloaded ERC20 functionality, token sale managment, transfer of ether payments to an external wallet, token creation, token destruction, KYC management, LockedToken proxy getters and also contains an unrelated catcher for any tokens transferred to the contract itself.

The number of tokens awarded in the sale is pegged to the value of ether in US Dollars on 8 June 2017. This value can be changed by the contract owner prior to the sale.

The default function is payable and routes payments to `proxyPayment(). proxyPayment()` is also payable and contains value exchange and token creation logic.

```
uint tokens = msg.value * tokensPerKEther / 10**uint(18 - decimals + 3);
```

For some non-obvious reason, token calculations are in the magnitude of kiloether which introduces additional magnitude modification math in the divisor of the conversion rate.

The `proxyPayment()` function allows for intermediary payment channels such as exchanges to buy tokens on behalf of holders and it is presumed such intermediaries will be required to properly accommodate this function. It is unknown to the author if any such arrangements have been made between OpenANX and intermediaries for the purpose of the crowdsale.

`proxyPayment()` finalises the token creation process by sending the funds to the fund raising wallet and throwing on a false return as standard practice.

```
    if (!wallet.send(msg.value)) throw;
```

This transfer can be made more readable with the solidity 0.4.11 `address.transfer(value)` function which will also throw upon a false return:

```
    wallet.transfer(msg.value)
```

As in the ERC20Token implementation, the overloading `transfer()` and `transferFrom()` functions are using `if/then` blocks to handle permissioning and parameter validation. While not in itself wrong, it does decrease the readability and raise the mental logic required to properly understand the strings of conditional logic. However, as discussed in the parent contract's review, there is no highly readable alternative like `requires()` which does not throw.

`finalize()`, `kycVerify()` and `addPreCommit()` functions appear sound, though it has been pointed out by another reviewer that the `PrecommitmentAdded()` event is not used and so will not be compiled into bytecode.

Tokens with a non-zero allowance for the `0x0` address can be burned by any caller to the `burnFrom()` function. This is to allow for the burning of tokens during an upgrade of the contract. There is however no further information regarding an upgrade path and without any upgrade demarcation status, it will likely be that any upgrade will not be uniform across all tokens.

The remaining functions are proxy getters for the `lockedTokens` contract and present no issues of concern.

The final function `transferAnyERC20Token()` is an extraneous token recovery function in the event that any ERC20Tokens are sent to the contract. In this case, the tokens are recovered to the contract owner address.

Conclusions

While the contract coding itself is commendable and of high quality with only a few minor bugs identified, numerous questions arise surrounding architectural decisions which do not appear to be answered in the White Paper.

A further concern is the brief period given for public code review (3 days) before the token sale goes live which has prevented a more comprehensive audit of the code.

Initial post sale token transfers require prior KYC verification of the sender and some consideration needs to be given as to how KYC verification is enacted outside the contract. It is not entirely clear why these initial transfers are KYC encumbered though I'm told it may be a regulatory requirement. The KYC requirement will likely present as a disincentive for token purchase. Many investors who may wish to remain anonymous may not realise this encumbrance exists. Such cases could lead to protracted negative publicity toward the project and frustration on behalf of the investor.

Token holding can become anonymous after the first transfer. KYC verification may make more sense when the owner wishes to redeem their OAX tokens for OpenANX governance powers though contract does not provide for this.

The token itself appears to have no intrinsic value apart from being redeemable for governance power with OpenANX. Lacking in intrinsic value and being burdened by KYC buy in requirements leaves a pessimistic view that the sale will be unattractive and the token may not perform well in the market. The 1 and 2 year locked tokens may be effectively worthless once they are unlocked leading to possible shortfalls in current and future funding.

While these opinions are beyond the scope of the code review, I feel they must be addressed and made public prior to the funding launch.