TST TOKEN SMART CONTRACT AUDIT RESULTS

FOR DATA REVOLUTION TECHNOLOGIES PTY LTD

06/27/2018

Made in Germany by chainsulting.de

Change history

Version	Date	Author	Changes
1.0	**20.06.2018**	**Y. Heinze**	**Audit created**
1.5	**24.06.2018**	**Y. Heinze**	**Vulnerability check**
2.0	**27.06.2018**	**Y. Heinze**	**Executive Summary**

Smart Contract Audit TST Token

Table of Contents

1. Disclaimer
2. About the Project and Company
3. Vulnerability Level
4. Overview of the audit
5. Attack made to the contract
6. Executive Summary
7. General Summary
8. Source Code – Smart Contracts

1. Disclaimer

The audit makes no statements or warrantees 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.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Data Revolution Technologies PTY LTD. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

2. About the Project and Company

Company address:

TOUCH Social Pty Ltd. 3/34 Florence St Teneriffe Brisbane QLD 4005 Australia	Data Revolution Technologies Pty Ltd. 240 Queen Street Brisbane QLD 4000 Australia

Company Check: https://abr.business.gov.au/ABN/View?abn=67625452539

Project Overview:

Data Revolution Technologies Pty Ltd, an innovations and software company, under the guidance of co-founders Patrick Heaton, Mark Francis, Tyrone Crook and, Aidan Pennell, has taken up to its name in “changing the way data is used and stored”.

Touch is a young and motivated collective bent on making a revolutionary wallet backed by eCommerce and Web 3.0. We aim to develop and release the Touch Wallet - Your all in one cryptocurrency wallet that allows real world payments via your smart phones NFC. Touch TST Tokens are also used within the Touch Social app and will be the next stage after the wallet development.

3. Vulnerability Level

0-Informational severity – A vulnerability that have informational character but is not effecting any of the code.

1-Low severity - A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

2-Medium severity – A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

3-High severity – A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.

4-Critical severity – A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.

4. Overview of the audit

The TST Token which contains 274 lines of Solidity code. All the functions and state variables are well commented using the natspec documentation for the functions which is good to understand quickly how everything is supposed to work.

Etherscan:

TST Token address:
https://etherscan.io/address/0x5e0af01930c8dc676a6dc7133bd86370a0be3953#code

Used Code from other Smart Contracts

1. SafeMath (Math operations with safety checks that throw on error)
https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/math/SafeMath.sol

2. ERC20Basic (Simpler version of ERC20 interface)
https://github.com/ethereum/EIPs/issues/179
https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/ERC20/ERC20Basic.sol

3. Standard ERC20 (Based on code by FirstBlood)
https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/ERC20/StandardToken.sol

4. MonolithDAO (allowance prove to spend funds)
https://github.com/MonolithDAO/token/blob/master/src/Token.sol

5. Attack made to the contract

Attack: Using the approve function of the ERC-20 standard
The approve function of ERC-20 might lead to vulnerabilities.

```
function approve(address _spender, uint256 _value) public returns(bool) {
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}
```

Severity: 2

Result / Recommendation:
Only use the approve function of the ERC-20 standard to change allowed amount to 0 or from 0 (wait till transaction is mined and approved).

The TST Smart Contract is secure against that attack

```
function approve(address _spender, uint256 _value) public ifUnrestricted onlyPayloadSize(2) returns
(bool sucess) {
                        // Can only approve when value has not already been set or is zero
                        require(allowed[msg.sender][_spender] == 0 || _value == 0);
                        allowed[msg.sender][_spender] = _value;
                        Approval(msg.sender, _spender, _value);
                        return true;
                }
```

https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit

Attack: Unchecked math
Solidity is prone to integer over- and underflow. Overflow leads to unexpected effects and can lead to loss of funds if exploited by a malicious account.

TST Token # 123

```
uint256 public maximumTokenIssue = 1000000000 * 10**18;
```

Severity: 2

Result / Recommendation:
Check against over- and underflow (use the SafeMath library).
The TST Smart Contract is secure against that attack

Attack: Unhandled Exception

A **call/send** instruction returns a non-zero value if an exception occurs during the execution of the instruction (e.g., out-of-gas). A contract must check the return value of these instructions and throw an exception.

Severity: 0

Result / Recommendation:

Catching exceptions is not yet possible.

Attack: Transactions May Affect Ether Receiver

A contract is exposed to this vulnerability if a miner (who executes and validates transactions) can reorder the transactions within a block in a way that affects the receiver of ether.

Severity: 1

Result / Recommendation:

The contract is not vulnerable to this vulnerability as the receiver of ether is **msg.sender**, which cannot be modified by previously executed transactions

TST Token # 194– 200

```
function approve(address _spender, uint256 _value) public ifUnrestricted onlyPayloadSize(2) returns
(bool sucess) {
                        // Can only approve when value has not already been set or is zero
                        require(allowed[msg.sender][_spender] == 0 || _value == 0);
                        allowed[msg.sender][_spender] = _value;
                        Approval(msg.sender, _spender, _value);
                        return true;
                }
```

Attack: Sending tokens (not Ethereum) to a Smart Contract

It can happen that users without any knowledge, can send tokens to that address. A Smart Contract needs to throw that transaction as an exception.

TST Token # 266 – 268

```
function () external payable {
            revert();
        }
```

https://etherscan.io/tx/0xdf8f843067178b51a43213c19f61f5d006245e214ea89007906beb7d4e08ae82

Severity: 1

Result / Recommendation:

The function of sending back tokens that are not whitelisted, is not yet functional. The proposal ERC223 can fix it in the future.

https://github.com/Dexaran/ERC223-token-standard

Attack: **ERC20 Short Address Attack**

this attack occurs when an attacker use an incomplete address which ends with zeros 0,'00'...

if an attacked generate an address like 0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa0000. he can provide 0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa (zeros omitted) to the transfer function This would actually cause a transfer of a value shifted by 16 bits, i.e. 65536 times larger than X, to attacker's Ethereum account.The msg.data would be like following : function transfer(address _to, uint256 _value)
0xa9059cbb0000000000000000000000007973500000000000000000000000000000
00000000000005150ac4c39a6f3f0000

Severity: 1

Result / Recommendation:

TST Token # 24 – 27

```
  modifier onlyPayloadSize(uint numWords) {
       assert(msg.data.length >= numWords * 32 + 4);
       _;
    }
```

The TST Smart Contract is secure against that attack
Source: https://blog.golemproject.net/how-to-find-10m-by-just-reading-blockchain-6ae9d39fcd95

6. Executive Summary

A majority of the code was standard and copied from widely-used and reviewed contracts and as a result, a lot of the code was reviewed before. It correctly implemented widely-used and reviewed contracts for safe mathematical operations. The audit identified no major security vulnerabilities, at the moment of audit.

7. General Summary

The issues identified were minor in nature, and do not affect the security of the contract.

Additionally, the code implements and uses a SafeMath contract, which defines functions for safe math operations that will throw errors in the cases of integer overflow or underflows. The simplicity of the audited contracts contributed greatly to their security. The minimalist approach in choosing which pieces of functionality to implement meant there was very little attack surface available.

We recommended is to Update the etherscan.io information with Logo/Website for example. That gives buyers more transparency.

TST Token
https://etherscan.io/tokenupdate?a=0x5e0af01930c8dc676a6dc7133bd86370a0be3953

8. Source Code – Smart Contracts

TST Token (TST)

```
/**
 * @title Moderated
 * @dev restricts execution of 'onlyModerator' modified functions to the contract moderator
 * @dev restricts execution of 'ifUnrestricted' modified functions to when unrestricted
 *      boolean state is true
 * @dev allows for the extraction of ether or other ERC20 tokens mistakenly sent to this
address
 */
contract Moderated {

    address public moderator;

    bool public unrestricted;

    modifier onlyModerator {
        require(msg.sender == moderator);
        _;
    }

    modifier ifUnrestricted {
        require(unrestricted);
        _;
    }

    modifier onlyPayloadSize(uint numWords) {
        assert(msg.data.length >= numWords * 32 + 4);
        _;
    }

    function Moderated() public {
        moderator = msg.sender;
        unrestricted = true;
    }

    function reassignModerator(address newModerator) public onlyModerator {
        moderator = newModerator;
    }

    function restrict() public onlyModerator {
        unrestricted = false;
    }

    function unrestrict() public onlyModerator {
        unrestricted = true;
    }

    /// This method can be used to extract tokens mistakenly sent to this contract.
    /// @param _token The address of the token contract that you want to recover
    function extract(address _token) public returns (bool) {
        require(_token != address(0x0));
        Token token = Token(_token);
        uint256 balance = token.balanceOf(this);
        return token.transfer(moderator, balance);
    }

    function isContract(address _addr) internal view returns (bool) {
        uint256 size;
        assembly { size := extcodesize(_addr) }
        return (size > 0);
    }
}
```

```
/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
contract Token {

    function totalSupply() public view returns (uint256);
    function balanceOf(address who) public view returns (uint256);
    function transfer(address to, uint256 value) public returns (bool);
    function transferFrom(address from, address to, uint256 value) public returns (bool);
    function approve(address spender, uint256 value) public returns (bool);
    function allowance(address owner, address spender) public view returns (uint256);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

}

/**
 * @title SafeMath
 * @dev Math operations that are safe for uint256 against overflow and negative values
 * @dev https://github.com/OpenZeppelin/zeppelin-
solidity/blob/master/contracts/math/SafeMath.sol
 */

library SafeMath {
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }

  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}

// @dev Assign moderation of contract to CrowdSale

contract Touch is Moderated {
       using SafeMath for uint256;

              string public name = "Touch. Token";
              string public symbol = "TST";
              uint8 public decimals = 18;

        uint256 public maximumTokenIssue = 1000000000 * 10**18;

              mapping(address => uint256) internal balances;
              mapping (address => mapping (address => uint256)) internal allowed;
```

```
	uint256 internal totalSupply_;

	event Approval(address indexed owner, address indexed spender, uint256
value);
	event Transfer(address indexed from, address indexed to, uint256 value);

	/**
	* @dev total number of tokens in existence
	*/
	function totalSupply() public view returns (uint256) {
		return totalSupply_;
	}

	/**
	* @dev transfer token for a specified address
	* @param _to The address to transfer to.
	* @param _value The amount to be transferred.
	*/
	function transfer(address _to, uint256 _value) public ifUnrestricted
onlyPayloadSize(2) returns (bool) {
	    return _transfer(msg.sender, _to, _value);
	}

	/**
	* @dev Transfer tokens from one address to another
	* @param _from address The address which you want to send tokens from
	* @param _to address The address which you want to transfer to
	* @param _value uint256 the amount of tokens to be transferred
	*/
	function transferFrom(address _from, address _to, uint256 _value) public
ifUnrestricted onlyPayloadSize(3) returns (bool) {
	    require(_value <= allowed[_from][msg.sender]);
	    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
	    return _transfer(_from, _to, _value);
	}

	function _transfer(address _from, address _to, uint256 _value) internal
returns (bool) {
		// Do not allow transfers to 0x0 or to this contract
		require(_to != address(0x0) && _to != address(this));
		// Do not allow transfer of value greater than sender's current
balance
		require(_value <= balances[_from]);
		// Update balance of sending address
		balances[_from] = balances[_from].sub(_value);
		// Update balance of receiving address
		balances[_to] = balances[_to].add(_value);
		// An event to make the transfer easy to find on the blockchain
		Transfer(_from, _to, _value);
		return true;
	}

	/**
	* @dev Gets the balance of the specified address.
	* @param _owner The address to query the the balance of.
	* @return An uint256 representing the amount owned by the passed address.
	*/
	function balanceOf(address _owner) public view returns (uint256) {
		return balances[_owner];
	}

	/**
	* @dev Approve the passed address to spend the specified amount of tokens on
behalf of msg.sender.
	*
	* Beware that changing an allowance with this method brings the risk that
someone may use both the old
```

```
             * and the new allowance by unfortunate transaction ordering. One possible
solution to mitigate this
             * race condition is to first reduce the spender's allowance to 0 and set the
desired value afterwards:
             * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
             * @param _spender The address which will spend the funds.
             * @param _value The amount of tokens to be spent.
             */
             function approve(address _spender, uint256 _value) public ifUnrestricted
onlyPayloadSize(2) returns (bool sucess) {
                    // Can only approve when value has not already been set or is zero
                    require(allowed[msg.sender][_spender] == 0 || _value == 0);
                    allowed[msg.sender][_spender] = _value;
                    Approval(msg.sender, _spender, _value);
                    return true;
             }

             /**
             * @dev Function to check the amount of tokens that an owner allowed to a
spender.
             * @param _owner address The address which owns the funds.
             * @param _spender address The address which will spend the funds.
             * @return A uint256 specifying the amount of tokens still available for the
spender.
             */
             function allowance(address _owner, address _spender) public view returns
(uint256) {
                    return allowed[_owner][_spender];
             }

             /**
             * @dev Increase the amount of tokens that an owner allowed to a spender.
             *
             * approve should be called when allowed[_spender] == 0. To increment
             * allowed value is better to use this function to avoid 2 calls (and wait
until
             * the first transaction is mined)
             * From MonolithDAO Token.sol
             * @param _spender The address which will spend the funds.
             * @param _addedValue The amount of tokens to increase the allowance by.
             */
             function increaseApproval(address _spender, uint256 _addedValue) public
ifUnrestricted onlyPayloadSize(2) returns (bool) {
                    require(_addedValue > 0);
                    allowed[msg.sender][_spender] =
allowed[msg.sender][_spender].add(_addedValue);
                    Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
                    return true;
             }

             /**
             * @dev Decrease the amount of tokens that an owner allowed to a spender.
             *
             * approve should be called when allowed[_spender] == 0. To decrement
             * allowed value is better to use this function to avoid 2 calls (and wait
until
             * the first transaction is mined)
             * From MonolithDAO Token.sol
             * @param _spender The address which will spend the funds.
             * @param _subtractedValue The amount of tokens to decrease the allowance by.
             */
             function decreaseApproval(address _spender, uint256 _subtractedValue) public
ifUnrestricted onlyPayloadSize(2) returns (bool) {
                    uint256 oldValue = allowed[msg.sender][_spender];
                    require(_subtractedValue > 0);
                    if (_subtractedValue > oldValue) {
                           allowed[msg.sender][_spender] = 0;
```

```
                } else {
                        allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
                }
                Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
                return true;
        }

        /**
        * @dev Function to mint tokens
        * @param _to The address that will receive the minted tokens.
        * @param _amount The amount of tokens to mint.
        * @return A boolean that indicates if the operation was successful.
        */
        function generateTokens(address _to, uint _amount) internal returns (bool) {
                totalSupply_ = totalSupply_.add(_amount);
                balances[_to] = balances[_to].add(_amount);
                Transfer(address(0x0), _to, _amount);
                return true;
        }
        /**
        * @dev fallback function - reverts transaction
        */
    function () external payable {
        revert();
    }

    function Touch () public {
            generateTokens(msg.sender, maximumTokenIssue);
    }

}
```