Better Betting Token Audit

AUTHOR: MATTHEW DI FERRANTE

2018-01-02

Audited Material Summary

The audit consists of the following file:

```
d262dcddf8ca2c0b6cca8c0cab84f871301baf4f709919d3ca112a7e6c5f397b
   betr_token.sol
```

The contract implements a token with some management functionality.

There are no major security issues with the contract.

betr_token.sol

The `BETR_TOKEN` contract is self contained, and implements an ERC20 Token with some ICO/management functionality, a fee system, and a service forwarder.

```
contract BETR_TOKEN {
    using SafeMath for uint256;
```

SafeMath is used throughout the whole contract.

Constructor

```
    function BETR_TOKEN() public {
        owner = msg.sender;
    }
```

The constructor sets the contract owner to the msg.sender.

Token Functionality

allowEscroe

```
    function allowEscrow(bool _choice) external returns(bool) {
      escrowAllowed[msg.sender] = _choice;
      return true;
    }
```

The `allowEscrow` function allows a caller to set whether an escrow is allowed for its address.

escrowFrom

```
    function escrowFrom(address _from, uint256 _value) external onlyEscrow
        returns(bool) {
      require (
        _from != address(0) &&
        balances[_from] >= _value &&
        escrowAllowed[_from] &&
        _value > 0
      );
      balances[_from] = balances[_from].sub(_value);
      balances[escrow] = balances[escrow].add(_value);
      Transfer(_from, escrow, _value);
      return true;
    }
```

The `escrowFrom` function allows the escrow contract to remove tokens from an escrowable address and transfer them to the `escrow` address.

On success, a `Transfer` event is emitted.

escrowReturn

```
    function escrowReturn(address _to, uint256 _value, uint256 _fee)
       external onlyEscrow returns(bool) {
        require(
            _to != address(0) &&
            _value > 0
        );
        if(_fee > 0) {
            require(_fee < totalSupply && _fee < balances[escrow]);
            totalSupply = totalSupply.sub(_fee);
            balances[escrow] = balances[escrow].sub(_fee);
        }
        require(transfer(_to, _value));
        return true;
    }
```

The `escrowReturn` function allows the escrow contract to transfer tokens from the `escrow`'s balance to a target `_to` balance. If the function is called with a fee greater than 0, then `fee` amounts of tokens are burned from the `escrow` balance and removed from the total supply.

ERC20 Functions

transfer

```
    function transfer(address _to, uint256 _value) public returns (bool) {
        require(
            _to != address(0) &&
            balances[msg.sender] >= _value &&
            balances[_to] + _value > balances[_to]
        );
        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        Transfer(msg.sender, _to, _value);
        return true;
    }
```

The `transfer` function implements the standard ERC20 function with some extra require checks.

transferFrom

```
    function transferFrom(address _from, address _to, uint256 _value)
       public returns (bool) {
        require (
          _from != address(0) &&
          _to != address(0) &&
          balances[_from] >= _value &&
          allowed[_from][msg.sender] >= _value &&
          balances[_to] + _value > balances[_to]
        );
        balances[_from] = balances[_from].sub(_value);
        balances[_to] = balances[_to].add(_value);
        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value
           );
        Transfer(_from, _to, _value);
        return true;
    }
```

The `transferFrom` function implements the standard ERC20 function with some extra require checks.

approve

```
    function approve(address _spender, uint256 _value) public returns (
        bool) {
        require(_spender != address(0));
        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
        return true;
    }
```

The `approve` function implements the standard ERC20 function, with an extra check ensuring the 0 address cannot be used as an argument.

Management fuctions

mint

```
    function mint(address _user, uint256 _tokensAmount) public
        onlyTgeIssuer tgeRunning returns(bool) {
        uint256 newSupply = totalSupply.add(_tokensAmount);
        require(
            _user != address(0) &&
            _tokensAmount > 0 &&
             newSupply < hardCap
        );
        balances[_user] = balances[_user].add(_tokensAmount);
        totalSupply = newSupply;
        Transfer(0x0, _user, _tokensAmount);
        return true;
    }
```

The `mint` function can only be called by the `TgeIssuer` address, and only while the ICO is live. It creates new tokens and assigns them to the chosen address,only up to hardCap tokens can be created.

reserveTokensGroup

```
    function reserveTokensGroup(address[] _users, uint256[] _tokensAmounts
        ) external onlyOwner {
        require(_users.length == _tokensAmounts.length);
        uint256 newSupply;
        for(uint8 i = 0; i < _users.length; i++){
```

```
            newSupply = totalSupply.add(_tokensAmounts[i].mul(10 **
                decimals));
            require(
                _users[i] != address(0) &&
                _tokensAmounts[i] > 0 &&
                newSupply < hardCap
            );
            balances[_users[i]] = balances[_users[i]].add(_tokensAmounts[i
                ].mul(10 ** decimals));
            totalSupply = newSupply;
            Transfer(0x0, _users[i], _tokensAmount[i]);
        }
    }
```

The `reserveTokensGroup` function takes an array of addresses and values, and adds tokens to those addresses, up to the maximum of hardCap.

It caan only be called by the owner.

reserveTokens

```
    function reserveTokens(address _user, uint256 _tokensAmount) external
        onlyOwner {
        uint256 newSupply = totalSupply.add(_tokensAmount.mul(10 **
            decimals));
        require(
            _user != address(0) &&
            _tokensAmount > 0 &&
            newSupply < hardCap
        );
        balances[_user] = balances[_user].add(_tokensAmount.mul(10 **
            decimals));
        totalSupply = newSupply;
        Transfer(0x0, _user, _tokensAmount);
    }
```

The `reserveTokens` function is the same as `reserveTokensGroup`, except it deals with single addresses instead of an array.

It can only be called by the owner.

startTge

```
    function startTge() external onlyOwner {
        tgeActive = true;
        if(tgeStartTime == 0) tgeStartTime = block.timestamp;
    }
```

The `startge` function allows the owner to begin the ICO.

stopTge

```
    function stopTge(bool _restart) external onlyOwner {
        tgeActive = false;
        if(_restart) tgeStartTime = 0;
    }
```

The `stopTge` function allows the contract owner to stop the ICO.

extendTge

```
    function extendTge(uint256 _value) external onlyOwner {
        tgeDuration = tgeDuration.add(_time);
    }
```

The `extendTge` function allows the contract owner to extend the token generation event deadline.

setEscrow

```
    function setEscrow(address _escrow) external onlyOwner {
        escrow = _escrow;
    }
```

The `setEscrow` function allows the owner to set the escrow address.

setTgeIssuer

```
    function setTgeIssuer(address _tgeIssuer) external onlyOwner {
        tgeIssuer = _tgeIssuer;
    }
```

The `setTgeIssuer` function allows the owner to set the `tgeIssuer` address.

Disclaimer

This audit concerns only the correctness of the Smart Contracts listed, and is not to be taken as an endorsement of the platform, team, or company.

Audit Attestation

/ This audit has been signed by the key provided on https://keybase.io/mattdf - and the signature is available on https://github.com/mattdf/audits/