
Better Betting Token Audit

AUTHOR: MATTHEW DI FERRANTE

2018-01-02

Audited Material Summary

The audit consists of the following file:

```
1 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.

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

SafeMath is used throughout the whole contract.

Constructor

```
1   function BETR_TOKEN() public {
2     owner = msg.sender;
3   }
```

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

Token Functionality

allowEscrow

```
1   function allowEscrow(bool _choice) external returns(bool) {
2     escrowAllowed[msg.sender] = _choice;
3     return true;
4   }
```

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

escrowFrom

```
1     function escrowFrom(address _from, uint256 _value) external onlyEscrow
      returns(bool) {
2         require (
3             _from != address(0) &&
4             balances[_from] >= _value &&
5             escrowAllowed[_from] &&
6             _value > 0
7         );
8         balances[_from] = balances[_from].sub(_value);
9         balances[escrow] = balances[escrow].add(_value);
10        Transfer(_from, escrow, _value);
11        return true;
12    }
```

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

```
1     function escrowReturn(address _to, uint256 _value, uint256 _fee)
      external onlyEscrow returns(bool) {
2         require(
3             _to != address(0) &&
4             _value > 0
5         );
6         if(_fee > 0) {
7             require(_fee < totalSupply && _fee < balances[escrow]);
8             totalSupply = totalSupply.sub(_fee);
9             balances[escrow] = balances[escrow].sub(_fee);
10        }
11        require(transfer(_to, _value));
12        return true;
13    }
```

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

```
1     function transfer(address _to, uint256 _value) public returns (bool) {
2         require(
3             _to != address(0) &&
4             balances[msg.sender] >= _value &&
5             balances[_to] + _value > balances[_to]
6         );
7         balances[msg.sender] = balances[msg.sender].sub(_value);
8         balances[_to] = balances[_to].add(_value);
9         Transfer(msg.sender, _to, _value);
10        return true;
11    }
```

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

transferFrom

```
1     function transferFrom(address _from, address _to, uint256 _value)
2         public returns (bool) {
3         require (
4             _from != address(0) &&
5             _to != address(0) &&
6             balances[_from] >= _value &&
7             allowed[_from][msg.sender] >= _value &&
8             balances[_to] + _value > balances[_to]
9         );
10        balances[_from] = balances[_from].sub(_value);
11        balances[_to] = balances[_to].add(_value);
12        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
13        Transfer(_from, _to, _value);
14        return true;
15    }
```

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

approve

```
1     function approve(address _spender, uint256 _value) public returns (
2         bool) {
3         require(_spender != address(0));
4         allowed[msg.sender][_spender] = _value;
5         Approval(msg.sender, _spender, _value);
6         return true;
7     }
```

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

```
1     function mint(address _user, uint256 _tokensAmount) public
2         onlyTgeIssuer tgeRunning returns(bool) {
3         uint256 newSupply = totalSupply.add(_tokensAmount);
4         require(
5             _user != address(0) &&
6             _tokensAmount > 0 &&
7             newSupply < hardCap
8         );
9         balances[_user] = balances[_user].add(_tokensAmount);
10        totalSupply = newSupply;
11        Transfer(0x0, _user, _tokensAmount);
12        return true;
13    }
```

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

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

```
5     newSupply = totalSupply.add(_tokensAmounts[i].mul(10 **
6         decimals));
7     require(
8         _users[i] != address(0) &&
9         _tokensAmounts[i] > 0 &&
10        newSupply < hardCap
11    );
12    balances[_users[i]] = balances[_users[i]].add(_tokensAmounts[i]
13        ].mul(10 ** decimals));
14    totalSupply = newSupply;
15    Transfer(0x0, _users[i], _tokensAmount[i]);
16 }
```

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

It can only be called by the owner.

reserveTokens

```
1     function reserveTokens(address _user, uint256 _tokensAmount) external
2         onlyOwner {
3         uint256 newSupply = totalSupply.add(_tokensAmount.mul(10 **
4             decimals));
5         require(
6             _user != address(0) &&
7             _tokensAmount > 0 &&
8             newSupply < hardCap
9         );
10        balances[_user] = balances[_user].add(_tokensAmount.mul(10 **
11            decimals));
12        totalSupply = newSupply;
13        Transfer(0x0, _user, _tokensAmount);
14    }
```

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

```
1     function startTge() external onlyOwner {
2         tgeActive = true;
3         if(tgeStartTime == 0) tgeStartTime = block.timestamp;
4     }
```

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

stopTge

```
1     function stopTge(bool _restart) external onlyOwner {
2         tgeActive = false;
3         if(_restart) tgeStartTime = 0;
4     }
```

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

extendTge

```
1     function extendTge(uint256 _value) external onlyOwner {
2         tgeDuration = tgeDuration.add(_time);
3     }
```

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

setEscrow

```
1     function setEscrow(address _escrow) external onlyOwner {
2         escrow = _escrow;
3     }
```

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

setTgeIssuer

```
1     function setTgeIssuer(address _tgeIssuer) external onlyOwner {  
2         tgeIssuer = _tgeIssuer;  
3     }
```

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/>