DataBrokerDAO Token Sale Audit

AUTHOR: MATTHEW DI FERRANTE

2017-09-15

Audited Material Summary

The audit consists mainly of the EarlyTokenSale contract and its usage of the MiniMeToken contract. The rest of the contracts have been through previous audits by the OpenZeppelin team and others.

EarlyTokenSale.sol

The EarlyTokenSale contract implements a token sale with a maximum raise of 28,125 ether with gas price capped at 50 Gwei. The ETH:Token ratio is 1:1200. It also prevents more than 1 call per 100 blocks from a specific address.

The contract inherits an interface from TokenController, and a set of helper functions from Controlled:

```
contract EarlyTokenSale is TokenController, Controlled
```

SafeMath from zeppelin-contracts is used throughout the entire contract by being associated to all uint256 operations:

```
using SafeMath for uint256
```

Some security notes:

- Per address gas or call limits are only a band-aid and incite spamming, sophisticated users will still be able to get around it by automating sends from many different addresses.
- Make sure the controller address is always set to a contract that understands the proper methods, if it ever does get set to a contract (e.g. a multisig)
- The authors opted not to employ ERC20 Short Address “Attack” protection following the advice from coinfabrik: https://blog.coinfabrik.com/smart-contract-short-address-attack-mitigation-failure/

Constructor

```
function EarlyTokenSale(
    uint _startFundingTime,
    uint _endFundingTime,
    address _vaultAddress,
    address _tokenAddress
) {
    require(_endFundingTime > now);
    require(_endFundingTime >= _startFundingTime);
```

```
    require(_vaultAddress != 0);
    require(_tokenAddress != 0);

    startFundingTime = _startFundingTime;
    endFundingTime = _endFundingTime;
    tokenContract = DataBrokerDaoToken(_tokenAddress);
    vaultAddress = _vaultAddress;
    paused = false;
}
```

The constructor takes start and ending times for the token sale and validates them, it also takes vault and token addresses and ensures they aren't the 0 address. These are all assigned to state variables with _tokenAddress being type cast to a DataBrokerDaoToken type, which inherits MiniMeToken. The state variable "paused" is set to false, enabling the token sale upon deployment.

Default Function

```
function () payable notPaused {
    doPayment(msg.sender);
}
```

The default function is simply a payable forwarder to doPayment with the notPaused modifier.

proxyPayment

```
function proxyPayment(address _owner) payable notPaused returns(bool
   success) {
    return doPayment(_owner);
}
```

proxyPayment is a payable forwarder to doPayment which allows buying tokens for an address different from msg.sender. Like the default function, this is only callable when the crowdsle is not paused.

onTransfer

```
function onTransfer(address _from, address _to, uint _amount) returns(bool
    success) {
    if ( _from == vaultAddress ) {
        return true;
```

```
    }
    return false;
}
```

onTransfer is a function used by MiniMe to check whether a transfer is allowed for a given Token implementation. In this case all transfers from the vaultAddress are allowed, and all others rejected.

Note: This check only works if “controller” is a contract, otherwise it’s skipped.

onApprove

```
function onApprove(address _owner, address _spender, uint _amount) returns
    (bool success) {
    if ( _owner == vaultAddress ) {
        return true;
    }
    return false;
}
```

onApprove is the same as onTransfer, except it catches on approve events instead of transfer events. Like onApprove, this allows all approvals as long as the owner is the vaultAddress, and rejects all others.

Note: This check only works if “controller” is a contract, otherwise it’s skipped.

doPayment

```
function doPayment(address _owner) internal returns(bool success) {
    require(tx.gasprice <= maxGasPrice);

    // Antispam
    // do not allow contracts to game the system
    require(!isContract(msg.sender));
    // limit the amount of contributions to once per 100 blocks
    require(getBlockNumber().sub(lastCallBlock[msg.sender]) >=
        maxCallFrequency);
    lastCallBlock[msg.sender] = getBlockNumber();

    // First check that the EarlyTokenSale is allowed to receive this
        donation
    require(startFundingTime <= now);
```

```
13      require(endFundingTime > now);
14      require(tokenContract.controller() != 0);
15      require(msg.value > 0);
16      require(totalCollected.add(msg.value) <= maximumFunding);
17
18      // Track how much the EarlyTokenSale has collected
19      totalCollected = totalCollected.add(msg.value);
20
21      //Send the ether to the vault
22      require(vaultAddress.send(msg.value));
23
24      // Creates an equal amount of tokens as ether sent. The new tokens are
            created in the ‘_owner‘ address
25      require(tokenContract.generateTokens(_owner, tokensPerEther.mul(msg.
            value)));
26
27      return true;
28  }
```

Main logic for token sale. Takes a payment and ensures the tx gas price is less than the limit, ensures the msg.sender is not a contract, and ensures that there was less than 1 payment from that msg.sender in the last 100 blocks.

Ensures token sale is still open, that the token contract controller is not the zero address, that the amount of ether sent is not 0, and that the total collected so far is less than the cap.

It adds the ether sent to the total collected and ensures that it is sent to the vaultAddress.

Finally it calls generateTokens wrapped around require.

This function is internal and hence cannot be called directly.

isContract

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

```
}
```

isContract simply checks that the addr isn't 0 and that the length of the code stored is greater than 0 using the EXTCODESIZE opcode.

finalizeSale

```
function finalizeSale() onlyController {
    require(now > endFundingTime || totalCollected >= maximumFunding);
    require(!finalized);

    uint256 reservedTokens = 225000000 * 0.35 * 10**18;
    if (!tokenContract.generateTokens(vaultAddress, reservedTokens)) {
        revert();
    }

    finalized = true;
}
```

Function that finalizes the sale. It can only be called after the sale expires or after the cap is reached, and only if the sale has not already been finalized.

It allocates platform and team tokens once the sale is over.

claimTokens

```
function claimTokens(address _token) onlyController {
    if (_token == 0x0) {
        controller.transfer(this.balance);
        return;
    }

    ERC20Token token = ERC20Token(_token);
    uint balance = token.balanceOf(this);
    token.transfer(controller, balance);
    ClaimedTokens(_token, controller, balance);
}
```

Standard function that allows tokens mistakenly sent to this address to be redeemed by the controller.

pauseContributions

```
function pauseContribution() onlyController {
    paused = true;
}
```

Sets “paused” to true when called by the controller.

resumeContributions

```
function resumeContribution() onlyController {
    paused = false;
}
```

Sets “paused” to false when called by the controller.

modifier: notPaused

```
modifier notPaused() {
    require(!paused);
    _;
}
```

Simply throws if paused is set to true.

MiniMeToken

The main underlying token handled by EarlyTokenSale is DataBrokerDaoToken which is just an instantiation of MiniMeToken.

MiniMeToken is used in the Status, Aragon and Swarm City tokens, and has been previously audited by multiple parties.

After review it appears that the DataBrokerDaoToken team have used this contract in an safe manner. The only relevant vulnerability from past audits is the ERC20 Short Address “Attack”.

List of past audits for MiniMeToken:

- CoinFabrik
- BlockchainLabs
- OpenZeppelin