Bounty0x Crowdsale Audit

ZK Labs Auditing Services

AUTHOR: MATTHEW DI FERRANTE

2017-12-15

Audited Material Summary

The audit consists of the Bounty0x Crowdsale contracts. The git commit hash of the reviewed files is `11adb7f7256f67cdf288ef4b2c500633cccfa806`.

The contracts implement a simple price crowdsale with a whitelisting period backed by a MiniMe Token.

SafeMath is used throughout and the contracts do not have any major security issues.

Bounty0xCrowdsale.sol

The `Bounty0xCrowdsale` contract implements the main logic for the crowdsale. It inherits from a few different contracts:

```
contract Bounty0xCrowdsale is KnowsTime, KnowsConstants, Ownable,
    BntyExchangeRateCalculator, AddressWhitelist, Pausable
```

Security

The contract does not have any security issues, but implements some unadvised functionality such as setting a maximum gas price and gas allowance per tx.

We recommend also sending the collected funds to a vault or multisig directly instead of having a withdraw function.

Constructor

```
    function Bounty0xCrowdsale(Bounty0xToken _bounty0xToken, uint
        _USDEtherPrice)
        BntyExchangeRateCalculator(MICRO_DOLLARS_PER_BNTY_MAINSALE,
            _USDEtherPrice, SALE_START_DATE)
        public
    {
        bounty0xToken = _bounty0xToken;
    }
```

The constructor explicitly calls the parent constructor for `BntyExchangeRateCalculator` with the exchange rate and sale start date, and assigns the MiniMe token address to `bounty0xToken`.

withdraw

```
    function withdraw(uint amount) public onlyOwner {
        msg.sender.transfer(amount);
        OnWithdraw(msg.sender, amount);
    }
```

The `withdraw` function allows the contract owner to withdraw raised ether from the crowdsale contract.

Default Function

```
    function () payable public whenNotPaused {
        uint time = currentTime();

        // require the sale has started
        require(time >= SALE_START_DATE);

        // require that the sale has not ended
        require(time < SALE_END_DATE);

        // maximum contribution from this transaction is tracked in this
           variable
        uint maximumContribution = usdToWei(HARD_CAP_USD).sub(
           totalContributions);

        // store whether the contribution is made during the whitelist
           period
        bool isDuringWhitelistPeriod = time < WHITELIST_END_DATE;

        // these limits are only checked during the limited period
        if (time < LIMITS_END_DATE) {
            // require that they have not overpaid their gas price
            require(tx.gasprice <= MAX_GAS_PRICE);

            // require that they haven't sent too much gas
            require(msg.gas <= MAX_GAS);

            // if we are in the WHITELIST period, we need to make sure the
                sender contributed to the presale
            if (isDuringWhitelistPeriod) {
```

```
26                  require(isWhitelisted(msg.sender));
27
28                  // the maximum contribution is set for the whitelist
                        period
29                  maximumContribution = Math.min256(
30                      maximumContribution,
31                      usdToWei(MAXIMUM_CONTRIBUTION_WHITELIST_PERIOD_USD).
                            sub(contributionAmounts[msg.sender])
32                  );
33              } else {
34                  // the maximum contribution is set for the limited period
35                  maximumContribution = Math.min256(
36                      maximumContribution,
37                      usdToWei(MAXIMUM_CONTRIBUTION_LIMITED_PERIOD_USD).sub(
                            contributionAmounts[msg.sender])
38                  );
39              }
40          }
41
42          // calculate how much contribution is accepted and how much is
                refunded
43          uint contribution = Math.min256(msg.value, maximumContribution);
44          uint refundWei = msg.value.sub(contribution);
45
46          // require that they are allowed to contribute more
47          require(contribution > 0);
48
49          // account contribution towards total
50          totalContributions = totalContributions.add(contribution);
51
52          // account contribution towards address total
53          contributionAmounts[msg.sender] = contributionAmounts[msg.sender].
                add(contribution);
54
55          // and send them some bnty
56          uint amountBntyRewarded = Math.min256(weiToBnty(contribution),
                bounty0xToken.balanceOf(this));
57          require(bounty0xToken.transfer(msg.sender, amountBntyRewarded));
58
59          if (refundWei > 0) {
60              msg.sender.transfer(refundWei);
61          }
62
```

```
63          // log the contribution
64          OnContribution(msg.sender, isDuringWhitelistPeriod, contribution,
                amountBntyRewarded, refundWei);
65      }
```

The default function is the crowdsale's main contribution function. It ensures that the function has been called between the sale's start and end time, and also enforces a hard cap on the total contribution amount. Having the `whenNotPaused` modifier, it can only be called if the crowdsale is not in the paused state.

During the "limits" period, gas price and gas amount and contribution per address are limited, and if in the whitelist period, only whitelisted addresses may contribute up to the maximum whitelist contribution amount.

If exceess ether (past any limits) is sent to the crowdsale, it is refunded via transfer.

On success, the purchased amount of BNTY tokens are sent to the user from the crowdfund contract, and an `OnContribution` event is emitted.

Bounty0xToken.sol

The `Bounty0xToken` contract is just an instantiation of MiniMe token:

```
1  contract Bounty0xToken is MiniMeToken {
2      function Bounty0xToken(address _tokenFactory)
3          MiniMeToken(
4              _tokenFactory,
5              0x0,                        // no parent token
6              0,                          // no snapshot block number from
                   parent
7              "Bounty0x Token",           // Token name
8              18   ,                      // Decimals
9              "BNTY",                     // Symbol
10             true                       // Enable transfers
11         )
12         public
13     {
14     }
15 }
```

Transfers are enabled by default, and no parent token is set.

KnowsConstants.sol & KnowsTime.sol

The `KnowsConstants` and `KnowsTime` are simple contracts that allow the crowdsale to inherit some variables and a function to abstract the `time`:

```
    function currentTime() public view returns (uint) {
        return now;
    }
```

Security

There are no security issues, beyond some unnecessary indirection/abstraction.

BntyExchangeRateCalculator.sol

The `BntyExchangeRateCalculator` contract is used by Bounty0xCrowdsale to retrieve the amount of tokens that should be allocated for a contribution.

It inherits from `KnowsTime` and `Ownable`:

```
contract BntyExchangeRateCalculator is KnowsTime, Ownable
```

Constructor

```
    function BntyExchangeRateCalculator(uint _bntyMicrodollarPrice, uint
        _USDEtherPrice, uint _fixUSDPriceTime)
        public
    {
        require(_bntyMicrodollarPrice > 0);
        require(_USDEtherPrice > 0);

        bntyMicrodollarPrice = _bntyMicrodollarPrice;
        fixUSDPriceTime = _fixUSDPriceTime;
        USDEtherPrice = _USDEtherPrice;
    }
```

The constructor ensures the microdollar and ether price are greater than 0, and assigns its arguments to the respective state variables.

setUSDEtherPrice

```
    function setUSDEtherPrice(uint _USDEtherPrice) onlyOwner public {
        require(currentTime() < fixUSDPriceTime);
        require(_USDEtherPrice > 0);

        USDEtherPrice = _USDEtherPrice;
    }
```

The `setUSDEtherPrice` function can be called by the contract owner to change the USD price of Ether any time after the "fixed USD price time" timeout. The function ensures this value can never be 0.

usdToWei

```
    function usdToWei(uint usd) view public returns (uint) {
        return WEI_PER_ETH.mul(usd).div(USDEtherPrice);
    }
```

The `usdToWei` function is a constant function that returns (10^18 * usd) / ETH_PRICE.

weiToBnty

```
    function weiToBnty(uint amtWei) view public returns (uint) {
        return USDEtherPrice.mul(MICRODOLLARS_PER_DOLLAR).mul(amtWei).div(
           bntyMicrodollarPrice);
    }
```

The `weiToBnty` function is a constant function that returns (ether price * 10^6 * wei contribution) / BNTY microdollar price.

AddressWhitelist.sol

The `AddressWhitelist` contract is used by `Bounty0xCrowdsale` to check whether an address is whitelist, and allow the contract owner an interface to add and remove addresses from the whitelist.

It is a simple contract that inherits from Ownable:

```
contract AddressWhitelist is Ownable
```

Constructor

```
    function AddressWhitelist() public {
    }
```

The constructor is empty.

isWhitelisted

```
    function isWhitelisted(address addr) view public returns (bool) {
        return whitelisted[addr];
    }
```

The `isWhitelisted` function is the read-only function used by `Bounty0xCrowdsale` to check whether an address is in the whitelist.

addToWhitelist

```
    function addToWhitelist(address[] addresses) public onlyOwner returns
       (bool) {
        for (uint i = 0; i < addresses.length; i++) {
            if (!whitelisted[addresses[i]]) {
                whitelisted[addresses[i]] = true;
                LogWhitelistAdd(addresses[i]);
            }
        }

        return true;
    }
```

The `addToWhitelist` function allows the contract owner to supply an array of addresses to add to the crowdsale whitelist.

removeFromWhitelist

```
    function removeFromWhitelist(address[] addresses) public onlyOwner
       returns (bool) {
        for (uint i = 0; i < addresses.length; i++) {
            if (whitelisted[addresses[i]]) {
```

```
                whitelisted[addresses[i]] = false;
                LogWhitelistRemove(addresses[i]);
            }
        }

        return true;
    }
```

The `addToWhitelist` function allows the contract owner to supply an array of addresses to remove from the crowdsale whitelist.

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/