(https://blog.coinfabrik.com/)

CRYPTOCURRENCY (HTTPS://BLOG.COINFABRIK.COM/CATEGORY/CRYPTOCURRENCY/)

Worldbit Token Sale Smart Contract Audit

Pablo Yabo (https://blog.coinfabrik.com/author/pyabo/)

November 16, 2017 (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/worldbit-token-sale-smart-contract-audit/)

CoinFabrik's Smart Contract Audit (https://www.coinfabrik.com) team audited WorldBit Token sale ICO's smart contracts.

The contracts audited are taken from the WorldBit repository on https://github.com/CoinFabrik/ico/tree/world-bit (https://github.com/CoinFabrik/ico/tree/world-bit). The audit is based on the commit *445cc0c28894a85cb58f54631666deafdd35d859*, and updated to reflect changes on *21abcc3bde6d64d4aee391544fc9332fc9f2749d*.

Contents

Summary

The audited contracts are:

- ERC20Basic: non-approval half of ERC20 interface.
- ERC20: Full ERC20 interface.
- FractionalERC20: Full ERC20 interface with decimals.
- Ownable: Interface for owning a contract with modifiers.
- Burnable: an internal interface for burning tokens.
- Mintable: an internal interface for minting tokens.
- MintableToken: Mintable Token Implementation.
- BasicToken: ERC20Basic implementation.
- StandardToken: ERC20 implementation.
- ReleasableToken: The crowdsale part of a token, modifiers to only allow certain addresses to transfer pre-release.
- UpgradeableToken: ERC20Basic implementation with upgrade capabilities, uses burning
- LostAndFoundToken: Allows other tokens to be retrieved from the contract
- CrowdsaleToken: Specific implementation of the token, which is Releasable, Mintable and Upgradable.
- Haltable: Allows contracts to halt, has modifiers to disallow operations when halted.
- TokenTranchePricing: Stores multiple prices over different overlapping periods of time.
- GenericCrowdsale: Base Crowdsale, with generic implementations of basic functions.
- Crowdsale: Specific implementation of the Crowdsale.

The WorldBit Token will have a total of 210,000,000 tokens ever issued, out of which 25% (52,500,000) will be offered for sale. 20% of sale tokens will be offered in the pre-sale period, while the rest will be sold on the ICO. The sale will always succeed since it doesn't have a minimum cap.

Regarding pricing, the number of tokens you get per ETH will decrease by 20 WBT each week at pre-sale, starting at 410 WBT per ETH. Once the ICO starts, it will decrease by 10 WBT roughly every week, until the ICO ends at 280 WBT.

Detailed findings

Minor severity

TokenTranchePricing start timestamp *require* is strange and doesn't work

This requires is strange:

```
require(init_tranches[start_offset].add(1 hours) > block.timestamp);
```

It means the first tranche is allowed to start one hour **before** the block timestamp. What one would probably want is requiring it to start one hour **after**, to have some time:

```
require(block.timestamp.add(1 hours) < init_tranches[start_offset]);
```

Even after changing that line it doesn't work as intended, since inside the *for* loop there's a *require* for each tranche:

```
require(block.timestamp < start && start.add(1 hours) <= end);
```

The first clause does not have the one-hour restriction, so each tranche after the first one is allowed to break that rule.

Fixed in commit cce0ebcdf83db2f0d46f2a4f30da6dbf5d5eb2ff

TokenTranchePricing does not use Safe Math

There is a *using* directive for *uint* but it is never used in the contract. It should either be used properly at every arithmetical operator or removed altogether since there are only operations at initialization and it is fairly simple to check for integrity.

```
function TokenTranchePricing(uint[] init_tranches) public {
  // Need to have tuples, length check
  require(init_tranches.length % tranche_size == 0);
  // A tranche with amount zero can never be selected and is therefore useless.
  // This check and the one inside the loop ensure no tranche can have an amount equal to zero.
  require(init_tranches[amount_offset] > 0);

  tranches.length = init_tranches.length / tranche_size;
  for (uint i = 0; i < init_tranches.length / tranche_size; i++) {
        // No invalid steps
        uint amount = init_tranches[i * tranche_size + amount_offset];
        uint start = init_tranches[i * tranche_size + start_offset];
        uint end = init_tranches[i * tranche_size + end_offset];
        require(block.number < start && start < end);
        // Bail out when entering unnecessary tranches
        // This is preferably checked before deploying contract into any blockchain.
        require(i == 0 ||
(end >= tranches[i - 1].end && amount > tranches[i - 1].amount) ||
                (end > tranches[i - 1].end && amount >= tranches[i - 1].amount));

        tranches[i].amount = amount;
        tranches[i].price = init_tranches[i * tranche_size + price_offset];
        tranches[i].start = start;

        tranches[i].end = end;
  }
}
```

Fixed in commit 63d4ddc82c0d548fa61a01262612e4fc4c0b0558

TokenTranchePricing inherits Ownable but never uses it

If there is no need for the *OnlyOwner* modifier in the contract this dependency should be removed.

```
contract TokenTranchePricing is Ownable {
```

Fixed in commit 0dbef35c8efabc6af3d81e8d6611399a6e579e7d

TokenTranchePricing has misleading comments

In *getCurrentPrice*, the documentation of the function says it returns 0 outside tranche ranges, but it actually reverts. There is also a return comment missing in *getCurrentTranche*.

```
/// @dev Get the current tranche or bail out if we are not in the tranche periods.
/// @param tokensSold total amount of tokens sold, for calculating the current tranche
/// @return {[type]} [description]
function getCurrentTranche(uint tokensSold) private constant returns (Tranche) {
  for (uint i = 0; i < tranches.length; i++) {
        if (tranches[i].start <= block.number && block.number < tranches[i].end && tokensSold <
        return tranches[i];
        }
  }
  // No tranche is currently active
  revert();
}

/// @dev Get the current price.
/// @param tokensSold total amount of tokens sold, for calculating the current tranche
/// @return The current price or 0 if we are outside tranche ranges
function getCurrentPrice(uint tokensSold) public constant returns (uint result) {
  return getCurrentTranche(tokensSold).price;
}
```

Fixed in commit 63d4ddc82c0d548fa61a01262612e4fc4c0b0558

Solidity compiler warnings

Compiling with solidity shows some warnings about shadowing and unused parameters. Having warnings is wrong, even when they are showing inexistent issues they may hide later important warnings.

Specifically, the function *calculateTokenAmount* in GenericCrowdsale and Crowdsale has an argument agent which it is never used.

```
  function calculateTokenAmount(uint weiAmount, address agent) internal constant returns (uint
    uint tokensPerWei = getCurrentPrice(tokensSold);
    uint maxAllowed = sellable_tokens.sub(tokensSold).div(tokensPerWei);
    weiAllowed = maxAllowed.min256(weiAmount);

    if (weiAmount < maxAllowed) {

      tokenAmount = tokensPerWei.mul(weiAmount);
    }
    // With this case we let the crowdsale end even when there are rounding errors due to the t
    else {
      tokenAmount = sellable_tokens.sub(tokensSold);
    }
  }
```

Fixed in commits 63d4ddc82c0d548fa61a01262612e4fc4c0b0558 and a425e29732d1ba9a65d786ac23364fa89dd4e851

Missing transfer event in burnTokens

Function *burnTokens* in StandardToken.sol does not generate a standard transfer event when it burns tokens. We recommend adding this event so external services can update the account balance.

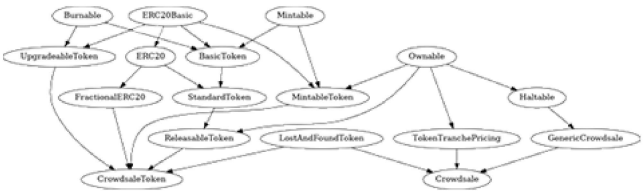
```
  function burnTokens(address account, uint value) internal {
    balances[account] = balances[account].sub(value);
    total_supply = total_supply.sub(value);
    Burned(account, value);
    Transfer(account, 0, value);
  }
```

Fixed in commit 1b6a90f43970427b3cbb886a85a9f198b35f881d

Enhancements

Unnecessary multiple inheritance

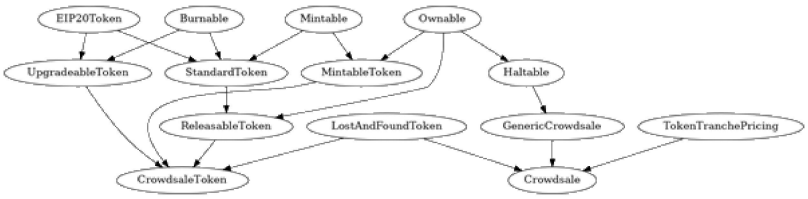
Inheritance has its benefits, but abusing it may lead to a diamond problem (https://en.wikipedia.org/wiki/Multiple_inheritance#The_diamond_problem). It can be tricky to follow function overrides and corresponding *super* calls on a complex scheme, which makes it harder to notice issues on the contracts:



(https://blog.coinfabrik.com/wp-content/uploads/2017/11/worldbit-audit-1-1-1.png)

The contracts BasicToken and StandardToken each implement a half of ERC20. This interface is also split between ERC20Basic, ERC20, and FractionalERC20. Also, FractionalERC20 only defines *decimals*, so it doesn't need to inherit ERC20. We suggest unifying these contracts to simplify the scheme.

Improved in commit b0ffde290532d223f003b58b844aae66e8f2aa6e. This is the result, including the TokenTranchePricing and some other recent commits:



(https://blog.coinfabrik.com/wp-content/uploads/2017/11/worldbit-audit-2.png)

Simplify token calculation

Function *calculateTokenAmount* in Crowdsale.sol is too complex, it can be simplified keeping the same functionality at lower gas costs.

```
function calculateTokenAmount(uint weiAmount, address agent) internal constant returns (uint
  uint tokensPerWei = getCurrentPrice(tokensSold);
  uint remainingTokens = sellable_tokens.sub(tokensSold);

  tokenAmount = tokensPerWei.mul(weiAmount);
  if (tokenAmount > remainingTokens) {
    tokenAmount = remainingTokens;
  }
  weiAllowed = tokenAmount.div(tokensPerWei);
}
```

Document Crowdsale constructor

Both contracts Crowdsale and GenericCrowdsale do not document the constructor arguments. These are the main contracts of the Crowdsale and they should be documented properly along with their constraints.

For example, the parameter start is a timestamp that requires being at least one hour after the block is mined. But there's no documentation of such a requirement.

Improved in commit 36c8ef479f2b222f60eb8d2d605cb48a907064ce

TokenTranchePricing constructor can be refactored

Consider doing something like this in the future to improve readability and performance.

```
/// @dev Contruction, creating a list of tranches
/// @param init_tranches Raw array of ordered tuples: (start amount, start timestamp, end time
function TokenTranchePricing(uint[] init_tranches) public {
  // Need to have tuples, length check
  require(init_tranches.length % tranche_size == 0);
  // A tranche with amount zero can never be selected and is therefore useless.
  // This check and the one inside the loop ensure no tranche can have an amount equal to zero
  require(init_tranches[amount_offset] > 0);
  require(block.timestamp.add(1 hours) < init_tranches[start_offset]);

  Tranche memory last_tranche;
  tranches.length = init_tranches.length.div(tranche_size);
  for (uint i = 0; i < tranches.length; i++) {
       // No invalid steps
       uint tranche_offset = i.mul(tranche_size);
       uint amount = init_tranches[tranche_offset.add(amount_offset)];
       uint start = init_tranches[tranche_offset.add(start_offset)];
       uint end = init_tranches[tranche_offset.add(end_offset)];
       uint price = init_tranches[tranche_offset.add(price_offset)];

       require(block.timestamp < start && start.add(1 hours) <= end);
       // Bail out when entering unnecessary tranches
       // This is preferably checked before deploying contract into any blockchain.
       require(i == 0 || (end >= last_tranche.end && amount > last_tranche.amount) ||
                       (end > last_tranche.end && amount >= last_tranche.amount));
       last_tranche = Tranche(amount, start, end, price);
       tranches[i] = last_tranche;
  }
}
```

Improved on commit 4035fe8c6284ff34601f646b6c7718036b4be21d

Unnecessary minting privilege

Since the sale has a fixed amount of tokens to sell, there is no need to keep the Crowdsale contract as a minting agent after the initial setup.

In Crowdsale's constructor it assigns to itself the minting privilege, but after the setup, the privilege is kept.

```
function Crowdsale(address team_multisig, uint start, uint end, address token_retriever, uint
  ...
  token.setMintAgent(address(this), true);

  // Tokens to be sold through this contract
  token.mint(address(this), sellable_tokens);

  // Remove unnecessary privilege
  token.setMintAgent(address(this), false);
```

Notes

- There is a refund function on LostAndFoundToken.sol in case someone sends tokens to the contract. We didn't find any security issue in such a risky function and can be only called by contract owner.
- WorldBit uses timestamps instead of block numbers, but only for long periods of time (Crowdsale dates, TokenTranchePricing dates). Although they are subject to a miner frontrunning attack it is very unlikely and it limited only to very popular ICOs.
- Remove commented out code in StandardToken.sol. Since modifier *onlyPayloadSize* is no longer used, there's no need to keep it.

- Functions *halt/unhalt* in Haltable can be merged in a single function with a *boolean* parameter.

Conclusion

Overall the contracts are simple enough and easy to audit. Most methods have only a few lines of code and are well documented. Overall code quality is very high.

References

- “WorldBit Token Repository” https://github.com/CoinFabrik/ico/tree/world-bit (https://github.com/CoinFabrik/ico/tree/world-bit)
- “ERC-20 Token Standard” https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md)

Do you want to know what is Coinfabrik Auditing Process?

Check our A-Z Smart Contract Audit Guide (https://blog.coinfabrik.com/smart-contract-audits-ultimate-security-guide/) or you could request a quote (https://www.coinfabrik.com#contact_us) for your project.

Related Posts

(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/dmtoken-token-sale-smart-contract-audit/)

DMToken Token Sale Smart Contract Audit (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/dmtoken-token-sale-smart-contract-audit/)

Coinfabrik was hired to audit the contract in terms of its security. First of all,…

(https://blog.coinfabrik.com/smart-contracts/smart-

contract-audit-smart-contracts/rightmesh-token-sale-smart-contract-audit-master/)

Rightmesh Token Sale Smart Contract Audit (Master)

(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/rightmesh-token-sale-smart-contract-audit-master/)

Coinfabrik was asked to audit the contracts for the RightMesh Token sale. In the first…

(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/realisto-token-sale-smart-contract-audit/)

Realisto Token Sale Smart Contract Audit

(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/realisto-token-sale-smart-contract-audit/)

By Ismael Bejarano and Pablo Yabo Coinfabrik smart contract audit's team was hired to review…

Tags:

Token (https://blog.coinfabrik.com/tag/token/)

worldbit (https://blog.coinfabrik.com/tag/worldbit/)

SHARE ON

(https://www.facebook.com/sharer/sharer.php?u=https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/worldbit-token-sale-smart-contract-audit/)
(https://twitter.com/intent/tweet?text=Worldbit%20Token%20Sale%20Smart%20Contract%20Audit&url=https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/worldbit-token-sale-smart-contract-audit/)
(https://pinterest.com/pin/create/button/?url=&media=https://blog.coinfabrik.com/wp-content/uploads/2017/11/worldbit.png&description=Worldbit+Token+Sale+Smart+Contract+Audit)
(https://www.linkedin.com/shareArticle?mini=true&url=https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/worldbit-token-sale-smart-contract-audit/&title=Worldbit%20Token%20Sale%20Smart%20Contract%20Audit&source=CoinFabrik%20Blog)

← PREVIOUS ARTICLE

RCN Network Smart Contract Audit (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/ripio-credit-network-rcn-smart-contract-audit/)

NEXT ARTICLE →

Survey of Decentralized Exchanges (https://blog.coinfabrik.com/exchange/survey-decentralized-exchanges/)

You may also like

(https://blog.coinfabrik.com/smart-contracts/magic-bridge-audit/)

Magic Bridge Audit (https://blog.coinfabrik.com/smart-contracts/magic-bridge-audit/)

(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/mintingfactoryv2-baseupgradablemarketplace-kodav3upgradablegatedmarketplace/)

2 months ago

Smart Contract Audit (https://blog.coinfabrik.com/category/smart-contracts/smart-contract-audit-smart-contracts/)

MintingFactoryV2, BaseUpgradableMarketplace & KODAV3UpgradableGatedMarketplace (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/mintingfactoryv2-baseupgradablemarketplace-kodav3upgradablegatedmarketplace/)

(https://blog.coinfabrik.com/feed/)

(https://ar.linkedin.com/company/coinfabrik)

(https://twitter.com/coinfabrik)

(https://www.youtube.com/channel/UC2GmjCr7aEz-il31kqOy9aw)

(https://www.facebook.com/CoinFabrik/)

(https://www.reddit.com/r/CoinFabrik/)

(https://github.com/coinfabrik)

Made with ♥ in Buenos Aires, Argentina.