BankexCoin Token Contract: Review

Mikhail Vladimirov and Dmitry Khovratovich

14th November, 2017

This document describes the issues, which were found in BankexCoin during the code review performed by ABDK Consulting.

1. Introduction

We were asked to review a set of contracts acquired from GitHub:

- Bankexpresaleescrow.
- BankexToken.
- BankexCrowdsale.

We also got a non-technical description of the contract architecture.

2. Bankexpresaleescrow

In this section we describe issues related to the token contract defined in the bankexpresaleescrow.sol.

2.1 EIP-20 Compliance Issues

This section lists issues of token smart contract related to the EIP-20 requirements.

1. Line 19: instead of the integer `uint` there should be `uint8` (accordings to EIP-20).

2.2 Documentation Issues

This section lists documentation issues, which were found in the token smart contract.

1. Line 6: the function name `IToken` is misleading. One may think that this interface defines EIP-20 token API while it actually defines different API which is non-standard.
2. Line 38: the field name `tokenPriceInWei` and the comment `Number of token per 1 Eth` are contradicting.
3. Lines 139, 140, 141: the numbers `11210762331838`, `12106537530266`, `13245033112582` are different from the numbers in the code.

2.3 Unclear Behavior

This section lists issues of the token smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision.

1. Line 59: it seems that method `icoToken = IToken(_icoToken)` could be called multiple times. It is unclear is it supposed to be this way or not.
2. Line 86: as a transaction property, instead of `owner` there probably should be `msg.sender`.

2.4 Suboptimal Code

This section lists suboptimal code patterns, which were found in the token smart contract.

1. Line 44: the methods with modifiers `if (msg.sender == owner) _;` being called by non-owner does nothing at all. Probably `require(msg.sender == owner)` should be used instead.
2. In the line 20 `totalSupply` was declared as `300000000`. So in line 136 `totalSupply` should probably be used.

2.5 Readability Issues

This section lists cases where the code is correct, but too involved and/or complicated to verify or analyze.

1. Line 20: taking decimals into account, `3000000e2` would probably be more readable.

2.6 Other Issues

This section lists stylistic and other minor issues which were found in the token smart contract.

1. Line 14: the smart contract `TokenEscrow` looks like it implements subset of the EIP-20 token API and `iToken` interface defined above, while none of these are explicitly specified.
2. Line 58: instead `address` there should be `IToken`.

3. BankexToken

In this section we describe issues related to the token contract defined in the BankexToken.sol.

3.1 Documentation Issues

This section lists documentation issues, which were found in the token smart contract.

1. Line 9: the contract `BankexToken` actually implements `IToken` interface defined in the bankexpresaleescrow.sol, but this is not explicitly specified here.

3.2 Unclear Behavior

This section lists issues of the token smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision.

1. Line 65: it is unclear why `approve` method is not allowed in a frozen state.
2. Line 69: it is unclear why `increaseApproval` method is not allowed in a frozen state.
3. Line 73: it is unclear why `decreaseApproval` method is not allowed in a frozen state.

3.3 Suboptimal Code

This section lists suboptimal code patterns, which were found in the token smart contract.

1. Line 77: method `transferFromOwner` could probably be united with `transfer`.

3.4 Readability Issues

This section lists cases where the code is correct, but too involved and/or complicated to verify or analyze.

1. Line 77: name of the method is confusing. It actually transfers from pbkxToken, not from the owner.

3.5 Other Issues

This section lists stylistic and other minor issues which were found in the token smart contract.

1. Line 4: in this line it should be checked if latest version of StandardToken is used.

4. BankexCrowdsale

In this section we describe issues related to the token contract defined in the BankexCrowdsale.sol.

4.1 Documentation Issues

This section lists documentation issues, which were found in the token smart contract.

1. Line 130: the semantics of `_presaleConversion` variable is not clear as documentation is missing both in this file and in BankExToken.sol

4.2 Unclear Behavior

This section lists issues of the token smart contract, where the contract behavior is unclear: the business logic might be violated here, but the documentation and functional requirements are not sufficiently documented to make a clear decision.

1. Line 139: since both, `startTime` and `endTime` are inclusive, `_startTime == _endTime` makes sense, so probably `>=` should be used here.
2. Line 227: maybe the function `destroy()` should be called only when it is finalized?

4.3 Suboptimal Code

This section lists suboptimal code patterns, which were found in the token smart contract.

1. Line 104: looks like `bool status` is always true and thus not needed.

4.4 Readability Issues

This section lists cases where the code is correct, but too involved and/or complicated to verify or analyze.

1. Line 192: instead of `for` a `while` statement would be more readable.

4.5 Major Flaws

This section lists major flaws, which were found in the token smart contract.

1. Line 157: `tranches[i].price` should be checked against zero to prevent division by zero in the future sale. Otherwise any sale in this tranche will fail.

5. Our Recommendations

Based on our findings, we recommend the following:

1. Fix the major flaw.
2. Make the token EIP-20 compliant.
3. Check the issues marked as “unclear behavior” against functional requirements.
4. Refactor the code to remove suboptimal parts.
5. Improve code readability.
6. Fix the documentation and other (minor) issues.