ABDK

LiveTree Token Contract: Review

Mikhail Vladimirov and Dmitry Khovratovich

22th November, 2017

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

1. Introduction

We were asked to review a set of contracts in a private repo. Here are the copies of contracts from commit 42fc36a1935587505e50dd9200a3837f7e04f1c4:

- CrowdsaleParameters.sol.
- LiveTreeCrowdsale.sol.
- LiveTreeCrowdsaleStub.sol.
- Migrations.sol
- SeedToken.sol
- ShortAddressAtackFix.sol
- TokenRecipient.sol

2. CrowdsaleParameters

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

2.1 Readability Issues

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

1. Lines 11-16: all constants have the same value. The contract would become more readable if this value is extracted into the constant itself. For example: `uint256 constant JAN_1_2019=1546300800`. And then this constant could be used to initialize other constants.
2. Line 20: instead of `1511136000` for readability should be `// 2017-11-20T00:00:00`.
3. Line 21: instead of `1511740800` there should be presaleStartDate + 1 weeks in total.

4. Line 22: instead of `1512345600` there should be presale EndDate + 1 weeks in total.
5. Line 23: instead of `1512950400` there should be generalSaleStartDate + 1 weeks in total.

3. LiveTreeCrowdsale.sol

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

3.1 Documentation and Readability Issues

This section lists documentation issues, which were found in the token smart contract, and cases where the code is correct, but too involved and/or complicated to verify or analyze.

1. Line 7: Usually, storage variable names start with the lower case letter for example `icoClosedManually`. Semantics of `ICOStagePeriod` storage variable is unclear without the documentation comment related to it.
2. Line 163: assumes `transferFrom` never returns false. This should be mentioned in a related comment.
3. Line 223: the comment is not correct.This method may be used when either ICO is closed manually or ICO is currently inactive.
4. Line 45: changing `tokenMultiplier` to `10` will improve readability.
5. Line 147: `1000000000000000000` could be changed to `1 ether`.

3.2 Arithmetic Overflow Issues

This section lists issues of the token smart contract related to the arithmetic overflows.

1. Line 148, 160: in the statement `amount * tokensPerEth` and `tokenAmount * weiPerEth` an overflow is possible.
2. Line 164: in the statement `amount - acceptedAmount` an underflow is possible.
3. Line 292: in the statement `*` and `-` an overflow is possible.

3.3 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 153: in `/ tokenMultiplier a` rounding error occurs. It would be better to multiply `tokenAmount` by `tokenMultiplier` before comparing it to the `remainigTokenBalance`, rather than dividing `remainingTokenBalance`.
2. Line 195, 204, 215: probably `closeICO`, `reopenICO` and `setAllowRefunds` methods should log some event to notify the participants that ICO was closed.

3. Line 216: statement `require(isICOClosed())` is unclear. Actually, it is possible to have both ICO open and the refunds allowed at the same time. In this case, you should just close ICO manually, allow the refunds and then reopen ICO.
4. Line 255: fallback function does not fit into 2300 gas when being called with no data. This violates recommendations in the Solidity documentation.
5. Line 93: inside this method `msg.value` refers to the value of the surrounding call, which is confusing. This method uses `msg.value` both explicitly and through the method argument. In practice these values are always the same, but the method is written as if they may differ. So, if the method actually needs to handle situation when these two values differ, then both values should be passed explicitly as method parameters. If the method does not need to handle such situation, it should use only parameter, but not `msg.value`.

3.4 Suboptimal Code

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

1. Line 19: assignment `uint private tokenMultiplier = 10` is confusing, because the real initial value is assigned to this storage variable in the constructor.
2. Line 31: `Debug(string message)` event is never used in the smart contract.
3. Line 46: `saleWalletAddress` variable never changes its value, so it would be more efficient to just use `CrowdsaleParameters.generalSaleAddress` instead.
4. Line 47: `presaleWalletAddress` never changes its value, so it would be more efficient to just use `CrowdsaleParameters.presalePoolAddress` instead.
5. Line 62: probably, it would be better to make the function public rather than internal.
6. Line 93: instead of `finney` it would be better to use `0 Wei` or just `0`.
7. Line 99: `= 1130` should probably be moved to the last `else` statement after all of the `if` statements.
8. Line 166: some contracts just take change, some just revert if change is not zero, some use formula that guarantee that there will be no change. Sending change back is inefficient and may fail if `msg.sender` is a proxy contract.
9. Line 184: probably `changeGeneralSaleDates` method should log an event to notify participant about the dates changing.

3.5 Major Flaws

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

1. Line 43: the value of `_reasonableCostsPercentage` is not checked for validity.
2. Line 266: smart contract have no `changeOwner` method definition. But usually in order to change an owner, you usually need to be the current owner. This line would work only if `LiveTreeCrowdsale` smart contract is the owner of the `tokenReward` token, which is unlikely, because `LiveTreeCrowdsale` gets created after the `tokenReward` token has been created.

3.6 Other Issues

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

1. Line 6: `CrowdsaleParameters.sol` should be imported explicitly.
2. Line 43: instead of `address` there should be `SeedToken`.
3. Line 236: instead of `0` there probably should be `address(this)`.

4. LiveTreeCrowdsaleStub.sol

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

4.1 Readability Issues

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

1. Line 15: instead of `2512950400` there should be `2049-08-19T01:46:40+00:00`.
2. Line 16: instead of `2513555200` there should be `2049-08-26T01:46:40+00:00`.
3. Line 23: instead of `1312950400` there should be `2011-08-10T04:26:40+00:00`.
4. Line 24: instead of `1313555200` there should be `2011-08-10T04:26:40+00:00`.

4.2 Unclear Behavior

This section lists issues of the 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 4: the name of contract `LiveTreeCrowdsaleStub` indicates that it might be a testing stuff. Perhaps, there is no need to review it.

5. Migrations

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

5.1 Suboptimal Code

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

1. Line 3: contract `Migrations` hasn't been used.

5.2 Other Issues

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

1. Line 1: the other contracts in the same folder specify Solidity version 0.4.15, while `Migrations` uses an older version.

6. SeedToken

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

6.1 EIP-20 Compliance Issues

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

1. Line 31: the names of event parameters `address indexed from, address indexed to, uint256 value` differ from those defined in EIP-20 which may lead to compatibility issues.
2. Line 211: parameter `this` will be logged as `from` parameter. EIP-20 recommends logging zero address in such cases and the same method logs `owner`, not `this` as `from` parameter of another similar event.
3. Line 297: `transfer` logs non-standard four-arguments `Transfer` event, which is fine, but does not log standard `Transfer` event which directly violates EIP-20 standard.
4. Line 373: instead `this` (according to EIP-20 recommendation) there should be zero address.

6.2 Documentation Issues

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

1. Line 18: in comment actually should be mentioned `mapping`, not `an array`.
2. Line 21,23: the semantic of `vestingBalanceOf` and `vestingTimes` is unclear without documentation comment.
3. Line 25: the mapping `allowed` has two keys and their semantics is unclear without documentation comment.
4. Line 26,49: the semantic of mappings `allowanceUsed` and `vestingTimesForPools` is unclear without documentation comment.
5. Line 45: the semantic of the array `addressByIndex` is unclear without documentation comment.
6. Line 87, 116: instead of `uintDecimals` could be just `uint(decimals)`.

6.3 Arithmetic Overflow Issues

This section lists issues of the token smart contract related to the arithmetic overflows.

1. Line 60-62: despite the description given in comment, using `SafeMath` is good practice even if overflow is not possible according to contract logic. In cases there is

a bug in contract logic, `SafeMath` will help to catch it during testing, not in production.
2. Line 186: `+=` overflow is possible. Input validation doesn't take balances `_to` into account.
3. Line 206, 208, 226, 227, 286: `+=` overflow is possible.
4. Line 283: `-=` underflow is possible.
5. Line 342: `-` underflow is possible.

6.4 Unclear Behavior

This section lists issues of 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 188: `addIndex(_to)` is added the address to index even if zero tokens were transferred. Perhaps, there is no need to do this.

6.5 Suboptimal Code

This section lists suboptimal code patterns found in token smart contract.

1. Line 7: in the body of `SeedToken` smart contract events are intermixed with storage variables without obvious reason. Separating them would improve readability. Also, public storage variables are intermixed with private ones.
2. Line 28: `private _totalSupply` could be made `public totalSupply` for contract simplicity.
3. Line 40: `Issuance` event should probably have an indexed address parameter of the tokens being used.
4. Line 43: `Burning` is common word for the process of destroying tokens. `Destruction` event should have an indexed parameters address of the tokens being destroyed and probably it should have the address that initiates the token destruction.
5. Line 53: instead of `address` there should be `SeedToken`.
6. Line 69: the condition `decimals <= 18` is always true. There is no need to check it.
7. Line 95: address of the contract that emitted the event is always part of the event, no need to put it into event parameter.
8. Line 118-133: the code would become more readable and less error-prone if `approve` method would be used here.
9. Line 203: `owner` parameter will be logged as `from` parameter of `VestingTransfer` event, once the creation of the new tokens is initiated. Probably, the zero address would be better to use here.
10. Line 256: the minimum payload size for `approveAndCall` method is greater than `2*32` because even empty bytes do occupy some space.
11. Line 260: there should probably be revoked unconsumed allowance.
12. Line 272: method `transferFrom` duplicates much of the logic of `transfer` method. Such duplication makes contract harder to read an is error-prone. It would be better to extract common logic into separate internal method.

13. Line 296: `addIndex(_to)` adds destination to index even if zero tokens were transferred.
14. Line 310: there is no need to make fallback function public. In recents version of Solidity, if there is no fallback function, all incoming transfers of ether as well as all calls to non-existing methods are reverted automatically.
15. Line 315: function `checkMyVesting` is inefficient. It updates storage again and again on every transfer even if nothing vested since last transfer. I would recommend to store vested amounts incrementally, so vestingBalanceOf[sender][k] will store not yet vested amount at time vestingTimes[k]. This way current vestingBalance could be found via binary search in logarithmic time without updating storage.

6.6 Dangerous Behavior

This section lists problems related to the code vulnerability. Problems of this type require additional attention, as they can result in a serious loss.

Line 172: the “short address” protection looks redundant. There are too many ways to call a smart contract incorrectly (for example, confusing the order of parameters), of which the “short address” issue is the most known and thus usually fixed. Moreover, if another function from the same contract (or from the one which inherited `SeedToken`) calls `approve` it will likely fail.

6.7 Moderate Flaws

This section lists moderate flaws found in the token smart contract.

1. Line 179: underflow is possible. Input validation compared `_value` with `accountBalance(msg.sender)` not with `balances[msg.sender]`.
2. Line 209: `Issuance(mintedAmount)` is logged even if zero tokens were minted.
3. Line 210: target address is added to index even if zero tokens were minted.
4. Line 366,371: underflow is possible.

6.8 Major Flaws

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

Line 239: the check `_value == 0 || allowanceUsed[msg.sender][_spender] == false` seems to forbid an approval if there was no spending after the last approval and if the new allowance is not zero. It does not protect from the double-approve attack since if the allowance was spent partially, the check will pass, and the attack will work still.

6.9 Other Issues

This section lists stylistic and other minor issues which were found in the token smart contract.
Line 10, 12, 14, 16: `standard`, `name`, `symbol` and `decimals` should be constant.

7. ShortAddressAtackFix

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

7.1 Documentation Issues

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

1. Line 4: the name of modifier `onlyPayloadSize` is confusing. It would to be better to rename it to `atLeastPayloadSize` or `minPayloadSize`.

7.2 Major Flaws

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

1. Line 5: for methods called internally `require(msg.data.length >= size + 4)` would produce unexpected results. This line requires attention.

8. TokenRecipient

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

8.1 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 4: perhaps, there is no need to use `_token`. It is always equal to `msg.sender`.

9. Our Recommendations

Based on our findings, we recommend the following:

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