ERT Token Contract: Final Review

Mikhail Vladimirov and Dmitry Khovratovich

30th October, 2017

This document describes issues found in ERT Token during code review performed by ABDK Consulting.

1. Introduction

We were asked to review a set of contract files, which on 12 October 2017 were provided at GitHub:

- ESports Freezing Storage.
- ESportsConstants.
- ESportsToken.
- ESportsMainCrowdsale.
- Zeppelin/crowdsale/crowdsale.
- ESportsBonusProvider.

We got additional documentation on the contracts from the eSports Whitepaper. We found several issues, and most of them were fixed or ruled out as business requirements. The other ones still present at the most recent code version are listed here. We certify that to the best of our knowledge none of them are critical and we are not aware of any others.

2. ESportsFreezingStorage

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

2.1 Suboptimal Code

In the `release` method it is impossible to distinguish an unsuccessful attempt to release non-zero value from the successful release of zero value.

2.2 Other Issues

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

1. Instead of freezing the storage it might be better to make the contract self-destruct itself after frozen tokens at released as it does not have any utility since then (line 8).

2. In Ethereum `uint256` is usually used for storing timestamps (line 10) so we recommend it here. Note that there is no storage or gas benefit in using smaller types as this contract does not allow their optimal packing.

3. ESportsConstants

All issues in ESportsConstants.sol were fixed.

4.ESportsToken.sol

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

4.1 Suboptimal Code

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

1. Creating new contract for time-locked tokens each time (line 73) is gas-consuming. It is possible to store all the time-locked tokens in the same contract, probably exactly this one, and then use state variables to keep a track on token owners and release times. If supplementary contracts are used nevertheless, we recommend killing them to reduce the ongoing state size.
2. The loop starting in line 96 may consume arbitrary large amount of gas, so `returnFrozenFreeFunds` method cannot be safely used from other contracts. Probably it would be better to store a total frozen amount per beneficiary and then update this stored value when necessary, rather than calculating every time.

4.2 Readability Issues

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

1. `SafeMath` is implicitly used In line 111 to check that `msg.sender` has enough tokens to burn. It would make code more readable if this check will be made explicitly.

5. ESportsMainCrowdsale

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

5.1 Documentation Issues

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

1. Code would become more readable if all the percentages applied to constant for the total amount of tokens (line 16).

5.2 Readability Issues

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

1. The `internal` non-public function in line 71 is among the `public` ones which makes it harder for the reader to realize which one is public API and which one is not.

6. Modified Zeppelin's Crowdsale

In this section we describe issues related to the token contract defined in zeppelin/crowdsale/Crowdsale.sol. The contract is a derivative from OpenZeppelin's crowdsale contract, which was modified to accommodate a different sale structure.

6.1 Documentation Issues

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

1. It should be added that this code differs from Zeppelin's codebase, as it is not obvious from the repository structure.
2. The `validPurchase` method name implies that it verifies if tokens can be purchased whereas it only verifies that at least one token can be purchased (line 156).

6.2 Suboptimal Code

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

1. If `buyTokens` method were `public` rather than `internal`, the tokens could be purchased from a contract which uses `transfer` to send Ether. Currently only direct transactions and `call` method are supported (line 90).

6.3 Other Issues

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

1. Fallback function `()` does not fit into 2300 gas when called with no data, which violates Solidity guidelines (line 85).
2. There is no provision for case if `msg.sender` is a contract whose fallback function is not payable or does not fit into 2300 gas (line 136).
3. Looks like `msg.value` is used as hidden parameter modifying behavior of `buyTokens` functions. Probably it would be better to make this parameter explicit such as bool `_isBTCPurchase` (line 105).

7. ESportsBonusProvider

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

7.1 Documentation Issues

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

1. Semantic of `releaseBonus`, `addDelayedBonus` and other methods is unclear without a documentation.

7.2 Suboptimal Code

1. The token counting operation in `addDelayedBonus` performs division and then multiplication. So it might end up in accumulating more and more errors. Probably it would be better to add `_amountTokens` values as is and divide by 10 inside `releaseBonus` method (line 51).

7.4 Other Issues

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

1. According to Solidity documentation, `require` is for validating input, state and output which is not the case here, so probably `revert` would be more appropriate (line 65, 75, 80).