Set Smart Contract: Review

Mikhail Vladimirov and Dmitry Khovratovich

15th May 2018

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

1. Introduction

We were asked to review a set of contracts in a repo with commit 5fd4a.

2. SafeMath

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

2.1 Suboptimal Code

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

Line 37: the function `isMultipleOf` reverts in case `b` is zero. It is unclear if it should just return false or not.

2.2 Arithmetic Overflow Issues

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

1. Line 12,20,36: functions `min`, `max` and `isMultipleOf` not look like "safe math", but rather like simply "math".
2. Line 42: the implementation `a.mul(b).div(base)` and `a.mul(base).div(b)` may overflow (and thus revert) in cases when result would actually fit into `uint256`.
3. Line 46: the implementation `a.mul(base).div(b)` may overflow (and thus revert) in cases where one would not usually expect overflow.
   Example: one would expect that `x/1` will always return `x`. Actually, `x` is large enough this particular implementation will revert.

2.3 Other Issues

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

Line 28,32: functions `getUint256Min` and `getUint256Max` should probably be constant.

3. Set

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

3.1 Documentation and Readability Issues

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

1. Line 7: the name of the contract `Set` not related to its content.
2. Line 8,9: the semantic of `issue` and `redeem` methods is not obvious from signature. It need to be described in documentation comment.
3. Line 8,9,18: the name of the method parameter `quantity` should be prefixed with underscore ('_').

3.2 Other Issues

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

1. Line 12, 17, 18: there is no need to prefix names of events parameters `_sender` and `_quantity` with underscore ('_').
2. Line 13,17: indexing by quantity looks a bit odd.

4. SetToken

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

4.1 EIP-20 Compliance Issues

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

Line 349,357: according to EIP-20, `Transfer` event from zero address should be logged.

4.2 Effectiveness Issue

The implementation of the SetToken contract uses simple and straightforward approach to redemption, and most of the redemption-related methods has O(N) or worse complexity, where N is the number of components. Effectively this means that number of components is limited. Though, it is possible to redesign the contract so there would be no methods with complexity higher than O(1) thus allowing virtually unlimited number of components.

The logic could be simplified by implementing two-phase redemption.

- At first phase Set tokens are burned, but no component tokens are transferred. Instead, user just gains right to withdraw certain amounts of component tokens from Set token smart contract.
- At the second phase user withdraws component tokens from smart contract one by one. Such approach will make it possible to have O(1) complexity for all redemption-related methods.

4.3 Documentation and Readability Issues

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

Line 24,25: there is no need to prefix names of events parameters `_address` and `_uint` with underscore ('_').

4.4 Imbalance Issue

Line 156: some token contracts, for example Paragon, deliver fewer tokens to the recipient than specified in `transfer/transferFrom`. As a result, the Set contract will have fewer tokens of this kind than the `transferValue` number, and will fail to return as many in the `redeem` function and its relatives. Other tokens charge a fee on top of the amount being sent. All these cases are not treated correctly by the existing logic, which may result in tokens unacquirable from the contract or unavailable to the user.
To mitigate this problem, the code should also count how many tokens actually transferred to or from the contract alongside calling `transfer` or `transferFrom` methods.

4.5 Suboptimal Code

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

1. Line 30: the name of field `isRedeemed` contradicts with name of the structure.
2. Line 39: component address `token address` should be replaced by component index to make contract logic simpler and more efficient.
3. Line 75: it might be better to use `isMultipleOf` from `SafeMathUint256` instead of `(_quantity % naturalUnit) == 0)`.
4. Line 99: there should be `isNonZero` instead of `_naturalUnit > 0`.

5. Line 102-103: the check, described in these lines, would be cheaper if addresses were going strictly in ascending order.
6. Line 117: to make this check cheaper just put `require(!isComponent[currentComponent])` before this line.
7. Line 126: this operation is not required in modern versions of Solidity. It's enough just to remove fallback function for the same result.
10. Line 207: it would be more effective to use bit mask in the element `address[] excludedComponents`.
11. Line 241: the field `isRedeemed` is used in the function `partialRedeem` only and seems to be universally `false` both before and after the call to `partialRedeem`. It might make sense to remove it and use a local array to deal with redeemed tokens.
12. Line 310, 314, 322: `componentCount(), getComponents() and getUnits()` are already public. There is no need for separating getter method that returns `components.length`.
13. Line 279: it would be more effective to use bit mask in `address[] componentsToRedeem`. After that `quantities` would be removed.

4.6 Unclear Behavior or Flaw

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 279: the method `uint[] quantities` allows specifying quantities. It is hard to say is this necessary or not.
2. Line 346: perhaps, in `quantity.div(naturalUnit)` should be the check that quantity is a multiple of natural unit.

4.7 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 189: the redeem implementation is dangerous. If one component token is broken or frozen, and cannot be transferred, then no tokens can be redeemed. Modern token contracts often has freeze transfers functionality, and owner of any component token may freeze redemptions of Set tokens.

4.8 Other Issues

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

1. Line 48: the index of event should be unique.
2. Line 156, 189: there should be `require()` instead of `assert`.
3. Line 345: the method `calculateTransferValue` should be defined with modifier `view`.

Our Recommendations

Based on our findings, we recommend the following:

1. Check the effectiveness and imbalance issues (Sections 4.2, 4.4).
2. Check the issues marked as “unclear behavior” against functional requirements.
3. Refactor the code to remove suboptimal parts.
4. Fix the documentation, readability and other (minor) issues.