Rosca Smart Contract: Review

Mikhail Vladimirov and Dmitry Khovratovich

28th November, 2017

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

1. Introduction

We were asked to review a contract ROSCA.sol, also available in the original repo.

2. ROSCA

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

Our major comment is that `roscaType` is set (Line 84) at the deployment time and cannot be changed afterwards. So, the code for the other ROSCA types won't be used ever, but will make a contract more expensive to deploy, harder to read, and more error prone. It would be better to move logic common for all ROSCA types into an abstract base contract, and then inherit three separate contracts from it, one per each ROSCA type.

2.1 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 37: It is unclear from the comment and the constant name that value is actually a percentage.
2. Line 225: the name of function `startRound` is misleading. Actually, this method not only starts a new round, but also finishes the previous one (if the current round is not the last one). So it would be better to rename it as `finishRound` or `nextRound`.
3. Line 31: instead of `900` it would be better to use `15 minutes to` increase readability.
4. Line 192, 429: a combination of four non-related requirements under a single `require()` invocation decreases the code readability.
5. Line 255: changing `if` to `else if` would make the contract logic easier for understanding.

2.2 Arithmetic Overflow Issues

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

1. Line 19: the contract `ROSCA` uses the mixture of integer types of different bit-withdraws. This makes the controlling of the potential overflows harder because Solidity usually truncates a result of arithmetic operation to withdraw of a wider parameter. So, this will overflow:
   — uint8 a = 100
   — uint8 b = 200
   — uint16 c = a + b; // c is now 44, not 300 as one may expect
2. Line 193: underflow is possible.

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 72: `uint16` allows at most 64K rounds at most. It is unclear if this behavior is adequate or not.
2. Line 88: the comment "`divided by the number of ROSCA participants`" is incorrect. It indicates that rounding errors are adding up. Accumulating non-divided total discounts and dividing accumulated value before use will increase precision.
3. Line 294: the "`block's timestamp`" is used to decide the winner. The problem with the timestamp is not only it is easy to manipulate with, but it also increases too slowly. In case the number of members is much greater than the average time (in seconds) between the rounds, it becomes more predictable who definitely will not win the next round. A block hash, if the manipulation is not a concern here, is a better solution.
   If, however, the manipulation with the winner index is (or will be) a concern, we recommend a cryptographically secure key generation protocol, such as by Gennaro et al.
4. Line 536: according to the comment "`one roundPeriodInSecs after * the end of the ROSCA`" members have only `roundPeriodInSecs` after the end of ROSCA to withdraw their funds, before the owner may take them. Is this behavior correct or not?
5. Line 566: for preventing re-entry it would be better to use `nonReentrant`.

2.4 Suboptimal Code

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

1. Line 40: the address variable `0x5bb8D0DfdAf3a03cF51F9c8623C160a4c021E522` should probably be set in constructor.
2. Line 190: passing the addresses of all of the members to the constructor effectively limits the maximum number of the members which a related contract may have. To

eliminate this limitation, there should be an ability to add more members after the contract was deployed.
3. Line 229: the condition `currentRound != 0` may be false only in if `currentRound` is overflowed, because `currentRound` is initially set to 1 and then only incremented.
4. Line 352: flag `winnerSelectedThroughBid` makes method overcomplicated. Probably, in case a winner was selected through the bid, it would be better to calculate `winnerIndex` from `winnerAddress` in a calling code and then pass correctly calculated winnerIndex to this method.
5. Line 384: condition `<=` is confusing, because `msg.value` could never be less than zero.
6. Line 611: sending ether via selfdestruct is not the same as sending them via send/transfer. Selfdestruct does not call the code of destination contract. Probably it would be better to try `send` first.

2.5 Major Flaws

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

1. Line 206: the loop `i < members_.length` will never end in a case when `members_.length >= 2^16`.
2. Line 207: while zero member address plays special role in the code of this smart contract, nothing prevents the deployer of this contract from adding zero address as one of member which would break the contract logic.
3. Line 262: `winnerIndex` is uninitialized local variable.
4. Line 302: due to the accumulated errors in `totalDiscounts` `>=` it may evaluate to false even if the member already came out of the debt.
5. Line 473: in case of using `totalDiscounts`, due to the accumulated errors, members will be able to withdraw less than they should be able to.
6. Line 609: smart contract will become self-destroyed even if `tokenContract.transfer(foreperson, balance)` transfer has failed, making tokens lost forever.

2.6 Other Issues

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

1. Line 45,49,50,55: parameters `user`, `winnerAddress`, `user` and `bidder` should probably be indexed.
2. Line 54: probably, the address of the user who tried to withdraw should be included into `LogCannotWithdrawFully` event as an indexed parameter.
3. Line 226: the cast `uint` looks redundant. Solidity will automatically case `uint16` to `uint256` here.

3. Our Recommendations

Based on our findings, we recommend the following:

1. Fix the major flaws.
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.