Ripio Token Audit

OPENZEPPELIN | OCTOBER 23, 2017

Security Audits

The Ripioteam asked us to review and audit their Ripio Credit Network Token(RCN) and crowdsale contracts. We looked at the code and now publish our results.

The audited contracts are located in the ripio/rcn-tokenrepository. The version used for this report is the commit `4bf441ae919f2580dcfeca59917b81bb30d2b856`.

Here's our assessment and recommendations, in order of importance.

Update: *The Ripio team has followed most of our recommendations and updated the contracts. The updated version is at commit* `c0fcee719ed9564992fd10423d2edba9e2e95f3`.

Critical Severity

No critical severity issues were found.

High Severity

No high severity issues were found.

Medium Severity

Reuse open source contracts

The contracts `StandardToken`, `MintableToken` and parts of `RCNCrowdsale` are very similar to code found in OpenZeppelin's `StandardToken`, `MintableToken` and `Crowdsale` contracts. Reimplementing functionality instead of reusing public and already

Use safe math

There are some unchecked math operations in the code (see thisand this, for example). It's always better to be safe and perform checked operations. Consider using a safe math library, or performing pre-condition checks on any math operation.

Update: *Fixed in* `1dc13ab` *and* `0daf25a`.

Low Severity

Token metadata should be in token contract

The public variables `name`, `symbol`, and `decimals` are definedin the `RCNCrowdsale` contract. They should be defined in the token contract, as suggested by ERC20. Define a new contract `RCNToken` inheriting from `MintableToken` and add the public variables there. Change the crowdsale contractto create an instance of `RCNToken` instead of `MintableToken`.

Update: *Fixed in* `be5255a`.

Using block numbers to specify start and end

The crowdsale contract uses block numbers to specify when it starts and when it ends. The current recommendation is to use timestamps instead. The risk of miner manipulation of timestamps is very low for this use case, and due to the Difficulty Bombit is now very difficult to correctly estimate future block times. Consider switching to timestamps.

Update: *Fixed in* `50a51b7`.

ERC20 compliance

ERC20 specifies `decimals` to be a value of type `uint8`. It is declaredas a `uint256` variable in `RCNCrowdsale`. Consider changing it to `uint8`. If so, it will be necessary to cast to `uint256` when using the variable for arithmetic such as when expressing token amounts as `400 * 10**decimals`. Consider defining a state variable `TOKEN_UNIT = 10 ** uint256(decimals)` to write `400 * TOKEN_UNIT` in these cases.

Update: *Fixed in* `0daf25a`.

Constructor parameter validation

Consider performing sanity checks to validate `RCNCrowdsale`'s constructor parameters. Check that `_fundingStartBlock < _fundingEndBlock` and that the addresses `_ethFundDeposit` and `_rcnFundDeposit` are not `0x0`.

Update: *Fixed in* `70a42f2`.

Notes & Additional Information

- Consider using `require` instead of `if (...) throw`. `throw` has been deprecated since Solidity 0.4.13.
- The comment in line 72 of `RCNCrowdsale` seems to be unrelated to the content of the line.
- Keep in mind that there is a possible attack vector on the `approve` / `transferFrom` functionality of ERC20 tokens, described here. Consider implementing one of the proposed mitigations, or using the ERC20 implementation from OpenZeppelin which already has one in place.
- The state variable name `raised` is somewhat misleading: it sounds like it stores the amount of ether raised, but it stores the amount of tokens minted. Furthermore, it is redundant because that value can already be found in the variable `totalSupply` which is updated in every `mint` operation. Consider removing `raised` altogether.
- The event `LogRefund` is not emitted anywhere (there is in fact no refund functionality). Consider removing it.
- The comment in line 75 of `RCNCrowdsale` is describing a previous version of the contract in which investors by default had a fixed cap, and whitelisted investors had no cap. To be accurate it should now read "if sender is not whitelisted *or* exceeds their cap".
- `mint` and `finishMinting` have a boolean return value meant to indicate success or failure. The return value is later ignored in `RCNCrowdsale`, which doesn't cause any problems because in this implementation it always return `true`. Nonetheless, ignored return values can cause problems in future changes to the code. Consider removing them.

Update: *Most of the suggestions were implemented in the updated version.*

follow best practices and reduce potential attack surface.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Ripio Credit Network Token contracts. We have not reviewed the related Ripio Credit Network project. The above should not be construed as investment advice. For general information about smart contract security, check out our thoughts here._

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits

OpenZeppelin