iEx.ec RLC Token Audit

APRIL 11, 2017 | IN SECURITY AUDITS | BY OPENZEPPELIN SECURITY

The iEx.ec team asked us to review and audit their new RLC token contract code. We looked at their contracts and now publish our results.

The audited contracts are at their iExecBlockchainComputing/rlc-token GitHub repo. The version used for this report is commit 3d9aa99ba33bb035c59740a621b1f21cd45cbac5. The main contract files are Crowdsale.sol and RLC.sol.

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

Update: iEx.ec team responded to this review here.

Severe

We haven't found any severe security problems with the code.

Potential problems

Use safe math

There are many unchecked math operations in the code. 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.

Timestamp usage

There's a problem with using timestamps and **now** (alias for **block.timestamp**) for contract logic, based on the fact that miners can perform some manipulation. In general, it's better not to rely on timestamps for contract logic. The solutions is to use **block.number** instead, and approximate dates with expected block heights and time periods with expected block amounts.

The Crowdsale.sol and RLC.sol contracts use timestamps in several places (for example, lines 69, 116, 145, and 295 of Crowdsale.sol and lines 37 and 49 of RLC.sol). The risk of miner manipulation, though, is really low. The potential damage is also limited: miners could only slightly manipulate when crowdfunding ends and starts. We recommend the team to consider the potential risk of this manipulation and switch to **block.number** if necessary.

For more info on this topic, see this stack exchange question.

Warnings

Use of send

Use of **send** is always risky and should be analyzed in detail. Two occurrences found in line 255 and line 297 of Crowdsale.sol.

- Always check send return value: OK.
- Consider calling send at the end of the function: Warning. Use at line 297 could be moved down.
- Favor pull payments over push payments: line 255

is a pull payment, line 297 is a push payment. Consider changing line 297 to use asyncSend.

For more info on this problem, see this note.

Usage of magic constants

There are several magic constants in the contract code. Some examples are:

- https://github.com/iExecBlockchainComputing/rlc-token/blob/3d9aa99ba33bb035c59740a621b1f21cd45cbac5/contracts/Crowdsale.sol#L295
- https://github.com/iExecBlockchainComputing/rlc-token/blob/3d9aa99ba33bb035c59740a621b1f21cd45cbac5/contracts/Crowdsale.sol#L148

Use of magic constants reduces code readability and makes it harder to understand code intention. We recommend extracting magic constants into contract constants.

Bug Bounty

Formal security audits are not enough to be safe. We recommend implementing an automated contract-based bug bounty and setting a period of time where security researchers from around the globe can try to break the contract's invariants. For more info on how to implement automated bug bounties with OpenZeppelin, see this guide.

Unused isMaxCapReached

isMaxCapReached is not used in Crowdsale.sol, but seems like it could be used in line 186 or in the finalize function.

Avoid duplicated code

Duplicate code makes it harder to understand the code's intention and thus, auditing the code correctly. It also increases the risk of introducing hidden bugs when modifying one of the copies of some code and not the others.

The logic in transfer, transferFrom, balanceOf, allowance and approve is very similar to the provided methods in StandardToken and could be refactored to avoid repetition. There is no need to rewrite the parent's contract logic: consider using the super keyword to access StandardToken implementation of methods, like so:

```
function transfer(address _to, uint _
  return super.transfer(_to, _value);
}
```

Furthermore, the burn method is a close rewrite of **transfer**, consider having burn call transfer instead.

Finally, Crowdsale.sol could use OpenZeppelin's Ownable instead of implementing the same idea.

Use latest version of Solidity

Current code is written for an old version of solc (0.4.8). We recommend changing the solidity version pragma for the latest version (`pragma solidity ^0.4.10;`) to enforce latest compiler version to be used.

Fail early and loudly

A simple yet powerful programming good practice is to make your code fail as promptly as possible. And be loud about it. We want to avoid a contract failing silently, or continuing execution in an unstable or inconsistent state. Consider changing all precondition checks to `throw`ing to follow this good practice.

Some parts of the code do this correctly, but we'd like to see more consistency on this pattern.

Some places where this could be improved are:

- https://github.com/iExecBlockchainComputing/rlc-token/blob/3d9aa99ba33bb035c59740a621b1f21cd45cbac5/contracts/Pausable.sol#L16
- https://github.com/iExecBlockchainComputing/rlc-token/blob/3d9aa99ba33bb035c59740a621b1f21cd45cbac5/contracts/Pausable.sol#L22
- https://github.com/iExecBlockchainComputing/rlc-token/blob/3d9aa99ba33bb035c59740a621b1f21cd45cbac5/contracts/Ownable.sol

Additional Information and Notes

- Good work using OpenZeppelin! 🙂
- Code is well-written, in general, and it was easy to understand intention of the developer. Nice work.
- RLC token is ERC20 compliant.
- Remove the **closeCrowdsaleForRefund** function (as the comment above it says). Not removing this function would be a severe security problem, but we mention it as an informational note because of the comment above it, which means the team is aware of this.
- Same with **finalizeTEST**.
- Remove commented code on line 149 of Crowdsale.sol.
- Same with line 294.
- Same with lines 260–281.
- Remove unnecessary code in lines 101–104 of RLC.sol. Since solidity 0.4.0, contracts will throw by

default when funds are sent, unless you add a `payable` fallback function.

- Remove unnecessary variable burnAddress, and just set the balances to 0 to burn tokens.
- You probably want the unlock function in RLC to have the onlyOwner modifier. Even though there's no real problem with other user to call the method, it's recommended to have full control over when those important methods are called.
- Unlock function in RLC is unnecessary, and **onlyUnlocked** modifier could use unlockBlock directly for its logic.
- Fix indentation of lines 250 to 258 of Crowdsale.sol.
- Some state variables of RLC.sol could be constant for added safety.
- Some functions on Crowdsale.sol could be declared constant.
- transferRLC function doesn't add any functionality and may add confusion.
- Comments in lines 160 and 161 of Crowdsale.sol have typos: "collcted".

Conclusions

No severe security issues were found. Some changes were recommended to follow best practices and reduce potential attack surface.

Update: iEx.ec team responded to this review here.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the RLC token. We have not reviewed the related iEx.ec project. The above should not be construed as investment advice or an offering of tokens. For general information about smart contract security, check out our thoughts here.

Security Audits

- If you are interested in smart contract security, you can continue the discussion in our forum, or even better, join the team 🚀
- If you are building a project of your own and would like to request a security audit, please do so here.

RELATED POSTS

SECURITY AUDITS

Metal Token Audit

The Metal team asked us to review and audit their new Metal Token contract code. We looked at their...

READ MORE

by OpenZeppelin Security

SECURITY AUDITS

Everus Token Audit

The Everus team asked us to review and audit their Everus Token (EVR) contract. We looked at the...

READ MORE

by OpenZeppelin Security