Ecomi Token Sale contracts Audit Report

Prepared by:

- Alex Tikonoff, alexf@blockchainlabs.nz
- Matt Lough, matt@blockchainlabs.nz

Report:

- June 06, 2018 – date of delivery
- June 27, 2018 – last report update

Preamble

This audit report was undertaken by BlockchainLabs.nz for the purpose of providing feedback to Ecomi.

It has subsequently been shared publicly without any express or implied warranty.

Solidity contracts were sourced from GitHub `17b96f`

We would encourage all community members and token holders to make their own assessment of the contracts.

Scope

The following contracts were subject for static analysis:

- OMICrowdsale.sol
- OMIToken.sol
- OMITokenLock.sol

Framework used

This project is using openzeppelin-solidity v1.8.0, which is not the latest version and lacks some of the useful features of latest Solidity releases, such as constructors, reverting reasons and emitting events.

Repository "zeppelin-solidity" was renamed to "openzeppelin-solidity" in May, 2018. If y uses `yarn` to install dependencies, the changes in the contracts "import" statements are required, since `yarn` distinguishe these repos and import paths from the contract ton't be found.

On the contrary, `npm` warns about this situation, but installs old "zeppelin-solidity" repository, so no extra actions are required.

No original OpenZeppelin Solidity framework contracts were changed.

Issues

Severity	Description
Minor	A defect that does not have a material impact on the contract execution and is likely to be subjective.
Moderate	A defect that could impact the desired outcome of the contract execution in a specific scenario.
Major	A defect that impacts the desired outcome of the contract execution or introduces a weakness that may be exploited.
Critical	A defect that presents a significant security vulnerability or failure of the contract across a range of scenarios.

Minor

- Contract defining variables are not defined by default `best practices`
  There is no check that `tokenAllowance`, `allowanceProvider`, `crowdsale` are valid. Consider addi them to the contract constructor.
  View on GitHub

  - [x] Fixed: 81daf7

- Old Solidity version `best practices`
  The current Solidity release version is 0.4.24. The project is using 0.4.18, which lacks some of the useful features of latest releases, such as constructors, reverting reasons and emitting events.
  View on GitHub

  - [x] Fixed: d32b56

- Zeppelin Solidity framework was renamed `testability`
  Repository "zeppelin-solidity" was renamed to the "openzeppelin-solidity" in May, 2018. If y uses yarn to install dependencies, the changes in the contracts "import" statements are required, since yarn distinguish these repos and import paths from the contract won't be found.
  View on GitHub

  - [x] Fixed: 81daf7

- Unnecessary limits checking `correctness`
  `for (uint256 i = 0; i < lock.locks.length; i = i.add(1)) {` – There is no r to use SafeMath.sol lab in this case since there is limits check already presented when checking `; i < lock.locks.lenght ;`
  View on GitHub

  - [x] Fixed: 75103c

- require() vs. modifiers `best practices`
  `require(!isFinalized);` These lines use aapproach which is different to the rest of project with `whenNotPaused` and `whenNotFinalized` modifiers in the same contract.
  View on GitHub

  - [x] Fixed: 1f3837

- Solidity variables should be used instead of hardcoded numbers `best practice`
  `uint day = 86400; tokenLock.lockTokens(_beneficiary, day.mul(7), _tokenAmount);`
  could be changed to
  `tokenLock.lockTokens(_beneficiary, 1 weeks, _tokenAmount);`
  View on GitHub

  - [x] Fixed: 3053fd

Moderate

- Multiple reverting `correctness`
  If `require(_release(_beneficiary))` fails by some reason, `releaseAll()` function will also fail. It could be better to log failed release and continue the loop.
  View on GitHub

  ✓ Fixed. The logic was moved to the web.

- Any address could be used as Crowdsale or AllowanceProvider addresses `correctness`

  ```
    function setCrowdsaleAddress (address _crowdsale) public onlyOwner
   returns (bool) {
                  crowdsale = _crowdsale;
          return true;
    }
  ```

  Consider validati that crowdsale contract is an actual crowdsale contract, not just an address.
  View on GitHub

  ✓ Fixed: 81daf7

- Finalization crowdsale could be incomplete `correctness`
  The Crowdsale `_finalization()` is an internal function and could be called only that lines 166, 170, 174.
  But, `_updatePurchasingState()` could be called only from `buyToken()` from Crowdsale.sol.
  That means if there are not enough purchases, the developers will be forced to buy their tokens themselves.
  View on GitHub

  ✓ Fixed: 2c1c3

- Variables assigned when it's possible to avoid them and thus save the gas `gas optimisation`
  Bariables that used not more than once could be removed in order to save on gas.
  View on GitHub

  ✓ Fixed: 1c2a01

Major

- None found

Critical

- Token transfer to the wrong account `correctness`
  The `transferFrom()` fsends tokens to the `msg.sender`, which is ok if the internal function `_release()` called from public function `releaseToken()`.
  But when `_release()` icalled from the `releaseAll()`, then msg.sender is an owner (not the actual beneficiary) and the token will be transferred to that owner's (contract) account.
  View on GitHub

  - [x] Fixed: 8393fd

Observations

No real token transfers

TockenLock contract does not transfer tokens during the Sale. All tokens are virtually deposited to the Lock contract and could be transferred to the customers after 7 days after the Sale is finished.

The process of sending tokens is possible only when particular AllowanceProvider Contract has that tokens on its balance.

That contract is not under audit, so we can not grant that AllowanceProvider will have required amount of tokens to distribute to buyers.

Exchange rate updated from outside

The token exchange rate is a subject to external changes and could be set by tContract Owner to any value. We encourage customers to check the rate thoroughly before buying.

WhitelistedCrowdsale.sol

Only whitelisted accounts allowed to check the token purchases from its own and other accounts, but Ethereum blockchain is transparent to everyone and anyone could check token purchases history for this project.

Nevertheless, WhitelistedCrowdsale.sol contract modifier `isWhitelisted` was used, just once and just to restrict purchase histoy check.

Latest Solidity versions benefits are not used

It is possible to use `emit` and `constructor` keywords, to increase readability but that is up to authors, use them or not.

Functions state mutability can be restricted to pure

Sfunctions can be marked explicitly with `pure` attribute to clarify that they do not change anything on the blockchain.

Conclusion

The developers demonstrated an understanding of Solidity and smart contracts. They were receptive to the feedback provided to help improve the robustness of the contracts.

We took part in carefully reviewing all source code provided

Overall we consider the resulting contracts following the audit feedback period adequate and any potential vulnerabilities have now been fully resolved. These contract has a low level risk of ETH and OMI being hacked or stolen from the inspected contracts.

___

Disclaimer

Our team uses our current understanding of the best practises for Solidity and Smart Contracts. Development in Solidity and for Blockchain is an emerging area of software engineering which still has a lot of room to grow, hence our current understanding of best practices may not find all of the issues in this code and design.

We have not analysed any of the assembly code generated by the Solidity compiler. We have not verified the deployment process and configurations of the contracts. We have only analysed the code outlined in the scope. We have not verified any of the claims made by any of the organisations behind this code.

Security audits do not warrant bug-free code. We encourage all users interacting with smart contract code to continue to analyse and inform themselves of any risks before interacting with any smart contracts.