SMART CONTRACTS EXPERTISE — RIGHT WAY TO SUCCESS
Qurrex Smart Contract Audit

If you have any questions concerning smart contract design and audit, feel free to contact zoia@oceanico.io

Content

Description of the complex of procedures for auditing a smart contract

1. Primary architecture review

- Checking the architecture of the contract.
- Correctness of the code.
- Check for linearity, shortness and self-documentation.
- Static verification and code analysis for validity and the presence of syntactic errors.

2. Comparison of requirements and implementation

- Checking the code of the smart contract for compliance with the requirements of the customer code logic, writing algorithms, matching the initial constant values.
- Identification of potential vulnerabilities

Terms of Reference for the creation of a smart contract

Token specification:

- Token symbol: QRX
- Token full name: Qurrex
- Token standard: ERC20
- Total supply: 70m (70000000)
- Sales supply: 55m (55000000)
- Hard Cap: 55m QRX
- Token burn: Yes

Non-public Pre-Sale smart contract specification:

- Sale supply: 17m (17000000)
- Start date: 07.02.2018 09:00:00 UTC
- Stop date: 09.02.2018 09:00:00 UTC
- Sale price for contribution:
    - from 1 to 10 ETH: 1 ETH = 840 QRX;
    - from 10 to 30 ETH: 1 ETH = 920 QRX;
    - from 30 to 150 ETH : 1 ETH = 1000 QRX;
    - from 150 and more ETH : 1 ETH = 1160 QRX. 1 ETH or BTC / LTC equivalent
- Min volume: 1 ETH or equivalent. For less values we will return funds with deducting transaction commission.
- The volume of credited tokens will be rounded up
- Accepted currency: ETH/BTC/LTC. Currency conversion will be processed with the rate of exchange actual for payment moment

List of audited files

Github: https://gist.github.com/siman/1080e7d8d225b8d4fe0331b21ce66596

The following 1 .sol files were the object of the audit:

- QurrexPrivateSale.sol

Review of smart contract #1

Qurrex smart contract review #1

https://gist.github.com/siman/1080e7d8d225b8d4fe0331b21ce66596#file-qurrexprivatesale-sol

1. Architecture Recommendations

1.1. Source code style & syntax corresponds to general recommendations. All needed places in the source code are commented.

1.2. Token contract contains functionality which does not relate to Token functionality (it's recommended to move it to crowdsale contract). For example, remove functionality related to buying buy BTC from Token to Crowdsale contact. Main rule: Token contract will not be changed ever, crowdsale is temporary solution. Not to include temporary functions to Token contract.

1.3. Contracts are overloaded with superfluous logic:

1.3.1. superfluous counters, the task of which is fulfilled by Events

1.3.2. duplicating restrictions

1.3.3. pre-generation of tokens

1.4. CrowdSale contract functionality is suspicious for the contributors. For example, the contract's owner can withdraw money but the tokens will not even be unlocked.

1.5. The code is not be optimized by gas usage.

1.6. Contract's methods are inconvenient for calling from transactions (details in paragraph 2.4).

1.7. The logic for calculating the rate of the token (bonuses / discounts) for buying buy not ETH (BTC, for example) is outside of the crowdsale contract. It's suspicious for the contributors and not recommended.

2. General Recommendations

2.1. The implementation of the token differs from the standard (https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/ERC20/StandardToken.sol) Instead of revert, it returns false. There may be problems when integrating the token with other systems, such as decentralized exchanges.

2.2. No Sense in creating the seller address, and accur the charge the totalSupply. You can do "minting" (mint) tokens. A good solution is offered here (https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/ERC20/MintableToken.sol). Easier implementation, fewer bugs, less gas.

2.3. MultiOwnable: array of owners usage is unclear. As a result, removeOwner requires a shift in the elements of the array.

2.4. Instead of revert, false is used, which is convenient only if the method calls another contract. The result of calling the method in the transaction is not visible (and is not stored in the blockchain), and only its status is visible (field "TxReceipt Status").

2.5. CommonBsToken: locked/unlocked implementation duplicate Pausable.

2.6. To ensure that the contract does not accept funds, you dont have to implement the payable function. You do not need to do its implementation with revert.

2.7. totalSales is present both in the sale and in the token. For the work of the contract, the field is useless. For statistics, it is better to use events.

2.8. totalSales indicates "Total amount of sales / contributions during this crowdsale", but in fact it is not the number of contributions, because user can make many purchases from one address, but totalSales counter will be increased every time.

2.9. maxCapInTokens duplicates the totalSupply functionality and partially saleLimit functionality.

2.10. Collection of backers is not used in the logic of the work of the contract.

2.11. 435: Superfluous result check. The sell method always returns as true, or as revert/assert.

2.12. Some view methods can be replaced with pure.

2.13. In BsTokenIssuer, the variables in the state are used for a group of nowhere-used fields "Reasons why tx can be rejected". We recommend that you replace them with constants, or with enum, or even remove them from the contract.

2.14. 641: using uint16 does not make sense, potentially leading to incorrect method operation (if the total value of transactions is greater than 32768).

2.15. Please note that beneficiary's address can not be changed without a complete re-creation of sales contracts.

3. Gas Recommendations

3.1. 309: if (saleLimit> 0) require (safeSub (saleLimit, safeAdd (tokensSold, _ value))> = 0); - it is important that safeSub does not give a negative value, but will indicate assert, which use all the gas.

3.2. Many immutable values (for example: token name, decimal, etc.) can be defined as constants and not stored in a contract state.

3.3. 511: superfluous line, the Ownable constructor will execute this code.

3.4. To store and transfer ENUM values, we don't recommend the usage of uint8. This leads to unnecessary converting uint to uint8. In the keys to the collections, we dont recommend using uint8, since in any case the key is bytes32 calculated from sha3 from the original value.

Results of contract audit

https://gist.github.com/siman/1080e7d8d225b8d4fe0331b21ce66596#file-qurrexprivatesale-sol

The information in this report is a list of recommendations what needs to be done to ensure the quality and security of the smart contract.

The OCEANICO experts conducted the verification of the smart contract. Based on the results, the customer's developers were given recommendations for optimizing the smart contract code.

This smart contract complies with the specifications specified in the terms of reference.

During the audit of the contract, critical errors and possible vulnerabilities were not identified.

If changes are made to the functionality of the contract, please submit the smart contract for re-examination to the OCEANICO experts (zoia@oceanico.io).