BlockchainLabs.NZ
Blockchain Solution Design & Development

KyberNetwork Smart Contracts Audit Report

Preamble

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

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

Solidity contracts were sourced from the public Github repo KyberNetwork/smart-contracts at commit 7c1170407b2dd6df061c9648c2ec9955d7cb6089

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, dynamic and functional analyses:

- KyberNetwork.sol
- KyberReserve.sol
- ConversionRates.sol
- ExpectedRate.sol
- FeeBurner.sol
- VolumeImbalanceRecorder.sol

Focus areas

The audit report is focused on the following key areas - though this is not an exhaustive list.

Correctness

- No correctness defects uncovered during static analysis?
- No implemented contract violations uncovered during execution?
- No other generic incorrect behaviour detected during execution?
- Adherence to adopted standards such as ERC20?

Testability

- Test coverage across all functions and events?
- Test cases for both expected behaviour and failure modes?
- Settings for easy testing of a range of parameters?
- No reliance on nested callback functions or console logs?
- Avoidance of test scenarios calling other test scenarios?

Security

- No presence of known security weaknesses?
- No funds at risk of malicious attempts to withdraw/transfer?
- No funds at risk of control fraud?
- Prevention of Integer Overflow or Underflow?

Best Practice

- Explicit labeling for the visibility of functions and state variables?
- Proper management of gas limits and nested execution?
- Latest version of the Solidity compiler?

Analysis

- Functional Analysis
- Test Coverage
- Work Paper

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

- Shouldn't be possible to set a rate 0 - `Bug` In the `Whitelist.sol` contract, the `setSgdToEthRate` takes a parameter `_sgdToWeiRate` and assigns it's value to the public variable `weiPerSgd` *without* checking that it might be 0 and set all subsequent whitelistings' caps to 0. #L40 View on GitHub

  Not Fixed
- Non standardised Naming - `Correctness` In the `Whitelist.sol` contract, the `setSgdToEthRate` takes a parameter `_sgdToWeiRate`and assigns it's value to the public variable `weiPerSgd`. These naming conventions for such a simple setter function change too much and might lead to a mistake in reading it or using it. View on GitHub

  Not Fixed
- All of the permissions are handled in the same file. - `Gas optimization` All the logic is loaded in contracts even when they only use a portion of the code. Some contracts only need to know about `admin`, others only about `alerters` and others only about `operators` We suggest having the logic split into different files under one folder and only request what is needed. View on GitHub

  Not Fixed
- No validation on for value in setQuantityFactor setter function - `Question` `setQuantityFactor` in `ExpectedRate.sol` has no validation of the `newFactor` value which is input by an operator. Is there an upper or lower bound for this value? #L25-L28 View on GitHub

  Fixed

- Setter function, setMinSlippageFactor, validates the existing value, not the new value - `BugsetMinSlippageFactor()` in `ExpectedRate.sol` checks that the value of the current `minSlippageFactorInBps` value is valid, but does not check the operator inputted value, `bps`. #L33 View on GitHub

  Fixed

- The getRate function could be a bit more DRY - `Best practice` For example this line #L219 There is a function `getBasicRate` which could be used the `rate = ...` line doesn't need to be included twice. You can expand this basic principle more to make the big `if (buy) { ...` case less intimidating with less repeating. This is a very minor issue and does not effect the security on the contract. View on GitHub

  Won't Fix

- recordImbalance function has unnecessary return - `Best practice` #L192 The function does not have a return value so adding a return statement is unneeded, and potentially confusing because one would assume that the `addBalance` function being called on the return line would return a value, which it doesn't either. This issue is very minor and does not affect the security of the contract. View on GitHub

  Won't Fix

Moderate

- None found

Major

- A malicious admin could deploy a malicious Feeburner Contract - The FeeBurner Contract could be changed to burn all of the `KNC` tokens of a reserve by modifying `burnReserveFees` and registering the new contract in the network. View on GitHub

  Mitigated - See point 8. in the Technical Details section of Kyber's post

Critical

- None found

Observations

During our audit we kept in mind this central statement:

"Provided that the users input reflects the users intent - i.e., min conversion rate, and the token address is of a reputable token, then even if network admin is malicious, user funds (beside gas costs) are never at risk"

One of the issues we found was around a malicious admin deploying a fake token, from the previous statement we can ignore this issue and assume there will be a front end solution for the user to confirm they always have the correct token address.

One issue worth highlighting is this one, a malicious admin could potentially burn the supply of KNC tokens from a Kyber Reserve. We are satisfied that public awareness of this issues is sufficient to mitigate this issue (See point 8. in the Technical Details section of Kyber's post)

The issues we spotted were to do with a Kyber Admin being malicious, not the Reserve Operators, we could not find any exploits for them to take a users tokens or ETH. One improvement may be splitting out the `KyberNetwork::setParams()`function into multiple functions and applying a different privilege level for the more important parameters.

Ideally a decentralized exchange would require buy-in from users and operators to effect any significant changes. In which case, changing a system contract address would be better managed by requiring a redeployment of the entire Kyber Network and convincing users that this is a good idea.

Kyber has clearly articulated their trust model is that Reserve Operators are required to trust Kyber Admins, however users are not required to trust either Reserve Operators or Kyber Admins.

Conclusion

The developers demonstrated a great 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, including both static and dynamic testing methodology.

We were satisfied that a user's ETH or ERC20 Tokens aren't at risk of being hacked or stolen in the interaction with Kyber Network contracts provided the user input is correct.

Overall we consider the resulting contracts following the audit feedback period to be robust and have not identified any critical vulnerabilities. Minor issues have been fixed where appropriate and the only Moderate issue has been mitigated.

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.