PUBLIC

Security Audit

of KyberNetwork's Smart Contracts

January 9, 2019

Produced for

by

Table Of Content

Foreword . . . 1

Executive Summary . . . 1

Audit Overview . . . 2

1. Scope of the Audit . . . 2

2. Depth of Audit . . . 2

3. Terminology . . . 2

Limitations . . . 4

System Overview . . . 5

Best Practices in KYBERNETWORK's project . . . 6

1. Hard Requirements . . . 6

2. Soft Requirements . . . 6

Security Issues . . . 7

1. Hijacking OrderbookReserve possible  ✓ Fixed . . . 7

2. Failed token approval  ✓ Fixed . . . 7

3. Dependency on MakerDAO price feed ✓ Acknowledged . . . 8

4. Possibility to steal tokens owned by Network  ✓ Fixed . . . 8

5. Gas implications when calling token contracts ✓ Addressed . . . 8

6. Possibility to manipulate the `OrderBook`  ✓ Fixed . . . 9

Trust Issues . . . 10

1. Users need to trust the traded tokens ✓ Acknowledged . . . 10

Design Issues . . . 11

1. Dead code in `OrderList`  ✓ Fixed . . . 11

2. Unnecessary loop iteration L . . . 11

3. Avoid assigning to function parameters L ✓ Fixed . . . 11

4. Struct and calculation optimization in `OrderIdData` L ✓ Acknowledged . . . 12

5. Unnecessary loop iteration L ✓ Fixed . . . 12

6. Duplicate getters for `TAIL_ID` and `HEAD_ID` L ✓ Fixed . . . 12

7. Consistent Handling of omitted return values L . . . 13

8. Specification mismatch L ✓ Addressed . . . 13

9. Possible out-of-gas exception L . . . 13

10. Inconsistent Naming in Interface L ✓ Fixed . . . 13

Recommendations / Suggestions . . . **15**

Disclaimer . . . **16**

Foreword

We first and foremost thank KYBERNETWORK for giving us the opportunity to audit their smart contracts. This documents outlines our methodology, limitations, and results.

– ChainSecurity

Executive Summary

The KYBERNETWORK smart contracts have been analyzed under different aspects, with a variety of custom and publicly available tools for automated security analysis of Ethereum smart contracts, as well as with expert manual review.

Overall, CHAINSECURITY found that KYBERNETWORK employs good coding practices and succeeded in expanding their business to permissionless exchanges.

However, CHAINSECURITY managed to uncover several issues including a critical security issue which allows anyone to block the listing of specific tokens. Therefore, these issues need to be addressed. CHAINSECURITY also discovered multiple design issues that when fixed, can improve the overall system. Finally, CHAINSECURITY added suggestions to further improve the smart contracts.

KYBERNETWORK has addressed all the critical and high severity issues by applying the fix to the corresponding code. CHAINSECURITY confirms that those issues are now fixed and resolved and does not contain further vulnerability.

Audit Overview

Scope of the Audit

The scope of the audit is limited to the following source code files. All of these source code files were received on December 10, 2018. Updated code post our intermediate report is received from the KyberNetwork on January 8, 2019 (corresponding to the commit `f710238fab6a9761a441bb5949f3c4a8bbd90563`):

File	SHA-256 checksum
./ExpectedRate.sol	c92137913f37fdae9826b71a2eed5bcd074250775214b97ddc18107e9ed91ba4
./ExpectedRateInterface.sol	183a729d1a96314299cad447956c4597a7170d4271356ae82870537aff432bd8
./FeeBurner.sol	c21fd936a99595872caa57e5543522c3fef4acc7a5870b41ee4b6f7bc4494b05
./FeeBurnerInterface.sol	e6b159541b0c46b5535ebde59a441a5498680e509dc827ab0de423b764afcf0a
./KyberNetwork.sol	91ead372e25257d86b73b719de0dc9e3b9bebc288a590263ca83ef5c5f363d2d
./KyberNetworkInterface.sol	273c07e80a40f7944d6832d681afe3066f2641463fbcd3f9b77fb69f9c24caeb
./KyberNetworkProxy.sol	7320bd6ae2f310c5d3d2f74e5e0d888be03c4b8b8fcc23ffc3a773a08bf25819
./KyberNetworkProxyInterface.sol	e5bb2a507699162ffed25d2b6c92ee45c746c650247031be92be311d6a553701
./KyberReserve.sol	759503e0ace2ca27c3ed2902d9ec6a12d8351c4f3bf14ad9e44c9c5d5a30398d
./KyberReserveInterface.sol	2b1c1d11ebd6929f9f1270099a78a47442d7e5de6017935c43e555f630f16944
./PermissionGroups.sol	b9049798efd7f635369e27552ef6cf33265c23b440c7bb0336d64978df8f87fa
./Utils.sol	5ce4c3e352dc54a8e102c32b566f1d968bc328a39f2f0f96fda8a48270c49a6c
./Utils2.sol	349239755016b74f3d173a6233ab59fbbc01bdd2c57c9a40f85c805451f61f2d
./Withdrawable.sol	fd9ddf8bee9eec20abc55767138e2815f2b9ac2b39807a54642f069967670f10
./permissionless/OrderIdManager.sol	fb78a86cee53973a0aba4acafe7c6c5172ed444bec9975d9fd4bf73e135c8a2b
./permissionless/OrderList.sol	5e2e15557cda336c18498b95a62e2244e0a82e6ea49fad9fea402c89d60a4a6f
./permissionless/OrderListFactory.sol	ddfa7ff75480670504f7149e714b70b5bcd3a013d178260384f6084bb113f39a
./permissionless/OrderListFactoryInterface.sol	21493aa35f108bd7f6ee1dfc2762bdba50ca5f035399c093e65433ce1a245ef2
./permissionless/OrderListInterface.sol	c9444baf186b48d00f4a6add9db057e31e063e0abc16b3ddfabd930da1f4c959
./permissionless/OrderbookReserve.sol	54f45e2593c5ee69fc5719c10b4e1960f8eb7c6ca15a6fae6acf5c56e113aaa5
./permissionless/OrderbookReserveInterface.sol	7f5d8b8b07f573c4ef0c88d84fb15664dfd480695ce9e996f9384a0243b1118c
./permissionless/PermissionlessOrderbookReserveLister.sol	7edfa1ffe10a3aeb743aa6fa5438a7372588dcbb80b3f340993d57f1d197719a

Depth of Audit

The scope of the security audit conducted by ChainSecurity was restricted to:

- Scan the contracts listed above for generic security issues using automated systems and manually inspect the results.
- Manual audit of the contracts listed above for security issues.

Terminology

For the purpose of this audit, we adopt the following terminology. For security vulnerabilities, we specify the *likelihood*, *impact* and *severity* (inspired by the OWASP risk rating methodology[1]).

Likelihood represents the likelihood of a security vulnerability to be encountered or exploited in the wild.

Impact specifies the technical and business related consequences of an exploit.

Severity is derived based on the likelihood and the impact calculated previously.

[1] https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

We categorize the findings into 4 distinct categories, depending on their severities:

- Low: can be considered as less important
-  Medium: should be fixed
-  High: we strongly suggest to fix it before release
-  Critical: needs to be fixed before release

These severities are derived from the likelihood and the impact using the following table, following a standard approach in risk assessment.

	IMPACT		
LIKELIHOOD	**High**	**Medium**	**Low**
High			
Medium			
Low			

During the audit concerns might arise or tools might flag certain security issues. If our careful inspection reveals no security impact, we label it as ✓ No Issue. If during the course of the audit process, an issue has been addressed technically, we label it as ✓ Fixed, while if it has been addressed otherwise by improving documentation or further specification, we label it as ✓ Addressed. Finally, if an issue is meant to be fixed in the future without immediate changes to the code, we label it as ✓ Acknowledged.

Findings that are labelled as either ✓ Fixed or ✓ Addressed are resolved and therefore pose no security threat. Their severity is still listed, but just to give the reader a quick overview what kind of issues were found during the audit.

Limitations

Security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a secure smart contract. However, auditing allows to discover vulnerabilities that were overlooked during development and areas where additional security measures are necessary.

In most cases, applications are either fully protected against a certain type of attack, or they lack protection against it completely. Some of the issues may affect the entire smart contract application, while some lack protection only in certain areas. We therefore carry out a source code review trying to determine all locations that need to be fixed. Within the customer-determined timeframe, CHAINSECURITY has performed auditing in order to discover as many vulnerabilities as possible.

System Overview

KYBERNETWORK runs an exchange for ERC20 tokens (and wrapped tokens) on the Ethereum blockchain. Everything is done on-chain except for querying the ETH to USD price (which is used to calculate minimum order sizes). This information is queried from the Maker DAO's API. As on common exchanges there are market makers (usually individuals which provide huge buy or sell orders to provide liquidity).

Until now, a market-maker-to-be needed KYBERNETWORK's permission to become a market maker. KYBERNETWORK's new update allows anyone to become a market maker without further permissions. This is managed by so called permissionless reserve contracts. There is one permissionless reserve per token (that manages the trading between Token $\rightarrow$ ETH and ETH $\rightarrow$ Token). The market makers on a permissionless reserve need to stake the token/ETH which they want to trade, prior to placing the market maker orders (also called maker orders). Additionally, KYBERNETWORK charges a fee for each fulfilled order which has to be paid by the maker. Therefore, a maker needs to stake enough KNC tokens(a multiple of the expected fee for the trades), to pay for the fees. Every trade costs some fee (in KNC tokens) which is deducted from the makers staked KNC tokens when the order is fulfilled.

From the perspective of a normal trader (non-market maker, also called taker) nothing changes. A taker places the order and it gets executed if there are matching maker orders. The taker can choose whether or not to use permissionless reserves.

Best Practices in KYBERNETWORK's project

Projects of good quality follow best practices. In doing so, they make audits more meaningful, by allowing efforts to be focused on subtle and project-specific issues rather than the fulfillment of general guidelines.

Avoiding code duplication is a good example of a good engineering practice which increases the potential of any security audit.

We now list a few points that should be enforced in any good project that aims to be deployed on the Ethereum blockchain. The corresponding box is ticked when KYBERNETWORK's project fitted the criterion when the audit started.

Hard Requirements

These requirements ensure that the KYBERNETWORK's project can be audited by CHAINSECURITY.

- ☑ The code is provided as a Git repository to allow the review of future code changes.
- ☑ Code duplication is minimal, or justified and documented.
- ☑ Libraries are properly referred to as package dependencies, including the specific version(s) that are compatible with KYBERNETWORK's project. No library file is mixed with KYBERNETWORK's own files.
- ☑ The code compiles with the latest Solidity compiler version. If KYBERNETWORK uses an older version, the reasons are documented.
- ☑ There are no compiler warnings, or warnings are documented.

Soft Requirements

Although these requirements are not as important as the previous ones, they still help to make the audit more valuable to KYBERNETWORK.

- ☑ There are migration scripts.
- ☑ There are tests.
- ☑ The tests are related to the migration scripts and a clear separation is made between the two.
- ☑ The tests are easy to run for CHAINSECURITY, using the documentation provided by KYBERNETWORK.
- ☑ The test coverage is available or can be obtained easily.
- ☑ The output of the build process (including possible flattened files) is not committed to the Git repository.
- ☑ The project only contains audit-related files, or, if not possible, a meaningful separation is made between modules that have to be audited and modules that CHAINSECURITY should assume correct and out of scope.
- ☑ There is no dead code.
- ☑ The code is well documented.
- ☑ The high-level specification is thorough and allow a quick understanding of the project without looking at the code.
- ☑ Both the code documentation and the high-level specification are up to date with respect to the code version CHAINSECURITY audits.
- ☑ There are no getter functions for public variables, or the reason why these getters are in the code is given.
- ☑ Function are grouped together according either to the Solidity guidelines[2], or to their functionality.

[2] https://solidity.readthedocs.io/en/latest/style-guide.html#order-of-functions

Security Issues

In the following, we discuss our investigation into security issues. Therefore, we highlight whenever we found specific issues but also mention what vulnerability classes do not appear, if relevant.

Hijacking OrderbookReserve possible  ✓ Fixed

The Orderbook Reserve setup is divided into three steps. According to KYBERNETWORK, this has to be done to avoid, to potentially exceed the block gas limit when deploying the necessary contracts. To complete the contract setup, a user calls the following methods on the `PermissionlessOrderbookReserveLister` contract (in the following order).

- The first call to `addOrderbookContract` method deploys a new `OrderbookReserve` contract.
- The second call to `initOrderbookContract` method initializes the `OrderbookReserve`.
- The third call to `listOrderbookContract` method lists the Orderbook contract.

An attacker can hijack a deployed `OrderbookReserve` which is not initialized yet. To accomplish this an attacker performs the following transactions:

- The attacker will deploy a `OrderListFactory` contract from his own account. Let us further refer to this as an `AttackerOrderListFactory` contract.
- An attacker keeps on listening for the event `TokenOrderbookListingStage` on the `PermissionlessOrderbookReserveLister` contract.
- As soon as a user calls `addOrderbookContract` method on `PermissionlessOrderbookReserveLister` contract, to add a new Orderbook Reserve for non-existing ERC20 token. This operation creates a new `OrderbookReserve` contract and triggers the `TokenOrderbookListingStage` event.
- The attacker can find the address of the newly created `OrderbookReserve` contract.
- Before the user would call `initOrderbookContract` on `PermissionlessOrderbookReserveLister` contract, the attacker will call `init` method on newly created `OrderbookReserve` and pass the address of `AttackerOrderListFactory` which he deployed already.

The operation described above, enables an attacker to hijack the newly created `OrderbookReserve` contract and inject his own implementation of the `OrderList` contract. Now, even if the user tries to call the `initOrderbookContract` function on `PermissionlessOrderbookReserveLister` (using the same ERC20 token), their transaction will always fail. Moreover, this ERC20 token would be locked forever, as a new reserve cannot be created for this token. Also, the `Operator` of the `KyberNetwork` will not be able to remove the reserve as it is not listed.

Likelihood: High
Impact: High

Fixed: KYBERNETWORK solved the problem by passing the `OrderListFactory` contract address to the `OrderbookReserve` constructor. This factory instance is used to create `OrderList` contracts. Hence code injection is not possible now. Thus, it fixes the vulnerability present in the code.

Failed token approval  ✓ Fixed

The method `listPairForReserve` present in `KyberNetwork` contract is making the call `token.approve`, to approve the maximal amount of tokens beforehand. The `approve` method of ERC20 standard also returns a `bool`, to let the caller know about the status of the approve call.

As per the new permissionless model, anyone would be able to provide the ERC20 token address and deploy the reserve. Hence it becomes more important to ensure the safety for the users.

CHAINSECURITY recommend to enclose the all the `token.approve`, `token.transfer` and the `token.transferFrom` function calls (if present in code) with a `require`, to ensure the calls' safety.

Likelihood: Medium
Impact: Medium

Fixed: KYBERNETWORK fixed the issue by enclosing `token.approve` function calls with `require`.

Dependency on MakerDAO price feed L ✓ Acknowledged

KYBERNETWORK is using MakerDAO's USD per ETH price feed contract by calling the `medianizer.peek` method (via `OrderbookReserve.setMinOrderSizeEth` method). In case the price feed contract is shut-down / changed or attacked, it would affect the `OrderbookReserve` contract. An overly high price returned by the price feed, would lower the minimum order size and thereby allow the creation of dust orders. An overly high price from the price feed would raise the minimum order size and block legitimate orders.

Likelihood: Medium
Impact: Low

Acknowledged: KYBERNETWORK is aware about the implications of this issue and would continue use the MakerDAO price feed.

Possibility to steal tokens owned by Network M ✓ Fixed

For technical reasons (e.g. transfer fees) the `KyberNetwork` contract owns a number of tokens. At the time of writing these tokens have a value of roughly 600 USD. These tokens of type D could be stolen as follows:

1. Register `OrderbookReserve` for malicious token M and insert some taker orders for M.
2. Insert taker order at very good rate for token D to ensure that the OrderbookReserve has the best rate for token D.
3. Trade $M \to D$ with a maximum amount so that a refund will be made.
   (a) The expected exchange rates are computed and the `OrderbookReserve` is chosen for D. The `expectedDestAmount` in D is X.
   (b) The refund is performed using `handleChange`.
   (c) During `handleChange` M is called.
   (d) M removes the very good entry from `OrderbookReserve` for D.
   (e) The `OrderbookReserve` for D returns $Y < X$ tokens instead of X because the `OrderBook` has changed.
   (f) The network sends X tokens of type D to the target address. Hence, $X - Y$ tokens were stolen from the network.

Note that the attacker controls most of the variables and can hence make sure that $X - Y$ is the amount of tokens that the network owns.

Likelihood: Low
Impact: High

Fixed: KYBERNETWORK added an additional `require` statement that enforces `OrderBook` consistency within one transaction. Therefore the "promised" conversion rate is now correctly enforced.

Gas implications when calling token contracts L ✓ Addressed

The newly listed tokens could try to use KYBERNETWORK's platform for improper purposes. During a call to `balanceOf` (which is part of a trade) tokens could consume large amounts of gas and subsequently ruin the user experience.

Additionally, tokens could theoretically behave differently when called by KYBERNETWORK in order to block being traded. Finally, tokens could discover that they are being "recovered" using the function `Withdrawable.withdrawToken` and consume extra gas.

CHAINSECURITY therefore recommends that KYBERNETWORK monitors the activity of these tokens, as it might otherwise negatively impact user behavior.

Likelihood: Low
Impact: Low

Addressed: KYBERNETWORK will monitor the listed tokens for suspicious behaviors.

Possibility to manipulate the `OrderBook` M ✓ Fixed

The `OrderBook` is internally represented as a sorted, doubly-linked list. When orders are removed, anticipating that they will be reused, they are not entirely deleted but simply unlinked. This allows removed orders to be referenced by functions such as `updateWithPositionHint`.

The insertion or update functions using a hint can reference these deleted orders. These may happen maliciously or accidentally (e.g. due to transaction reorderings). Once it happens the linking of the `OrderBook` is broken, as multiple valid orders will have the same `nextId` pointer.

Likelihood: Low
Impact: High

Fixed: KYBERNETWORK resolved this issue by invalidating the links of a removed order and thereby preventing them to be used as hints.

Trust Issues

The issues described in this section are not security issues but describe functionality which requires additional trust.

Users need to trust the traded tokens L ✓ Acknowledged

Given that any tokens can be listed, users need to be aware that they must trust token contracts and token owners when buying or selling these tokens. KYBERNETWORK puts a number of good checks in place to protect users.

However, these checks are useless against directly malicious tokens, e.g. tokens that report inconsistent balances. There are no checks that KYBERNETWORK can put in place to prevent this.

Acknowledged: KYBERNETWORK is aware of this issue and will communicate it to users.

Design Issues

The points listed here are general recommendations about the design and style of KYBERNETWORK's project. They highlight possible ways for KYBERNETWORK to further improve the code.

Dead code in `OrderList` M ✓ Fixed

The `OrderList` contract implements `Withdrawable` which has the following methods:

- `withdrawToken`: To withdraw tokens.
- `withdrawEther`: To withdraw Ether.

These methods are only callable from the `admin` role. But in this case, admin of this contract is the `OrderbookReserve` contract. This contract does not implement any method, to call the above listed methods. Hence, they are dead code in the contract. Thus, instead of inheriting from `Withdrawable`, `OrderList` could inherit directly from `PermissionGroups`. This change would save roughly $120{,}000$ gas during a single `OrderList` deployment. Please note: By inheriting from `PermissionGroups`, there will be still superfluous functionality left.

Similarly, `OrderList` also inherits from the `Utils2` contract and doesn't use any of the methods of this contract. Removing `Utils2` from the inheritance saves an additional $65{,}000$ gas during deployment of a single `OrderList` contract.

Fixed: KYBERNETWORK removed the dead code (`Withdrawable` and `Utils2`) from the `OrderList` contract, which now only inherits from the `PermissionGroups` contract.

Unnecessary loop iteration L

The methods `getEthToTokenMakerOrderIds` and `getTokenToEthMakerOrderIds` present in the contract `OrderbookReserve` are using a for loop to prepare the active `orderList` array which contains the active `orderId`. However, once the `orderList` array is full, the loop keeps iterating. Hence, it wastes gas.

```
585 for (uint32 i = 0; i < NUM_ORDERS; ++i) {
586     if ((makerOrders.takenBitmap & (uint(1) << i) > 0)) orderList[activeOrder
        ++] = makerOrders.firstOrderId + i;
587 }
```

OrderbookReserve.sol

CHAINSECURITY recommends to add some documentation whether these functions are only intended for off-chain calls, in which case no change is required. However, if these will be called on-chain, than the loop iteration should be optimized.

Avoid assigning to function parameters L ✓ Fixed

KYBERNETWORK should avoid assigning values to function parameters. This is done in the `OrderbookReserve` contract in the following functions:

- `bindOrderStakes` function overwrites `weiAmount` parameter.
- `takePartialOrder` function overwrites `orderSrcAmount` and `orderDstAmount` parameters.

Fixed: KYBERNETWORK partially fixed the issue, wherever it was feasible given the limitations on the number of local variables.

Struct and calculation optimization in `OrderIdData` L ✓ Acknowledged

KYBERNETWORK uses a `uint256` to represent their `uint32` bitmap.

```
5 struct OrderIdData {
6     uint32 firstOrderId;
7     uint takenBitmap;
8 }
```

OrderIdManager.sol

KYBERNETWORK could consider using a `uint32` to represent the `takenBitmap` variable. This would primarily optimize the `OrderIdData` struct. The newly organized struct would require less storage slots. Additionally, such a change could also simplify the return statement `return(uint32(uint(freeOrders.firstOrderId)+ uint32(i));`, present in the `fetchNewOrderId` function.

Acknowledged: KYBERNETWORK is aware about this and will not make code changes, considering future enhancements or upgrades.

Unnecessary loop iteration L ✓ Fixed

The function`KyberNetwork.listPairs` contains a loop to add or remove a reserve. When removing a reserve by passing `add = false`, the else block will be executed and remove the corresponding reserve from the array. However, it keeps iterating over the remaining array entries. This is unnecessary.

```
454 for (i = 0; i < reserveArr.length; i++) {
455     if (reserve == reserveArr[i]) {
456         if (add) {
457             break; //already added
458         } else {
459             //remove
460             reserveArr[i] = reserveArr[reserveArr.length - 1];
461             reserveArr.length--;
462         }
463     }
464 }
```

KyberNetwork.sol

As this method is used for adding reserves, KYBERNETWORK should consider fixing it, to use the optimum number of iterations. This would also reduce the gas consumption.

Fixed: KYBERNETWORK solved the issue by breaking the loop once the item is removed from the `reserveArr` array.

Duplicate getters for `TAIL_ID` and `HEAD_ID` L ✓ Fixed

Inside the `OrderList` contract there are duplicate getters for the state variables `TAIL_ID` and `HEAD_ID`. These variables are public and therefore have built-in getters, but at the same time, the functions `getTailId()` and `getHeadId()` exist. The duplicate functions unnecessarily inflate bytecode size.

As a side note, it is not entirely clear why getters for these rather internal variables are needed, given that they are currently not used.

Fixed: KYBERNETWORK has removed the explicit getter methods `getHeadId()` and `getTailId()` from the `OrderList` contract.

Consistent Handling of omitted return values L

Omitted return values are handled differently inside the code. Here are two code examples:

```
682 (updatedWithHint, noUse2) = list.updateWithPositionHint(orderId, newSrcAmount,
        newDstAmount, hintPrevOrder);
```

OrderbookReserve.sol

```
873 (isSuccess,) = list.updateWithPositionHint(orderId, orderSrcAmount,
       orderDstAmount, HEAD_ID);
```

OrderbookReserve.sol

In one case the return value is explicitly omitted, while in the other it is assigned to a "useless" variable. More consistent handling might improve the readability of the code. Finally, it is not entirely clear why these return values were added given that they seem to be ignored in all cases.

Specification mismatch L ✓ Addressed

As per the specification, the function `depositKncForFee(uint amount)` takes only the argument `amount`. However, looking at the code, the function `depositKncForFee(address maker, uint amount)` takes two arguments, `maker` and `amount`. According to the function definition present in the code, it allows anyone to call this function and deposit KNC token for any beneficiary address.

KyberNetwork should either fix the specification or correct the function definition present in the code if it was not the intention.

Addressed: KyberNetwork has updated the function name as `depositKncForFee(address maker, uint amount)` in the specification document.

Possible out-of-gas exception L

The Method `OrderList.findPrevOrderId` has a while loop which iterates over `orders`. If there are many orders present in the linked list, it would cause an out-of-gas exception. The method `findPrevOrderId` is called from many different places.

To address this, the `OrderbookReserve` contract allows in the function `getEthToTokenAddOrderHint` to pass a `hint` argument, to find the `orderId` more efficiently. However, in case that the order book is very busy, hints might fail due to concurrent transactions. In that case the problem persists. However, this problem cannot lead to a deadlock as removals are always possible.

Finally, to tackle the issue, KyberNetwork could perform gas optimizations inside the `compareOrders` function which is called inside the loop and currently has more multiplications than necessary:

```
213         if (s2 * d1 < s1 * d2) return -1;
214         if (s2 * d1 > s1 * d2) return 1;
```

OrderList.sol

As the compiler does not recognize that the same value are being multiplied two times, it performs the same multiplication multiple times. By caching the results, per invocation of `compareOrders` roughly 13 gas can be saved, which is not a lot, but might add up over time.

Inconsistent Naming in Interface L ✓ Fixed

The `ExpectedRateInterface` has the following function

```
6 interface ExpectedRateInterface {
7     function getExpectedRate(ERC20 src, ERC20 dest, uint srcQty, bool
          useOnlyPermissioned) public view
```

ExpectedRateInterface.sol

However, in the `ExpectedRate` contract the interface is implemented as follows:

```
43     function getExpectedRate(ERC20 src, ERC20 dest, uint srcQty, bool
           usePermissionless)
```

ExpectedRate.sol

It appears that in these two definitions the final `bool` field has opposite meanings.

Fixed: KYBERNETWORK changed the parameter name present in `ExpectedRateInterface` interface to match the definition.

Recommendations / Suggestions

☐ The contracts listed below, import contracts and/or interfaces and also include plain text contract(s) and/or interfaces in the same file. CHAINSECURITY suggest according to the solidity best practices the separation of each contract into a single file. KYBERNETWORK's approach is neither consistent nor following the best practices.

- `PermissionlessOrderbookReserveLister.sol`
- `OrderbookReserve.sol`

☑ The file `OrderListFactoryInterface.sol` defines name as `OrderFactoryInterface` as interface. The contract or interface name does not match with the filename.

CHAINSECURITY recommends using an appropriate name for the interface or file so that it matches and make the code more readable.

☑ Each time a trade is called by calling `OrderbookReserve.trade` it triggers two if statements with identical conditions:

```
188 OrderListInterface list = (srcToken == ETH_TOKEN_ADDRESS) ? tokenToEthList
        : ethToTokenList;
189
190 if (srcToken == ETH_TOKEN_ADDRESS) {
191     require(msg.value == srcAmount);
192 } else {
193     require(msg.value == 0);
194     require(srcToken.transferFrom(msg.sender, this, srcAmount));
195 }
```

OrderbookReserve.sol

The code could be optimized to have calculations in single if statement. This would also reduce a small amount of gas cost.

☐ There are some functions which have their visibility set to `public` and they accept an array as an argument.

```
293 function addOrderBatch(bool[] isEthToToken, uint128[] srcAmount, uint128[]
        dstAmount, uint32[] hintPrevOrder, bool[] isAfterPrevOrder)
```

OrderbookReserve.sol

```
357 function updateOrderBatch(bool[] isEthToToken, uint32[] orderId, uint128[]
        newSrcAmount, uint128[] newDstAmount, uint32[] hintPrevOrder)
```

OrderbookReserve.sol

CHAINSECURITY recommends changing the visibility of these functions to `external`[3]. This makes the function more efficient in terms of gas costs.

☐ When a user performs a trade using `KyberNetworkProxy.tradeWithHint` function, during the course of the full execution of the function, it would emit multiple events. Out of these, two events in particular `TradeExecute` and `ExecuteTrade` have different parameters, but their names are misleading. KYBERNETWORK should provide appropriate names to these events.

Post-audit comment: KYBERNETWORK has fixed some of the issues above and is aware of all the implications of those points which were not addressed. Given this awareness, KYBERNETWORK has to perform no more code changes with regards to these recommendations.

[3] https://solidity.readthedocs.io/en/v0.4.24/contracts.html#visibility-and-getters

Disclaimer