Quantstamp Security Assessment Certificate

August 27th 2020 — Quantstamp Verified

IDEX

This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type	Exchange
Auditors	Leonardo Passos, Senior Research Engineer Sebastian Banescu, Senior Research Engineer Jan Gorzny, Blockchain Researcher Kevin Feng, Software Engineer
Timeline	2020-06-08 through 2020-07-21
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Main Controls and Governance
Documentation Quality	High
Test Quality	High

Source Code	
Repository	Commit
idex-contracts	94082d
idex-contracts	4c9fb73
idex-contracts	226c568
idex-contracts	a6290a2
idex-contracts	512c092e

Total Issues	**11** (7 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	**3** (3 Resolved)
Informational Risk Issues	**8** (4 Resolved)
Undetermined Risk Issues	0 (0 Resolved)

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.
Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Quantstamp has performed an audit on the Solidity contracts in IDEX's exchange; the audited contracts comprise the on-chain components of the exchange. In total, 11 issues have been identified, as well as minor improvements in the code (pointed out as best practices). No high risk issue has been identified. Most issues tend to be informational; generally, those could be easily addressed with enhanced documentation. Privileged operations do exist in the audited contracts and are properly acknowledged in the given specs. Nonetheless, Quantstamp suggests a mitigation plan and clear information on how to address the hypothetical situation of an attacker gaining a privileged role. On the testing side of things, the given test suite has high coverage; nonetheless, some [few] functions lack unit tests. Specification is of good quality, and so is the code and overall inline comments. Still, some functions lack documentation or could be given more detailed documentation. Last, but not least, we suggest documenting all on-chain and off-chain component interactions to further aid independent assessment and overall auditing. After the initial report, IDEX has collaborated with Quantstamp in understanding the reported issues as well as fixing and/or acknowledging them. From the initial set of 11 issues, 7 have been resolved, whereas the remaining 4 have been acknowledged. The latter are all informational issues, and do not pose security risks.

ID	Description	Severity	Status
QSP-1	Integer Overflow / Underflow	Low	Fixed
QSP-2	Missing Input Validation	Low	Fixed
QSP-3	Users Cannot Withdraw their Fractional Part of Ether Deposits	Low	Fixed
QSP-4	Some Functions Lack Unit Tests	Informational	Acknowledged
QSP-5	Old ERC20 Tokens May Not Work With IDEX	Informational	Fixed
QSP-6	Unlocked Pragma	Informational	Fixed
QSP-7	Race Conditions / Front-Running	Informational	Fixed
QSP-8	Unused Imports	Informational	Fixed
QSP-9	Gas Usage / `for` Loop Concerns	Informational	Acknowledged
QSP-10	Privileged Roles and Ownership	Informational	Acknowledged
QSP-11	Token De-registration Not Possible	Informational	Acknowledged

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Truffle v4.1.12
- Ganache v1.1.0
- SolidityCoverage v0.5.8
- Slither v0.6.6

Steps taken to run the tools:

1. Installed Truffle: `npm install -g truffle`
2. Installed Ganache: `npm install -g ganache-cli`
3. Installed the solidity-coverage tool (within the project's root directory): `npm install --save-dev solidity-coverage`
4. Ran the coverage tool from the project's root directory: `./node_modules/.bin/solidity-coverage`
5. Installed the Slither tool: `pip install slither-analyzer`
6. Run Slither from the project directory: `slither .`s

Findings

QSP-1 Integer Overflow / Underflow

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/libraries/AssetUnitConversions.sol`, `contracts/libraries/UUID.sol`

Description: Integer overflow/underflow occur when an integer hits its bit-size limit. Every integer has a set range; when that range is passed, the value loops back around. A clock is a good analogy: at 11:59, the minute hand goes to 0, not 60, because 59 is the largest possible minute. Integer overflow and underflow may cause many unexpected kinds of behavior and was the core reason for the `batchOverflow` attack.

Exploit Scenario:

- `AssetUnitConversions.sol` (L22): the statement `return uint256(quantityInPips) * uint256(10)**(assetDecimals - 8)` can overflow, as there is no upper bound check on the value of `assetDecimals` and `quantityInPips`.
- `UUID.sol` (L26): the subtraction statement could lead to an integer underflow, especially when called by [unaware] end users via `invalidateOrderNonce()`.

Recommendation: Use `SafeMath` to perform arithmetic operations - if overflow/underflow occurs, transactions safely revert.

Update: According to IDEX, in the `AssetUnitConversions.sol` issue, "*we would need to list a token with over 57 decimals for this to present an actual risk to users. We have never encountered a token over 18 decimals, so even without the added checks, there is not a practical risk to users here*". As for the issue in `UUID.sol`, "*for a UUID with a timestamp before 1/1/1970, the extracted timestamp would underflow and be far in the future. The user would thus not be able to place orders from that wallet, but no funds would ever be at risk, and the user can still withdraw those funds and redeposit with a different wallet*". Quantstamp is pleased with the justification above, and agreed to lower the severity of the issue from `High` to `Low`; furthermore, the issue has been promptly fixed, and it is marked as such in this report.

QSP-2 Missing Input Validation

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/Exchange.sol`, `contracts/Governance.sol`, `contracts/libraries/AssetRegistry.sol`

Description:

- `contracts/Exchange.sol`: `wallet` should be checked to be different from `0x0` in `loadBalanceInAssetUnitsByAddress`, `loadBalanceInPipsBySymbol`, and `loadBalanceInAssetUnitsBySymbol`.
- `contracts/Governance.sol`: Functions `setCustodian`, `initiateExchangeUpgrade`, and `initiateGovernanceUpgrade` do not check if the given address parameter is a contract.
- `libraries/AssetRegistry.sol`: Function `registerToken` does not check if `tokenAddress` is different from `0x0`. Moreover, it does not check if `tokenAddress` is a contract address.
- `libraries/AssetRegistry.sol`: Function `registerToken` does not check if symbol is different from the empty string.

Recommendation: Add checks fixing the enumerated cases. For checking whether an address is a contract, refer to `AddressUtils`.

Update: All recommended checks have been placed in the code.

QSP-3 Users Cannot Withdraw their Fractional Part of Ether Deposits

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/Exchange.sol`

Description: The `depositEther()` function is payable and can receive arbitrary amounts of `ether`. This function calls `deposit()`, which first converts the amount to pips and then back into `ether`. Such conversion is required so that the transfer of `ether` to the Custodian contract does not contain the “dust”, i.e. values lower than 1/10^8 `ether`. However, the dust is locked in the `Exchange` contract for each user deposit, without letting users ever having them back.

Recommendation: Invest in public knowledge (external facing documentation) such that users are aware of the potential issue of having the dust of their `ether` deposits being locked in the `Exchange` contract; alternatively, check that `msg.value` does not contain more than 8 decimal digits. If so, reject the transaction.

Update: IDEX has provided external facing documentation to educate users of the given issue. See the docs here.

QSP-4 Some Functions Lack Unit Tests

Severity: *Informational*

Status: Acknowledged

File(s) affected: `contracts/libraries/Signatures.sol`, `contracts/libraries/AssetTransfers.sol`

Description: All the functions in `Signatures.sol` and `AssetTransfers.sol` lack unit tests; however, we acknowledged that they are being indirectly checked by `deposit` and `withdraw` testing flows in the test suite. We suggest, nonetheless, having dedicated unit tests for all functions, testing different input combinations.

Recommendation: Implement (thorough) unit tests for all functions.

> **Update:** To avoid the complexity of fixturing unit tests, IDEX decided it would be better to fully cover the functionality of the aforementioned functions through other tests.

QSP-5 Old ERC20 Tokens May Not Work With IDEX

Severity: *Informational*

Status: Fixed

Description: Since Solidity 0.4.22, some non-ERC20 compliant token contracts that used to work in older versions no longer work now. Since many token contracts implement the `transfer` function by either returning `true` or reverting the call altogether, some contract developers deemed the return value unnecessary, writing contracts with a `transfer` function with signature `function transfer(address to, uint amount) public { ... }`. All non-ERC20 compliant contracts like the latter case that used to work before 0.4.22 will fail in IDEX, as it relies on Solidity 0.6.8. More info can be found here.

Recommendation: This is merely an educational point. Nonetheless, we suggest scripting token registration to check whether target tokens contracts implement the ERC20 standard, with matching function signatures. Needless to say this check is restricted to tokens with a known ABI and/or source code (e.g., as published in some blockchain explorers, like Etherscan).

> **Update:** IDEX changed the logic to address the issue. Specifically, IDEX does not expect any return value from `transfer` calls; since the target ERC20 may still not throw upon an error, IDEX assures the transfer has been made by checking if the balance, after the transfer call, is updated accordingly. If not, the transaction reverts. Quantstamp agrees with such approach; thus, this issue is marked as fixed.

QSP-6 Unlocked Pragma

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/*.sol`, `contracts/libraries/*.sol`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.4.*`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked." For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version.

Recommendation: Except for `Migrations.sol`, where the Solidity version is already locked, all other files *DO NOT* lock the Solidity version. For those, lock the version to `0.6.8`.

> **Update:** All contracts are now locked to `0.6.8`.

QSP-7 Race Conditions / Front-Running

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/Exchange.sol`,

Description: Contrary to the comment on L546, calling `exitWallet` does not immediately disable deposits. Deposits may still occur after one calls `exitWallet` if deposit transactions are mined first (e.g, if they are given higher gas).

Recommendation: Adjust comment accordingly; for instance, saying that deposits are not possible after the transaction containing a call to exitWallet is mined.

> **Update:** Comment has been changed accordingly. It is now stated as "*Calling exitWallet disables deposits immediately on mining [...]*", which addresses our initial miscommunication concern.

QSP-8 Unused Imports

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/libraries/AssetUnitConversions.sol`

Description:

- `SafeMath` is imported but never used.
- `IERC20` is imported but never used.

Recommendation: In `AssetUnitConversions`, use `SafeMath` to perform arithmetic operations and prevent overflow/underflow cases; remove the unnecessary import of `IERC20`.

> **Update::** The unused imports herein discussed have been fixed accordingly.

QSP-9 Gas Usage / `for` Loop Concerns

Severity: *Informational*

Status: Acknowledged

File(s) affected: `contracts/AssetRegistry.sol`

Description: The `loadAssetBySymbol` function inside `AssetRegistry.sol` contains a for-loop over all assets registered with the same symbol (see L100). If there are too many assets registered with the same symbol, this function would revert due to reaching the block-gas-limit. Although this is unlikely to be an issue in practice (the number of token contract update are likely to be small), in theory, reaching the gas-block-limit is still possible.

Recommendation: Do a gas analysis and determine how many assets with the same symbol can be added before reaching the block gas limit. Then add a check and error message in the asset registration function to notify the end-user (admin in this case) that adding more tokens with this symbol will cause the `loadAssetBySymbol` function to fail.

> **Update:** According to IDEX "*In practice this is not an operational concern as only IDEX can add tokens to the registry, and the number of swapped symbol contracts is very low. We have not encountered more than 3 different addresses associated with a single symbol during the course of operating IDEX 1.0.*"

QSP-10 Privileged Roles and Ownership

Severity: *Informational*

Status: Acknowledged

File(s) affected: `contracts/Exchange.sol`, `contracts/Governance.sol`,

Description: Smart contracts will often have `owner` and/or `admin` roles to designate the person with special privileges to make modifications to the smart contract and/or other components (e.g., dispatcher wallet). However, this centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

Recommendation: Quantstamp recommends investment in a contingency plan to outline the actions in case an attacker gains privileged access.

> **Update:** IDEX has stated that they have currently working on a contingency plan.

QSP-11 Token De-registration Not Possible

Severity: *Informational*

Status: Acknowledged

File(s) affected: `contracts/libraries/AssetRegistry.sol`

Description: Currently, after registering a new token, it cannot be de-registered. Thus, if an incorrect token registration occurs, it cannot be undone unless one performs a new contract deployment and migrates all correctly registered tokens.

Recommendation: To be able to remove a token that should not have been registered, add a function to de-register a token.

> **Update:** According to IDEX, "*this is by design. The consequences of incorrectly listing the precision of an asset is severe. Allowing tokens to be unlisted would open up the possibility of an attacker with access to privileged accounts to unlist and relist with a different precision. Rather than open this security risk, we require two independent transactions in the token registration process for redundancy.*". Quantstamp considers this a fair assessment; nonetheless, the issue is in theory possible, and IDEX acknowledges it.

Adherence to Specification

Overall, Quantstamp found the code to reflect most of what was defined in the specification. We found a single issue for which the code does not reflect the spec:

- In the Governance contract, it is stated that "Contract Upgrade Period: 1 week" is a fixed parameter. However, we did not find this parameter being set to 1 week anywhere in the code base. According to IDEX, "*this is set as a parameter on deployment of the Governance contract.*". As the parameter is not being set in the migrations file, we assume this is handled externally, by scripts not currently located in the Github repo.

Code Documentation

Overall, Quantstamp finds the code documentation in the audited smart contracts to be of high quality.

Adherence to Best Practices

With respect to the initial audit of commit `94082d`, Quantstamp suggests adhering to the following best practices:

- Use NatSpec format for all public interfacing contracts and functions. Currently, this is not consistent. At the very least, for each public or external function, provide a `@notice`, document each parameter using `@param` (if any), as well as the return value using `@return` (if any). See the Solidity docs for further details. **Fixed**

- `contracts/libraries/Interfaces.sol`: Document the fields of all `structs`. **Fixed**

- `contracts/libraries/AssetRegistry.sol`: On L53, it says “Block timestampInMs is seconds”. Probably meant `block.timestamp is in seconds`. **Fixed**

- `contracts/libraries/AssetUnitConversions.sol`: `SafeMath` is imported but never used; same with `IERC20`. Fix imports accordingly. **Fixed**

- In Solidity 6, events with indexed string parameters log a hash value, not the raw one (raw values are only logged as it for fixed-size types). Make sure whoever consumes these events hash expected string values, comparing them with the the hash in the logged event. **Fixed**

- `contracts/Exchange.sol`: In `setChainPropagationPeriod`, `newChainPropagationPeriod` should be checked to be different than `_chainPropagationPeriod`; otherwise the function just wastes gas. **Not fixed**

- `contracts/Custodian.sol`: In `setGovernance`, `newGovernance` should be checked to be different than `_governance`; otherwise the function just wastes gas. **Not fixed**

- `contracts/Custodian.sol`: In `setExchange`, `newExchange` should be checked to be different than `_exchange`; otherwise the function just wastes gas. **Not fixed**

- `contracts/Exchange.sol`: On one hand, the two require statements on L619-620 check if the wallet exit was finalized. However, the error messages indicate that trades are not allowed on exited wallets: `require(!isWalletExitFinalized(buy.walletAddress), 'Buy wallet exited')` and

• `require(!isWalletExitFinalized(sell.walletAddress), 'Sell wallet exited')`. The `isWalletExitFinalized` function returns false if the wallet is exited, but not yet finalized. Hence, trades can be executed with exited wallets, which are not yet finalized. The same applies to L485. On the other hand, the require statement on L370 in `Exchange.sol` checks if the wallet exit was initiated, regardless if it was finalized or not. However, the error message is the same as above: `require(!_walletExits[wallet].exists, 'Wallet exited')`. Clarify the error message to something like: "... wallet exit finalized" (L485, L619-620) and "... wallet exit initiated" (L370). **Fixed**

• `contracts/Exchange.sol`: Typo in code comment on L964; `registerAsset` -> `registerToken`. **Fixed**

• Unnamed magic numbers should be avoided; they don't convey any meaning to the reader who is new to the code. For instance, in `contracts/Exchange,sol`:

  • `1000` (L273, L301, L356) - **Fixed** (consider using a `constant`)

  • `10**8` (L821) - **Fixed** (consider using a `constant`)

  • `(24 * 60 * 60) * 1000` (L439) - **Fixed** (consider using a `constant`)

  • `10000` (L1040) - **Fixed** (consider using a `constant`)

• `contracts/libraries/Signatures.sol`: the loop on L111-113 is unnecessary as `hyphenBytes.length` is always equal to 1. **Fixed**

• Use interface type other than raw address for params whenever possible (e.g., `tokenAddress` is actually a `IERC20 Token`). **Fixed**

• `contracts/Custodian.sol`: there should be an event to monitor `receive()` ; at least it would be consistent with `withdraw`. **Not fixed**

Test Results

Test Suite Results

Quantstamp found the test suite of the audited contracts to be of high quality. Nonetheless, as we reported in issue QSP-2, functions in `Signatures.sol` and `AssetTransfers.sol` do not have unit tests; they are, nonetheless, tested within the flow of other tests.

```
Contract: Exchange (tokens)
  registerToken
     ✓ should work (951ms)
     ✓ should revert when token has too many decimals (814ms)
     ✓ should revert for ETH address (736ms)
     ✓ should revert for blank symbol (797ms)
     ✓ should revert when already finalized (991ms)
  confirmTokenRegistration
     ✓ should work (780ms)
     ✓ should revert for unknown token address (862ms)
     ✓ should revert when already finalized (881ms)
     ✓ should revert when symbols do not match (812ms)
     ✓ should revert when decimals do not match (813ms)
  loadAssetBySymbol
     ✓ should work for ETH (655ms)
     ✓ should work for registered token (934ms)
     ✓ should revert when no token registered for symbol (918ms)
     ✓ should revert when no token registered for symbol prior to timestamp (906ms)
  assetUnitsToPips
     ✓ should succeed (118ms)
     ✓ should truncate fractions of a pip (41ms)
     ✓ should revert on uint64 overflow
     ✓ should revert when token has too many decimals
  pipsToAssetUnits
     ✓ should succeed (104ms)
     ✓ should revert when token has too many decimals

Contract: Exchange (deposits)
   ✓ should revert when receiving ETH directly (202ms)
  depositEther
     ✓ should work for minimum quantity (857ms)
     ✓ should revert below minimum quantity (665ms)
  depositTokenBySymbol
     ✓ should work for minimum quantity (1005ms)
     ✓ should revert for ETH (990ms)
     ✓ should revert for exited wallet (1157ms)
     ✓ should revert when token quantity above wallet balance (1118ms)
     ✓ should revert for unknown token (668ms)
  depositTokenByAddress
     ✓ should work for minimum quantity (1006ms)
     ✓ should work for minimum quantity with non-compliant token (933ms)
     ✓ should revert for ETH (955ms)
     ✓ should revert for unknown token (721ms)
     ✓ should revert when token skims from transfer (964ms)

Contract: Custodian
  deploy
     ✓ should work (65ms)
     ✓ should revert for invalid exchange address (67ms)
     ✓ should revert for non-contract exchange address (65ms)
     ✓ should revert for invalid governance address (75ms)
     ✓ should revert for non-contract governance address (104ms)
  receive
     ✓ should work when sent from exchange address
     ✓ should revert when not sent from exchange address
  setExchange
     ✓ should work when sent from governance address (240ms)
     ✓ should revert for invalid address (74ms)
     ✓ should revert for non-contract address (73ms)
     ✓ should revert when not sent from governance address (68ms)
  setGovernance
     ✓ should work when sent from governance address (149ms)
     ✓ should revert for invalid address (75ms)
     ✓ should revert for non-contract address (73ms)
     ✓ should revert when not sent from governance address (69ms)
  withdraw
     ✓ should work when sent from exchange (76ms)
     ✓ should revert withdrawing ETH not deposited (52ms)
     ✓ should revert withdrawing tokens not deposited (120ms)
     ✓ should revert when not sent from exchange (44ms)
```

```
Contract: Exchange (exits)
  exitWallet
     ✓ should work for non-exited wallet (672ms)
     ✓ should revert for wallet already exited (668ms)
  withdrawExit
     ✓ should work for ETH (743ms)
     ✓ should revert for wallet not exited (646ms)
     ✓ should revert for wallet exit not finalized (709ms)
     ✓ should revert for asset with no balance (740ms)

Contract: Exchange (tunable parameters)
    ✓ should deploy (162ms)
    ✓ should revert when receiving ETH directly (186ms)
  loadBalanceInAssetUnitsByAddress
     ✓ should revert for invalid wallet (610ms)
  loadBalanceInPipsByAddress
     ✓ should revert for invalid wallet (625ms)
  loadBalanceInPipsBySymbol
     ✓ should revert for invalid wallet (620ms)
  loadBalanceInAssetUnitsBySymbol
     ✓ should revert for invalid wallet (632ms)
  setAdmin
     ✓ should work for valid address (202ms)
     ✓ should revert for empty address (210ms)
     ✓ should revert for setting same address as current (691ms)
     ✓ should revert when not called by owner (216ms)
  removeAdmin
     ✓ should work (652ms)
  setCustodian
     ✓ should work for valid address (613ms)
     ✓ should revert for empty address (215ms)
     ✓ should revert after first call (647ms)
  setChainPropagationPeriod
     ✓ should work for value in bounds (652ms)
     ✓ should revert for value out of bounds (787ms)
  setDispatcher
     ✓ should work for valid address (678ms)
     ✓ should revert for empty address (212ms)
     ✓ should revert for setting same address as current (681ms)
  removeDispatcher
     ✓ should set wallet to zero (734ms)
  setFeeWallet
     ✓ should work for valid address (936ms)
     ✓ should revert for empty address (206ms)
     ✓ should revert for setting same address as current (691ms)

Contract: Governance
    ✓ should deploy (76ms)
  setAdmin
     ✓ should work for valid address (99ms)
     ✓ should revert for empty address (104ms)
  setCustodian
     ✓ should work for valid address (594ms)
     ✓ should revert for empty address (103ms)
     ✓ should revert for non-contract address (102ms)
     ✓ should revert after first call (630ms)
     ✓ should revert when not called by admin (663ms)
  initiateExchangeUpgrade
     ✓ should work for valid contract address (820ms)
     ✓ should revert for invalid contract address (641ms)
     ✓ should revert for non-contract address (107ms)
     ✓ should revert for same Exchange address (666ms)
     ✓ should revert when upgrade already in progress (862ms)
  cancelExchangeUpgrade
     ✓ should work when in progress (878ms)
     ✓ should revert when no upgrade in progress (635ms)
  finalizeExchangeUpgrade
     ✓ should work when in progress and addresses match (995ms)
     ✓ should revert when no upgrade in progress (889ms)
     ✓ should revert on address mismatch (1315ms)
     ✓ should revert when block threshold not reached (964ms)
  initiateGovernanceUpgrade
     ✓ should work for valid contract address (823ms)
     ✓ should revert for invalid contract address (657ms)
     ✓ should revert for non-contract address (115ms)
     ✓ should revert for same Governance address (670ms)
     ✓ should revert when upgrade already in progress (806ms)
  cancelGovernanceUpgrade
     ✓ should work when in progress (807ms)
     ✓ should revert when no upgrade in progress (671ms)
  finalizeGovernanceUpgrade
     ✓ should work when in progress and addresses match (905ms)
     ✓ should revert when no upgrade in progress (832ms)
     ✓ should revert on address mismatch (765ms)
     ✓ should revert when called before block threshold reached (753ms)

Contract: Exchange (invalidations)
  invalidateOrderNonce
     ✓ should work on initial call (652ms)
     ✓ should work on subsequent call with a later timestamp (694ms)
     ✓ should revert for nonce with timestamp too far in the future (690ms)
     ✓ should revert on subsequent call with same timestamp (692ms)
     ✓ should revert on subsequent call before block threshold of previous (751ms)
     ✓ should revert for non-V1 UUID (634ms)

SafeMath64
  add
     ✓ adds correctly
     ✓ reverts on addition overflow (39ms)
  sub
     ✓ subtracts correctly
     ✓ reverts if subtraction result would be negative
  mul
     ✓ multiplies correctly
     ✓ multiplies by zero correctly
     ✓ reverts on multiplication overflow (40ms)
  div
     ✓ divides correctly
     ✓ divides zero correctly
     ✓ returns complete number result on non-even division
     ✓ reverts on division by zero

Contract: Exchange (trades)
  executeTrade
     ✓ should work for matching limit orders (1482ms)
```

```
      ✓ should work near max pip value (1870ms)
      ✓ should revert above max pip value (1670ms)
      ✓ should work for matching limit orders with 2 decimal base asset (1513ms)
      ✓ should work for matching limit orders without exact price match (1494ms)
      ✓ should work for matching stop limit orders (1547ms)
      ✓ should work for matching stop market orders (1698ms)
      ✓ should work for matching maker limit and taker market order on quote terms (1543ms)
      ✓ should work for partial fill of matching limit orders (1471ms)
      ✓ should work for partial fill of matching maker limit and taker market order in quote terms (1499ms)
      ✓ should revert for self-trade (1145ms)
      ✓ should revert for limit order with quoteOrderQuantity (1527ms)
      ✓ should revert when fill base net and fee do not sum to gross (1521ms)
      ✓ should revert when fill quote net and fee do not sum to gross (1502ms)
      ✓ should revert for limit order overfill (1799ms)
      ✓ should revert for market order overfill on quote terms (1891ms)
      ✓ should revert when not called by dispatcher (1167ms)
      ✓ should revert for exited buy wallet (1367ms)
      ✓ should revert for exited sell wallet (1184ms)
      ✓ should revert for invalidated buy nonce (1429ms)
      ✓ should revert for invalidated sell nonce (1320ms)
      ✓ should revert for unconfirmed base asset (893ms)
      ✓ should revert for invalid signatureHashVersion (1333ms)
      ✓ should revert for invalid signature (1296ms)
      ✓ should revert for excessive taker fee (1324ms)
      ✓ should revert for excessive maker fee (1317ms)
      ✓ should revert for zero base quantity (1078ms)
      ✓ should revert for zero quote quantity (1137ms)
      ✓ should revert when buy limit price exceeded (1088ms)
      ✓ should revert when sell limit price exceeded (1146ms)
      ✓ should revert when base and quote assets are the same (1066ms)
      ✓ should revert when maker fee asset not in trade pair (1313ms)
      ✓ should revert when taker fee asset not in trade pair (1330ms)
      ✓ should revert when maker and taker fee assets are the same (1076ms)
      ✓ should revert on double fill (1795ms)

  Contract: Exchange (trades)
    executeTrade
      ✓ should revert when buy order base asset is mismatched with trade (1473ms)
      ✓ should revert when sell order base asset is mismatched with trade (1603ms)

  Contract: UUID
    getTimestampInMsFromUuidV1
      ✓ should work for current timestamp (66ms)
      ✓ should work for 0 (56ms)
      ✓ should revert for wrong UUID version (57ms)
      ✓ should revert for timestamp before Unix epoch (62ms)

  Contract: Exchange (withdrawals)
    withdraw
      ✓ should work by symbol for ETH (554ms)
      ✓ should work by address for ETH (539ms)
      ✓ should work by symbol for token (874ms)
      ✓ should work by symbol for non-compliant token (816ms)
      ✓ should deduct fee (1000ms)
      ✓ should revert for unknown token (548ms)
      ✓ should revert when token skims from transfer (1066ms)
      ✓ should revert for invalid signature (522ms)
      ✓ should revert for exited wallet (507ms)
      ✓ should revert for excessive fee (478ms)
      ✓ should revert for double withdrawal (624ms)


  180 passing (2m)
```

Code Coverage

As the coverage data indicates, all test coverage metrics reached 100% (the best score possible).

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
Custodian.sol	100	100	100	100	
Exchange.sol	100	100	100	100	
Governance.sol	100	100	100	100	
Owned.sol	100	100	100	100	
contracts/libraries/	100	100	100	100	
AssetRegistry.sol	100	100	100	100	
AssetTransfers.sol	100	100	100	100	
AssetUnitConversions.sol	100	100	100	100	
Interfaces.sol	100	100	100	100	
SafeMath64.sol	100	100	100	100	
Signatures.sol	100	100	100	100	
UUID.sol	100	100	100	100	
contracts/test/	100	100	100	100	
AssetsMock.sol	100	100	100	100	
ExchangeMock.sol	100	100	100	100	
GovernanceMock.sol	100	100	100	100	
NonCompliantToken.sol	100	100	100	100	
SafeMath64Mock.sol	100	100	100	100	
SkimmingTestToken.sol	100	100	100	100	
TestToken.sol	100	100	100	100	
UUIDMock.sol	100	100	100	100	
All files	**100**	**100**	**100**	**100**	

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
622884eb268ac6af27b7cb6cbfa5e2cf18a0b5f6f11f8ddb3a772aa80d08aefd ./contracts/Exchange.sol
308a63eacc22d07ba28283aca36058ff2fdeea7399fd27c79ed46622a78410e9 ./contracts/Owned.sol
2e9bbace7731240a8c8d1380feece3520e13d20ae2eba893a91b5bc2278ab0a7 ./contracts/Migrations.sol
dd02e77ec375dc3f4de98510f3041d3c5fcd88e68021d7127c27ee3b5babb9c0 ./contracts/Governance.sol
2cc0b12d725c2c19c5322728e9ca30b823cbda51072f86b68c0f54273cef1cb0 ./contracts/Custodian.sol
df70e30548ecd8d6462fa161ddda3a64a61552e14ffdac7826c73234b3b5a5e2 ./contracts/libraries/UUID.sol
4edbd9f4df1b0a908f235b70c012b04122504a2b759af66ba832dad61f10eabd ./contracts/libraries/AssetRegistry.sol
c49b7b0fade2a695e075b5be68566b52f935a08f7b24ff2a3669f85305dd06d9 ./contracts/libraries/AssetTransfers.sol
76beefb2f18a941f7f1d29ababbda9dc625344625b57c545ad4176ac65191d3b ./contracts/libraries/AssetUnitConversions.sol
f893f868a085f9cfaece60c6ed0ac503aab19c49fdbbb9ee04d785e238f1967b ./contracts/libraries/Signatures.sol
d00869131016ab0450c73038806d39e4a363160a2ec2c58da779d3ee96496a10 ./contracts/libraries/Interfaces.sol
27e8ba2e34fea7cfa925144cf583b5733c9e85633aba25949554f05a51e3b23f ./contracts/libraries/SafeMath64.sol
0fe80f1a797fc4318090baa9481612a6dc05aa3f2e31fc9ea7174adeb70f192d ./contracts/test/UUIDMock.sol
b041485ec310b3ba8820f5df0a6bfc1c394f8417f09863c84abc648d62da15aa ./contracts/test/SkimmingTestToken.sol
72c01e0f81148dbe26ba7cbd5c29e788b4ed28a1c3d242deea6187fa194bac63 ./contracts/test/NonCompliantToken.sol
814511e0e99c5b6aae85e57382b28c8197b9159f163b3654c6c8eae416e2e9e4 ./contracts/test/SafeMath64Mock.sol
03b02a1a5e88b5696168239ef03e8a23f660d162e966a5e390db1e8150fe27dd ./contracts/test/ExchangeMock.sol
9ab8e1d766976a2c078d580349415e67a89c171bdb0b880db2f52902345958a7 ./contracts/test/TestToken.sol
5872601338af067d4d59397761f74384e8066be144a495bcec516e4938931a43 ./contracts/test/GovernanceMock.sol
25b12c32f151fb0d2fb7f3b04ab209f020c9e4b3105d60c92c6898665992d136 ./contracts/test/AssetsMock.sol
```

Tests

```
504dfe01ab404d8df6e6de5b60fe77fac1a6ab5d8b741ac656b53c82241dd543 ./test/governance.ts
49aa630e39a5d7d4691042865ed6fa50fef543acf0bce6c9c63f5177487e0e6c ./test/uuid.ts
b9520c67931243d764ccf0fd7d680d083b1006e0ff664f015d25edb4931c7abc ./test/helpers.ts
70371690b6058b1ec92388efe525416ff77a21971980a8331af55276503de738 ./test/exit.ts
792dae46a7e1684b7bd53666d0c2314442a0666fd683cee821b50cbb6e2ab49c ./test/assets.ts
7bffe49d090d5e9da54c46ac06e846cb335582a9bc8bf4d52f434a120659ecd8 ./test/exchange.ts
decf60f49b23fd838b04d1828a8ac42ff68f910a3a92a0355d95a8d2a3cb8a02 ./test/trade-block-time.ts
bd5b00b8269043413101f50c8837ae3d277f5b0a17a3d7e3e150c6ce286183df ./test/invalidate.ts
281673074612fc041cbd36274f4ae6491020973dbb2fc73f39a93fe278454610 ./test/custodian.ts
cabc3d49615ed6a4787aeae95f8fc69baafc3d16d01087142ac4e906069d8d11 ./test/trade.ts
bd6640e1ccd31d1b409a47b19c903d321acaa2ca26f967e4c92775a2237f3992 ./test/deposit.ts
79ab05c9770bb9c8af2b96f56d3089f6ac1f88b1bb8c957d951ea63b06a7d964 ./test/withdraw.ts
0cda1b915ed0f2f8ceecfc297fa4a44d2aff88357167399149caa5c82ad9a80c ./test/safemath.ts
```

Changelog

- 2020-06-08 - Initial report
- 2020-06-25 - Reaudit report
- 2020-06-27 - Reaudit report (2)
- 2020-07-10 - Reaudit report (3)
- 2020-07-21 - Final report

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp, Inc. (Quantstamp). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.