PUBLIC

Code Assessment

of the System contracts v1.2 Smart Contracts

December 8, 2022

Produced for

Q

by

CHAINSECURITY

Contents

1 Executive Summary 3

2 Assessment Overview 5

3 Limitations and use of report 9

4 Terminology 10

5 Findings 11

6 Resolved Findings 13

7 Notes 17

1 Executive Summary

Dear Q Blockchain,

Thank you for trusting us to help the Q Blockchain with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of System contracts v1.2 according to Scope to support you in forming an opinion on their security risks.

Q Blockchain implements EVM chain with a delegated proof of stake (DPoS) consensus mechanism, on-chain governance framework, built-in stablecoin system, and numerous other features. This assessment focused on changes that were performed on top of previously reviewed version. Notable changes are a switch of solidity compiler version, new price feed oracles and crypto wallet key protection mechanism for the on-chain stakers.

The most critical subjects covered in our audit are functional correctness, upgradeability and usability. Security regarding all the aforementioned subjects is good.

The general subjects covered are code complexity and event handling. Security regarding those subjects is good.

In summary, we find that the codebase provides a satisfactory level of security. The remaining acknowledged but not fixed issues do not immediately impair the system, however, we still suggest addressing them in the future. Over time their significance might change and cause more serious consequences.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	1
• Code Corrected	1
Medium-Severity Findings	0
Low-Severity Findings	10
• Code Corrected	7
• Risk Accepted	3

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the `contracts` folder of the System contracts v1.2 repository based on the documentation files. Following files from the repository contracts folder were part of the assessment scope:

```
    /common/AddressStorage.sol
    /common/AddressStorageStakes.sol
    /common/AddressStorageStakesSorted.sol
    /common/CompoundRateKeeper.sol
    /common/CompoundRateKeeperFactory.sol
    /common/AddressStorageFactory.sol
    /ContractRegistry.sol
    /defi/BorrowingCore.sol
    /defi/DefiParams.sol
    /defi/GSNPaymaster.sol
    /defi/LiquidationAuction.sol
    /defi/oracles/FxPriceFeed.sol
/defi/oracles/FxPriceFeedMedianizer.sol
/defi/oracles/FxPriceFeedSM.sol
    /defi/Saving.sol
    /defi/SystemBalance.sol
    /defi/SystemDebtAuction.sol
    /defi/SystemSurplusAuction.sol
    /defi/token/StableCoin.sol
    /defi/TokenBridgeAdminProxy.sol
/governance/AccountAliases.sol
    /governance/AParameters.sol
/governance/ARootNodeApprovalVoting.sol
    /governance/ASlashingEscrow.sol
    /governance/constitution/ConstitutionVoting.sol
/governance/ContractRegistryAddressVoting.sol
/governance/ContractRegistryUpgradeVoting.sol
    /governance/EmergencyUpdateVoting.sol
    /governance/experts/AExpertsMembership.sol
    /governance/experts/AExpertsMembershipVoting.sol
    /governance/experts/AExpertsParametersVoting.sol
    /governance/GeneralUpdateVoting.sol
    /governance/rootNodes/RootNodesSlashingVoting.sol
    /governance/rootNodes/Roots.sol
    /governance/rootNodes/RootsVoting.sol
    /governance/validators/Validators.sol
    /governance/validators/ValidatorsSlashingVoting.sol
    /governance/VotingWeightProxy.sol
/tokeneconomics/ATimeLockBase.sol
    /tokeneconomics/DefaultAllocationProxy.sol
/tokeneconomics/PushPayments.sol
```

```
    /tokeneconomics/QHolderRewardPool.sol
    /tokeneconomics/QHolderRewardProxy.sol
    /tokeneconomics/QVault.sol
    /tokeneconomics/RootNodeRewardProxy.sol
    /tokeneconomics/SystemReserve.sol
    /tokeneconomics/ValidationRewardPools.sol
    /tokeneconomics/ValidationRewardProxy.sol
/tokeneconomics/Vesting.sol
/tokeneconomics/WithdrawAddresses.sol
```

The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	26 October 2022	e765d34507dcb03ddac483e84aa846488f7a682e	Initial Version
2	7 November 2022	cad269b460664024502102879cb191ed7fc7ef13	Switch to hardhat
3	5 December 2022	480966e36c3f0f40ff3959d3c13e563a6acea17c	Version 3
4	8 December 2022	ac355b9e13684ab7b9f595b576c76150e225c98b	Version with fixes

For the solidity smart contracts, the compiler version `0.8.9` was chosen.

2.1.1 Excluded from scope

Any contracts not mentioned above and the mock and testing contracts might rely on scope contracts are not part of the scope. Imported libraries are assumed to behave according to their specification and are not part of the assessment scope.

2.2 System Overview

This system overview describes the initially received version ((Version 1)) of the contracts as defined in the Assessment Overview. Please consider our previous audit report for a more detailed overview of the system. This assessment was focused on the review of the following new features to the Q Blockchain system contracts:

1. Medianized price feed
2. Delayed price feed
3. Account aliases
4. Withdraw addresses
5. Root node approval voting
6. Other upgrades

This system overview describes the new features brought by the updated codebase.

2.2.1 Medianized price feed

The FxPriceFeedMedianizer is an implementation of the IFxPriceFeed interface. The FxPriceFeedMedianizer aggregates rates from multiple subfeed addresses. Subfeeds must call submit within roundTime seconds from the start of the round. Once the minSubmissionsCount limit of submissions is reached, the new median rate is computed and the round is closed. FxPriceFeedMedianizer.exchangeRate getter will start providing this new median return rate. New submission will reopen a new round. If within roundTime seconds from the start of the round minSubmissionsCount limit of submissions was not reached, the round closes without a exchangeRate update.

2.2.2 Delayed price feed

The FxPriceFeedSM is an implementation of the IFxPriceFeed interface. When created, the FxPriceFeedSM.priceFeed is set to another IFxPriceFeed implementation contract. The FxPriceFeedSM itself can be seen as an intermediate contract that provides a delayed exchange rate of the priceFeed contract. Any caller of the FxPriceFeedSM.updateExchangeRate function can set the _nextExchangeRate based on priceFeed.exchangeRate. After the next updateExchangeRate this rate will become _currentExchangeRate that FxPriceFeedSM.exchangeRate returns. At least 1 hour should pass between two successful updateExchangeRate calls. Owner of FxPriceFeedSM can:

- Change the priceFeed
- Pause the contract exchange rate updates
- Set isPriceValid boolean flag. The isPriceValid getter can be used by other contracts to know if the exchangeRate provided by the contract is valid or not.

2.2.3 Account aliases

The AccountAliases contract is a Q blockchain system contract that the node clients will query to determine the role of a given address. Validators need to hold significant value as a stake, but at the same time, they need to actively sign the blocks with their key. This creates an undesired mix of hot wallet functionality with the need for cold wallet security. One of the usages of AccountAliases is to allow the separation of validator stake address from block signer address. This is done by the setAlias(address _alias, uint256 _role) functionality. The msg.sender address can designate a certain alias address for a specific role. Q blockchain node clients will rely on these aliases for block validation. Aliases are one-to-one associated with main accounts. An attempt to reuse the same alias address for a different main address will fail. The alias address must first call reserve function to reserve the alias for the main address. This is needed to prevent the "hijacking" of alias addresses by malicious parties. The reserved main address can always override any existing association for this alias address, thus preventing hijacking. The following functions exist for querying the association table:

- resolve - returns the alias address for the given main address and role.
- resolveBatch - batch version of resolve.
- resolveReverse - returns the main address for the given alias address and role.
- resolveBatchReverse - batch version of resolveReverse.

2.2.4 Withdraw addresses

The `WithdrawAddresses` contract resolves the same problem as `AccountAliases`, but for validator addresses that have already operated in an undesired way for some time. Thus they can be potentially compromised. With the help of `WithdrawAddresses`, the main address as `msg.sender` can call the `change` function to set up an alias address. The owner of the `WithdrawAddresses` contract must call `finalize` to confirm the alias address. After the finalization, `Validators` and `Roots` contracts query this alias address from `WithdrawAddresses` during the `withdraw` function execution. Once the alias address is finalized, it cannot be changed. It is assumed that the owner of `WithdrawalAddress` will be the Q foundation-associated party that will perform proof of ownership and verify that the `change` was performed by the legit owner of the validator or root address.

2.2.5 Root node approval voting

Two new voting contracts were added to the Q system:

- `ContractRegistryAddressVoting` - allows Root nodes to vote on proposals to change the address entries in the `ContractRegistry` contract. The proposal can be created by the owner of the contract.
- `ContractRegistryUpgradeVoting` - allows Root nodes to vote on proposals to change the address of the proxy implementation in the `ContractRegistry` contract. The proposal can be created by the owner of the contract.

Similar to the other voting schemas on Q Blockchain, a proposal for a change must be created first. The root nodes have to approve it before the expiration time. Once the required majority of the root nodes have approved it, it will be executed.

2.2.6 Other upgrades

Multiple other upgrades were performed compared to the previous audit report version, such as:

- Solidity version was changed to `0.8.9`
- The OpenZeppelin `SafeMath` library was discarded due to the previous point.
- Strings that are used for `ContractRegistry` query are now defined in `Globals` file.
- Initial total supply of native Q tokens was set to 1 billion, instead of 10 billion.

In the findings section, we have added a version icon to each of the findings to increase the readability of the report.

2.2.7 Trust model

The following new roles appear in the updated version of the system:

- `FxPriceFeedMedianizer` owner: assumed to be fully trusted.
- `FxPriceFeedMedianizer` subfeed: assumed to provide correct price submission.
- `FxPriceFeedSM` owner: assumed to be fully trusted.
- `WithdrawAddresses` owner: assumed to be fully trusted.
- `ContractRegistryUpgradeVoting` owner: assumed to be fully trusted.
- `ContractRegistryAddressVoting` owner: assumed to be fully trusted.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of 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 are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	**Impact**		
	High	Medium	Low
High	**Critical**	**High**	**Medium**
Medium	**High**	**Medium**	**Low**
Low	**Medium**	**Low**	**Low**

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Trust: Violations to the least privilege principle

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	3

- Constitution and Experts Keys Risk Accepted
- FxPriceFeedMedianizer Price Feed Manipulation Risk Accepted
- FxPriceFeedMedianizer Subfeeds Consistency Risk Accepted

5.1 Constitution and Experts Keys

Design Low Version 1 Risk Accepted

All the registry keys for `common`, `governance`, and `tokeneconomics` have been grouped in `Globals.sol` as constant. However, it is not the case with other keys like the `constitution` and `governed` parameters keys. An accidental typo in the string constant can lead to potential system misconfiguration.

Risk accepted:

Q Blockchain accepts the risk and decides to leave the code as it is.

5.2 `FxPriceFeedMedianizer` Price Feed Manipulation

Design Low Version 1 Risk Accepted

Since all the data submitted by the subfeeds is immediately visible on-chain, malicious subfeed can decide what data to submit and manipulate the outcome of the computed round. By submitting a value above or below the current median the subfeed provider has limited control over the outcome. Assuming that subfeeds are trusted, this has a limited likelihood of happening.

Risk accepted:

Q Blockchain accepted the risk and states:

```
The subfeeds are considered trustworthy, also the price manipulation is quite limited in
range since the resulting price is a median and by definition insensitive to extreme values.
Therefore the impact of a malicious subfeed would be minimal.
```

5.3 `FxPriceFeedMedianizer` Subfeeds Consistency

Design **Low** **Version 1** **Risk Accepted**

Multiple `subFeed` parties need to provide the rate in a given round to compute the `exchangeRate`.

The following problems can potentially arise during the lifetime of this contract:

1. Due to the concurrent nature of the blockchain, subfeeds cannot control when their `submit()` call will be included in the chain history. In addition, the `submit` reverts if the same subfeed `msg.sender` tries to republish the rate in the same round. The `submit` transaction submitted at the very start of the new round may be counted in the previous round, due to random delays between transaction creation and block confirmation.
2. The subfeeds rates provided to the `FxPriceFeedMedianizer` contract are not sanitized. It is possible to provide subfeeds that do not match the pair, decimals, or base token address. The only sanity check performed on the submitted rate is the `rate >=minRateValue` check. A misconfiguration on the subfeed side can lead to a wrong `exchangeRate` as a result.

Risk accepted:

Q Blockchain accepts the risk and states:

```
1. We don't see a downside of their submission just going an earlier or later round.

2. If a subfeed reports the price for a wrong asset it will probably always be far
   from the median. So, it does not immediately affect the price but the owner can
   detect this and remove the subfeed. We consider this also low severity and would
   accept the risk.
```

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	1

- FullMath Operation Correctness Issue Code Corrected

Medium-Severity Findings	0
Low-Severity Findings	7

- Reward Allocation Can Be Blocked Code Corrected
- Code Duplication Code Corrected
- Floating Dependencies Versions Code Corrected
- Reentrancy Possibility on Root Node Approval Voting Code Corrected
- Unused Functionality and Libraries Code Corrected
- FxPriceFeedMedianizer Missing Events Code Corrected
- FxPriceFeedMedianizer Rate and Update Time Code Corrected

6.1 FullMath Operation Correctness Issue

Design High Version 1 Code Corrected

The `FullMath.mulDiv(uint a, uint b, uint denominator)` function performs the `floor(a×b÷denominator)` operation. A similar function exists in the UniswapV3 core codebase. However due to the change of the solidity compiler version from `0.7.6` to `0.8.9`, some modifications were made to account for the default SafeMath arithmetic operations behavior. This computation happens in `FullMath.mulDiv` code:

```
prod0 := add(prod0, mul(prod1, twos))
```

While Uniswap library performs this operation:

```
prod0 |= prod1 * twos;
```

Please note that this operation occurs in the `unchecked` block.

As a result, the `FullMath.mulDiv` computation yields undesired results.

For example, this assignment of arguments causes overflow and panic in the `FullMath.mulDiv`.

```
a = 2**255
b = 2**255
denominator = 57896044618658100000000000000000000000000000000000000000000000000000000000001
```

The correct computation should yield `5789604461865809542357098500868799828994258986484 4074485766219939100404438900`.

Code corrected:

The `FullMath.mulDiv` function has been replaced by the OpenZeppelin v4.8.0 implementation. It uses Solidity compiler above `0.8.0` version.

6.2 Reward Allocation Can Be Blocked

Trust Low Version 3 Code Corrected

In Version 3 of the code in `ValidationRewardProxy` and `RootNodeRewardProxy` the `Allocated` event was added:

```
emit Allocated(beforeAllocation - address(this).balance);
```

The allocation happens with the help of the `PushPayments` contract that performs a call with 30000 gas. Malicious Validator or Root in their fallback function can transfer an amount that is greater than `beforeAllocation` back to the reward proxy. This way the computation of the event argument will overflow. As a result, the allocation of rewards will be blocked. The severity of this issue is low since Roots are trusted and Validators can be slashed.

Code corrected:

A variable has been added to aggregate the allocated amounts instead of computing a difference.

6.3 Code Duplication

Design Low Version 1 Code Corrected

1. In the `Validators` contract, the call to `registry.mustGetAddress(RKEY__VOTING_WEIGHT_PROXY)` is inlined multiple times, while the calls to `registry.mustGetAddress(RKEY__VALIDATORS_SLASHING_VOTING)` are grouped under the `_getSlashingVotingAddress()` view function. Having a single getter function for the voting weight proxy registry key would be consistent with the rest of the codebase.
2. In `BorrowingCore`, the function `clearVault(address, _vaultId, _amountToClear, _beneficiary)` basically duplicates the functionality of `transferCol`, followed by `_clearVault`.

Code corrected:

1. The inlined calls have been replaced by more consistent calls to `_getConstitutionParametersAddress()` and `_getVotingWeightProxyAddress()`.
2. Functions `clearVault` and `transferCol` have been marked as deprecated and are kept for backward compatibility.

6.4 Floating Dependencies Versions

Design | Low | Version 1 | Code Corrected

The versions of `@opengsn` and `@openzeppelin` in `package.json` is not fixed. This could break the codebase if a new version has breaking changes. Upgradeable contracts that rely on proxy pattern can be rendered broken if the new version of the dependency contract introduces or changes the order of defined storage fields.

Code corrected:

The versions of `@opengsn` and `@openzeppelin` have been fixed to `2.2.4` and `4.3.3`.

6.5 Reentrancy Possibility on Root Node Approval Voting

Design | Low | Version 1 | Code Corrected

In the function `ARootNodeApprovalVoting._execute`, called by `ARootNodeApprovalVoting.approve`, there is a call to `onExecute`.

Since the `ARootNodeApprovalVoting` is abstract, contracts that inherit from it may execute arbitrary external code and re-enter during this call. The function `ARootNodeApprovalVoting.approve` can be called again the because `_proposal.executed` is set to true only after the call to `onExecute`. However, relying on the trust model, should not be an issue since the function can only be called by trusted root nodes when the majority is reached. Nevertheless, the Checks-Effects-Interactions pattern is violated.

Code corrected:

The flag `_proposal.executed` is set to `true` before the call to `onExecute`. The function adheres to the `check effect interaction` pattern now.

6.6 Unused Functionality and Libraries

Design | Low | Version 1 | Code Corrected

The following list contains problems due to imports, inheritance and usage of the libraries and contracts that are not needed. The redundant code can be removed.

- The `SafeMath` library is not used anymore in the code since the compiler version is now `0.8.9`. However, some `SafeMath` library imports are left in the source code, as well as some `using SafeMath for uint256`.
- The `PushPayments` contract is `Initializable` but the functionality is never used and can be removed.

Code partially corrected:

- The `SafeMath` library has been completely removed.

- The PushPayments contracts Initializable functionality while not used in the current version might be needed in the future. To prevent problems with storage layout it is necessary to keep the Initializable functionality of the PushPayments.

6.7 FxPriceFeedMedianizer Missing Events

Design | Low | Version 1 | Code Corrected

The FxPriceFeedMedianizer contract does not emit events when certain state changes occur. For example:

- addSubFeed
- removeSubFeed
- setMinSubmissionsCount
- setMinRateValue
- _closeRound

The lack of such events complicates the reproduction of the contract state off-chain. Thus, SubFeed providers might need custom solutions for monitoring the chain to know when a new rate must be submitted.

Code corrected:

The events ExchangeRateUpdated, SubFeedAdded, SubFeedRemoved, MinSubmissionsCountSet, and MinRateValueSet have been added in FxPriceFeedMedianizer.

6.8 FxPriceFeedMedianizer Rate and Update Time

Design | Low | Version 1 | Code Corrected

New exchangeRate is computed only after a round where minSubmissionsCount limit of submissions was reached.

External protocols and systems can mistakenly use stale exchangeRate if updateTime is not properly inspected. There is no default way to get both the rate and update time in a single call

---

Code corrected:

A getter returning both the rate and the time has been added.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Finalized Withdraw Address

Note **Version 1**

For an address to be finalized in `WithdrawAddresses`, a root/validator node must provide proof of ownership to Q Development AG. One of the limitations set by Q Blockchain is to disallow withdrawals to the main account.

7.2 Gas Optimization

Note **Version 1**

1. The majority of the updates of the form `x = x +/- y` have been optimized to `x +/-= y`, but there remains some unoptimized variable updates. An unexhaustive list is:
   - `BorrowingCore.depositCol`
     `userVaults[msg.sender][_vaultId].colAsset = userVaults[msg.sender][_vaultId]`
   - `BorrowingCore.getAggregatedTotals`:
     `_totalsInfo.outstandingDebt = _totalsInfo.outstandingDebt + _colOutstandingD`
   - `Saving.deposit`:
     `aggregatedNormalizedCapital = aggregatedNormalizedCapital + (_newNormalizedC`
   - `Saving.withdraw`:
     `aggregatedNormalizedCapital = aggregatedNormalizedCapital - (normalizedCapit`
2. The modifier `ARootNodeApprovalVoting.onlyRoot` takes an address as an argument, but is only used with `msg.sender`. Removing the argument and using directly `msg.sender` will save gas.

7.3 Storage Variables Visibility

Note **Version 1**

Some variables that are currently internal in contracts that are not inherited can be made private. Examples are: `AExpertsMembershipVoting.registry`, `Saving.registry`, `ValidationrewardPool.registry`, `FxPriceFeedMedianizer.pair`, or `Saving.stc`.