Quantstamp Security Assessment Certificate

March 10th 2022 — Quantstamp Verified

# Kresko Protocol

This audit report was prepared by Quantstamp, the leader in blockchain security.

## Executive Summary

| | |
|---|---|
| Type | DeFi Synthetics |
| Auditors | Mohsen Ahmadvand, Senior Research Engineer<br>Jan Gorzny, Blockchain Researcher<br>Cristiano Silva, Research Engineer |
| Timeline | 2021-11-17 through 2022-02-08 |
| EVM | Celo |
| Languages | Solidity |
| Methods | Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review |
| Specification | None |
| Documentation Quality | Medium |
| Test Quality | Undetermined |

Source Code

| Repository | Commit |
|---|---|
| kresko-protocol | df89fb1 |
| kresko-protocol | 6da14b7 (diff audit) |
| kresko-protocol | 5730bf9 (re-audit) |
| kresko-protocol | 3b2b305 (diff audit of the liquidation logic) |
| kresko-protocol | c9e58d0 (re-re-audit of the fixes) |

| | |
|---|---|
| Total Issues | **25** (10 Resolved) |
| High Risk Issues | **5** (3 Resolved) |
| Medium Risk Issues | **5** (4 Resolved) |
| Low Risk Issues | **5** (2 Resolved) |
| Informational Risk Issues | **2** (0 Resolved) |
| Undetermined Risk Issues | **8** (1 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

This audit was conducted on a subset of files in the Kresko repository, viz. `Kresko.sol`, `KreskoAsset.sol`, `NonRebasingWrapperToken.sol`, and `FixedPoint.sol`. It is worthwhile to mention that `tests` were excluded from the scope.
The provided documentation misses some aspects such as ownership and roles and the economics of the `NonRebasingWrapperToken` contract.
We identified several issues ranging from unprotected delegate call, owner's role in the mint and burn of assets, susceptibility to flash loans, and other logical pitfalls. Given the criticality of the liquidation logic, it needs to be verified using formal or empirical evaluations. That is, we cannot guarantee the correctness without the requested evaluations. The gas usage can be further optimised.

| ID | Description | Severity | Status |
|---|---|---|---|
| QSP-1 | `batch` allows anyone to run arbitrary transactions, for instance, to deplete the funds | ⇈ High | Fixed |
| QSP-2 | `KreskoAsset.mint` and `KreskoAsset.burn` are limited to `onlyOwner` | ⇈ High | Fixed |
| QSP-3 | `liquidate` falls short when there is not enough collateral to pay back | ⇈ High | Acknowledged |
| QSP-4 | Privileged Roles and Ownership | ⇈ High | Acknowledged |
| QSP-5 | Underspecified `NonRebasingWrapperToken` | ⇈ High | Fixed |
| QSP-6 | `IERC20.transferFrom` and `IERC20.transfer` may not return the expected value | ^ Medium | Fixed |
| QSP-7 | `chargeBurnFee` undercharges if burn fees exceed the available collateral balances | ^ Medium | Fixed |
| QSP-8 | Potential precision loss (truncation) errors | ^ Medium | Fixed |
| QSP-9 | `initialize` functions can be frontrun | ^ Medium | Acknowledged |
| QSP-10 | Gas-Intensive book keeping | ⌄ Low | Acknowledged |
| QSP-11 | Reentrancy risk in `withdrawUnderlying` | ⌄ Low | Fixed |
| QSP-12 | Unchecked inputs | ⌄ Low | Acknowledged |
| QSP-13 | Gas Usage / `for` Loop Concerns | ⌄ Low | Acknowledged |
| QSP-14 | Ownership can be renounced | ⌄ Low | Fixed |
| QSP-15 | Non-indexed events impose expensive lookups on clients | ○ Informational | Acknowledged |
| QSP-16 | Unlocked Pragma | ○ Informational | Acknowledged |
| QSP-17 | High liquidation incentive bounds | ? Undetermined | Acknowledged |
| QSP-18 | `KreskoAsset`'s supply is limited and thus susceptible to whale/flash loan attacks | ? Undetermined | Acknowledged |
| QSP-19 | `seizeAmount` can be greater than a collateral's available balance | ? Undetermined | Fixed |
| QSP-20 | `liquidate` can be frontrun | ? Undetermined | Acknowledged |
| QSP-21 | Asset duplicates are not prevented | ? Undetermined | Acknowledged |
| QSP-22 | Assumed number of decimals | ? Undetermined | Acknowledged |
| QSP-23 | Reliance on single oracle per asset | ? Undetermined | Acknowledged |
| QSP-24 | Liquidator cannot be the borrower | ^ Medium | Fixed |
| QSP-25 | Unverified liquidation | ? Undetermined | 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:

- Slither 0.8.1

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .`

# Findings

## QSP-1 `batch` allows anyone to run arbitrary transactions, for instance, to deplete the funds

**Severity:** *High Risk*

**Status:** Fixed

**Description:** The `Kresko` contract inherits from `BoringBatchable` contract. `BoringBatchable` comes with a batch function which is greatly unrestricted:

```
function batch(bytes[] calldata calls, bool revertOnFail) external payable {
       for (uint256 i = 0; i < calls.length; i++) {
           (bool success, bytes memory result) = address(this).delegatecall(calls[i]);
           if (!success && revertOnFail) {
               revert(_getRevertMsg(result));
           }
       }
   }
```

The `delegatecall` in the stated function can run transactions on the Kresko storage. The access control on the `batch` function is set to `external` enabling any user to modify Kresko's storage. An attacker can steal all the funds deposited into the Kresko protocol. Moreover, the `batch` function can call `destruct` leading to fund losses.

**Recommendation:** It is not clear what is intended to achieve with the `batch` function. Regardless of the intention, it may not have access to the `Kresko`'s storage. Moreover, it is not clear why any user should be able to run such a `batch` function. Consider limiting the access to the `batch` function (e.g., `onlyOperator`).

## QSP-2 `KreskoAsset.mint` and `KreskoAsset.burn` are limited to `onlyOwner`

Severity: *High Risk*

**Status:** Fixed

**File(s) affected:** `contracts/Kresko.sol:488`, `contracts/KreskoAsset.sol:29~31`

**Description:** The `mintKreskoAsset` and `burnKreskoAsset` functions call mint and burn on the input `_kreskoAsset`. However, the modifier of the `KreskoAsset.mint` and `KreskoAsset.burn` are set to `onlyOwner`. That is, the `Kresko` contract has to become the owner of the `KreskoAsset` contracts after deployment. We assume that the Kresko team plans to call `transferOwnership` on KreskoAssets with the address of the deployed Kresko contract. However, this is not explicitly done in the reviewed contracts.

**Recommendation:** Ensure that the owner is set properly for all the `KreskoAsset` contracts. Consider disjointing the minter role from the owner role (who is assigned to the deployer automatically). Best is to carry out an owner/minter check ensuring that Kresko can call mint and burn upon adding `KreskoAssets` to the protocol (through `addKreskoAsset`).

## QSP-3 `liquidate` falls short when there is not enough collateral to pay back

Severity: *High Risk*

**Status:** Acknowledged

**Description:** Some crypto assets can drop substantially in value before liquidation. It is not clear what is supposed to happen if the users' collateral is not sufficient to pay an underwater position. In the specs, it is stated that `KreskoAsset` will be swapped to pay such debts. It is, however, not integrated into the contract.

**Recommendation:** Consider adding the logic for liquidating underwater positions with insufficient collateral to pay back their debts. One possibility is to enable liquidators to buy KreskoAssets at a discounted rate upon unpayabe position liquidations. The debt needs to be added to the protocol debt and possibly paid with burning fees or other protocol fees.

**Update:** Kresko response:

> We acknowledge that sufficient capital must be available to facilitate good samaritan liquidations. Some potential solutions we've considered for future versions include: As recommended, accrued protocol fees could be used as underwater liquidation liquidity. A token insurance pool where token holders lock Kresko tokens to be used for underwater liquidations in return for a yield generated from supply inflation. Kresko protocol acts as the liquidator of last resort, directly minting Kresko tokens to raise the collateral required to back liquidations.

## QSP-4 Privileged Roles and Ownership

Severity: *High Risk*

**Status:** Acknowledged

**Description:** Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.
The owner can set `feeRecipient` (an externally owned account), set incentive, and liquidation factors giving them the power to potentially underwater some positions followed with liquidating their collaterals.

**Recommendation:** 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. Kresko should explicitly state the power of the owner in their public-facing documentations.

**Update:** Kresko response:

> Added explicit comments on the contract owner's privileged role in #32. Future user facing documentation will acknowledge the owner's privileged role.

## QSP-5 Underspecified `NonRebasingWrapperToken`

Severity: *High Risk*

**Status:** Fixed

**Description:** The provided specs does not include any explanation over the usage and economies of the `NonRebasingWrapperToken.sol`. This severely limits the understanding of the auditor of the intended use.

**Recommendation:** Add sufficient specs for the missing parts.

## QSP-6 `IERC20.transferFrom` and `IERC20.transfer` may not return the expected value

Severity: *Medium Risk*

**Status:** Fixed

**File(s) affected:** `contracts/Kresko.sol`, `contracts/misc/NonRebasingWrapperToken.sol`

**Description:** Some ERC-20 Tokens do not comply with returning true if a transfer goes through, e.g., USDT (see here). Consequently, `depositCollateral`, `depositUnderlying`, and `withdrawUnderlying` will always revert for all non-compliant ERC-20 tokens (see a list here.

**Recommendation:** Consider using SafeERC20 wrapper to handle noncompliant ERC20 tokens, safely.

## QSP-7 `chargeBurnFee` undercharges if burn fees exceed the available collateral balances

Severity: *Medium Risk*

**Status:** Fixed

**File(s) affected:** `contracts/Kresko.sol:951~1025`

**Description:** The `_chargeBurnFee` function will undercharge if the available collateral balances are short (e.g., due to liquidations).

**Recommendation:** Ensure that liquidation pays burning fees. Consider reverting if the fees can not be paid in full.

## QSP-8 Potential precision loss (truncation) errors

Severity: *Medium Risk*

**Status:** Fixed

**File(s) affected:** `contracts/Kresko.sol`, `contracts/misc/NonRebasingWrapperToken.sol`

**Description:** The reviewed contracts carry out multiplication after division. This can cause precision loss or truncation in the calculations:

1. `contracts/Kresko.sol:1055~1058`

```
// Seize amount = (repay amount USD / exchange rate of collateral asset) * liquidation incentive.
FixedPoint.Unsigned memory seizeAmount =
    _kreskoAssetRepayAmountUSD
        .div(oraclePrice) // Denominate seize amount in collateral type
        .mul(liquidationIncentive); // Apply liquidation percentage
```

1. `contracts/misc/NonRebasingWrapperToken.sol:156~165`

```
        FixedPoint.Unsigned memory shareOfToken =
            FixedPoint.Unsigned(_nonRebasingAmount).div(FixedPoint.Unsigned(_totalSupply));
        uint256 underlyingBalance = underlyingToken.balanceOf(address(this));
        // Because shareOfToken has a max rawValue of 1e18, this calculation can overflow
        // if underlyingBalance has a value of ((2^256) - 1) / 1e18. For an underlying
        // token with 18 decimals, this means this contract can at most tolerate an underlying
        // balance of ((2^256) - 1) / 1e18 / 1e18 = 115792089237316195423570985008687907853269
        // whole tokens. This is more than enough for any reasonable token, though
        // keep this in mind if the underlying has many decimals or a very low value.
        return shareOfToken.mul(FixedPoint.Unsigned(underlyingBalance)).rawValue;
```

**Recommendation:** Consider to multiply before divide.

## QSP-9 `initialize` functions can be frontrun

Severity: *Medium Risk*

**Status:** Acknowledged

**File(s) affected:** `contracts/Kresko.sol`, `contracts/KreskoAsset.sol`, `contracts/misc/NonRebasingWrapperToken.sol`

**Description:** The `initialize` function that initializes important contract state can be called by anyone. The attacker can initialize the contract before the legitimate deployer, hoping that the victim continues to use the same contract. In the best case for the victim, they notice it and have to redeploy their contract costing gas. It it worthwhile to mention that `initialize` sets the contract owner. That is, attackers can potentially set themselves as the owner of the contract by frontrunning the `initialize` function call.

**Recommendation:** Use the constructor to initialize non-proxied contracts. For initializing proxy contracts deploy contracts using a factory contract that immediately calls `initialize` after deployment or make sure to call it immediately after deployment and verify the transaction succeeded.

**Update:** Kresko response:

> We acknowledge the potential downsides of having the initializer function frontrun and consider them insufficient to warrant changes. In cases of initialization frontrunning, Kresko's deployment scripts will revert and a redeployment will take place.

## QSP-10 Gas-Intensive book keeping

Severity: *Low Risk*

**Status:** Acknowledged

**File(s) affected:** `contracts/Kresko.sol:114~126`

**Description:** As constructed Kresko stores and furthermore maintains some states that can be computed with simple lookup functions.
``` solidity
mapping(address => mapping(address => uint256)) public collateralDeposits;
/// @notice Mapping of account address to an array of the addresses of each collateral asset the account /// has deposited. mapping(address => address[]) public depositedCollateralAssets;
/* ===== General state - Kresko Assets ===== */
/// @notice Mapping of Kresko asset token address to information on the Kresko asset. mapping(address => KrAsset) public kreskoAssets;
/// @notice Mapping of Kresko asset symbols to whether the symbol is used by an existing Kresko asset. mapping(string => bool) public kreskoAssetSymbols; ```
```
Storing state is expensive in gas. There is also an overhead of maintaining the values throughout the contract functions. If there is a quick way to compute needed values, it should be favoured over writing state variables.

**Recommendation:** Consider implementing view functions to obtain persisted values in `kreskoAssetSymbols` and `depositedCollateralAssets` programmatically.

**Update:** Kresko response:

> Using public view getDepositedCollateralAssets means the protocol must loop over each enabled collateral asset to retrieve the addresses of all collateral assets the user has deposited. This makes internal contract calls that need the user's array of collateral addresses more expensive since the amount of collateral assets in the protocol will likely be larger than unique collateral addresses a user has in the depositedCollateralAssets and equal in the best case.

## QSP-11 Reentrancy risk in `withdrawUnderlying`

Severity: *Low Risk*

**Status:** Fixed

**File(s) affected:** `contracts/misc/NonRebasingWrapperToken.sol#91-111`

**Description:** The underlying token is assumed to be trusted hence the low severity. It is nevertheless a better practice to make any external call after state changes.

```
Reentrancy in NonRebasingWrapperToken.withdrawUnderlying(uint256) (contracts/misc/NonRebasingWrapperToken.sol#91-111):
    External calls:
    - require(bool,string)(underlyingToken.transfer(msg.sender,underlyingAmount),NRWToken: underlying transfer out failed) (contracts/misc/NonRebasingWrapperToken.sol#99)
    State variables written after the call(s):
    - _burn(msg.sender,_nonRebasingWithdrawalAmount) (contracts/misc/NonRebasingWrapperToken.sol#108)
        - _balances[account] = accountBalance - amount (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#281)
    - _burn(msg.sender,_nonRebasingWithdrawalAmount) (contracts/misc/NonRebasingWrapperToken.sol#108)
        - _totalSupply -= amount (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#283)
```

**Recommendation:** Consider adding non-rentrant or moving the transfer after the `_burn` call.

## QSP-12 Unchecked inputs

Severity: *Low Risk*

**Status:** Acknowledged

**File(s) affected:** `NonRebasingWrapperToken.sol`

**Description:** NonRebasingWrapperToken.sol: `initialize` does not check that `_underlyingToken` is not a zero address. Symbol and name are also not checked against empty string.

**Recommendation:** Add a `require` statement to check that the inputs.

**Update:** Partially fixed - symbol and name are still not checked.

## QSP-13 Gas Usage / `for` Loop Concerns

Severity: *Low Risk*

**Status:** Acknowledged

**File(s) affected:** `kresko.sol`

**Description:** Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. For example, if a `for` loop requires too much gas to exit, then it may prevent the contract from functioning correctly entirely. It is best to break such loops into individual functions as possible. Kresko comes with several for loops.

**Recommendation:** Make sure loops are safe and have a small bound.

**Update:** Kresko response:

> Extensive gas testing was performed in #40, giving us confidence in the current design. We will mitigate the existing gas limitations by keeping the number of whitelisted assets on each chain below the chain's respective gas limit.

## QSP-14 Ownership can be renounced

Severity: *Low Risk*

**Status:** Fixed

**Description:** The `Ownable.sol` has the `function renounceOwnership()`. Although it can only be called by the `owner`, such a function leaves the contract without an owner rendering contracts useless. Particularly, KreskoAssets can no longer be burnt or minted in case of ownership renounces.

**Recommendation:** Consider using a two-step ownership transfer scheme and overriding the `renounceOwnership` function.

## QSP-15 Non-indexed events impose expensive lookups on clients

Severity: *Informational*

**Status:** Acknowledged

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

**Description:** Kresko's events do not utilize indexes upon event attributes. It is often useful to have indexed keys (e.g., account address) in events to provide a better user experience on the client side.

**Recommendation:** Consider adding `indexed` keyword to the wallet address in the events.

**Update:** Partially fixed. Indexes are added, but event `KreskoAssetAdded` indexes a string `symbol`. This doesn't work! See, e.g., https://ethereum.stackexchange.com/questions/6840/indexed-event-with-string-not-getting-logged

## QSP-16 Unlocked Pragma

Severity: *Informational*

**Status:** Acknowledged

**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".
Different versions of Solidity is used:

```
- Version used: ['>=0.8.4', '^0.8.0', '^0.8.4']
- ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3)
- ^0.8.0 (node_modules/@openzeppelin/contracts/security/ReentrancyGuard.sol#3)
- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#3)
- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3)
- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#3)
- ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3)
- ^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/SafeMath.sol#3)
- ^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/SignedSafeMath.sol#3)
- >=0.8.4 (contracts/BasicOracle.sol#2)
- >=0.8.4 (contracts/Kresko.sol#2)
- >=0.8.4 (contracts/KreskoAsset.sol#2)
- >=0.8.4 (contracts/interfaces/IOracle.sol#2)
- ^0.8.4 (contracts/libraries/FixedPoint.sol#2)
- >=0.8.4 (contracts/misc/NonRebasingWrapperToken.sol#2)
- >=0.8.4 (contracts/test/RebasingToken.sol#2)
```

**Recommendation:** 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.

**Update:** The submitted fixes do not perform locking correctly. The locking should target a specific version.

## QSP-17 High liquidation incentive bounds

Severity: *Undetermined*

**Status:** Acknowledged

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

**Description:** The liquidation fee min and max bounds seem to be pretty large. We are not sure about the economical implications of such large margins of incentives. Such high incentives can potentially subject a larger portion of users' collateral to unhealthy ratio leading to larger funds being underwater.

1. Minimum incentive bound is set to 100%. This mean that the liquidator is set to recieve a minimum of 100% fees upon liquidation!
2. Why the maximum is set to 150%? To the best of my knowledge, in Compound the liquidation incentive is well below 10%...
3. What if the liquidation fees are larger than the balance (deposited collateral)?

```
/// @notice The minimum configurable liquidation incentive multiplier.
    uint256 public constant MIN_LIQUIDATION_INCENTIVE = 1e18; // 100%

/// @notice The maximum configurable liquidation incentive multiplier.
    uint256 public constant MAX_LIQUIDATION_INCENTIVE = 1.5e18; // 150%

// TODO: consider implications
```

1. There is a TODO in the code indicating potential implication overlooks: `consider implications`

**Recommendation:** Consider acceptable and tested liquidation incentive ranges.

**Update:** Kresko response:

> The min/max liquidation incentive range is 0-50%, as opposed to the 100-150% range noted in the audit report. To clarify the values we added additional documentation and updated relevant variable names in #44.

## QSP-18 `KreskoAsset`'s supply is limited and thus susceptible to whale/flash loan attacks

Severity: *Undetermined*

**Status:** Acknowledged

**Description:** It is not clear how the value (price) of `KreskoAsset`s are set. If it is left to market (supply and demand), it can have consequences on open positions. `KreskoAsset`s with limited supply can be targeted by whale attackers. An attacker can buy a large portion of the supply inflating the price temporarily. The price raise can potentially render user positions unhealthy. The attacker can then buy users collateral at a discounted rate by liquidating unhealthy positions. The same attack can be mounted with flash loans if the Oracle's feeds are easily manipulatable.

**Recommendation:** Ensure that the price cannot be easily manipulated by whale attackers by having a large enough supply. One possibility is to fix the floor for `KreskoAsset`s by offering a buy back mechanism at a fixed price in case of market manipulations.

**Update:** Kresko response:

> Price manipulation is unlikely because price data is sourced from an established oracle instead of an AMM. Oracle data providers are incentivized to provide correct data and if some providers do have malicious intent, the majority of providers for a given feed would need to provide false data for it to reflect in the aggregated price. While it's still possible for this to happen, we regard it as highly unlikely and are satisfied with current oracle security.

## QSP-19 `seizeAmount` can be greater than a collateral's available balance

Severity: *Undetermined*

**Status:** Fixed

**File(s) affected:** `contracts/Kresko.sol:1193`

**Description:** Given the high liquidation incentives (which was reflected in a separate issue) and the fact that liquidation takes strictly one target paying collateral asset, it is very likely that the calculated `siezeAmount` exceeds the balance. In this case, the collateral subtract statement would result in underflow causing a transaction revert:

`collateralDeposits[_account][_collateralAssetToSeize] = collateralDeposit - _seizeAmount;`

This would cost gas for a failing transaction. In some cases liquidation may require multiple transactions to liquidate unhealthy positions whose debt cannot be paid by a single collateral asset.

**Recommendation:** Consider adding a view function that can be externally called by liquidators to compute the `seizeAmount` facilitating off-chain automations.

**Update:** The issue is fixed. However, the fix does not follow our recommendation.
Kresko response:

> Implemented handling of liquidations where the seized amount is greater than a collateral's available balance in #45.

## QSP-20 `liquidate` can be frontrun

Severity: *Undetermined*

**Status:** Acknowledged

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

**Description:** The `liquidate` function can be fronrun by liquidators to harvest the incentives.

**Recommendation:** Consider having an auction scheme for the liquidation process.

**Update:** Kresko response:

> Implemented an additional liquidation mechanism to allow non-profitable liquidations to be made by a good-hearted third party or protocol backers, we introduced parameter `_keepKrAssetDebt` for function liquidate in #50 which can be used by the liquidator to only receive pure profit as collateral instead of the whole seized amount, reducing friction and risk for the liquidator. The optimized liquidation mechanism helps mitigate the risk of positions going underwater.

## QSP-21 Asset duplicates are not prevented

Severity: *Undetermined*

**Status:** Acknowledged

**File(s) affected:** `Kresko.sol`

**Description:** In `addKreskoAsset`, the same asset name could be added twice. When an asset is added, it deploys a new contract, and the existence of that contract's address is stored. However, calling this multiple times with the same data deploys multiple instances of the contract, so you could have more than one copy.

**Recommendation:** Don't check the existence of the deployed contract. Instead, check the existence of an asset with that symbol name in a mapping (not a mapping to a struct)

**Update:** Kresko response:

> Duplicate asset symbols are prevented by the kreskoAssetDoesNotExist modifier on function addKreskoAsset.

## QSP-22 Assumed number of decimals

**Severity:** *Undetermined*

**Status:** Acknowledged

**File(s) affected:** `NonRebasingWrapperToken.sol`

**Description:** There is a comment "This is more than enough for any reasonable token, though keep this in mind if the underlying has many decimals or a very low value." (Line 144)

**Recommendation:** If more than 18 decimals are not supported, explicitly reject such assets.

**Update:** Kresko response:

> In order to reasonably support irregular collateral assets with greater than 18 decimals the calculation was updated in #52. Additional code exists throughout the protocol to handle assets with irregular decimals e.g. Kresko.sol's `_toCollateralFixedPointAmount`.

## QSP-23 Reliance on single oracle per asset

**Severity:** *Undetermined*

**Status:** Acknowledged

**File(s) affected:** `Kresko.sol`

**Description:** Reliance on a single oracle per asset may be problematic for flash loans. One oracle may be easier to manipulate than multiple.

**Recommendation:** Aggregate several results, or clarify how the oracles work to ensure that they cannot be easily manipulated.

**Update:** Kresko response:

> The oracle price data is aggregated from several data providers. The majority of data providers for a given feed would have to provide false data for it to reflect in the aggregated data

## QSP-24 Liquidator cannot be the borrower

**Severity:** *Medium Risk*

**Status:** Fixed

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

**Description:** As constructed the contract allows a borrower to act as liquidator for their own undercollaterlized positions. This may lead to economical issues in the protocol in which borrowers seize their own collateral at a discounted rate. With the modified `liquidate` function that enables paying back Kresko debt, i.e., `_keepKrAssetDebt=false`, having identical borrower and liquidator may lead to a faulty state.

**Recommendation:** Add a check in the liquidation logic to prevent borrowers from liquidating their own positions. See a comparable check in the Compound protocol.

## QSP-25 Unverified liquidation

**Severity:** *Undetermined*

**Status:** Acknowledged

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

**Description:** We are unable to fully verify the correctness of the liquidation logic based on the provided tests. In our time-boxed review, we did not find any exploits (other than those that are listed in this report). Nevertheless, liquidation is a critical part of the protocol and therefore further evaluations need to be conducted to verify its correctness.

**Recommendation:** We strongly recommend formal verifications. If not, consider running simulations with real-world data using liquidation transactions of comparable protocols.

**Update:** Our recommendations were not addressed but further test cases including flash loan tests were added.
Response from the Kresko team:

> Removed the closeFactor from the protocol and replaced it with a new `calculateMaxLiquidatableValueFor(address _account)` function that simply returns the missing collateral USD amount for the user to be considered healthy and added additional liquidation and protocol solvency tests in #68

# Automated Analyses

### Slither

We analyzed the slither findings and included the true-positives to the reported issues.

# Adherence to Best Practices

1. `Kresko.sol`: the comment on `isAccountLiquidatable` is not correct: it returns true if the collateral value is less than the minimum account collateral value. This is not defined as the health factor anywhere in the contract. This phrase appears in other comments, too.

2. In `Kresko.withdrawCollateral(...)` the code reverts if the amount is greater than the deposited collateral in L418. One way to avoid another call to this function is to adjust the withdrawal value:`_amount = (_amount <= depositAmount ? _amount : depositAmount);`

3. The cost of the revert operation is proportional to the string length passed as parameter. Consider using the newly introduced gas-efficient revert statements (see here).

# Test Results

**Test Suite Results**

The tests were not in the audit scope

# Code Coverage

We strongly advise reaching 100% coverage for all the contracts.

| File | % Stmts | % Branch | % Funcs | % Lines | Uncovered Lines |
|---|---|---|---|---|---|
| contracts/ | 97.49 | 89.83 | 98.15 | 97.56 | |
| BasicOracle.sol | 83.33 | 50 | 80 | 85.71 | 27 |
| Kresko.sol | 97.78 | 90.35 | 100 | 97.83 | … 3,1235,1237 |
| KreskoAsset.sol | 100 | 100 | 100 | 100 | |
| contracts/interfaces/ | 100 | 100 | 100 | 100 | |
| IKreskoAsset.sol | 100 | 100 | 100 | 100 | |
| INonRebasingWrapperToken.sol | 100 | 100 | 100 | 100 | |
| IOracle.sol | 100 | 100 | 100 | 100 | |
| contracts/libraries/ | 13.86 | 25 | 14.93 | 13.86 | |
| Arrays.sol | 100 | 100 | 100 | 100 | |
| FixedPoint.sol | 9.38 | 0 | 13.64 | 9.38 | … 764,765,766 |
| contracts/misc/ | 100 | 91.67 | 100 | 100 | |
| NonRebasingWrapperToken.sol | 100 | 91.67 | 100 | 100 | |
| contracts/utils/ | 100 | 100 | 100 | 100 | |
| OwnableUpgradeable.sol | 100 | 100 | 100 | 100 | |
| **All files** | **75.84** | **83.55** | **57.35** | **76.4** | |

# 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

```
fd330bd2335a2bfbefb41b27a4b2c0ee9a6674b3bb4f4993acbd523ef0a77836 ./NonRebasingWrapperToken.sol
1e8c768dedc780d882418f6503db036f15cb7811b2721955c8d19b5ab28df421 ./Kresko.sol
d59ee202bb89fde0ba1bf87267948280959cc51966a84b28a58d7718d5cc6c0f ./KreskoAsset.sol
1fbe58ba41f2c419b4e4afc6f354d1eab2feb3060a3b2a0b2bece4acc670a3eb ./contracts/interfaces/INonRebasingWrapperToken.sol
f53aa44b32fb0dedd18cd6444627521d67fb1f8ff6666be8392574f21ae482a8 ./contracts/interfaces/IKreskoAsset.sol
b8392f7b794539b945d565ad8dc86a3f23ae8256f83a4421bef587609376f35e ./contracts/libraries/FixedPoint.sol
```

### Tests

```
9ac1e2096092f21489ee0e5a3c5d095a2a3685bb6052e35cd6eb923532d8e611 ./test/Kresko.ts
a96a0b35d71b5f80b1bb36a5a3bc5d7daf52491e166b9ffa4d205c11198bd447 ./test/KreskoAsset.ts
0d256c3a97ea4ffcd5b202c2d535d2984f677f367eea759d56427c1641c5346c ./test/misc/NonRebasingWrapperToken.ts
```

## Changelog

- 2021-11-20 - Initial report
- 2021-12-15 - Diff audit
- 2022-01-26 - Re-audit
- 2022-02-08 - Diff audit of the liquidation logic
- 2022-03-10 - Re-re-audit liquidation logic fixes

## 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.