SOLIDIFIED

Audit Report for Kresko Labs Pte. Ltd - March 5, 2024

Summary

Audit Report prepared by Solidified covering the Kresko Synthetic Asset Protocol.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on February 19, 2023, and the results are presented here.

Audited Files

The source code has been supplied in a public GitHub repository:

https://github.com/kreskohq/kresko-protocol

Commit number: `92ae69d62b58f2dac566068f41b114ef073097e5`

Fixes received at commit: `e400396366e3d790d9231b3c722cac86b8dd1cd5`

The scope of this follow-up audit was limited to the following areas of the protocol:

- Primary oracle change from RedStone to Pyth.
- Expanded pause state coverage to include SCDP/SDI functionality.

Intended Behavior

The contracts implement a non-custodial, capital-efficient synthetic asset protocol that runs on the EVM. It facilitates creating and managing securely collateralized synthetic assets using smart contracts written in Solidity.

Specifically, users can mint the internal KISS stablecoin token by depositing various collateral assets into the KISS vault as well as deposit collateral into the shared collateralized debt position (SCDP) that have their value combined, enabling users to swap KISS tokens with synthetic assets referred to as Kresko Assets using the backing of the shared CDP. Moreover, users can participate in liquidations in case of an unhealthy SCDP.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Medium-High	In several places, token amounts are potentially rebased and thus require normalization. Any such oversight might lead to unintended consequences.
Code readability and clarity	High	The code is easy to read and, in general, very clear. Many naming conventions, code patterns, and other best practices were implemented.
Level of Documentation	Medium	Almost all functions are well documented with NatSpec and inline comments, which greatly enhance the overall understanding of the codebase. Additionally, the Gitbook documentation provides an in-depth explanation of how the protocol is intended to work. However, the recent changes that are part of this audit have

		not been sufficiently documented. Thus, certain assumptions, such as using multiple collateral tokens for the SCDP, had to be made.
Test Coverage	Medium-High	Due to the inability to determine the test coverage with the hardhat-coverage plugin, as well as having tests fragmented across Hardhat and Foundry, it was not possible to verify the overall test coverage. Nevertheless, based on our subjective assessment, we consider the test coverage to be medium-high. However, we recommend extending the test suite with more granular and detailed tests to ensure edge cases are covered.

Issues Found

Solidified found that the Kresko Labs contracts contain 0 critical issues, 0 major issues, 8 minor issues, and 3 informational notes.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	setAssetIsSharedOrSwappedCollateralSCDP can set isSharedOrSwappedCollateral to true even if both conditions are not true	Minor	Acknowledged
2	CommonConfigFacet.initializeCommon does not set minimum debt value by default	Minor	Acknowledged
3	Fallback oracle can not be utilized in case of stale price	Minor	Acknowledged
4	MinterLiquidationFacet.liquidate does not have pause functionality	Minor	Resolved
5	SCDPSWapFacet.cumulateIncomeSCDP does not check for asset pause state	Minor	Acknowledged
6	Pyth oracle confidence interval is not utilized	Minor	Acknowledged
7	Unsafe type casting from signed to unsigned integer in the normalizePythPrice and invertNormalizePythPrice functions	Minor	Acknowledged
8	Incorrect oracle type used for retrieving the Pyth price via CommonStateFacet.getPythPrice	Minor	Resolved
9	SCDPSwapFacet.previewSwapSCDP returns a value even if the assets are paused	Note	Acknowledged
10	Possible array index out of bounds in PythScript.getMockPythPayload	Note	Acknowledged

11	Possible use of residual ETH to pay Pyth's oracle feed update fee	Note	Acknowledged

Critical Issues

No critical issues were found.

Major Issues

No major issues were found.

Minor Issues

1. `setAssetIsSharedOrSwappedCollateralSCDP` can set `isSharedOrSwappedCollateral` to `true` even if both conditions are not true

In `SCDPConfigFacet.sol`, a new setter function was introduced to set the asset as either shared or swapped collateral SCDP. The boolean `isSharedOrSwappedCollateral` is used in other places to represent the conditional `(asset.isSharedCollateral || asset.isSwapMintable)`.

However, this admin-callable setter function can set the boolean value to `true` even if neither of the two conditions is `true`. This can cause issues in other parts of the protocol that depend on the individual flags `isSharedCollateral` and `isSwapMintable`.

Recommendation

We recommend adding a check to `setAssetIsSharedOrSwappedCollateralSCDP` to also update the other individual flags in case the boolean `isSharedOrSwappedCollateral` is set or unset.

Status
Acknowledged

2. `CommonConfigFacet.initializeCommon` does not set minimum debt value by default

In `CommonConfigFacet.sol`, the `setMinDebtValue` function was removed. This means by default `minDebtValue` will be 0, and users will be able to create very small positions (i.e., "dust" positions) until the admin manually calls `MinterConfigFacet.setMinDebtValue`.

Recommendation
We recommend setting by default the minimum debt value to a global non-zero parameter that can be later adjusted by the admin.

Status
Acknowledged. Team's response: *"This is set during the call to the MinterConfigFacet.initializerMinter function"*.

3. Fallback oracle can not be utilized in case of stale price

---

The `safePrice` function in `Prices.sol` retrieves the asset price from the provided primary and secondary oracle feeds using safety checks for deviation, staleness, and L2 sequencer uptime. If either the primary or secondary oracle returns a price of 0 while the other returns a non-zero price, the `safePrice` function will fall back to the non-zero price. Otherwise, it will revert.

However, the oracle price setter functions `pythPrice`, `aggregatorV3Price`, and `API3Price` revert if the price is stale instead of returning `0`. Consequently, the fallback oracle can not be utilized if one of the oracles returns a stale price, requiring both oracles always to return a

non-stale price. This defeats the purpose of a two-oracle system, where one is supposed to act as a fallback oracle.

Recommendation

We recommend returning `0` instead of reverting in the above-mentioned oracle price setter functions to allow the fallback oracle to be utilized.

Status

Acknowledged. Team's response: *"This is intentional. We use push oracles only to check that the price deviation is within desired range. The secondary oracles are push based (eg. CL/API3) and they require quite high price movement for updates, even in L2's.*

Since we will be using a very low stale time for the pull-based primary oracle (<10sec) - it is expected that stale prices will happen. If we allowed users to execute with the reference price when the primary price is "stale" it is very likely to lead to latency arbitrage scenarios where liquidity depositors in the shared model would pay for the profits.

We think these scenarios are more effective to mitigate than preparing for the scenario where the primary oracle is constantly unperformant.".

4. `MinterLiquidationFacet.liquidate` does not have pause functionality

In `MinterLiquidationFacet.sol`, the `liquidate` function does not have pause functionality, which means assets can be liquidated through the regular flow even if the SCDP liquidation mechanism is paused.

If an asset is set, at the same time, as a "minter mintable"/"minter collateral" and as an "active shared collateral"/"swap mintable", it may cause unwanted liquidations because of the inability to pause in one of the liquidations flow.

Recommendation

We recommend applying the same pause logic to the two flows.

Status
Resolved

5. SCDPSWapFacet.cumulateIncomeSCDP does not check for asset pause state

---

In `SCDPSwapFacet.sol`, the `cumulateIncomeSCDP` function is meant to accumulate fees to deposits as a fixed, instantaneous income.

However, this function does not check if the asset is paused. As a result, any user can trigger the `cumulateIncome` function and update the `currFeeIndex` liquidity index. This means anyone can make `_assetAddr` accrue fees, but this does not seem to cause any impact other than a positive update on the liquidity index. After unpause, the index will be higher than before the pause, which might be confusing or unwanted, assuming deposits will also have been paused.

Recommendation
We recommend also pausing `cumulateIncomeSCDP` when other functions that change the `currFeeIndex` are paused.

Status
Acknowledged. Team's response: *"This is intentional".*

6. Pyth oracle confidence interval is not utilized

The `pythPrice` function in `Prices.sol` retrieves the asset price from the Pyth oracle feed via the `getPriceNoOlderThan` function. The returned `IPyth.Price` data includes, besides the price and the exponent, the confidence interval of the price. This confidence interval represents the estimated uncertainty of the reported price and is intended to achieve 95% coverage, i.e., the price publisher expresses the belief that this interval contains the "true" price with 95%

probability. By using and checking the confidence interval, the stability of the price and the current market volatility can be assessed, allowing the protocol to react accordingly.

However, the `pythPrice` function does not utilize the confidence interval to check for price uncertainty, missing an opportunity to react to the current market volatility.

For more information, please refer to the following resources:

- Pyth Network Best Practices - Confidence Interval
- Pyth Primer: Don't Be Pretty Confident. Be Pyth Confident

Recommendation
We recommend utilizing the confidence interval returned by the Pyth oracle.

Status
Acknowledged

7. Unsafe type casting from signed to unsigned integer in the `normalizePythPrice` and `invertNormalizePythPrice` functions

---

The `normalizePythPrice` and `invertNormalizePythPrice` functions in `Prices.sol` normalize the Pyth oracle price returned by the `getPriceNoOlderThan` function to 18 decimals. However, as the price is cast from `int64` to `uint64` in lines `123` and `136` without checking if the value is negative, the resulting price can be unexpectedly large.

As this inflated asset price will most likely deviate from the other oracle price, this invalid price will be detected in the `deducePrice` function, preventing the invalid price from being used in the protocol. Thus, we classify this finding as minor.

Recommendation
We recommend checking if the Pyth oracle price is negative before casting it to a `uint64`.

Status
Acknowledged

8. Incorrect oracle type used for retrieving the Pyth price via CommonStateFacet.getPythPrice

In CommonStateFacet.sol, the getPythPrice function is intended to return the Pyth price but is using the Enums.OracleType.Redstone oracle as the OracleType in line 60 instead of Enums.OracleType.Pyth, resulting in the inability to return the correct Pyth price due to missing the necessary Pyth feed configuration.

Recommendation
We recommend changing the OracleType to Enums.OracleType.Pyth in line 60.

Status
Resolved

Informational Notes

9. SCDPSwapFacet.previewSwapSCDP returns a value even if the assets are paused

In `SCDPSwapFacet.sol`, the `previewSwapSCDP` function is meant to preview the amount out to be received after a swap.

However, this function does not rely on the asset's swap pause state to return the expected output amount. As a result, it can lead to unwanted consequences if the user relies on it to perform a swap.

Recommendation

We recommend returning 0 in case the asset is paused.

Status

Acknowledged

10. Possible array index out of bounds in PythScript.getMockPythPayload

---

In `PythScript.sol`, the `getMockPythPayload` function iterates over two arrays, `_ids`, and `_prices`, while reading from both. There is the possibility of an index out of bounds if both arrays are not the same length.

Recommendation

We recommend adding a check to ensure that both arrays are the same length, similar to line `85`.

Status

Acknowledged

11. Possible use of residual ETH to pay Pyth's oracle feed update fee

---

In `Utils.sol`, the `handlePythUpdate` function does not ensure that `msg.value` is equal to the Pyth oracle feed update fee returned by the `getUpdateFee` function. As a result, a user is able to use any residual native ETH tokens that have been left in the contract due to providing empty update data in the `SCDPFacet` contract to make up or pay the update fee to update a Pyth price feed.

Recommendation

We recommend adding a check to the `handlePythUpdate` function to ensure that `msg.value == updateFee` to ensure that the user sends the correct amount for the `updateFee`.

Status
Acknowledged

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Kresko Labs Pte. Ltd. or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Oak Security GmbH