MakerDAO: Spark-PSM

Security Review

Cantina Managed review by:
Christoph Michel, Lead Security Researcher
M4rio.eth, Security Researcher

September 9, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

The Maker Protocol, also known as the Multi-Collateral Dai (MCD) system, allows users to generate Dai (a decentralized, unbiased, collateral-backed cryptocurrency soft-pegged to the US Dollar) by leveraging collateral assets approved by the Maker Governance, which is the community organized and operated process of managing the various aspects of the Maker Protocol.

From Aug 21st to Aug 22nd the Cantina team conducted a review of spark-psm on commit hash fcde7093.

The Cantina team reviewed MakerDAO's spark-psm changes holistically on commit hash 6324e261 and determined that all issues were resolved and no new issues were identified.

The team identified a total of **7** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 1
- Low Risk: 2
- Gas Optimizations: 0
- Informational: 4

3 Findings

3.1 Medium Risk

3.1.1 Depositors might be unable to withdraw desired asset

Severity: Medium Risk

Context: Global scope

Description: The PSM3 allows deposits of three assets. It allows anyone to swap between these three assets. A depositor may not expect to always be immediately able to withdraw the asset they deposited (or an asset of their choice) as the contract's balance might have been depleted by a swap.

A malicious user can deliberately DoS withdrawals by sandwiching the withdrawal, swapping out the withdrawal asset, and later swapping it back in. As there are no fees only cheap L2 gas costs are to be paid. They could even leave `1 wei` in the contract balance to burn a rounded-up user's share balance each time for maximum impact.

Recommendation: Consider adding a `withdrawAnyAsset(address receiver, uint256 maxValueToWithdraw)` function that chooses the `asset` to withdraw based on the highest value asset in the contract's balance, converts `maxValueToWithdraw` to the asset's decimals and calls `withdraw(asset, receiver, maxAssetToWithdraw)`.

Maker: Acknowledged. This is an interesting edge case. This will be a contract executing the `deposit/withdrawal`, so there will be no issue as we can put the check-3-asset logic inside the spell or other contract if it becomes a problem. We don't expect third-party LPs, and there is private transaction submission in the extreme case.

Cantina Managed: Acknowledged. If the contract can check the current balances of the 3 assets before deciding what asset to withdraw, then it should be fine (for you). And for small depositors I honestly don't think anyone would attempt this.

3.2 Low Risk

3.2.1 `PSM3.withdraw` breaks non-decreasing share price invariant

Severity: Low Risk

Context: PSM3.sol#L165, LpHandler.sol#L83

Description: One of the main invariants is that LPs never lose value to a `swap`, `deposit`, or `withdraw` action, i.e., the share price `totalAssets() / totalSupply` does not decrease after any of these actions.

This invariant is currently broken by the `withdraw` function. We can measure the share price `totalAssets() * 1e18 / totalSupply` by `convertToAssetValue(1e18)`. Apply the following changes to the current `test/invariant/handlers/LpHandler.sol` to check this invariant:

```
  function withdraw(uint256 assetSeed, uint256 lpSeed, uint256 amount) public {
      // ...

      // 2. Cache starting state
      uint256 startingConversion = psm.convertToShares(1e18);
      uint256 startingValue = psm.totalAssets();
+     uint256 startingSharePrice = psm.convertToAssetValue(1e18);

      // ...

      // 5. Perform action-specific assertions
      assertApproxEqAbs(
          psm.convertToShares(1e18), startingConversion, 1e12, "LpHandler/withdraw/conversion-rate-change"
      );

+     assertGe(psm.convertToAssetValue(1e18), startingSharePrice, "LpHandler/withdraw/share-price-decrease");

      assertLe(psm.totalAssets(), startingValue + 1, "LpHandler/withdraw/psm-total-value-increase");

      // ...
  }
```

LPs can currently lose value when another LP withdraws.

Recommendation: Consider adding the same checks also to `LpHandler.deposit` and `SwapHandler.swap*` test suite. The culprit of the failing invariant is this line in `previewWithdraw`:

```
sharesToBurn = _convertToSharesRoundUp(_getAssetValue(asset, assetsWithdrawn));
```

Note how `_getAssetValue(asset, assetsWithdrawn)` rounds down and `_convertToSharesRoundUp` rounds up. We want the final `sharesToBurn` to round up. Consider adding an `_getAssetValueUp` function and use it instead of `getAssetValue_` in the computation:

```
// should be cleaned up before using
function _getAssetValueUp(address asset, uint256 amount) internal view returns (uint256) {
    if      (asset == address(asset0)) return _divUp(amount * 1e18, _asset0Precision);
    else if (asset == address(asset1)) return _divUp(amount * 1e18, _asset1Precision);
    else if (asset == address(asset2)) return _divUp(_divUp(amount *
↪   IRateProviderLike(rateProvider).getConversionRate(), 1e9), _asset2Precision);
    else revert("PSM3/invalid-asset");
}
```

The invariant will then pass but the test suite will fail in the `afterInvariant` → `_withdrawAllPositions` → `assertEq(psm.totalAssets(), 0)` check. This should be further investigated.

Maker: Fixed in commit 6324e261. The code was changed to round up the asset value computation for `asset2`. Checks in the constructor were added to ensure that `asset0` and `asset1` decimals are at most 18. Meaning, rounding up `asset0` and `asset1` asset values is the same as rounding them down, fixing the issue (asset value just stretches the precision to 1e18 for `asset0` and `asset1`).

The share price invariant was not added to the `SwapHandler`, only to the `LpHandler`.

Cantina Managed: Verified.

3.2.2 No amount received checks on `PSM3.deposit`/`PSM3.withdraw`

Severity: Low Risk

Context: PSM3.sol#L111, PSM3.sol#L128

Description: The `PSM3.deposit` function takes an `assetsToDeposit` parameter and in return mints `newShares` share tokens to the recipient. The amount of `newShares = amount / sharePrice` minted depends on the current share price (`totalAssets() / totalShares`). The `totalAssets()` again depend on the current `asset2` conversion factor if the contract has a non-zero `asset2` balance.

The current conversion factor on L2 can be temporarily out of sync with the real conversion factor and change abruptly, see issue "Out-of-sync DSRAuthOracle's pot data leads to conversion factor jumps" (from the XChain DSR Oracle Security Review). The depositor will receive fewer shares if the conversion factor (and therefore the share price) is temporarily higher.

Similarly, a withdrawer will overpay and burn more shares when calling `withdraw(maxAssetsToWithdraw)` to receive their `maxAssetsToWithdraw = sharesToBurn * sharePrice` assets when the conversion factor (and therefore the share price) is temporarily lower.

Example: The PSM3 has the following state: `1.5e6` USDC, `1.5e18` DAI, `2e18` sDAI and a conversion rate `chi = 1.6e27`. This gives a `totalAssets()` of `1.5e18 + 1.5e18 + 1.6 * 2e18 = 6.2e18`.

A depositor checks the conversion rate `pot.chi()` on ETH mainnet and sees it is `1.5e27` which would lead to a `totalAssets()` of `6e18`. They deposit `0.6e18` DAI and expect to receive 10% of the current total shares. As the conversion rate is temporarily out-of-sync they only receive `0.6e18 / 6.2e18 = 9.67%` of the current total shares. If they had waited until the conversion rate is in sync again, they would have received more shares for their deposit amount.

Recommendation: Consider adding a `minShares` parameter to `deposit` and check that `newShares >= minShares`. For `withdraw`, consider adding a `maxSharesToBurn` parameter that is to be understood relative to the `maxAssetsToWithdraw` parameter, then scale it down to be relative to the actual `assetsWithdrawn` variable (`_maxSharesToBurn = maxSharesToBurn * assetsWithdrawn / maxAssetsToWithdraw`) and check `sharesToBurn <= _maxSharesToBurn`.

Alternatively, ensure the conversion rate is always accurate and does not experience sudden jumps on L2.

Maker: Acknowledged. We accept this risk. Normal operation will be such that whenever the DSR changes, all other chains will receive the new pot values within ~20 minutes. Discrepancies can happen, but they will be small enough in practice that this is not a concern. Please let us know if you see any edge cases we are missing.

Cantina Managed: Acknowledged.

3.3 Informational

3.3.1 Missing `RateProvider` check in the deploy script

Severity: Informational

Context: PSM3Deploy.sol#L18

Description: In the `PSM3Deploy.sol` contract, when the PSM contract is deployed, a `RateProvider` is passed as a parameter. However, there is a missing check to verify whether the rate provider has been properly initialized.

Recommendation: Consider adding a check to see if `rateProvider.getConversionRate()` returns a value different than 0.

Maker: Fixed in commit 6324e261.

Cantina Managed: Verified.

3.3.2 Typographical issue in `IPSM3.sol`

Severity: Informational

Context: IPSM3.sol#L126

Description: There is a typographical issue in the `IPSM3.sol` file:

```
/**
 *  @return amountOut    Resulting mount of the asset that will be received in the swap.
```

Recommendation: Consider solving the typo by changing the `mount` to `amount`.

Maker: Resolved in commit 6324e261.

Cantina Managed: Verified.

3.3.3 `_divUp(0, 0)` returns 0 instead of reverting

Severity: Informational

Context: PSM3.sol#L351

Description: Calling `_divUp(x=0, y=0)` returns 0 whereas Solidity's in-built division would revert with a division-by-zero error. Note that `_divUp` is called with `y` set to an `assetPrecision` or the `rate`. The `rate` can be zero when coming from a `DSROracleBase` rate provider that hasn't received any pot data yet.

Recommendation: Consider reverting on division by zero. Furthermore, consider following the recommendation of "DSROracle's pot data starts uninitialized" (from the XChain DSR Oracle Security Review) to avoid all computations with a zero rate in the PSM3.

Maker: We want to switch to OZ math functions for this. Fixed in commit 6324e261.

Cantina Managed: Verified.

3.3.4 PSM3 treats `asset0` and `asset1` as 1-to-1 leading to potential LP losses in a depeg event

Severity: Informational

Context: PSM3.sol#L257-L263

Description: The `PSM3` contract values `asset0` and `asset1` (expected to be USDC and DAI) the same when swapping and for the LPs' `totalAssets()`. When there is a depeg event of one token, there is a natural arbitrage using external markets (the same as with other Maker PSMs). The PSM3 will end up filled with the less valuable token only. The LPs (here users that called `deposit` and received shares) will suffer this loss in case the assets don't repeg.

Recommendation: Depositors should be aware of this risk and withdraw liquidity in time to reduce risk if they don't expect the assets to repeg.

Maker: Acknowledged. This is expected behaviour. Same with the PSM on Mainnet. We rely on kill switches to halt withdraws during sustained depeg events.

For example, let's say March 2023 USDC depeg happens again, we no longer want users to be able to exit USDC into NST/sNST we would remove all NST/sNST liquidity as the LP and wait for peg to return (or eat the loss).

Judging temporary vs sustained depeg is very subjective and requires human analysis. The default is to assume all assets are pegged with manual intervention otherwise.

Cantina Managed: Acknowledged.