CANTINA

Maker Dao - Follow up

Security Review

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

November 24, 2023

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 Oct 19th to Oct 25th the Cantina team conducted a review of dss-lite-psm on commit hash 3ec57f35. The team identified a total of **7** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 3
- Gas Optimizations: 0
- Informational: 4

3 Findings

3.1 Low Risk

3.1.1 Transferring gem out of `pocket` breaks PSM

Severity: Low Risk

Context: DssPocket.sol#L97

Description: All gem ERC20 for the PSM is held in a separate `DssPocket` contract. Its wards can approve the gem for any spender. If a spender besides the `DssLitePsm` transfers out gem, the fees accounting in `DssLitePsm.cut()` will not work correctly anymore and fees will be lost. Furthermore, the `DssLitePsm` debt cannot be cleared naturally anymore as part of the DAI-swapped-to-Gem disappeared.

Recommendation: Only the `DssLitePsm` should be `hope`d to avoid breaking the accounting in `DssLitePsm`.

Maker DAO: Acknowledged.

Cantina Managed: Acknowledged.

3.1.2 `gush` ignores total `vat.Line()` debt ceiling

Severity: Low Risk

Context: DssLitePsm.sol#L472

Description: The `trim` function currently only considers its own ilk-specific debt ceiling `ilk.line`, ignoring global total debt ceiling `vat.Line()`. There are cases when it should trim *more* DAI than it currently does if it would respect the global total bad debt `vat.debt() - vat.Line()`.

One might expect that trimming more than `gush()` would always result in being able to perform a non-zero `fill()` but this is currently not the case because of this difference in how `fill` and `trim` handle `vat.Line()`.

Recommendation: Consider taking into account the `vat.debt() - vat.Line()` difference when `trim`ming, or explain why it's fine for the PSM not to help with bringing the global total system debt back in line:

```
wad = _min(
    _max(
        // To avoid two extra SLOADs it assumes urn.art == ilk.Art.
        _subcap(Art, tArt),
-       _subcap(Art, line / RAY)
+       _max(
+           _subcap(Art, line / RAY),
+           _subcap(vat.debt(), vat.Line())  / RAY
+       )
    ),
    // Cannot burn more than the current balance.
    dai.balanceOf(address(this))
);
```

Maker DAO: Acknowledged. The global debt ceiling (`Line`) is more of a lock of last resort to prevent inflating the Dai supply indefinitely in case there's ever a bug in the system.

Under normal conditions, `Line = sum_of_line + buffer`, where the buffer is arbitrary. It is not even an actual variable in the system, but we try to keep it around 600M Dai (I'm not sure about why). Usually, a change in an `ilk` debt ceiling is bundled with an equivalent change in the global one (i.e.: `AutoLine`).

Even in the unlikely scenario when `vat.debt() > vat.Line()`, unwinding the `DssLitePsm` even further wouldn't solve the issue, as all other collateral types in the system don't have the same behavior. There is no particular reason why this contract should respond differently than other vault-like structures.

Cantina Managed: Acknowledged.

3.1.3 Fees can be `trim`med

Severity: Low Risk

Context: DssLitePsm.sol#L478

Description: The accrued fees that haven't been sent to `vow` (`chug`ged) yet are tracked in `cut()` as `cash - debt` where `cash` is the sum of the swapped Gem balance and the DAI balance (minted DAI + accrued DAI fees). The `trim` function allows removing excess DAI (`gush`) when the balance exceeds the debt limits. This excess DAI can include the accrued fees which will then be temporarily unavailable to be `chug`ged.

Note that calling `trim()` (and `fill()`) never changes the `cash + gem.balanceOf(pocket) * to18ConversionFactor - art` part of `cut()` as the DAI `cash` and `art` are decreased (increased) in tandem by the same `wad` amount. Therefore, the fees are only temporarily lost but can be retrieved once the liquidity crunch has recovered (for example, if Gem is swapped back to DAI, or changes in debt ceilings allow calling `fill` again).

Example:

- `buf`: 1M DAI.
- `urn.Art`: 2M DAI.
- GEM balance: 1M GEM.
- DAI balance: 1.1M DAI = 1M (as 1M of 2M `Art` was swapped to GEM) + 0.1M fees.

Setting `ilk.line` to 0.5M, `gush()` will return `min(_subcap(Art, line / RAY), daiBalance) = min(1.5M, 1.1M) = 1.1M`. The entire DAI balance including the 0.1M fees will be trimmed.

Recommendation: Consider always performing a `chug()` if necessary before `trim()` such that adding to the surplus buffer takes priority over decreasing `art`.

Maker DAO: Acknowledged. This is yet another case we did not want to keep track of the fees to save gas on storage. The bookkeeping functions will be under automation through the keeper network. We'll make sure that the `chug()` job has higher priority than the `trim()` one.

Cantina Managed: Acknowledged.

3.2 Informational

3.2.1 Donating `gem` to `DssPocket` inflates `cut()`

Severity: Informational

Context: DssLitePsm.sol#L490-L495

Description: The collected protocol fee is returned by `DssLitePsm.cut()`, which calculates the fees dynamically. The dynamic fee calculation is used because there is no fee storage variable that gets updated during `buyGem()` and `sellGem()` executions. Below is the `cut()` function which shows returned value is linearly correlated to Psm's dai balance and Pocket's gem balance:

```
function cut() public view returns (uint256 wad) {
    (, uint256 art) = vat.urns(ilk, address(this));
    uint256 cash = dai.balanceOf(address(this));

    wad = _min(cash, cash + gem.balanceOf(pocket) * to18ConversionFactor - art);
}
```

This makes `cut()` manipulable by way of token donations.

Donating dai to Psm is not an issue, because fees are denominated and collected in dai. During a dai donation, `cut()` will increase equal to the donation amount and that can be moved to `vow` on the next `chug()`. This has no significant effect on the rest of the system.

Donating gem to Pocket can be an issue. Because it increases the `chug()`gable `cut()` without increasing dai. When `chug()` is called equivalent amount of dai to the donated gem is moved to `vow` as fee.

Recommendation: If Maker team desires that donated gem goes to `vow` as dai, then no change is required. However, if they believe the donated gem should not leave the system as dai as a fee, then tracking the fees in a storage variable that gets updated with each buy and sell appears to be the best solution.

Maker DAO: Acknowledged. We want to keep gas usage as low as possible, that's why there is no storage variable tracking the amount of gem ever deposited.

Cantina Managed: Acknowledged.

3.2.2 Depegging between the GEM and DAI

Severity: Informational

Context: DssLitePsm.sol

Description: The `DssLitePsm` assumes that the exchange rate between the `GEM` and `DAI` is always 1-1. As this is acceptable during normal market conditions, it can create a good opportunity for MEV during volatile conditions. If a depeg happens, users will start dumping gems into the protocol, gaining more DAI.

Recommendation: We do not have a recommendation on the current code. We recommend to use the Debt Ceiling Instant Access Module (AutoLine) to control the debt ceiling of a gem which will restrict the maximum growth of the debt ceiling on a certain timeframe. Furthermore, because the AutoLine contains the `exec` function which can be triggered by anyone, meaning it can increase the debt ceiling in a permissionelss way, if the depegging lasts for a long period, an emergency spell should be in place to zero de debt ceiling for that specific gem until the depegging starts to heal to an acceptable ratio.

Maker DAO Acknowledged. This module is meant to be used with `DCIAM`.

Cantina Managed: Acknowledged.

3.2.3 Small swaps can avoid paying fees

Severity: Informational

Context: DssLitePsm.sol#L330 | DssLitePsm.sol#L376

Description: During a swap, from GEM to DAI (`buyGem`) or DAI to GEM (`sellGem`), a fee is paid based on `tout`/`tin` parameters. Because the fee calculation is rounding down, one can split its swap into smaller swap to pay 0 fees and obtain a better price.

- Sell gem:

```
fee = daiOutWad * tin_ / WAD;
```

- Buy gem:

```
fee = daiInWad * tout_ / WAD;
```

We emphasize that this might be not economically viable for Ethereum Mainnet due to the transaction cost, this can become a viable path for L2s.

Recommendation: Consider rounding up when calculating the fees.

Maker DAO: Acknowledged. This contract is meant to be used only on Ethereum Mainnet, so we don't care about the L2 scenarios.

Cantina Managed: Acknowledged.

3.2.4 Asymmetric fee structure

Severity: Informational

Context: DssLitePsm.sol

Description: `buyGem()` adds fees on top of the input token while `sellGem()` removes the fees from the output token. This is an asymmetric design in which the same values of `tin` and `tout` represent slightly different fees.

For example, assume 10% wad is set for both `tin` and `tout`:

- `sellGem()`: Selling 100 gem gets **90 dai** with fee and 100 dai without fee.
- `buyGem()`: Selling 100 dai gets ~**90.9 gem** with fee and 100 gem without fee.

Recommendation: Due to the system working with the expectation that fees are always collected from dai, no change is recommended. However, governance should be aware of this asymmetric behaviour when setting `tin` and `tout` fees.

Maker DAO: Acknowledged. This is by design, and it's already the mechanism in place for the existing version. Collecting fees in Dai makes it easier for the protocol to incorporate them, as any other token would have to be converted to Dai before being incorporated into the surplus buffer.

Cantina Managed: Acknowledged.

4 Additional Comments

The Cantina team reviewed MakerDao's dss-lite-psm changes holistically on commit hash 3ec57f35fdd910ab765379c324a4dc2a7c08d54a and determined that all issues were acknowledged and no new issues were identified.