CANTINA

Maker DAO - dss-flappers

Security Review

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

July 3, 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 Jun 5th to Jun 11th the Cantina team conducted a review of dss-flappers on commit hash 4b0cb1f5.

The specific scope of the review included the `Splitter`, `SplitterMom`, `FlapperUniV2`, `FlapperUniV2SwapOnly`, `OracleWrapper` contracts and their corresponding deployment scripts.

The team identified a total of **5** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 0
- Gas Optimizations: 2
- Informational: 3

3 Findings

3.1 Gas Optimization

3.1.1 `OracleWrapper` can be replaced with an immutable variable within Flappers

Severity: Gas Optimization

Context: FlapperUniV2.sol#L148, FlapperUniV2SwapOnly.sol#L123

Description: `OracleWrapper` contract is used to scale down an oracle price to handle decimal differences. These wrappers are only used in Flapper contracts' `exec()` functions to calculate the minimum buy amounts. The below snippet is from `FlapperUniV2SwapOnly.exec()` function in which it is expected that `pip` is an `OracleWrapper` that returns the amount based on token's true decimal count:

```
require(_buy >= lot * want / (uint256(pip.read()) * RAY / spotter.par()),
↪  "FlapperUniV2SwapOnly/insufficient-buy-amount");
```

Calling an intermediate contract like this incurs minimum `2600` extra gas instead of calling the oracle directly.

Recommendation: An immutable divisor can be used within the Flapper contracts directly. Consider the below alternative snippet from `exec()` where `pip` is the primary oracle and not the wrapper, and the immutable `divisor` is directly incorporated into the Flapper.

```
require(_buy >= lot * want / ((uint256(pip.read()) / divisor) * RAY / spotter.par()),
↪  "FlapperUniV2SwapOnly/insufficient-buy-amount");
```

Maker: From the perspective of the flapper, the oracle divisor is an implementation detail of the oracle which does not directly concern the burn engine and is therefore best abstracted away, outside of the flapper contract. The gas savings does not seem to be large enough here to justify breaking this principle. In the future the re-denominated token may have its own oracle, at which point this leftover `divisor` in the code could be confusing to the reader.

Cantina Managed: Acknowledged.

3.1.2 Uniswap pair token order detection can be simplified

Severity: Gas Optimization

Context: FlapperUniV2.sol#L79

Description: The flapper UniV2 contracts check if `dai` is the first token by `daiFirst = pair.token0() == dai`.

Recommendation: As both tokens are known in the constructor (token pair verification is performed in the deployment scripts), one can avoid the external `token0()` call by ordering the pairs as Uniswap does.

```
- daiFirst = pair.token0() == dai;
+ daiFirst = _dai < _gem;
```

Maker: Doing `daiFirst = pair.token0() == dai` avoids requiring the additional context of the `UniswapV2Factory` (which is where the token ordering is implemented) so could be argued to make the code slightly more readable, as only `UniswapV2Pair` is then required to fully understand and validate the implementation.

Cantina Managed: Acknowledged.

3.2 Informational

3.2.1 Rewards duration should always be equal with the `hop`

Severity: Informational

Context: FlapperInit.sol#L171

Description: The rewards duration in the farm is set to the `hop` in the init script and the enforcement for the `cfg.hop` to be equal with the `flapper.hop()` is also performed on FlapperInit.sol#L166.

If the `hop` is modified in the future, there is currently no mechanism at the Splitter level to enforce consistency between the hop and the rewards duration. As a result, misconfigurations could occur where the `hop` and rewards duration have different values.

Recommendation: We recommend that both the `hop` and the rewards duration to be changed at the same time if the `hop` will ever be changed.

Maker: Acknowledged. We aim to only support the initial setup in these init libraries, as otherwise the scope is very broad. There are other mechanisms to ensure certain frequent gov actions are grouped together such as `dss-exec-lib`, which can be used here if needed.

Cantina Managed: Acknowledged.

3.2.2 Missing checks in the initialization scripts.

Severity: Informational

Context: FlapperInit.sol#L192, FlapperInit.sol#L125

Description: Various checks are missing from the `FlapperInit` script:

- The `want` is checked to not be too low but should also be checked to not be > WAD (FlapperInit.sol#L192).
- The `bump` is checked to be multiple of RAY but is not checked if it's 0 (FlapperInit.sol#L125).

Recommendation: Consider adding the missing checks in the initialization scripts.

Maker: Acknowledged:

- A `want` that is too low could be exploited and cause loss of funds for the DAO. A `want` that is too high could cause reverts but despite the temporary DoS (until the configuration gets fixed), it would not cause loss of funds. In general the sanity checks performed in init scripts aim to prevent security issues. Anything more than that is considered nice-to-have but not mandatory.
- We chose not to check that `bump` is greater than 0 as a `bump` of zero is semantically correct and does not pose a security risk.

Cantina Managed: Acknowledged.

3.2.3 Flapper's swap could be sandwiched

Severity: Informational

Context: FlapperUniV2.sol#L148

Description: The gem to dai swaps in `FlapperUniV2` and `FlapperUniV2SwapOnly` could be sandwiched:

1. Attacker can frontrun buying `gem`, lowering the `_buy` variable until it equals the RHS of the slippage check: `_buy == _sell * want / (uint256(pip.read()) * RAY / spotter.par())`.
2. Let flapper run `exec` to buy gem at the increased price.
3. attacker backruns selling the DAI bought in step 1.

The attacker's profit is the flapper's loss, therefore the flapper bought fewer gem tokens. The swap is easy to sandwich as anyone can call `vow.flap` once its requirements are met. The profitability of the attack depends on the liquidity of the pair, the trade size (`bump`), and the price deviation threshold `want`.

Recommendation: Ensure the chosen parameters make the attack unprofitable. Consider performing swaps through a private

Maker: Maker's risk unit is setting parameters while having sandwich attacks in mind. See their lastest update.

Cantina Managed: Acknowledged.