PUBLIC

Code Assessment

of the FlapperUniV2SwapOnly Smart Contracts

July 27, 2023

Produced for

by

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help MakerDAO with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of FlapperUniV2SwapOnly according to Scope to support you in forming an opinion on their security risks.

MakerDAO implements a new Flapper contract for the Maker Core contract `Vow` that is used to convert DAI surplus. In comparison to the old Flapper contract, the DAI are only swapped on a Uniswap v2 pair and the proceedings sent to a predefined `receiver` address instead of deposited into the pair as liquidity.

The most critical subjects covered in our audit are functional correctness and frontrunning. Functional correctness is high and frontrunning is only possible to a small extent determined by the `want` factor.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the FlapperUniV2SwapOnly repository based on the documentation files. The scope consists of the following solidity smart contracts:

1. ./src/FlapperUniV2SwapOnly.sol

The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	20 July 2023	70246121db072856fe1219089c89ad24f600dc99	FlapperUniV2SwapOnly code delivered

For the solidity smart contracts, the compiler version `0.8.16` was chosen.

2.1.1 Excluded from scope

Any other file not explicitly mentioned in the scope section. In particular tests, scripts, external dependencies, and configuration files are not part of the audit scope.

UniswapV2 is not in scope of this review.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

MakerDAO offers *FlapperUniV2SwapOnly*, a new version of the flapper contract. *FlapperUniV2SwapOnly* uses Uniswap V2 to periodically exchange the surplus DAI accumulated in the Vow for Gem tokens. The acquired Gem tokens are then sent to a receiver.

The contract *FlapperUniV2SwapOnly* differs from the previously deployed *FlapperUniV2* because it does not use the acquired Gem to supply liquidity to the Uniswap V2 pool, and instead only takes Gem from the existing Uniswap pool.

2.2.1 FlapperUniV2SwapOnly

The contract is used by the Vow to exchange surplus DAI to Gem tokens, expected to be MKR, in order to redistribute the surplus DAI to MKR holders through MKR buy-back.

Initialization of the contract consists in contract creation, during which immutable variables are set, and calling of `file()` which allows a privileged account to set the following state variables of the contract:

- `pip`: The price oracle used as a reference to protect the Flapper against excessive slippage.
- `hop`: The cooldown period between successive calls of `kick()`.

- `want`: The relative anti-slippage parameter, the output amount value, estimated with the oracle, has to be at least `want` times the input value.

Contract deployment also sets the `live` variable to `1`. Call of `cage()` by a `ward` irreversibly sets `live` to `0`.

All the functionality exposed by the contract is guarded by the `auth` modifier, only allowing `wards` to call, however unprivileged users are expected to be able to trigger calls of `kick()` through `Vow.flap()`. The Vow allows users to call `flap()` when a surplus (and no debt) is present.

The main function `kick()` exposes the following functionality:

Its argument `lot` is a RAD amount of DAI that is intended to be exchanged on UniswapV2 for the Gem. The output amount of Gem corresponding to a swap of size `lot` is computed from the current reserves of the Uniswap pool. The output amount from Uniswap is compared with the ideal output amount for `lot`, as indicated by the price reported by the oracle `pip`. If the output is lower than the `want` threshold, execution reverses to prevent an excessive amount of slippage. When the Flapper is set in the Vow, it is given rights to move DAI in the Vat on behalf of the Vow through `hope()`. The Flapper therefore moves `lot` from the Vow to itself, and exits the `lot` amount of ERC20 DAI token to the Uniswap pool through DaiJoin. Finally `swap()` is called on the Uniswap pool to obtain Gem tokens. The receiver of `swap()` is set to the global `receiver` variable, which is expected to be a PauseProxy controlled by the Governance.

Besides the main `kick()` function, the contract exposes the following functionality:

`rely()` and `deny()`, to add or remove accounts from the `wards` mapping.

`cage()`, to set the `live` variable to 0 and disable the contract permanently.

`file()`, to set the `pip` oracle address, the `hop` cooldown period length and the `want` anti-slippage parameter.

2.2.2 Roles & Trust Model

Wards are trusted, maliciously operating wards can empty the Vow DAI surplus. It is assumed that the Governance chooses sane parameters for `want`, `hop` and `pip`.

in particular the wards are `Vow`, that will call `kick()` and `cage()` in the event of system shutdown, and a Governance proxy that will select flapper parameters. The Governance proxy is trusted not to call `kick()` directly.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe our findings. The findings are split into these different categories:

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

6.1 Call to Vow.flap() Can Be Sandwiched

Note Version 1

The parameters for the FlapperUniV2 deployed at time of the audit are a `lot` of `5000` DAI, a cooldown period of 1577 seconds between calls to `Vow.flap()`, and 98% slippage tolerance through `want`.

An MEV searcher can therefore sandwich the `Vow.flap()` call and extract up to 2% of the `5000` DAI every 26 minutes. The gas cost of calling `flap()` is around 20$ at the current gas price of 30 Gwei. Assuming the sandwich attack gas cost is within the same order of magnitude (one frontrunning transaction and one backrunning transaction, on warm token addresses, and warm Uniswap pool), we expect a MEV searcher to extract a profit of around $50 per call. This amounts to a possible loss for the protocol of around $2800 daily.