PUBLIC

Code Assessment

of the SparkLend Advanced Smart Contracts

January 03, 2024

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 SparkLend Advanced according to Scope to support you in forming an opinion on their security risks.

MakerDAO implements two new price oracles that can be used within the SparkLend protocol, one has a fixed price and the other has a capped price. In the extended scope, two interest rate strategies have been added. One sets the base rate using a rate source, while the other targets a specific rate at optimal utilization.

The most critical subjects covered in our audit are functional correctness and precision of arithmetic operations. Security regarding all the aforementioned subjects is high. After the intermediate report all issues have been resolved.

The general subjects covered are specification, gas efficiency, and trustworthiness. Security regarding all the aforementioned subjects is high.

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	1
• Code Corrected	1
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 SparkLend Advanced repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	11 Dec 2023	2980932a2bed29387eaa9f43cda06c458ffd5e50	Initial Version
2	19 Dec 2023	2835e3f09efe4f2dcf880e23427b5318fbc5a2d0	Extended Scope
3	03 Jan 2024	277ea9d9ad7faf330b88198c9c6de979a2fad561	Fixes

For the solidity smart contracts, the compiler version `0.8.20` was chosen.

The following contracts are in the scope of this review:

```
CappedOracle.sol
FixedPriceOracle.sol
```

Starting with Version 2, the scope has been extended to include the following contracts:

```
interfaces/IRateSource.sol
PotRateSource.sol
RateTargetBaseInterestRateStrategy.sol
RateTargetKinkInterestRateStrategy.sol
VariableBorrowInterestRateStrategy.sol
```

2.1.1 Excluded from scope

Any file not explicitly listed above as well as third-party libraries are out of the 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.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

SparkLend Advanced offers a suite of features enhancing security and streamlining the automation of governance processes.

2.2.1 Oracles

MakerDAO offers two new price oracles that can be used within the SparkLend protocol. The oracles have the following interface:

```
interface IOracle {
    function latestAnswer() external view returns (int256);
    function decimals() external pure returns(uint8);
}
```

The price returned is an integer.

Capped Oracle

This oracle returns the minimum price with 8 decimals between a `maxPrice` set during deployment and the price returned by the `source`, which is assumed to have 8 decimals. The price can be queried with `latestAnswer()`. Note that the consumer of this price feed should be able to handle negative prices.

Fixed Price Oracle

This oracle returns a constant price set during deployment with 8 decimals. The price can be queried with `latestAnswer()`.

2.2.2 Rate Sources

PotRateSource

This adapter converts the DSR (DAI Savings Rate, per-second compound interest factor in RAY) querried from the POT into APR (Annual Percentage Rate in RAY).

2.2.3 Custom Interest Rate Strategies

VariableBorrowInterestRateStrategy

Modified version of Aave's `DefaultReserveInterestRateStrategy` with the stable borrow logic removed.

The assumption made is that the stable borrow functionality remains inactive and therefore, its associated functions are not expected to be used. The majority of features connected to the stable borrow functionality have been effectively deactivated by always returning a value of zero. An exception exists for `getBaseStableBorrowRate()`. In order to align closely with the default version, this function returns the value corresponding to variable slope 1.

`calculateInterestRates()` implements the calculation of the interest rates depending on the passed reserve state. The function returns the `liquidityRate` (the supplyRate), the `stableBorrowRate` (present due to the requirements of the interface, always zero since this functionality has been removed) and the `variableBorrowRate`.

The `currentVariableBorrowRate` is determined based on the borrow usage ratio and the 2-slope interest rate model which adjusts the rate depending on whether the borrow usage ratio exceeds the optimal usage ratio.

The `currentLiquidityRate` in this function depends on the current variable borrow rate, the supply usage ratio, and the reserve factor of the protocol.

RateTargetBaseInterestRateStrategy

Derives from the `VariableBorrowInterestRateStrategy` and modifies `_getBaseVariableBorrowRate()` to align with an external rate source (`Rate_source.getAPR()`), with the addition of a constant spread. This strategy is expected to be used for the DAI market, with the external rate source being the annualized DSR.

RateTargetKinkInterestRateStrategy

Derives from the `VariableBorrowInterestRateStrategy` and modifies `_getVariableRateSlope1()` to set the variable slope 1 rate to match an external rate source (`Rate_source.getAPR()`) and a constant spread. This strategy is expected to be used for ETH, it is expected to track the staked ETH yield minus some spread.

2.2.4 Trust Model

There is no privileged role in these contracts. The origin of the information (pricing, rates), namely the base oracle or the `Pot` are completely trusted.

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 any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Correctness: Mismatches between specification and implementation

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 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

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

Critical-Severity Findings	0
High-Severity Findings	1

- Decimals Mismatch in CappedOracle Code Corrected

Medium-Severity Findings	0
Low-Severity Findings	0

6.1 Decimals Mismatch in `CappedOracle`

Correctness High Version 1 Code Corrected

CS-SPRKADV-002

The function `CappedOracle.decimals` returns `8` regardless of the decimals of the `source`, that is expected to be a Chainlink price oracle. This creates a decimals mismatch when the source is an `ETH` pair. For example, the Chainlink price oracle for `stETH/ETH` has 18 decimals (https://etherscan.io/address/0x86392dc19c0b719886221c78ab11eb8cf5c52812#readContract#F3). If the price of an `ETH` pair is computed with the decimals of the `CappedOracle`, the price will be either heavily over-evaluated if `maxPrice` has `18` decimals, or always capped to `maxPrice` if it has `8` decimals.

Code corrected:

The oracle is designed to work with source oracles that have 8 decimals. An additional check has been added to the constructor to enforce this 8-decimal precision in the source oracle, preventing possible misconfiguration and resulting decimal mismatch.

6.2 Gas Optimizations

Informational Version 1 Code Corrected

CS-SPRKADV-001

1. The `vars.totalDebt = params.totalStableDebt + params.totalVariableDebt;` computation can ignore the total stable debt, as it should always be 0.

Code corrected:

`totalDebt` has been removed and instead `params.totalVariableDebt` is now used directly.