PUBLIC

Code Assessment

of the SparkLend Freezer Smart Contracts

January 03, 2024

Produced for

MAKER

by

CHAINSECURITY

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

MakerDAO implements an emergency spells system for SparkLend.

The most critical subjects covered in our audit are functional correctness, integration in the underlying system, and access control. Security regarding all the aforementioned subjects is high.

The general subjects covered are specification and gas efficiency. 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	0
Medium-Severity Findings	0
Low-Severity Findings	1
• Acknowledged	1

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 Freezer 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	a1ca29c5d1e8c816398b5095a4da7d50e3d5db5d	Initial Version

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

The following contracts are in the scope of this review:

```
SparkLendFreezerMom.sol
interfaces:
    IExecuteOnceSpell.sol
    ISparkLendFreezerMom.sol
spells:
    EmergencySpell_SparkLend_FreezeAllAssets.sol
    EmergencySpell_SparkLend_FreezeSingleAsset.sol
    EmergencySpell_SparkLend_PauseAllAssets.sol
    EmergencySpell_SparkLend_PauseSingleAsset.sol
```

2.1.1 *Excluded from scope*

The Sparklend protocol, third-party libraries with which the contracts in scope can interact with, and all files not explicitly listed above 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.

MakerDAO offers SparkLend Freezer, a module to execute emergency governance spells for SparkLend, where the spells are executed through the `SparkLendFreezerMom` contract.

2.2.1 *SparkLendFreezerMom*

This contract is assumed to be a risk and emergency admin of the pool. The contract exposes the following function to execute actions on the pool:

- `freezeAllMarkets`: calls `setReserveFreeze` on all the reserve assets of the pool, the frozen/unfrozen status is given by a boolean flag passed as parameter.

- `freezeMarket`: calls `setReserveFreeze` on a specific reserve assets passed as parameter, the paused/unpaused status is given by a boolean flag passed as parameter.
- `pauseAllMarkets`: calls `setReservePause` on all the reserve assets of the pool, the paused/unpaused status is given by a boolean flag passed as parameter.
- `pauseMarket`: calls `setReservePause` on a specific reserve passed as parameter, the paused/unpaused status is given by a boolean flag passed as parameter.

Freezing a reserve blocks the supply, borrow, and rate mode swap. Pausing a reserve blocks all the interaction with the asset in the pool.

The caller of these functions must be either the `owner`, a `ward`, or an address whitelisted by the `authority`. In practice these functions will be called either by wards or the Emergency Spells.

The `owner` can transfer and renounce the ownership of the contract, set/unset an address in the `wards` mapping, and set the `authority` address.

2.2.2 Spells

All the spells can only be used once, if the spell needs to be reused, a new contract must be deployed. The currently available spells are:

- freeze all assets: sets the spell as `executed` and calls `SparkLendFreezerMom.freezeAllMarkets` with the `freeze` flag set to `true`.
- freeze a single asset: sets the spell as `executed` and calls `SparkLendFreezerMom.freezeMarket` with the `freeze` flag set to `true` and the target `reserve` set during deployment.
- pause all assets: sets the spell as `executed` and calls `SparkLendFreezerMom.pauseAllMarkets` with the `pause` flag set to `true`.
- pause a single asset: sets the spell as `executed` and calls `SparkLendFreezerMom.pauseMarket` with the `pause` flag set to `true` and the target `reserve` set during deployment.

These spells must be deployed and given the required privileges in `SparkLendFreezerMom`. Afterwards any account can trigger `execute()` which will carry out its execution.

2.2.3 Trust Model

- Wards: Accounts holding the ward role have access to all privileged functionalities of this contract. These parties are assumed to act honestly and correctly at all times.
- Owner: The account holding the owner role has access to all privileged functionalities of this contract. This party is assumed to act honestly and correctly at all times.
- Authority: The smart contract holding the `authority` role is expected to work correctly at all times by authorizing only the legitimate spells execution

and blocking all the other illegitimate spell execution requests.

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:

- Design: Architectural shortcomings and design inefficiencies

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	1

- Pool and Configurator May Not Match Acknowledged

5.1 Pool and Configurator May Not Match

Design Low Version 1 Acknowledged

CS-SPRKFRZR-001

Nothing enforces the pool address to be the one set in the configurator. If the `pool` address and the `poolConfigurator` pool were to differ, the `SparkLendFreezerMom` contract may not work as expected.

Acknowledged:

Client states:

```
Acknowledged, no change. Will ensure configuration is correct with adequate end
to end testing.
```

6 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

6.1 Events Emission Inconsistency

Informational **Version 1** **Risk Accepted**

CS-SPRKFRZR-002

In the functions `rely` and `deny`, the event is emitted after the storage update, but in the functions `setAuthority` and `setOwner` the event is emitted before. A consistent codebase is easier to maintain and understand.

Risk accepted:

Client states:

```
Acknowledged, no change. This pattern was done on purpose to efficiently emit
the previous and new values of the setters for setAuthority and setOwner before
the value is updated in storage.
```

6.2 Gas Optimizations

Informational **Version 1** **Acknowledged**

CS-SPRKFRZR-003

1. In the functions `SparkLendFreezerMom.freezeAllMarkets` and `SparkLendFreezerMom.pauseAllMarkets`, the length of `reserves` can be cached before the loop to save gas.
2. The incrementation of the index in the function `SparkLendFreezerMom.freezeAllMarkets` and `SparkLendFreezerMom.pauseAllMarkets` can be unchecked.

Acknowledged:

Client chose to not implement gas optimizations.

6.3 Unused Code

Informational **Version 1** **Acknowledged**

CS-SPRKFRZR-004

1. The function `SparkLendFreezerMom.isAuthorized` has a branch `src == address(this)` that cannot be reached.

Acknowledged:

Client acknowledged the code is unused, `isAuthorized()` is a common function present in multiple contracts and intentionally not modified.