PUBLIC

Code Assessment

of the Spark User Actions Smart Contracts

June 5, 2024

Produced for

by

Contents

1 Executive Summary

Dear all,

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

SparkDAO implements a contract that batches actions of the PSM and the savings token.

The most critical subjects covered in our audit are functional correctness and precision of arithmetic operations. Security regarding all the aforementioned subjects is high.

The general subjects covered are unit testing and documentation. Both are good.

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 Spark User Actions 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	29 May 2024	48e6628514af0a126b66b5a5ba4ceab982f51e98	Initial Version
2	05 June 2024	4fa86e5cb73bb8486e85d9df77a332353e3ee7a9	NatSpec fixes

For the solidity smart contracts, the compiler version `0.8.25` was chosen.

The following file is in scope:

```
src/PSMVariant1Actions.sol
```

2.1.1 Excluded from scope

All other files such as tests and scripts are not in scope. The interacted with contracts are expected to work correctly and their correctness is out of scope.

2.2 System Overview

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

SparkDAO offers a peripheral contract that acts as a wrapper for batching PSM (PSM implementation) swaps and ERC-4626 deposits/withdrawals (e.g. sDAI).

The `PSMVariant1Actions` contract implements the following function:

1. `swapAndDeposit`: Pulls the PSM's gem token from the caller and sells it to the PSM for its `dai` token which it it then deposits to the ERC-4626 savings token.
2. `withdrawAndSwap`: Withdraws an amount of the `dai` from the caller's savings so that it can be swapped for an exact amount of the gem.
3. `redeemAndSwap`: Redeems an exact amount of the savings token and swaps the received underlying token to the gem. Note that this operation might leave some dust in the contract.

All operations implement slippage protection in case the PSM's `tout` variable and, thus, the fees change. The slippage protection for `swapAndDeposit` does not consider fees on deposit. Note that these are not expected (e.g. sDAI has no fees).

2.2.1 Roles and Trust Model

All the interacted with addresses `psm`, `dai`, `gem` and `savingsToken` are expected to be the correct addresses and are trusted and expected to work correctly. Users are untrusted.

There are no privileged roles defined for the contract.

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:

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	0
Medium-Severity Findings	0
Low-Severity Findings	0
Informational Findings	1

- Partially Inaccurate Function Description Code Corrected

6.1 Partially Inaccurate Function Description

Informational Version 1 Code Corrected

CS-SPRKUA-001

Function `withdrawAndSwap` can be used to get an exact amount of `gem` tokens out. While the function description clearly states this, the description starts with:

```
Withdraw a specified amount of `dai` from the `savingsToken` and swap for `gem` in the PSM.
```

This is not accurate. The `dai` amount is not specified but the output amount of `gem` is.

Code corrected:

The comment has been adjusted.