PUBLIC

Code Assessment

of the Spark ALM Controller Smart Contracts

October 22, 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 ALM Controller according to Scope to support you in forming an opinion on their security risks.

SparkDAO implements the Spark ALM Controller, a suite of contracts of the Spark Liquidity Layer designed to manage and control the flow of liquidity originating from DSS Allocator.

The most critical subjects covered in our audit are functional correctness, access control, and the integration with CCTP. The general subjects covered are gas efficiency, documentation and composability. 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	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 ALM Controller 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	15 Sep 2024	7a0535ee07815a2c8604d58469393c56a62d5b81	Initial Version
2	26 Sep 2024	2087250fc5988d3f0117f23e3f959f9423d04909	Second Version
3	03 Oct 2024	c76b422b053dd055aeb2cd555acfe353f05b316e	Fixes And Deployment Scripts
4	07 0ct 2024	342fe537b020ffa8ea7fcedf166b59b7ede21232	Fix Deployment Scripts
5	08 0ct 2024	52deda866ec8abdeaae9ace8574457d3e4209c36	Setting USDS And sUSDS Rate Limits
6	22 Oct 2024	6058f68f79520eb06ea8eded146da13039c47525	Bump Version

For the solidity smart contracts, the compiler version `0.8.21` was chosen.

The files in scope were:

```
src/
    ALMProxy.sol
    ForeignController.sol
    MainnetController.sol
    RateLimitHelpers.sol
    RateLimits.sol
    interfaces/
        IALMProxy.sol
        IRateLimits.sol
        CCTPInterfaces.sol
```

In Version 3 the following files were further added to scope:

```
deploy/
    ControllerDeploy.sol
    ControllerInit.sol
    ControllerInstance.sol
```

2.1.1 Excluded from scope

All other files are out of scope. It is assumed that USDC and CCTP will work honestly as documented.

In addition, the inherent centralization risks of USDC are out of the scope of this review:

- USDC is deployed behind a proxy, and its implementation can be upgraded by an admin.
- CCTP relies on a set of centralized offchain signers to provide the bridging attestation.

Note that the deployment script is in scope. However, governance should validate the deployment.

2.2 System Overview

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

At the end of this report section, we have added subsections for each of the changes according to the versions.

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 Spark ALM Controller, a set of on-chain components of the Spark Liquidity Layer designed to manage and control the flow of liquidity between Ethereum mainnet and L2s by leveraging DSS Allocator.

On each chain, the following contracts are deployed:

1. ALMProxy: Entity that holds funds and interacts with external contracts (e.g. DssAllocator, PSM). Thus, it holds the required privileges to interact with other contracts.
2. Controllers: Dictate which operations an ALMProxy shall perform. Note that multiple controllers could point to the same ALMProxy.
3. RateLimits: Computes limits for liquidity flows.

All of the contracts inherit the standard AccessControl and grant the `DEFAULT_ADMIN_ROLE` role to an `admin` which can configure other roles.

2.2.1 ALMProxy

The ALMProxy provides the following privileged functions that are restricted to addresses with the `CONTROLLER` role:

1. `doCall()`: triggers a call from ALMProxy to a target contract with `msg.value`.
2. `doCallWithValue()`: triggers a call from ALMProxy to a target contract with a specific `value`.
3. `doDelegateCall()`: triggers a delegatecall from ALMProxy to a target contract.

2.2.2 Controllers

MainnetController defines operations in the context of the mainnet ALMProxy that are restricted to addresses with the `RELAYER` role:

1. `mintUSDS()` / `burnUSDS()`: leverages AllocatorVault to mint or burn (`draw()` or `wipe()`) USDS.
2. `depositToSUSDS()` / `withdrawFromSUSDS()`: wraps USDS to SUSDS or vice-versa.
3. `redeemFromSUSDS()`: this further provides the option to unwrap SUSDS by specifying the shares to burn.

4. `swapUSDSToUSDC()` / `swapUSDCToUSDS()`: leverages the PSM to swap between USDS and USDC without fees (`buyGemNoFee` and `sellGemNoFee`).
5. `transferUSDCToCCTP()`: leverages CCTP (Circle's Cross-Chain Transfer Protocol) to bridge USDC to a recipient (expected to be another ALMProxy) on a foreign domain.

ForeignController defines operations in the context of a foreign domain ALMProxy that are restricted to addresses with the `RELAYER` role:

1. `depositPSM()` / `withdrawPSM()`: deposits / withdraws specific assets to / from Spark PSM.
2. `transferUSDCToCCTP()`: leverages CCTP (Circle's Cross-Chain Transfer Protocol) to bridge USDC to a recipient (expected to be another ALMProxy) on a foreign domain.

Both controller types provide the following administrative functions:

1. `freeze()`: restricted to the `FREEZER` role to pause all the operations defined above.
2. `reactivate()`: restricted to the `DEFAULT_ADMIN_ROLE` to reactivate the operations.
3. `setMintRecipient()`: set the token recipient of a destination domain when bridging USDC with CCTP.

In addition, except interaction with SUSDS, all other operations are subject to specific limits defined by Rate Limits.

2.2.3 Rate Limits

RateLimits defines a limit on a rate for a given key. The rate limit will linearly grow from `lastAmount` with `slope` over the time elapsed (tracked with `lastUpdated`), and is capped `maxAmount`. The full rate limit data or the current rate limit can be queried from `getRateLimitData()` and `getCurrentRateLimit()`, respectively.

The data can be set by the `DEFAULT_ADMIN_ROLE` with `setRateLimitData()` (two signatures available) or `setUnlimitedRateLimitData()`.

The following functions are introduced to update the rate limit by the `CONTROLLER` role:

1. `triggerRateLimitDecrease()`: It deducts an amount from the current rate limit and sets `lastUpdated` to `block.timestamp`. In case of the rate limit is insufficient, the call will revert.
2. `triggerRateLimitIncrease()`: It adds an amount to the current rate limit (capped by the `maxAmount`) and sets `lastUpdated` to `block.timestamp`.

2.2.4 Deployment Scripts

In version 3 (see Changes in Version 3), deployment scripts have been added that can be used in governance spells.

MainnetControllerDeploy and ForeignControllerDeploy libraries both offer the functions `deployController` and `deployFull` to deploy on mainnet and on the foreign chain, respectively. While `deployController` solely deploys a controller with a given ALM proxy and rate limit contract, the `deployFull` function deploys the ALM proxy and rate limit contract as well.

The MainnetControllerInit library implements functionality to initialize the ALM controller architecture on mainnet. Three functions are offered:

1. `subDaoInitController`: For SubDAO. Sanity checks along with assigning the expected roles and setting rate limits for the expected keys. Additionally, allows to revoke the roles of a previously used controller contract.
2. `subDaoInitFull`: For SubDAO. `subDaoInitController` with additional sanity checks on the ALM proxy and the rate limit contract along with setting the required access control on the allocator infrastructure.

3. `pauseProxyInit`: For Sky Governance. Calls `kiss` on the PSM to allow fee-less swaps on the PSM. Note that no checks are performed. Thus, this should be performed only after the expected spell to setup the ALM controller have been performed by the SubDAO.

Note that the current version of the mainnet controller will be only able to use CCTP to bridge to Base.

The ForeignControllerInit library implements functionality to initialize the ALM controller architecture on a foreign chain (at the time of writing: Base). Only `init` is offered. Essentially it performs similar actions to `subDaoInitFull` in the MainnetControllerInit. However, customized to the ForeignController contract. Note that only USDC will be usable in the integration with the Spark PSM on L2.

Note that L1 addresses are not published to the Chainlog (which is intended).

2.2.5 Changes in Version 2

The changes below were introduced in version 2 of the codebase:

1. The `transferUSDCToCCTP` function now has an additional limit on the amount of USDC that can be bridged to a given domain.
2. The MainnetController's `swapUSDCToUSDS` function now uses the PSM's `fill` function to allow filling the PSM if needed. As a consequence, swaps will be repeated as long as the full USDC amount has not been swapped (with filling happening before the swap). In case the USDC amount cannot be swapped, a revert will occur as before.
3. The unused immutables `usds` and `susds` have been removed from the `ForeignController`, hence these two addresses cannot be queried from the controller anymore.

2.2.6 Changes in Version 3

The following changes were introduced in version 3 of the codebase:

1. Function `receive()` has been added to the `ALMProxy` to support receiving ETH.
2. Events will be emitted when changing the controller's `active` status and setting the mint recipient on a destination domain.
3. Deployment scripts have been included in scope, see Deployment Scripts.

2.2.7 Changes in Version 4

Now, on the foreign controller's initialization, limits for USDS and sUSDS are set up for PSM deposits and withdrawals.

2.2.8 Roles & Trust Model

ALMProxy: The admin is fully trusted, otherwise, it can setup controllers and trigger any calls with the privilege of ALMProxy. The `CONTROLLER` is also trusted.

In addition, the ALMProxy requires several roles to operate, which are assumed to be setup properly by governance, for instance:

1. It requires `bud` role to swap without fee on DssPSMLite.
2. It requires `wards` role on AllocatorVault to `draw()` and `wipe()` USDS.
3. It needs sufficient allowance from AllocatorBuffer to move minted USDS.

MainnetController and ForeignController: The admin is fully trusted, otherwise they can DoS the Controller, or steal the bridged money on the destination domain by changing the mint recipient. The `RELAYER` is semi-trusted, and they can only change the liquidity allocation in the worst case. The `FREEZER` is also semi-trusted which can temporarily DoS the Controller in the worst case.

As of version 3, deployments scripts are in scope. Before initializing the contracts, governance should always carefully examine whether the deployed contracts match the expectations.

RateLimits: The admin is fully trusted to configure the limit data and CONTROLLER correctly.

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	7

- Mint Recipient Is Not Initialized Code Corrected
- Sanity Checks In Deployment Scripts Code Corrected
- Unused Parameters Code Corrected
- ALM Proxy Cannot Receive Code Corrected
- Constructor Parameters Code Corrected
- Events Code Corrected
- Gas Optimizations Code Corrected

6.1 Mint Recipient Is Not Initialized

Informational Version 3 Code Corrected

CS-SPRKALM-007

In the controller initialization library, the `RateLimit` of bridging tokens with CCTP has been configured, however, the mint recipients are not. As a result, USDC cannot be bridged after initialization and another spell is required to set the mint recipients.

Code corrected:

Mint recipients are now configured in the initialization library.

6.2 Sanity Checks In Deployment Scripts

Informational Version 3 Code Corrected

CS-SPRKALM-005

1. The initialization code doesn't check if the Foreign Controller is active.
2. The status of the Spark PSM is not validated in the Foreign Controller intialization library. The Spark ALM would be subjected to Share Inflation Attack if the deployer of Spark PSM does not make the proper first deposit.

3. In the initialization library for both the Mainnet and Foreign controllers, there is no validation to ensure that the new controller address (`controllerInst.controller`) is different from the old controller address (`params.oldController`). If both addresses are the same, the script will first grant the necessary permissions to the controller and then immediately revoke them. As a result, the controller address will not obtain the `CONTROLLER` role.

Code corrected:

Code has been corrected to perform the respective checks.

6.3 Unused Parameters

Informational **Version 3** **Code Corrected**

CS-SPRKALM-006

The intialization function `init()` of the Foreign Controller takes as parameters `params.usds` and `params.susds`. However, these parameters are not used in the function.

Code corrected:

Code has been corrected by removing `usds` and `susds` from the `AddressParams` struct.

6.4 ALM Proxy Cannot Receive

Informational **Version 1** **Code Corrected**

CS-SPRKALM-001

The ALM proxy contract is intended to be used for use-cases beyond the implementations of the current controllers. In the future, scenarios might exist where a controller requires that the proxy can receive ETH (e.g. by withdrawing from WETH). However, such use cases are not possible to implement due to the lack of a `receive` function.

Code corrected:

Function `receive()` has been added to support receiving ETH.

6.5 Constructor Parameters

Informational **Version 1** **Code Corrected**

CS-SPRKALM-002

The constructor of the mainnet controller receives `buffer` as an input. However, the `buffer` could be retrieved from the `vault`. Ultimately, retrieving the buffer on-chain could make the code more consistent (e.g. `dai` is retrieved from `daiUsds`).

Code corrected:

Code has been corrected to retrieve `buffer` from the `vault`.

6.6 Events

Informational | Version 1 | Code Corrected

CS-SPRKALM-003

The controller contracts lack events on important state changes. More specifically no event is emitted on

1. `setMintRecipient()`
2. `freeze()`
3. `reactivate()`

which involve important state changes.

For other functions, such as `MainnetController::mintUSDS` no event is emitted. Note that the relevant events can be retrieved from the external contracts. However, that is also true for CCTP which emits `DepositForBurn` making `CCTPTransferInitiated` redundant. Nevertheless, emitting an event on every action could also be reasonable to easily allow distinguishing which controller (of the potentially many) initiated a certain sequence of calls.

Code corrected:

The following events have been added to the privileged functions in both mainnet and foreign controllers:

1. event Frozen.
2. event MintRecipientSet.
3. event Reactivated.

6.7 Gas Optimizations

Informational | Version 1 | Code Corrected

CS-SPRKALM-004

In the MainnetController's `swapUSDSToUSDC` and `swapUSDCToUSDS` functions, `to18ConversionFactor` is always queried. However, the factor is expected to be a constant and could be made an `immutable`.

Code corrected:

The conversion factor has been set as an immutable in the constructor.

7 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.

7.1 Special Cases Handling

Note Version 1

The ALM's functionality can be extended by allowing new controllers. Some currently unresolvable scenarios, could be resolved in the future if needed. For example:

1. Assume it is desired that for an L2, all funds are bridged back to mainnet. However, in case the PSM3 never holds sufficient USDC to bridge back to L1, funds will remain on L2. As a result, another controller could be whitelisted that initiates redeeming the PSM shares against the other two assets to then bridge them back to mainnet through the respective bridges.
2. The mint recipient for CCTP could be blacklisted. That effectively could DoS the USDC bridging. In that case, a new controller could be added that allows calling CCTP's `replaceDepositForBurn` to resolve the issue.

Ultimately, some unlikely (and intentionally unhandled) issues may arise with the existing controllers. To resolve such issues, new controllers can be added.