



# Security Assessment Final Report



## Spark ALM v1.8.0

October 2025

Prepared for Spark

## Table of Contents

<b>Project Summary</b>	<b>3</b>
Project Scope	3
Project Overview	3
Protocol Overview	3
Findings Summary	4
Severity Matrix	4
<b>Detailed Findings</b>	<b>5</b>
Medium Severity Issues	6
M-01 Compromised relay can drain proxy by incurring losses upon ERC4626 withdrawal	6
M-02 Aave and ERC4626 are vulnerable to reentrancy attacks	7
Low Severity Issues	8
L-01 Overloaded maxSlippages allows bypassing of access control	8
L-02 Configuration validation gap can lead to permanently stuck funds	9
L-03 Relay can exceed ratelimits by frontrunning a setRateLimitData transaction	10
L-04 Relay can swap outside of rate limits by adding liquidity, imbalancing the pool, and then burning liquidity	11
Informational Issues	11
I-01. Partial AccessControlEnumerable implementation	12
I-02. Grammatical natspec errors	12
I-03. Error message for depositERC4626 check should be clarified	13
I-04. Not resetting the withdrawal rate limit upon deposits might leave the proxy with tokens temporarily locked in the vault	14
I-05. MainnetController::requestWithdrawFromWstETH() could be more gas efficient by rate-limiting wstETH amount	15
I-06. Inconsistent Cross-Chain Event Emission	16
I-07. LIMIT_OTC_SWAP assumes all traded assets are worth the same	16
I-08. OTC swap functionality on Spark ALM Controller would allow immediate loss of funds in case of relay compromise	17
<b>Disclaimer</b>	<b>19</b>
<b>About Certora</b>	<b>19</b>

# Project Summary

## Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Spark ALM	<a href="#">v1.8.0-beta.0</a>	3144c91	EVM
Spark ALM	<a href="#">v1.8.0</a>	7be9593	EVM

## Project Overview

This document describes the findings of the manual code review of **Spark ALM Controller v1.8.0**. The work was undertaken from **October 20th** to **October 31st**.

The scope of the review are all contracts in the below folder:

`spark-alm-controller/src/...`

## Protocol Overview

The Spark ALM Controller is an automated liquidity management system within the Sky ecosystem that aims to maintain optimal liquidity levels across EVM chains. It operates through a system where one proxy is holding funds while controllers execute operations from an automated relayer while respecting the limits set by Sky's governance.

The version 1.8.0 update adds OTC buffer functionality for off-chain swaps with exchanges while aiming to limit counterparty exposure, wStETH integration for staking operations, and comprehensive slippage protections for many deposit functions..

## Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	2	2	2
Low	4	4	0
Informational	8	8	2
<b>Total</b>	<b>14</b>	<b>14</b>	<b>4</b>

## Severity Matrix

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

# Detailed Findings

ID	Title	Severity	Status
M-01	Compromised relayer can drain proxy by incurring losses upon ERC4626 withdrawal	Medium	Fixed
M-02	Aave and ERC4626 are vulnerable to reentrancy attacks	Medium	Fixed
L-01	Overloaded maxSlippages allows bypassing of access control	Low	Acknowledged
L-02	Configuration validation gap can lead to permanently stuck funds	Low	Acknowledged
L-03	Relayer can exceed ratelimits by frontrunning a setRateLimitData transaction	Low	Acknowledged
L-04	Relayer can swap outside of rate limits by adding liquidity, imbalancing the pool, and then burning liquidity	Low	Acknowledged

## Medium Severity Issues

### M-01 Compromised relay can drain proxy by incurring losses upon ERC4626 withdrawal

Severity: **Medium**

Impact: **High**

Likelihood: **Low**

Files:  
MainnetController.sol

Status: Fixed

#### Description:

MainnetController::withdrawERC4626() allows a user to withdraw a given amount of tokens from an ERC4626, without checking for inflated shares.

This attack can lead to a loss of virtually 100% of the deposited tokens.

#### Exploit Scenario:

A compromised relay can use this to drain the proxy through the following path:

- Deposit to an inflated vault, e.g., deposit 1e18 assets and receive 1 share
- Withdraw an amount of assets such that amount % shares != 0, e.g. withdraw 1 wei of assets, which will burn 1 share

#### Recommendations:

Suggested mitigations:

- Remove withdrawERC4626() and keep only redeemERC4626()
- Add a slippage check on withdrawERC4626() similar to depositERC4626()
- Convert the requested amount to shares (via IERC4626.convertToShares()) and redeem that amount of shares instead

**Customer's response:** Implemented a `maxExchangeRate` which will cause the depositERC4626 to fail if the rate between shares and assets is too extreme.

**Fix Review:** The max exchange rate mechanism introduced in [commit d97b92c](#) effectively mitigates this issue, by preventing deposits on excessively inflated vaults.

**M-02 Aave and ERC4626 are vulnerable to reentrancy attacks**

Severity: <b>Medium</b>	Impact: <b>High</b>	Likelihood: <b>Low</b>
Files: MainnetController#499	Status: Fixed	

**Description:**

``depositERC4626()`` and ``depositAave()`` do important checks after interacting with the pool:

After the deposit, ``depositERC4626()`` checks:

- `IERC4626(token).convertToAssets(shares) >= amount * maxSlippages[token] / 1e18,`

Similarly, ``depositAave()`` checks:

- `uint256 newATokens = IERC20(aToken).balanceOf(address(proxy)) - aTokenBalance;`

Both of these checks are done **after** interacting with the external pool, which means if such a pool has a reentrancy vector, like a pre or post deposit hook, the relayer will be able to reenter the controller before these functions are triggered.

**Exploit scenario:**

A compromised relayer can exploit this in the following way:

- Deposit to an inflated vault with a reentrancy hook
- Reenter ``MainnetController::withdrawERC4626()``, withdrawing the shares at a loss
- Still, during the reentrancy, make a donation to the ERC4626 vault, increasing the share rate
- Let the execution continue, which will pass the slippage/inflation check noted above

**Recommendations:** Add Reentrancy Guards to all ERC4626 and Aave functions

**Customer's response:** Fixed.

**Fix Review:** Reentrancy guards were added to all state changing functions in the MainnetController and ForeignController contracts in commit [37e2b93](#).

## Low Severity Issues

### L-01 Overloaded maxSlippages allows bypassing of access control

Severity: **Low**

Impact: **Low**

Likelihood: **High**

Files:  
MainnetController#180

Status: Acknowledged

#### Description:

The `mapping (address pool => uint256 maxSlippage) maxSlippages` is heavily overloaded since it is not only used for Curve pools but also for ERC4626 vaults, Aave aTokens, and OTC exchanges.

Since it is used as an access control whitelist in multiple functions, this allows potential pollution where using a pool in an `otcswap` function or an `aToken` in a `pool` function can partially pass access control validation.

The contract utilizes multiple layers of validation, and currently any incorrect use is blocked by the `ratelimits` checks, which are based on the action string and the address.

While there is no possible exploitation today, this vulnerability weakens the security layer provided by using the mapping as an access control mechanism and can give developers a false sense of security. This is especially dangerous since the protocol is continuously evolving and expanding with new functionalities and features.

**Recommendations:** We strongly recommend implementing a clear separation of concerns and having a `maxSlippages` mapping per type of asset.

**Customer's response:** Acknowledged, the code will be refactored in the v2 version.

**Fix Review:** It would be advisable to have no further changes before the v2 version refactor. A fragile security layer will often allow vulnerabilities to slip in due to minor modifications.

### L-02 Configuration validation gap can lead to permanently stuck funds

Severity: **Low**

Impact: **High**

Likelihood: **Low**

Files:  
MainnetController#255

Status: Acknowledged

#### Description:

The `setOTCBuffer` function lacks validation to ensure the provided address is a legitimate `OTCBuffer` instance. The function validates that the exchange is non-zero and that `exchange != otcBuffer`, but it does not verify that the buffer address implements the expected interface or behavior.

While this is an operation by a trusted actor, the presence of existing input validations points to the recognized possibility of user input error. Since the consequences of an error are quite grave, all funds sent by the OTC actor would be permanently stuck, we believe a check to ensure that the address is an instance of `otcBuffer` is merited.

**Recommendations:** Add an input validation check that ensures the address is an instance of `otcBuffer`

**Customer's response:** Acknowledged.

**Fix Review:** Acknowledged.

### L-03 Relayer can exceed ratelimits by frontrunning a setRateLimitData transaction

Severity: <b>Low</b>	Impact: <b>Medium</b>	Likelihood: Low
Files: RateLimits#52	Status: Acknowledged	

#### Description:

Governance can adjust ratelimits through the `setRateLimitData` calls, of which there are 3:

- 1st: `maxAmount` and `lastAmount` are set as input variables
- 2nd: `maxAmount` is the input variable and also used for `lastAmount`
- 3rd: special case of setting unlimited rates

Specifically for the 2nd call, there is an opportunity for a relayer to massively bypass the intended `rateLimits` set by Sky Governance.

#### Exploit Scenario:

Assume:

- Current `rateLimit` = \$3M
- Governance transaction to set it to \$4M with the second function.
- Relayer Bob frontruns the transaction and uses the entire \$3M limit. `Funds spent` = \$3M
- Governance call is executed: `lastAmount == maxAmount == $4M`
- Bob uses the entirety of the new `rateLimit`. `Funds spent` = \$3M + \$4M = \$7M

This scenario is possible regardless of whether it is an increase or a decrease and merely requires that there was a previous limit and that the new limit is not 0.

**Recommendations:** Whenever a new `rateLimit` is set, the `lastAmount` should remain the same and allow the natural recharging to take it to its new limit.

**Customer's response:** Acknowledged.

**Fix Review:** Acknowledged.

### L-04 Relayer can swap outside of rate limits by adding liquidity, imbalancing the pool, and then burning liquidity

Severity: **Low**

Impact: **Medium**

Likelihood: Low

Files:

Status: Acknowledged

#### Description:

CurveLib::removeLiquidity() doesn't check the swap rate limits, such that a relayer can atomically:

- Imbalance the pool with a flashloan
- Add liquidity with the pool's imbalanced supply
- Imbalance the pool to the opposite side
- Remove liquidity with the pool original supply
- Rebalance the pool and return the flashloan

This would allow them to swap more than intended

#### Recommendations:

This is complex to mitigate, as it can occur naturally – i.e., the relayer adds liquidity, the balance between assets changes due to regular swaps occurring, and then the relayer removes liquidity in a different ratio.

Implementing a hard swap rate could DoS legitimate attempts to remove liquidity in this scenario.

Given this tradeoff, the recommended mitigation is to consider if it is better to have the flexibility (i.e., keep the code as is) or protection (i.e., check for swap rate limits upon removing liquidity).

**Customer's response:** Acknowledged.

**Fix Review:** Acknowledged.

## Informational Issues

---

### I-01. Partial AccessControlEnumerable implementation

#### Description:

In the recent update, the OZ AccessControl contract is replaced (but not used) by AccessControlEnumerable in:

- MainnetController
- ForeignController
- OTCBuffer

But AccessControl remains in:

- ALMProxy
- RateLimits

**Recommendation:** It is recommended to harmonize the type of OZ AccessControl contract across the protocol.

**Customer's response:** Acknowledged.

**Fix Review:** Acknowledged.

## I-02. Grammatical natspec errors

### Description:

In [interfaces/ILayerZero.sol#26](#):

Rust

```
* @dev These amounts can change dynamically and are up the the
specific oft implementation.
*/
```

The “the” should be replaced with “to”.

In [libraries/PSMLib.sol#154](#) :

Rust

```
function _rateLimited(IRateLimits rateLimits,bytes32 key, uint256
amount) internal {
```

There needs to be a space after the comma “rateLimits, bytes32”

**Customer’s response:** Fixed.

**Fix Review:** The spelling errors were fixed in commit [247af78](#).

### I-03. Error message for depositERC4626 check should be clarified

#### Description:

The `maxSlippages` require statement appears at first sight to check for a potential slippage problem with an ERC4626 vault. However, as explained by the client, the goal is not to check for slippage but to ascertain that the vault has not been inflated in preparation for a vault inflation attack. As written, it is very difficult to understand this for anyone not involved with coding this feature.

**Recommendation:** The error message should indicate the type of issue the require statement is guarding against.

```
Rust
require(
    IERC4626(token).convertToAssets(shares) >= amount * maxSlippages[token] / 1e18,
    "ForeignController/inflated-shares"
);
```

**Customer's response:** Fixed.

**Fix Review:** The error message was clarified in commit [f590ef6](#).

#### **I-04. Not resetting the withdrawal rate limit upon deposits might leave the proxy with tokens temporarily locked in the vault**

##### **Description:**

When the relayer withdraws from a ERC4626 vault, or from Aave, their deposit rate limit is reset, allowing them to deposit more immediately.

However, when the relayer deposits to such a vault, their withdrawal rate limit is not reset.

This might leave the proxy with tokens temporarily locked in the vault:

- Assuming deposit rate limit == withdrawal rate limit ==  $1e18$
- Deposit  $1e18$  tokens to the vault
  - Deposit rate limit is now 0
- Withdraw  $1e18$  tokens from the vault
  - Deposit rate limit is restored to  $1e18$
  - Withdrawal rate limit is now 0
- Deposit  $1e18$  tokens to the vault
  - Deposit rate limit is now 0
- Cannot withdraw now, as the withdrawal rate limit is still 0

##### **Recommendation:**

Consider resetting withdrawal upon deposit, or documenting that the current design is intended.

**Customer's response:** Acknowledged.

**Fix Review:** Acknowledged.

### I-05. MainnetController::requestWithdrawFromWstETH() could be more gas efficient by rate-limiting wstETH amount

#### Description:

The rate limiting is currently done based on a stETH, which requires the following conversion:

Rust

```
@> _rateLimited(  
    LIMIT_WSTETH_REQUEST_WITHDRAW,  
    IWstETHLike(Ethereum.WSTETH).getStETHByWstETH(amountToRedeem)  
);
```

#### Recommendation:

This could be more efficient by implementing the rate limit on a wstETH basis instead

**Customer's response:** Acknowledged.

**Fix Review:** Acknowledged.

### I-06. Inconsistent Cross-Chain Event Emission

**Description:** Events are only emitted upon changes to state variables and CCTP cross-chain transactions. The recently added transferTokenLayerZero function does emit an event even though it also transfers funds cross-chain.

**Recommendation:** For consistency and observability, LZ should also emit an event.

**Customer's response:** Acknowledged.

**Fix Review:** Acknowledged.

## **I-07. LIMIT\_OTC\_SWAP assumes all traded assets are worth the same**

### **Description:**

The rate limit is based on token amounts, so e.g., if governance allows  $1,000e18$  tokens, the operator can swap:

- $1,000e18$  USDS, worth \$1,000 or
- $1,000e18$  ETH, worth \$4 million

### **Recommendation:**

Only whitelist USD pegged stablecoins and document this intention to avoid error, or introduce per-token limits.

**Customer's response:** Acknowledged.

**Fix Review:** Acknowledged.

## **I-08. OTC swap functionality on Spark ALM Controller would allow immediate loss of funds in case of relayer compromise**

### **Description:**

``MainnetController::otcSend()`` allows the relayer to move up to “allowed amount” to “allowed exchange”.

This is intended and necessary for the OTC swap functionality, since OTC swaps happen offchain and cannot be done entirely through smart contracts.

The safety of the funds depends on how the intended OTC swap systems work. Some possible threats related to this include:

- Can the relayer choose the destination address of the swap out tokens?
- Can the relayer choose any swap out tokens? If a swap is done to a highly volatile, low liquidity token, even though by the time of the swap, they are equivalent in price, this can lead to severe loss of funds very quickly

This is outside the scope of this audit, but it worries us greatly since failures in the off-chain system could very easily be abused by a compromised relayer to exploit the on-chain system when it comes to the OTC swaps.

**Recommendation:**

If possible, use exchanges that allow to mitigate the threats listed above.

**Customer's response:** Acknowledged.

**Fix Review:** Acknowledged.

# Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

# About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.