Sky: Spark ALM Controller

Security Review

Cantina Managed review by:
M4rio.eth, Lead Security Researcher
Xmxanuel, Lead Security Researcher

January 6, 2026

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Sky Protocol is a decentralised protocol developed around the USDS stablecoin.

From Dec 15th to Dec 19th the Cantina team conducted a review of spark-alm-controller on commit hash 9ef81e88 (tag v1.9.0-beta.1). The team identified a total of **12** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	4	1	3
Gas Optimizations	0	0	0
Informational	8	4	4
Total	**12**	**5**	**7**

The Cantina Managed team reviewed Sky's spark-alm-controller holistically on commit hash 3dbc7cb0 (tag v1.9.0) and concluded that all findings were addressed and no new vulnerabilities were identified.

3 Findings

3.1 Low Risk

3.1.1 `bytes32` vs `bytes25` poolId mismatch allows inconsistent limits for the same Uniswap v4 pool

Severity: Low Risk

Context: UniswapV4Lib.sol#L56

Description: Spark uses a `bytes32 poolId` in `UniswapV4Lib` and to configure tick limits and rate limits.

However, Uniswapv4 `PositionManager` internally uses only `bytes25` to identify pools. The Uniswap v4 `PositionManager` calculates the unique pool ID by hashing the defining parameters of a pool and taking the first 25 bytes to store them in a mapping.

This creates a mismatch where different `bytes32 poolId` values used for tick limits and rate limits in Spark can map to the same `Uniswap` pool.

The existing `_requirePoolIdMatch` function in `UniswapV4Lib` verifies that the provided `bytes32 poolId` correctly corresponds to the derived `PoolKey` hash.

The validation is only performed in `increasePosition` and `decreasePosition`. The `mintPosition` and `swap` functions do not include this check.

This allows governance to accidentally configure different `tick limits` or `rate limits` for `bytes32` values that resolve to the same Uniswap pool. A relayer could then choose which `poolId` to pass, selecting whichever limits are most favorable.

Recommendation: Add `_requirePoolIdMatch(poolId, poolKey)` to both `mintPosition` and `swap` functions for consistency with `increasePosition` and `decreasePosition`. This would prevent multiple different `bytes32 poolId` values from mapping to the same `bytes25` Uniswap pool ID.

Sky: Fixed in commit 3d2d0dde.

Cantina Managed: Fixed.

3.1.2 Uniswap v4 interactions use `block.timestamp` as deadline

Severity: Low Risk

Context: UniswapV4Lib.sol#L166-L244, UniswapV4Lib.sol#L510-L517

Description: The Uniswap v4 integration passes `block.timestamp` as the `deadline` when calling:

- The universal router `execute(...)` (swaps), and...
- The position manager `modifyLiquidities(...)` (mint/increase/decrease liquidity).

According to the Uniswap v4 docs, `block.timestamp` should not be used:

Using `block.timestamp` effectively disables deadline protection, as the check

```
require(block.timestamp <= deadline)
```

will always pass at execution time.

We understand the trust assumption is to limit the relayer's degree of freedom.

However, a configurable deadline would still benefit the relayer by allowing them to specify how long a pending transaction remains valid in the mempool. This provides additional MEV protection.

Recommendation: Add an explicit `deadline` parameter to the relevant controller entrypoints (for example `swapUniswapV4`, `mintPositionUniswapV4`, `increaseLiquidityUniswapV4`, `decreaseLiquidityUniswapV4`).

Sky: Acknowledged.

Cantina Managed: Acknowledged.

3.1.3 Relayer can grief MetaMorpho vault liquidity by changing queues and reallocations without notional bounds

Severity: Low Risk

Context: ForeignController.sol#L118-L134, ForeignController.sol#L486-L524

Description: In `ForeignController`, the `RELAYER` role can call the following MetaMorpho allocator functions:

- `setSupplyQueueMorpho`.
- `updateWithdrawQueueMorpho`.
- `reallocateMorpho`.

These calls are only gated by `rateLimitExists(...)`, which checks that a rate limit is configured, but does **not** apply any actual notional or frequency bound to the action. As a result, a malicious or compromised relayer can repeatedly reshape how a MetaMorpho vault deploys and withdraws funds without being limited by the rate limit system.

This is risky because MetaMorpho withdrawal liquidity depends on the underlying Morpho markets. If assets are allocated into markets that are close to fully utilized (borrowed out), withdrawals can fail due to insufficient available liquidity in those markets. Since MetaMorpho withdraw behavior depends on its withdraw queue ordering, the relayer can also reorder the withdraw queue to prefer stressed / illiquid markets first, increasing the probability that vault withdrawals revert.

A relayer can intentionally liquidity-grief by:

- Reallocating assets away from liquid markets into markets that are likely to become fully utilized.
- Configuring withdraw queue ordering to pull from illiquid markets first.
- Concentrating exposure such that predictable market stress causes withdrawals/redemptions to revert.

This can effectively "*trap*" funds inside the vault for an unknown duration until liquidity returns (borrowers repay or new liquidity is supplied).

Recommendation: As there is no easy solution to this, the easiest would be to move these operations to a more trusted role, maybe a new role that is in between `DEFAULT_ADMIN` and `RELAYER`.

Sky: Acknowledged. After talking with the risk team we determined that the operational efficiency of using the `RELAYER` vs `DEFAULT_ADMIN_ROLE` (Sky governance) is preferable at least for the significant future. At some point maybe we change our allocators to be a higher quorum multisig but we deem them good for now

Cantina Managed: Acknowledged.

3.1.4 Ethena delegates can persist even after relayer removed

Severity: Low Risk

Context: MainnetController.sol#L738-L754, MainnetController.sol#L352-L356

Description: A `RELAYER` can call `setDelegatedSigner(address)` which forwards to `ethenaMinter.setDelegatedSigner(...)` via the proxy.

If a relayer is later found malicious and removed via `removeRelayer`, this only revokes the relayer's role in `MainnetController`. It does not automatically undo any delegated signer permissions that were previously set on the external Ethena minter.

This creates a "*lingering authorization*" risk: a compromised or malicious relayer could preemptively add many delegated signers. Even if the relayer is removed afterwards, those delegated signers may remain authorized at the Ethena minter level.

Recommendation: There is not an easy solution to this, for now i suggest to keep tracking of all delegate signers and remove them in case the relayer performs such attack. Would recommend to documenting this in the readme at the `Trust Assumptions and Attack Mitigation` section

Sky: Acknowledged. Documented it in the README, at commit hash 3dbc7cb0.

Cantina Managed: Acknowledged.

3.2 Informational

3.2.1 `_decreaseLiquidity` does not enforce `amount0Min` / `amount1Min` beyond Uniswap slippage checks

Context: UniswapV4Lib.sol#L346

Description: The `_decreaseLiquidity` function has `amount0Min` and `amount1Min` parameters but does not use them in the function body. These minimum amount values are already encoded in the calldata and forwarded to the Uniswap `PositionManager`, relying entirely on Uniswap's implementation to enforce slippage protection.

```
function _decreaseLiquidity(
    address        proxy,
    address        rateLimits,
    bytes32        poolId,
    address        token0,
    address        token1,
    uint256        amount0Min,  // unused
    uint256        amount1Min,  // unused
    bytes   memory callData
)
```

Recommendation: The `_decreaseLiquidity` could explicitly verify that the actual received token amounts meet the minimum thresholds.

```
require(endingBalance0 - startingBalance0 >= amount0Min, "MC/amount0-below-min");
require(endingBalance1 - startingBalance1 >= amount1Min, "MC/amount1-below-min");
```

Sky: Fixed in commit 7b59cf5e by removing the unused parameters from the `_decreaseLiquidity` function.

Cantina Managed: Fixed.

3.2.2 `_approveWithPermit2` allows `uint256` amounts exceeding `Permit2` `uint160` limit

Severity: Informational

Context: UniswapV4Lib.sol#L250

Description: The `_approveWithPermit2` function accepts a `uint256` amount parameter, but Permit2 internally uses `uint160` for allowance amounts. If an amount greater than `type(uint160).max` is passed, it will silently truncate when cast to `uint160`.

Recommendation: Add an explicit bounds check to prevent silent truncation:

```
require(amount <= type(uint160).max, "MC/amount-too-large-for-permit2");
```

Sky: Fixed in commit d54b80d8.

Cantina Managed: Fixed.

3.2.3 Inconsistent Uniswap v4 `PositionManager` action usage in `increasePosition`

Severity: Informational

Context: UniswapV4Lib.sol#L454

Description: The `increasePosition` operation uses two `CLOSE_CURRENCY` `PositionManager` actions after `INCREASE_LIQUIDITY`, while `mint` uses `SETTLE_PAIR` and `decreasePosition` uses `TAKE_PAIR`. Since increasing liquidity always requires settling tokens to the pool, using a single `SETTLE_PAIR` would be more consistent and explicit.

```
bytes memory actions = abi.encodePacked(
    uint8(Actions.INCREASE_LIQUIDITY),
    uint8(Actions.CLOSE_CURRENCY),
    uint8(Actions.CLOSE_CURRENCY)
);
```

Recommendation: Use `SETTLE_PAIR` for `increasePosition` to align with the other operations, instead of two `CLOSE_CURRENCY` actions.

Sky: Fixed in commit 61cbc57.

Cantina Managed: Fixed.

3.2.4 Inconsistent token approval handling (legacy `_approve` / raw `approve`) instead of `ApproveLib`

Severity: Informational

Context: PSMLib.sol#L57-L140, CCTPLib.sol#L46-L105, ForeignController.sol

Description: Some parts of the codebase use `ApproveLib.approve`, while other parts still use older approval patterns such as direct `IERC20.approve` via `proxy.doCall` (for example in `PSMLib` and `CCTPLib`), or custom `_approve` helper logic (for example in `ForeignController`).

Recommendation: Standardize token approval logic across the codebase by using `ApproveLib.approve` everywhere an approval is performed via the proxy (except where Permit2-specific logic is required).

Sky: Acknowledged, but `ApproveLib` is primarily used for handling `USDT` approvals.

Cantina Managed: Acknowledged.

3.2.5 `UniswapV4Lib.swap` is not split into internal helpers, unlike other actions

Severity: Informational

Context: UniswapV4Lib.sol#L166-L244

Description: `UniswapV4Lib.swap` currently inlines multiple responsibilities in a single function:

- Building the router "*actions/params/inputs*" payload.
- Performing slippage checks.
- Triggering the rate limit decrease.
- Managing Permit2 approvals (set approval, execute, reset approval).
- Executing the router call.

Other flows in the same library (for example liquidity changes) are structured as:

- One helper that builds calldata (`_getDecreaseLiquidityCallData`, `_getMintCalldata`, `_getIncreaseLiquidityCallData`).
- One helper that performs the external interaction and surrounding bookkeeping (`_decreaseLiquidity`, `_increaseLiquidity`).

Keeping `swap` monolithic makes the integration harder to review and maintain as it's different than the rest of the functions.

Recommendation: Refactor the swap path to match the structure used elsewhere in `UniswapV4Lib` by splitting it into two internal functions, for example:

- `_getSwapCallData(...)` (or `_getSwapInputs(...)`): builds the `commands` + `inputs` payload for `IUniversalRouterLike.execute`.
- `_swap(...)`: performs rate limit accounting, Permit2 approval management, and the external router call

Sky: Acknowledged.

Cantina Managed: Acknowledged.

3.2.6 Uniswap v4 hook support is not defined or enforced

Severity: Informational

Context: UniswapV4Lib.sol#L166-L244, UniswapV4Lib.sol#L408-L508, MainnetController.sol#L323-L346

Description: Uniswap v4 pools can be deployed with hooks. Hooks can execute arbitrary logic during swaps and liquidity modifications.

The current integration always passes empty hook data (`hookData: bytes("")` / `""`) when:

- Swapping (`UniswapV4Lib.swap`).
- Minting/increasing/decreasing liquidity (calldata builders use `""` for hook data).

This means, currently we can not manage liquidity on the pools with hooks. We should enforce this at set limits time to avoid whitelisting such pools.

The Spark team has stated that in the future might want to accept also pools with hooks which increases the risk.

Hooks can materially change outcomes. In particular, liquidity decrease relies on `amount0Min` / `amount1Min` provided by the relayer and does not apply an additional safety check based on value or slippage. A malicious hook could cause the decrease flow to return less value than expected, effectively allowing the pool/hook behavior to drain the liquidity.

Recommendation: If hooks are **not intended to be supported initially**, enforce that at the code level. Add a check when configuring pools (recommended location: `MainnetController.setUniswapV4TickLimits`) that rejects pools with hooks:

```
require(address(poolKey.hooks) == address(0), "MC/pool-has-hooks");
```

If hooks may be supported in the future, document and implement a clear trust model allowing only specific reviewed hooks that won't affect the liquidity handling.

Sky: Acknowledged. This will be enforced by governance rather than through a require check.

Cantina Managed: Acknowledged.

3.2.7 `setUniswapV4TickLimits` does not validate tick bounds and can configure non-existent pools

Severity: Informational

Context: MainnetController.sol#L323-L346

Description: `setUniswapV4TickLimits` only checks that:

- Either all values are zero (meaning “disabled”), or..
- `maxTickSpacing > 0` and `tickLowerMin < tickUpperMax`.

It does not validate that `tickLowerMin` / `tickUpperMax` are within Uniswap's valid tick bounds (`-887272` to `887272`).

In addition, the function accepts an arbitrary `poolId` without checking whether the pool actually exists in the Uniswap v4 Position Manager.

Recommendation: Add stricter validation in `setUniswapV4TickLimits`, for example:

- If enabling (non-zero limits), require:
  - `tickLowerMin >= MIN_TICK`.
  - `tickUpperMax <= MAX_TICK`.
  - and keep the existing `tickLowerMin < tickUpperMax` check.

- Also validate that `poolId` exists before storing limits, for example by querying the Uniswap v4 Position Manager for the `PoolKey` and requiring it matches a valid/initialized pool (or at minimum that the returned `PoolKey` is not the default empty key).

Sky: Acknowledged. We won't fix it but we will review through governance process.

Cantina Managed: Acknowledged.

3.2.8 Minor issues, Typos & Documentation

Severity: Informational

Context: *(See each case below)*

Description:

UniswapV4Lib.sol#L105: The current comment only mentions transferred Uniswap liquidity positions as a reason for checking tick limits in `increasePosition`. However, positions minted under historical limits that have since been changed face the same restriction. Update the comment to reflect both cases:

```
// When adding funds, validate the position's tick range against current tick
↪  limits.
// The position may have been transferred to the proxy or minted under historical
↪  limits.
// Such positions can still be decreased, but increasing liquidity is only allowed
↪  if the ticks stay within limits.
```

UniswapV4Lib.sol#L339: The comment in `UniswapV4Lib._increaseLiquidity` stating that it is unnecessary to reset the `PositionManager` approval in `permit2` is incorrect. The `_approveWithPermit2` is called with `amount0Max` / `amount1Max`. If `increaseLiquidity` doesn't use the full amounts (e.g., due to an added buffer in case the pool state changes), the Position Manager can retain a non-zero Permit2 allowance (and ERC20 approval to Permit2). Update the comment to clarify that it is needed to fully clean up remaining allowances.

UniswapV4Lib.sol#L324, UniswapV4Lib.sol#L370: typo: `equally (i.e. 1.00000 USDC ...` ⇒ `equally (i.e. 1.000000 USDC ...`

MainnetController.sol#L793: `cooldownSharesSUSDe` function could use internal helper function `_rateLimited` to decrease the limit.

Recommendation: Consider addressing the aforementioned issues.

Sky: Fixed in 3fe30725, f8e16e4.

Cantina Managed: Fixed.