Sky: Spark ALM Controller v1.5.0-beta.0

Security Review

Cantina Managed review by:

Christoph Michel, Lead Security Researcher

Mario.eth, Lead Security Researcher

July 23, 2025

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	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

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 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.

On Jul 7th the Cantina team conducted a review of spark-alm-controller v1.5.0-beta.0 on commit hash 9f24f17f. The team identified a total of **9** issues in the following risk categories:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	2	2	0
Gas Optimizations	1	1	0
Informational	5	4	1
Total	**9**	**8**	**1**

The Cantina team reviewed Sky's spark-alm-controller changes holistically on commit hash 38da4568 (corresponding to tag v1.5.0) and determined that all findings were resolved and no new issues were identified.

3 Findings

3.1 Medium Risk

3.1.1 `ForeignController.transferTokenLayerZero` is missing approval to the Sky OFT adapter

Severity: Medium Risk

Context: ForeignController.sol#L239

Description: The `ForeignController.transferTokenLayerZero` function is supposed to be compatible with:

1. The official USDT0 adapter.
2. The L2 Sky OFT adapters (mint-and-burn) for USDS, sUSDS, (SKY?).

While the `USDT0` adapter does not require approvals as the adapter has special burn rights in the USDT0 token, the Sky OFT adapter does not have special `burn` privileges for USDS/sUSDS/Sky as these tokens technically do not support burn privileges.

Whenever, `burn` (or `transferFrom`) from the Sky OFT adapter is called, the caller must have approved the Sky OFT adapter to burn their tokens.

This approval is missing in the `ForeignController.transferTokenLayerZero`.

```
// USDS burn
function burn(address from, uint256 value) external {
    uint256 balance = balanceOf[from];
    require(balance >= value, "Usds/insufficient-balance");

    // calling burn always requires approval as msg.sender (Sky OFT adapter) != from (almProxy)
    if (from != msg.sender) {
        uint256 allowed = allowance[from][msg.sender];
        if (allowed != type(uint256).max) {
            require(allowed >= value, "Usds/insufficient-allowance");

            unchecked {
                allowance[from][msg.sender] = allowed - value;
            }
        }
    }
    // ...
```

Recommendation: Similar to `MainnetController.transferTokenLayerZero`, consider adding the approval call to `ForeignController.transferTokenLayerZero`:

```
_approve(ILayerZero(oftAddress).token(), oftAddress, amount);
```

Note that LayerZero OFT adapters also implement an `approvalRequired()` function that returns `true` in case the caller has to approve the contract for sending:

```
/**
 * @notice Indicates whether the OFT contract requires approval of the 'token()' to send.
 * @return requiresApproval Needs approval of the underlying token implementation.
 */
function approvalRequired() external pure virtual returns (bool) {
    return true;
}
```

This function could also be checked and the approval only performed if it returns `true`.

- This would correctly **skip** the approvals for L2 USDT0 Adapter.
- This would correctly **set** the approvals for L1 USDT0 Adapter.
- This would correctly **set** the approvals for both Sky `MintAndBurnOFTAdapter` (L2) and escrow-type `OFTAdapter` (L1) adapters.

This could be implemented on both `MainnetController` and `ForeignController`.

Sky: Added `approvalRequired` in both `MainnetController` and `ForeignController` in PR 123.

Spearbit: Fix verified.

3.2 Low Risk

3.2.1 Compromised relayer can use `transferTokenLayerZero` to burn ETH in ALM Proxy

Severity: Low Risk

Context: MainnetController.sol#L813

Description: A compromised relayer can use the `transferTokenLayerZero` function to bridge 1 wei of one of the allowed tokens. Each LayerZero bridge action burns a fixed gas fee from the ALM Proxy. This can be repeated several times.

Recommendation: If this attack is a concern (should depend on how much native tokens are kept in the ALM Proxy), consider adding a second "rate limit" to `transferTokenLayerZero` that restricts the number of calls that can be made to that function. Each `transferTokenLayerZero` invocation would consume 1.0 units (for example 1e18) of this rate limit.

Sky: Fixed in PR 127.

Spearbit: Fix verified.

3.2.2 `_approve` ignores the `success` bool return value

Severity: Low Risk

Context: MainnetController.sol#L846-L857

Description: The `_approve` logic was changed such that it retries a failed `approve` call by a sequence of **two** `approve` calls, the first resetting the approval to 0, and the second one retrying the original approval.

```
function _approve(address token, address spender, uint256 amount) internal {
    bytes memory approveData = abi.encodeCall(IERC20.approve, (spender, amount));

    // Call doCall on proxy to approve the token
    ( bool success, bytes memory data )
        = address(proxy).call(abi.encodeCall(IALMProxy.doCall, (token, approveData)));

    // Decode the first 32 bytes of the data, ALMProxy returns 96 bytes
    bytes32 result;
    assembly { result := mload(add(data, 32)) }

    // Decode the result to check if the approval was successful
    bool decodedSuccess = (data.length == 0) || result != bytes32(0);

    // If call succeeded with expected calldata, return
    if (success && decodedSuccess) return;

    // If call reverted, set to zero and try again
    proxy.doCall(token, abi.encodeCall(IERC20.approve, (spender, 0)));
    proxy.doCall(token, abi.encodeCall(IERC20.approve, (spender, amount)));
}
```

> This is done to support tokens like USDT that require resetting the approval to 0 first before being able to approve non-zero values.

However, the check if the first approval succeeded or not is incorrect.

When the low-level call to the `almProxy` via `address(proxy).call(abi.encodeCall(IALMProxy.doCall, (token, approveData)))` is performed, the `data` is the **abi encoding** of the approve-call-return-data, not the approve-call-return-data itself.

Meaning, the actual `bytes` approve-call-return-data is encoded again, such that `data` consists of the following fields `offset ++ data.length ++ data.bytes`:

```
0000000000000000000000000000000000000000000000000000000000000020 offset to data (0x20)
0000000000000000000000000000000000000000000000000000000000000020 data.length (32)
0000000000000000000000000000000000000000000000000000000000000001 data bytes (bytes32(1))
```

An `approve` call that does not fail returns 96 `data` bytes. By reading `add(data, 32)` into `result`, the **offset** is read, not the actual approve-call-return-boolean.

This leads to a false positive for tokens that return `false` on approve, the offset `0x20` is interpreted as a successful `result` and the approval-retry code does not run.

Proof of Concept:

Here's an ERC20 token having the same USDT-like approval behavior but returns `false` instead of reverting on the initial check. Add the code to `spark-alm-controller/test/mainnet-fork/Approve.t.sol`:

```
import { ERC20 } from "openzeppelin-contracts/contracts/token/ERC20/ERC20.sol";

contract ERC20ApproveFalse is ERC20 {
    constructor(string memory name_, string memory symbol_) ERC20(name_, symbol_) {}

    function approve(address spender, uint256 value) public virtual override returns (bool) {
        // USDT-like resetting to 0 required. but returns false instead of reverting
        if ((value != 0) && (allowance(msg.sender, spender) != 0)) {
            return false;
        }

        return super.approve(spender, value);
    }
}


contract CantinaApproveTest is MainnetControllerApproveSuccessTests {
    function test_approveCustom() public {
        ERC20ApproveFalse mock = new ERC20ApproveFalse("Mock", "MOCK");
        _approveTest(address(mock), harness);
    }
}
```

Recommendation:

The same issue also applies to `ForeignController._approve`, `CurveLib._approve`.

Consider changing to an implementation that checks the actual `success` boolean instead of the offset:

```
function _approve(address token, address spender, uint256 amount) internal {
    bytes memory approveData = abi.encodeCall(IERC20.approve, (spender, amount));

    // Call doCall on proxy to approve the token
    ( bool success, bytes memory data )
        = address(proxy).call(abi.encodeCall(IALMProxy.doCall, (token, approveData)));

    if (success) {
        // data is the ABI-encoding of the approve call bytes return data, need to decode it first
        bytes memory approveCallReturnData = abi.decode(data, (bytes));
        // approve was successful if 1) no return value or 2) true return value
        if (approveCallReturnData.length == 0 || abi.decode(approveCallReturnData, (bool))) {
            return;
        }
    }

    // If call was unsuccessful, set to zero and try again
    proxy.doCall(token, abi.encodeCall(IERC20.approve, (spender, 0)));

    // 1. either this
    proxy.doCall(token, abi.encodeCall(IERC20.approve, (spender, amount)));
    // 2. Alternatively, if the _approve function should revert on a final failed approve for tokens that
    ↪  return false:
    bytes memory approveCallReturnData = proxy.doCall(token, abi.encodeCall(IERC20.approve, (spender, amount)));
    require(approveCallReturnData.length == 0 || abi.decode(approveCallReturnData, (bool)),
    ↪  "MainnetController/approve-failed");
}
```

Sky: Fixed in PR 126.

Spearbit: Fix verified.

3.3 Gas Optimization

3.3.1 _approve can re-use approveData in third call

Severity: Gas Optimization

Context: MainnetController.sol#L861

Description: The _approve logic was changed such that it retries a failed approve call by a sequence of **two** approve calls, the first resetting the approval to 0, and the second one retrying the original approval.

Recommendation: For the third call, consider re-using the same calldata as for the first call, to save on additional calldata encoding cost.

```
proxy.doCall(token, abi.encodeCall(IERC20.approve, (spender, 0)));
- proxy.doCall(token, abi.encodeCall(IERC20.approve, (spender, amount)));
+ proxy.doCall(token, approveData);
```

Sky: Fixed in PR 131.

Spearbit: Fix verified.

3.4 Informational

3.4.1 Quirks in OFT implementations

Severity: Informational

Context: MainnetController.sol#L810-L814, ForeignController.sol#L270-L274

Description: When integrating LayerZero OFTs we currently assume uniform logic across all the implementations but that is not always true, some implementations have quirks that needs to be assessed. For example in the USDT0 the _credit logic does not redirect the amount to 0xdead if _to is address(0) or token itself:

- Address 0xcd979b10a55fcdac23ec785ce3066c6ef8a479a4 implementation:

```
    function _credit(
           address _to,
           uint256 _amountLD,
           uint32 /*_srcEid*/
       ) internal virtual override returns (uint256 amountReceivedLD) {
           // @dev Unlock the tokens and transfer to the recipient.
           innerToken.safeTransfer(_to, _amountLD);
           // @dev In the case of NON-default OFTAdapter, the amountLD MIGHT not be == amountReceivedLD.
           return _amountLD;
       }
```

In the SKY's implementation we redirect to 0xdead:

```
 function _credit(
        address _to,
        uint256 _amountLD,
        uint32 _srcEid
    ) internal virtual override whenNotPaused returns (uint256 amountReceivedLD) {
        if (_to == address(0x0) || _to == address(innerToken)) _to = address(0xdead); // transfer fn has
        ↪   restrictions

        // Check and update the rate limit based on the source endpoint ID (srcEid) and the amount in local
        ↪   decimals from the message.
        _checkAndUpdateRateLimit(_srcEid, _amountLD, RateLimitDirection.Inbound);

        // @dev Unlock the tokens and transfer to the recipient.
        innerToken.safeTransfer(_to, _amountLD);

        // @dev In the case of NON-default OFT, the _amountLD MIGHT not be == amountReceivedLD.
        return _amountLD;
    }
```

Recommendation: Treat every new OFT as bespoke. During onboarding, review the logic and add an integration tests asserting expected behavior.

Sky: Acknowledged.

Spearbit: Acknowledged.

3.4.2 Minor issues

Severity: Informational

Context: See each case below

Description: The following issues are minor and we aggregated them into one informational issue:

- Unused import `IMetaMorpho`: MainnetController.sol#L11.
- Unused import `AccessControl`: CurveLib.sol#L5.
- Unused import `RateLimitHelpers`: PSMLib.sol#L9.
- The `IATokenWithPool` interface is defined in both the Mainnet and Foreign controllers. Consider extracting it into a designated file.
- Typo `multipled` ⇒ `multiplied`: CurveLib.sol#L97.
- MainnetController.sol#L7-9 It is recommended to not use files that are meant as production to be used for production code, as stated in the forge `README`:

  > Forge Standard Library is a collection of helpful contracts and libraries for use with Forge and Foundry. It leverages Forge's cheatcodes to make writing tests easier and faster, while improving the UX of cheatcodes.

  Consider using OpenZeppelin interfaces for `IERC20`/ `IERC4626` and for `IERC7540` consider using just defining it in your `interfaces` directory

Recommendation: Consider resolving the aforementioned issues.

Sky: Fixed in commit c90f7baf.

Spearbit: Fix verified.

3.4.3 Deployment scripts set `mintRecipients` (for CCTP bridging) but no LayerZero recipients

Severity: Informational

Context: MainnetControllerInit.sol#L155-L157

Description: The init deployment scripts set the controller's `mintRecipients` for CCTP bridge transfers. However, the recipients for the similar LayerZero bridge transfers are not set.

Recommendation: Consider allow-listing LayerZero recipients by calling `setLayerZeroRecipient` in the init scripts after `setMintRecipient`.

> **The same issue applies to `ForeignControllerInit`.**

Sky: PR 129.

Spearbit: Fix verified.

3.4.4 `ILayerZero` interface improvements

Severity: Informational

Context: ILayerZero.sol#L61

Description:

1. The `ILayerZero` interface uses `uint` types for the `MessagingFee` instead of the standard `uint256` type name.
2. The `quoteSend` function is not declared `view` even though it is `view` in OFT adapters. Currently, when using this interface, no `staticcalls` might be performed.

Recommendation: Consider addressing the mentioned inconsistencies.

Sky: Fixed in PR 124.

Spearbit: Fix verified.

3.4.5 Consistent `IRateLimits rateLimits` parameter order

Severity: Informational

Context: PSMLib.sol#L156-L162

Description: Most internal helper functions for the library contracts in `/libraries` (`_approve`, `_rateLimited`) take the controller state variables as the first parameter.

Recommendation: For consistency, consider changing `PSMLib`'s `_rateLimited(bytes32 key, uint256 amount, IRateLimits rateLimits)` and `_cancelRateLimit(bytes32 key, uint256 amount, IRateLimits rateLimits)` functions to take `IRateLimits rateLimits` as the first parameter. This would match `CCTPLib`.

Sky: Fixed in PR 122.

Spearbit: Fix verified.