

Sky: Spark ALM Controller

Security Review

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

February 25, 2026

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 Low Risk . . . 4
  - 3.1.1 Uniswap V4 `increaseLiquidity` could use `CLOSE_CURRENCY` instead of `SETTLE_PAIR` . . . 4
  - 3.1.2 Missing slippage protection on weETH withdrawal request . . . 4
- 3.2 Informational . . . 5
  - 3.2.1 EtherFi `deposit`s can leave `eETH` dust in `almProxy` . . . 5
  - 3.2.2 No minimum weETH withdrawal amounts could lead to withdrawal griefing . . . 6
  - 3.2.3 `quoteSend` should be proxied through `almProxy` for consistency with `quoteOFT` . . . 7
  - 3.2.4 Missing explicit ownership check before claiming WeETH withdrawal . . . 7
  - 3.2.5 Claim weTH rate limit creates unnecessary bottleneck for withdrawals . . . 8
  - 3.2.6 Minor issues, Typos & Documentation . . . 8

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 Feb 7th to Feb 9th the Cantina team conducted a review of spark-alm-controller on commit hash 5379267e (tag v1.10.0-beta.0). The team identified a total of **8** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	2	2	0
Gas Optimizations	0	0	0
Informational	6	3	3
Total	**8**	**5**	**3**

The Cantina Managed team reviewed Sky's spark-alm-controller holistically on commit hash 984ec546 (tag v1.10.0) and concluded that all findings were addressed and no new vulnerabilities were identified.

3 Findings

3.1 Low Risk

3.1.1 Uniswap V4 `increaseLiquidity` could use `CLOSE_CURRENCY` instead of `SETTLE_PAIR`

Severity: Low Risk

Context: UniswapV4Lib.sol#L445-L457

Description: The `_getIncreaseLiquidityCallData` function is currently using the `SETTLE_PAIR` action. However, this function reverts in case there's a positive delta, which could happen naturally due to automatically collecting fees or as the result of other actions that have been previously batched with the transaction.

Recommendation: Consider using `CLOSE_CURRENCY` on both tokens instead of `SETTLE_PAIR`. `CLOSE_CURRENCY` is the safer option as it either takes or settles, depending on positive or negative delta.

Sky: Fixed in commit 402bb08c.

Cantina Managed: Fix verified.

3.1.2 Missing slippage protection on weETH withdrawal request

Severity: Low Risk

Context: WEETHLib.sol#L95-L140

Description: The `requestWithdraw()` function performs two rate-dependent conversions without slippage protection:



- Conversion 1 (L109-118): `weETH.unwrap(weETHShares)` converts weETH shares to eETH amount using the current weETH/eETH exchange rate.
- Conversion 2 (L130-139): `liquidityPool.requestWithdraw()` internally calls `sharesForAmount(eETHAmount)` to determine the `shareOfEEth` stored in the withdrawal request.

Neither conversion has slippage protection. While `depositToWeETH()` includes a `minSharesOut` parameter (L92), the withdrawal path has no equivalent `minEETHShares` check.

At claim time, the user receives `min(amountOfEEth, amountForShare(shareOfEEth))` - always the worse rate. If either rate moves unfavorably between request and execution, the locked shares could represent less ETH than expected.

Recommendation: Add a `minEETHShares` parameter to protect against cumulative rate exposure across both conversions

Sky: Fixed in commit 984ec546.

Cantina Managed: Verified. Note `claimWithdrawal` also computes `claimWithdraw` → `getClaimableAmount` →

```
payoutEthAssets = min(request.eEthAssetsAtRequest,
↪  LiquidityPool.amountForShare(request.eEthSharesAtRequest)) - fee
```

So in theory you could also have a slippage check there in case the `amountForShare` drops below and you want to wait until it recovers.

Sky: Acknowledged.

3.2 Informational

3.2.1 EtherFi `deposits` can leave `eETH` dust in `almProxy`

Severity: Informational

Context: WEETHLib.sol#L67-L88

Description: The EtherFi deposit path is `WETH` → `eETH` shares → `eETH` amount → `weETH`, and the code converts through:

- `liquidityPool.deposit()` (returns `eETHShares`).
- `amountForShare(eETHShares)` (computes `eETHAmount`).
- `weETH.wrap(eETHAmount)`.

This share → amount → share roundtrip rounds down twice and can leave residual `eETH` dust in the `almProxy` in the end.

It is even possible to deposit non-zero WETH but receive 0 weETH and leave eETH dust in the `almProxy`. A compromised relayer could abuse this to (temporarily) lock up the funds. Retrieving the eETH dust would require a `MainnetController` update.

Recommendation: There is no easy way to avoid this while keeping the deposit path as WETH → weETH as it's a result of how the EtherFi integration works. Consider updating the `MainnetController` with a wrap eETH → weETH function in case the residual eETH dust ever gets to a non-negligible amount.

Proof of Concept:

```
MainnetController.depositToWeETH(amount = 2)
└─ WEETHLib.deposit(...)
   ├─ LiquidityPool.deposit{value: 2}
   │  └─ _sharesForDepositAmount(2)
   │     shares = floor(2 / 1.5) = 1 share
   │  ← returns eETHShares = 1
   │
   ├─ LiquidityPool.amountForShare(eETHShares = 1)
   │  amount = floor(1 * 1.5) = 1 asset
   │  // convert the returned shares back to an asset amount
   │
   └─ WeETH.wrap(eETHAmount = 1)
      ├─ LiquidityPool.sharesForAmount(1)
      │  weETHShares = floor(1 / 1.5) = 0
      │
      └─ eETH.transferFrom(ALMProxy, WeETH, amount = 1)
         └─ EETH._transfer(amount = 1)
            sharesToTransfer = floor(1 / 1.5) = 0
```

```
// SPDX-License-Identifier: AGPL-3.0-or-later
pragma solidity >=0.8.0;

import { IEETHLike } from "../../src/libraries/WEETHLib.sol";

import "./ForkTestBase.t.sol";

interface IWEETHLike is IERC20 {
    function eETH() external view returns (address);
}

contract MainnetControllerWeETHTestBase is ForkTestBase {

    IWEETHLike weETH = IWEETHLike(Ethereum.WEETH);

    IEETHLike eETH;

    function setUp() public override {
        super.setUp();
```

```
        eETH = IEETHLike(weETH.eETH());
    }

    function _getBlock() internal override pure returns (uint256) {
        return 23469772; //  September 29, 2025
    }

}

contract MainnetControllerDepositToWeETHDustTest is MainnetControllerWeETHTestBase {

    function test_depositToWeETH_smallAmountMintsZeroWeETH() external {
        bytes32 key = mainnetController.LIMIT_WEETH_DEPOSIT();

        vm.prank(Ethereum.SPARK_PROXY);
        (bool ok, ) = address(rateLimits).call(
            abi.encodeWithSignature(
                "setRateLimitData(bytes32,uint256,uint256)",
                key,
                2,
                0
            )
        );
        require(ok, "rateLimit set failed");

        deal(Ethereum.WETH, address(almProxy), 2);

        vm.prank(relayer);
        uint256 shares = mainnetController.depositToWeETH(2, 0);

        assertEq(shares, 0);
        // received 0 shares
        assertEq(weETH.balanceOf(address(almProxy)), 0);
        // eETH dust in almProxy, temporarily unrecoverable
        assertEq(eETH.shares(address(almProxy)), 1);
    }

}

//  MAINNET_RPC_URL=https://rpc.flashbots.net MAINNET_API_KEY=foo
↪  OPTIMISMSCAN_API_KEY=foo BASESCAN_API_KEY=foo GNOSISSCAN_API_KEY=foo
↪  ARBISCAN_API_KEY=foo WORLD_CHAIN_API_KEY=foo UNICHAIN_API_KEY=foo forge test
↪ --match-path test/mainnet-fork/weETHDust.t.sol
```

Sky: Acknowledged. This dust accrual is negligible and if ever became a real issue it could be fixed with a `MainnetController` upgrade as mentioned

Cantina Managed: Acknowledged.

3.2.2 No minimum weETH withdrawal amounts could lead to withdrawal griefing

Severity: Informational

Context: WEETHLib.sol#L95-L139

Description: `requestWithdraw` forwards the full unwrapped `eETHAmount` to EtherFi's `LiquidityPool.requestWithdraw(...)` without enforcing any minimum request size. A compromised relayer can submit many tiny withdrawals, resulting in many withdraw-request NFTs that must later be claimed one-by-one, creating operational overhead / gas grief for the withdrawal module.

Recommendation: Consider adding a minimum weETH withdrawal shares amount or acknowledging this issue as the cost and lack of incentives make this attack unlikely even if the relayer is compromised.

Sky: Acknowledged, agreed on cost and lack of incentives here.

Cantina Managed: Acknowledged.

3.2.3 `quoteSend` should be proxied through `almProxy` for consistency with `quoteOFT`

Severity: Informational

Context: LayerZeroLib.sol#L87

Description: While the `quoteOFT` function is currently being called through the `almProxy`, and not the `MainnetController`, the `quoteSend` function is not.

```
( ,, OFTReceipt memory receipt ) = abi.decode(
    proxy.doCall(
        oftAddress,
        abi.encodeCall(ILayerZero.quoteOFT, (sendParams))
    ),
    (OFTLimit, OFTFeeDetail[], OFTReceipt)
);

MessagingFee memory fee = ILayerZero(oftAddress).quoteSend(sendParams, false);
```

While both functions' quotes do currently not depend on specific `msg.sender` values, if anything, the `quoteSend` function is more likely to quote on it in the future as it already uses `msg.sender` in case it's a compose message (which is always false for the current `sendParams`).

Recommendation: For consistency with `quoteOFT`, consider also calling `quoteSend` via `almProxy`.

Sky: Fixed in commit bd6f42b5.

Cantina Managed: Fix verified.

3.2.4 Missing explicit ownership check before claiming WeETH withdrawal

Severity: Informational

Context: WEETHModule.sol#L71-L96

Description: The `claimWithdrawal()` function validates that the request is valid and finalized, but does not verify that the `WEETHModule` actually owns the withdrawal NFT before attempting to claim:

```
function claimWithdrawal(uint256 requestId) external returns (uint256 ethReceived) {
    require(msg.sender == almProxy(), "WeEthModule/invalid-sender");

    // ... get addresses ...

    require(
        IWithdrawRequestNFTLike(withdrawRequestNFT).isValid(requestId),
        "WeEthModule/invalid-request-id"
    );

    require(
        IWithdrawRequestNFTLike(withdrawRequestNFT).isFinalized(requestId),
        "WeEthModule/request-not-finalized"
    );

    // MISSING: ownership check

    IWithdrawRequestNFTLike(withdrawRequestNFT).claimWithdraw(requestId);
    // ...
}
```

While the `WithdrawRequestNFT.claimWithdraw()` will revert it's better to revert early with our custom error.

Recommendation: Add ownership validation before claiming:

```
  require(
      IERC721(withdrawRequestNFT).ownerOf(requestId) == address(this),
      "WeEthModule/not-owner"
  );
```

Sky: Acknowledged, we will just rely on the etherFi error handling here.

Cantina Managed: Acknowledged.

3.2.5 Claim weTH rate limit creates unnecessary bottleneck for withdrawals

Severity: Informational

Context: WEETHLib.sol#L159-L163

Description: The `claimWithdrawal()` function applies a rate limit that decreases by the amount of ETH received:

```
// NOTE: weETHModule is enforced to be correct by the rate limit key
_rateLimited(
    rateLimits,
    RateLimitHelpers.makeAddressKey(LIMIT_WEETH_CLAIM_WITHDRAW, weETHModule),
    ethReceived
);
```

The comment states the rate limit key validates the weETHModule address, which is correct. However, tracking the `ethReceived` amount creates an unnecessary bottleneck:

Scenario:

1. REQUEST limit: 100 ETH equivalent.
2. CLAIM limit: 10 ETH.
3. Relayer creates 10 withdrawal requests of 10 weETH each (consumes REQUEST limit).
4. Requests are finalized by EtherFi
5. Relayer can only claim 10 ETH before CLAIM limit is exhausted.
6. Remaining 90 ETH worth of fulfilled requests sit in NFTs, waiting for claim limit to be regenerated.

Recommendation: You can use `rateLimitExists` just to verify the `weETHModule`.

Sky: Fixed in commit 4ad35877.

Cantina Managed: Fix verified.

3.2.6 Minor issues, Typos & Documentation

Severity: Informational

Context: (*See each case below*)

Severity: Informational

Description:

- ☐ ForeignController.sol#L68-L78, MainnetController.sol#L143-L172: Combination of keccak256 inline hashes and hashes in libraries, this should be standardized.

  WEETHModule.sol#L65: The `_authorizeUpgrade` should not be view.

  WEETHModule.sol#L53-54: The errors should use `WeEthModule` because that's what we use in the rest of the module.

Recommendation: Consider fixing the issues mentioned above.

Sky: Fixed in commit 10656c8b. Not addressing library issues in this release, will do when we have 100% libraries in an upcoming release.

Cantina Managed: Fix verified.