# Sprinter Stash Contracts
## Security Review


Cantina Managed review by:

**Blockdev**, Security Researcher
**Ladboy233**, Security Researcher



June 4, 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

Sprinter is next-generation solver-based infrastructure designed to optimize crosschain transactions.

From May 22nd to May 27th the Cantina team conducted a review of sprinter-stash-contracts on commit hash 8d793484. The team identified a total of **10** 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 | 1 | 1 | 0 |
| Informational | 5 | 2 | 3 |
| **Total** | **10** | **4** | **6** |

# 3 Findings

## 3.1 Low Risk

### 3.1.1 Use `forceApprove()`

**Severity:** Low Risk

**Context:** SprinterLiquidityMining.sol#L43

**Description:** The contract uses the standard `approve()` function for setting ERC20 token allowances. This approach can cause issues with tokens that require setting the allowance to zero before updating it to a new value (e.g., USDT), potentially resulting in failed transactions.

**Recommendation:** Use `forceApprove()`.

**Sprinter:** Fixed in PR 66.

**Cantina Managed:** Fix verified.

### 3.1.2 `Across` transfer slippage check does not work if input token is not same as the output token

**Severity:** Low Risk

**Context:** Repayer.sol#L137-L139

**Description:** When triggering `Across` transfer, all the parameter is decoded from `extraData`.

```
function initiateTransferAcross(
    IERC20 token,
    uint256 amount,
    address destinationPool,
    Domain destinationDomain,
    bytes calldata extraData
) internal {
    token.forceApprove(address(ACROSS_SPOKE_POOL), amount);
    (
        address outputToken, // Can be set to 0x0 for automapping by solvers.
        uint256 outputAmount,
        address exclusiveRelayer,
        uint32 quoteTimestamp, // Validated in the spoke pool
        uint32 fillDeadline, // Validated in the spoke pool
        uint32 exclusivityDeadline
    ) = abi.decode(extraData, (address, uint256, address, uint32, uint32, uint32));
    require(outputAmount >= (amount * 9980 / 10000), SlippageTooHigh());
    ACROSS_SPOKE_POOL.depositV3(
        address(this),
        destinationPool,
        address(token),
        outputToken,
        amount,
        outputAmount,
        domainChainId(destinationDomain),
        exclusiveRelayer,
        quoteTimestamp,
        fillDeadline,
        exclusivityDeadline,
        "" // message
    );
}
```

Check the migration guide for more details. The code enforces a slippage check.

```
require(outputAmount >= (amount * 9980 / 10000), SlippageTooHigh());
```

This check works if the input token is the same as the output token. This check does not work for the two case below:

1. When the input token and output token has different decimals in source chain and destination chain. In etherscan, the USDT has 6 decimals. In binance smart chain, the USDT has 18 decimals, then the code cannot use input token amount to validate output token amount.

2. When the input token is not the same as output token. If user bridge 100 USDC in exchange for WETH in other blockchain.
   - USDC has 6 decimals.
   - WETH has 18 decimals.
   - 100 USDC = 10 * 10 ** 6 = 100000000 wei.

If the check is applied,

```
require(outputAmount >= (amount * 9980 / 10000), SlippageTooHigh());
```

the output WETH amount is 0.0000000001 ETH * 0.98, which is a dust of WETH.

**Recommendation:** Consider remove the check.

```
require(outputAmount >= (amount * 9980 / 10000), SlippageTooHigh());
```

**Sprinter:** Acknowledged.

**Cantina Managed:** Acknowledged.

#### 3.1.3 The `Shares` token decimal should match asset decimals

**Severity:** Low Risk

**Context:** LiquidityHub.sol#L230-L238

**Description:**

```
contract ManagedToken is IManagedToken, ERC20Permit
```

The `Shares` token is a ManagedToken, but the `Shares` token is always 18 decimals. However, the asset token and share token can have different decimials. The `Shares` token can be transferred or trade. Smart contracts or UIs expecting 18-decimal precision may perform math incorrectly (e.g., dividing/multiplying by 1e18 instead of 1e6), leading to miscalculations.

**Recommendation:** Consider make sure the `Shares` token decimal match asset decimals instead of hard-code the `ManagedToken` decimal to 18.

**Sprinter:** Acknowledged.

**Cantina Managed:** Acknowledged.

#### 3.1.4 CCTP v2 doesn't have `depositForBurnWithCaller()` function

**Severity:** Low Risk

**Context:** CCTPAdapter.sol#L31

**Finding Description:** `depositForBurnWithCaller()` function isn't present in CCTP v2 (only in v1). While, there is no indication that v1 is deprecated, it should be explicitly considered which CCTP version should be supported.

**Recommendation:** v1 and v2 has some differences listed in CCTP docs, and here is the list of supported domains in both versions: domains. Consider which version to support based on these differences.

**Sprinter:** Acknowledged, we work with V1 for now.

**Cantina Managed:** Acknowledged.

### 3.2 Gas Optimization

#### 3.2.1 `isIncrease` is checked twice

**Severity:** Gas Optimization

**Context:** LiquidityHub.sol#L99-L100

**Description:** The highlighted block can be refactored to check for `isIncrease` only once.

**Recommendation:**

```
uint256 newAssets
if (isIncrease) {
    require(amount <= _assetsIncreaseHardLimit(assets), AssetsExceedHardLimit());
    newAssets = assets + amount;
} else {
    newAssets = assets - amount;
}
```

**Sprinter:** Fixed in PR 66.

**Cantina Managed:** Fix verified.

## 3.3 Informational

### 3.3.1 Use `1e9` to represent 10^9

**Severity:** Informational

**Context:** LiquidityMining.sol#L21

**Description:** The contract uses the value `1000000000` to represent the precision of $10^9$. While this representation is functionally correct, it reduces readability.

**Recommendation:** Replace the large integer literal with scientific notation `1e9`.

**Sprinter:** Fixed in PR 66.

**Cantina Managed:** Fixed verified.

### 3.3.2 Inconsistent Use of `msg.sender` and `_msgSender()`

**Severity:** Informational

**Context:** *(No context files were provided by the reviewer)*

**Description:** The contract uses both `msg.sender` and `_msgSender()` inconsistently. While both return the caller currently , overriding `_msgSender()` will break this compatibility.

**Recommendation:** Use `msg.sender` or `_msgSender()` consistently.

**Sprinter:** Fixed in PR 66.

**Cantina Managed:** Fixed verified.

### 3.3.3 `multicall()`'s behavior differs from `SafeERC20.safeTransferFrom()`

**Severity:** Informational

**Context:** CensoredTransferFromMulticall.sol#L24

**Finding Description:** Behavior on different return values of `.transferFrom` here vs if you used `.safeTransferFrom`:

- `true` → succeeds vs succeeds.
- no return value → succeeds vs succeeds.
- `false` → succeeds vs reverts.

So it differs when the call returns `false` as the return value.

**Recommendation:** If compatibility with `SafeERC20`'s behavior is important, consider reverting when the `transferFrom()` call returns `false` It can happen with some weird ERC20 token.

**Sprinter:** Acknowledged.

**Cantina Managed:** Acknowledged.

### 3.3.4 Consider try catch the `IERC20Permit(asset()).permit` external call

**Severity:** Informational

**Context:** LiquidityHub.sol#L176-L194

**Description:**

```
IERC20Permit(asset()).permit(
    _msgSender(),
    address(this),
    assets,
    deadline,
    v,
    r,
    s
);
```

If another consume and frontrun the permit signature before `depositWithPermit`, the `depositWithPermit` will revert because the signature nonce is already consumed.

**Recommendation:** Consider try catch the `permit` external call.

**Sprinter:** Acknowledged.

**Cantina Managed:** Acknowledged.

### 3.3.5 `CCTPAdapter` needs to be deployed on same address on all domains

**Severity:** Informational

**Context:** CCTPAdapter.sol#L36

**Description:** Since the `destinationCaller` argument in `depositForBurnWithCaller()` is the address on the source domain, CCTP will only allow that address to process the transfer on the destination domain. This, it's essential to deploy this adapter on the same address on all domains.

**Recommendation:** If it's EVM chain, ensure that the adapter is deployed using CREATE2 and that CREATE2 generates the same address (while not known, EVM incompatibility may cause an issue).

**Sprinter:** Acknowledged.

**Cantina Managed:** Acknowledged.