CANTINA

Velodrome
Security Review

Cantina Managed review by:

0xLeastwood, Lead Security Researcher
Jonatas Martins, Associate Security Researcher

October 2, 2025

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 Gas Optimization . . . 4
  - 3.1.1 Use `VelodromeTimeLibrary` instead of direct calculation . . . 4
- 3.2 Informational . . . 4
  - 3.2.1 Time restriction consistency between `Swapper` and `Relays` . . . 4
  - 3.2.2 Use `Ownable2Step` instead of `Ownable` . . . 4
  - 3.2.3 Add comment for arbitrary `_commands` and `_inputs` parameters . . . 5
  - 3.2.4 Missing upper bound on slippage . . . 5
  - 3.2.5 Dutch style auctions improve swap efficiency and keeper role dependency . . . 5

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

Velodrome Finance is a next-generation AMM that combines the best of Curve, Convex and Uniswap, designed to serve as Optimism's central liquidity hub. Velodrome NFTs vote on token emissions and receive incentives and fees generated by the protocol.

From Sep 14th to Sep 16th the Cantina team conducted a review of relay-private on commit hash a7f8f335. The team identified a total of **6** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	0	0	0
Gas Optimizations	1	0	1
Informational	5	2	3
Total	**6**	**2**	**4**

3 Findings

3.1 Gas Optimization

3.1.1 Use `VelodromeTimeLibrary` instead of direct calculation

Severity: Gas Optimization

Context: Swapper.sol#L85

Description: (*Originally found by the Velodrome team*) The `Swapper` contract calculates the beginning of the epoch using:

```
uint256 _epoch = block.timestamp - (block.timestamp % WEEK);
```

This calculation can be simplified by using the `VelodromeTimeLibrary.epochStart()` function.

Recommendation: Replace the direct calculation with the library function:

```
-   uint256 private constant WEEK = 7 days;
  // ...

  function swapToTargetToken(...){
    // ...
    if (!_isAutoCompounder) {
-       uint256 _epoch = block.timestamp - (block.timestamp % WEEK);
-       amountTokenEarned[_relay][_epoch] += _amountOut;
+       amountTokenEarned[_relay][VelodromeTimeLibrary.epochStart(block.timestamp)] += _amountOut;
    }
    // ...
  }
```

Velodrome: The issue has been acknowledged and will not be fixed due to a mismatch in Solidity versions. The Swapper-related contracts are using a newer Solidity version than the library, so the epoch calculation logic was replicated in the Swapper contract to avoid version downgrades.

Cantina Managed: Acknowledged.

3.2 Informational

3.2.1 Time restriction consistency between `Swapper` and `Relays`

Severity: Informational

Context: Swapper.sol#L61

Description: The `AutoCompounder.swapTokenToVELOWithOptionalRoute()` and `AutoConverter.swapTokenToTokenWithOptionalRoute()` functions include time restrictions that prevent keepers from calling them before the second hour of an epoch begins.

However, the `Swapper` contract lacks this time restriction, allowing keepers to call the `swapToTargetToken` function at any time during an epoch. To maintain consistency between contracts, this time restriction should also be implemented in the `Swapper` contract.

Recommendation: Add the same time restriction validation to the `swapToTargetToken` function to ensure consistency with the Relays.

Velodrome: Issue has been addressed in commits:

- SC change + unit tests: 97371d2.
- Fuzz tests: c0ffee5.

Cantina Managed: Fix verified.

3.2.2 Use `Ownable2Step` instead of `Ownable`

Severity: Informational

Context: Swapper.sol#L24

Description: The `Swapper` contract use the standard `Ownable` implementation from OpenZeppelin, which allows the owner to transfer ownership directly to a new address in a single transaction. This introduces a risk where the ownership of the contract could be accidentally transferred to an incorrect address.

The `Ownable2Step` pattern provides a more secure ownership transfer mechanism that requires a two-step process whereby the owner initiates a transfer, and the new owner must accept it. This prevents accidental transfers to incorrect or inaccessible addresses.

Recommendation: Replace the current `Ownable` implementation with `Ownable2Step`:

```
-   import "@openzeppelin/contracts/access/Ownable.sol";
+   import "@openzeppelin/contracts/access/Ownable2Step.sol";

-   contract Swapper is Ownable {
+   contract Swapper is Ownable2Step {
    // ...
  }
```

Velodrome: The finding has been acknowledged and will not be fixed. The Swapper's owner will be verified prior to granting the Swapper the permissions required for upkeeping operations.

Cantina Managed: Acknowledged.

3.2.3 Add comment for arbitrary `_commands` and `_inputs` parameters

Severity: Informational

Context: ISwapper.sol#L26-L27

Description: The `swapToTargetToken` function accepts arbitrary `_commands` and `_inputs` parameters set by the Keeper, who has a trusted role. While the Keeper is responsible for providing correct values to ensure proper swap functionality, the code lacks to document that these parameters are arbitrary but trusted by the Keeper.

Recommendation: Add explicit documentation or comments clarifying that `_commands` and `_inputs` parameters are provided by the trusted Keeper role. This comment prevents any potential misunderstanding of the input parameters.

Velodrome: Addressed in commit c407268

Cantina Managed: Fix verified.

3.2.4 Missing upper bound on slippage

Severity: Informational

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

Description: The `Swapper` contract uses the keeper role to facilitate swaps to the relay's target token, using Uniswap's universal router interface to do this. This keeper role is trusted and operated initially by the Velodrome team. To avoid significant reliance on this role, including an upper bound on slippage can help protect relay contracts from being intentionally sandwiched due to compromise.

Recommendation: Ensure it is intended for the keeper role to not have any designated max slippage as per the `AutoCompounder` and `AutoConverter` implementations.

Velodrome: The finding has been acknowledged and will not be fixed. The Keeper infrastructure will be managed internally and slippage will be handled off-chain, to prevent excessive slippage while avoiding additional oracle checks on-chain.

Cantina Managed: Acknowledged.

3.2.5 Dutch style auctions improve swap efficiency and keeper role dependency

Severity: Informational

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

Description: The `Swapper` contract relies on a trusted keeper role to facilitate swaps from arbitrary tokens held by the relay to some target token. While most input variables are checked, the keeper can choose

which slippage to set and therefore is not enforced to optimally route the swap. In terms of design, there could be considerable improvement in efficiency and dependency on trusted roles by moving to an dutch auction style token sale.

By taking this path, at a given interval, the dutch auction will find the optimal conversion rate for which `_inputToken` → `_targetToken` without needing to manage slippage controls on-chain. There is one caveat in the fact that this does introduce latency but because rewards and other accruals are usually done on an epoch basis within Velodrome, this is not overtly concerning.

Recommendation: Consider making efforts to further decentralize the `Swapper` contract in order to remove it's dependency on permissioned roles.

Velodrome: The finding has been acknowledged and will not be fixed. The Keeper infrastructure for this project will be managed internally, so the same Keeper flow will be preserved in the Swapper and Relays for consistency.

Cantina Managed: Acknowledged.