Capricorn

Security Review

Solo review by:
Slowfi, Security Researcher

November 20, 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**
- 2.1 Scope . . . 3

3 Findings **5**
- 3.1 Gas Optimization . . . 5
  - 3.1.1 Redundant conditional on `cardinalityNext` during observation initialization . . . 5
- 3.2 Informational . . . 5
  - 3.2.1 Initial `IncreaseObservationCardinalityNext` emit does not indicate completed observation initialization . . . 5
  - 3.2.2 Semantically clearer use of `cardinalityNext` in `observations.grow` . . . 6

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

A security review is 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 the review 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 a 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

From Nov 15th to Nov 16th the security researchers conducted a review of capricorn-cl-core and capricorn-cl-periphery on commit hashes 65ee0b24 and 65ee0b24. A total of **3** issues were identified:

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	2	2	0
Total	**3**	**2**	**1**

2.1 Scope

The security review had the following components in scope for capricorn-cl-core and capricorn-cl-periphery on commit hashes 65ee0b24 and ff220a10:

```
├── capricorn-cl-core
│   └── contracts
│       ├── CapricornCLFactory.sol
│       └── CapricornCLPool.sol
└── capricorn-cl-periphery
    └── contracts
        ├── base
        │   ├── BlockTimestamp.sol
        │   ├── ERC721Permit.sol
        │   ├── LiquidityManagement.sol
        │   ├── Multicall.sol
        │   ├── PeripheryImmutableState.sol
        │   ├── PeripheryPayments.sol
        │   ├── PeripheryPaymentsWithFee.sol
        │   ├── PeripheryValidation.sol
        │   ├── PoolInitializer.sol
        │   └── SelfPermit.sol
        ├── examples
        │   └── PairFlash.sol
        ├── interfaces
        │   ├── external
        │   │   ├── IERC1271.sol
        │   │   ├── IERC20PermitAllowed.sol
        │   │   └── IWETH9.sol
        │   ├── IERC20Metadata.sol
        │   ├── IERC721Permit.sol
        │   ├── IMulticall.sol
        │   ├── INonfungiblePositionManager.sol
        │   ├── INonfungibleTokenPositionDescriptor.sol
        │   ├── IPeripheryImmutableState.sol
        │   ├── IPeripheryPayments.sol
        │   ├── IPeripheryPaymentsWithFee.sol
        │   ├── IPoolInitializer.sol
        │   ├── IQuoter.sol
        │   ├── IQuoterV2.sol
        │   ├── ISelfPermit.sol
        │   ├── ISwapRouter.sol
        │   ├── ITickLens.sol
```

```
│   └── IV3Migrator.sol
├── lens
│   ├── CapricornInterfaceMulticall.sol
│   ├── Quoter.sol
│   ├── QuoterV2.sol
│   ├── README.md
│   └── TickLens.sol
├── libraries
│   ├── BytesLib.sol
│   ├── CallbackValidation.sol
│   ├── ChainId.sol
│   ├── HexStrings.sol
│   ├── LiquidityAmounts.sol
│   ├── NFTDescriptor.sol
│   ├── NFTSVG.sol
│   ├── OracleLibrary.sol
│   ├── Path.sol
│   ├── PoolAddress.sol
│   ├── PoolTicksCounter.sol
│   ├── PositionKey.sol
│   ├── PositionValue.sol
│   ├── SqrtPriceMathPartial.sol
│   ├── TokenRatioSortOrder.sol
│   └── TransferHelper.sol
├── NonfungiblePositionManager.sol
├── NonfungibleTokenPositionDescriptor.sol
├── SwapRouter.sol
├── test
│   ├── Base64Test.sol
│   ├── LiquidityAmountsTest.sol
│   ├── MockObservable.sol
│   ├── MockObservations.sol
│   ├── MockTimeNonfungiblePositionManager.sol
│   ├── MockTimeSwapRouter.sol
│   ├── NFTDescriptorTest.sol
│   ├── NonfungiblePositionManagerPositionsGasTest.sol
│   ├── OracleTest.sol
│   ├── PathTest.sol
│   ├── PeripheryImmutableStateTest.sol
│   ├── PoolAddressTest.sol
│   ├── PoolTicksCounterTest.sol
│   ├── PositionValueTest.sol
│   ├── SelfPermitTest.sol
│   ├── TestCallbackValidation.sol
│   ├── TestCapricornCLCallee.sol
│   ├── TestERC20.sol
│   ├── TestERC20Metadata.sol
│   ├── TestERC20PermitAllowed.sol
│   ├── TestMulticall.sol
│   ├── TestPositionNFTOwner.sol
│   └── TickLensTest.sol
└── V3Migrator.sol
```

3 Findings

3.1 Gas Optimization

3.1.1 Redundant conditional on `cardinalityNext` during observation initialization

Severity: Gas Optimization

Context: CapricornCLPool.sol#L282

Description: The function `initialize` from contract `CapricornCLPool` contains a conditional around the `IncreaseObservationCardinalityNext` event:

```
if (cardinalityNext != INITIAL_OBSERVATION_CARDINALITY) {
    emit IncreaseObservationCardinalityNext(cardinalityNext,
    ↪  INITIAL_OBSERVATION_CARDINALITY);
}
```

Given the current initialization flow, `observations.initialize` sets both `cardinality` and `cardinalityNext` to `1`, while `INITIAL_OBSERVATION_CARDINALITY` is set to `3`. Under these parameters, the condition `cardinalityNext != INITIAL_OBSERVATION_CARDINALITY` is always satisfied at initialization, so the `if` statement does not change the behavior of the function.

This pattern does not introduce a functional issue but adds a branch that is redundant for the current configuration and slightly reduces readability by suggesting configuration-dependent behavior that does not occur in practice.

Recommendation: Consider to:

- Remove the conditional and emit `IncreaseObservationCardinalityNext` unconditionally during initialization, or...
- Explicitly document that the conditional is only relevant for alternative configurations where `INITIAL_OBSERVATION_CARDINALITY` might equal the initial `cardinalityNext`.

Capricorn: This was done to be rigorous, since `INITIAL_OBSERVATION_CARDINALITY` was used as a configuration file and could be 1. It is possible to remove this in this case that `INITIAL_OBSERVATION_CARDINALITY=3`, but the team prefers to keep this here for safety/ease of reasoning, especially since we don't expect this to meaningfully affect gas.

Slowfi: Acknowledged by Capricorn team.

3.2 Informational

3.2.1 Initial `IncreaseObservationCardinalityNext` emit does not indicate completed observation initialization

Severity: Informational

Context: CapricornCLPool.sol#L283

Description: The pool initialization logic from contract `CapricornCLPool` emits the `IncreaseObservationCardinalityNext` event with `INITIAL_OBSERVATION_CARDINALITY` (around line 283):

```
emit IncreaseObservationCardinalityNext(cardinalityNext,
↪  INITIAL_OBSERVATION_CARDINALITY);
```

However, at this point the observation history is still of length 1, because no in-range liquidity update or price movement has yet triggered `observations.write`. As a result, off-chain tooling that relies on this event as a signal that the TWAP observation buffer is "ready" may read and use TWAPs that are effectively based on a single observation.

This behavior does not affect on-chain correctness, but it can cause off-chain indexers, oracles, or analytics systems that assume the first `IncreaseObservationCardinalityNext` event only occurs after an explicit `increaseObservationCardinalityNext` call (and after multiple observations exist) to misinterpret the state of the pool's observation history.

Recommendation: Consider to make the semantics explicit for integrators by one or more of the following:

- Consider to document that the initial `IncreaseObservationCardinalityNext` emit during pool initialization does not imply that the observation buffer already contains multiple timestamp-separated observations, and that off-chain components should wait for at least two distinct writes before trusting TWAP data.
- Consider to add an inline comment near the emit clarifying that observation history remains length-1 until the first in-range liquidity update or price change triggers `observations.write`.

Capricorn Exchange: Fixed in `capricorn-cl-core` PR 1.

Slowfi: Fix verified.

3.2.2 Semantically clearer use of `cardinalityNext` in `observations.grow`

Severity: Informational

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

Description: Found by Capricorn team.

The function `initialize` from contract `CapricornCLPool` pre-allocates oracle slots by calling `observations.grow`. The previous version used:

```
// Pre-allocate oracle slots so the pool creator pays
observations.grow(cardinality, INITIAL_OBSERVATION_CARDINALITY);
```

and this was updated to:

```
// Pre-grow oracle slots.
observations.grow(cardinalityNext, INITIAL_OBSERVATION_CARDINALITY);
```

Immediately after `observations.initialize`, both `cardinality` and `cardinalityNext` hold the same value, so the two calls are functionally equivalent in this context. The change to `cardinalityNext` aligns better with the intended semantics, since subsequent logic and events (such as `IncreaseObservationCardinalityNext`) conceptually work with `cardinalityNext` as the target observation capacity.

This is not a functional or security issue, but a small readability and consistency improvement between the variable used in `observations.grow` and the rest of the observation configuration logic.

Recommendation: Consider to:

- Keep using `cardinalityNext` in the `observations.grow` call at initialization, as it better reflects the intended target cardinality while preserving the same behavior as the previous implementation.

Capricorn: Fixed in `capricorn-cl-core` PR 1.

Slowfi: Fix verified.