OpenZeppelin | security

Uniswap Continuous Clearing Auction Audit

January 28, 2026

Table of Contents

Table of Contents 2

Summary 3

Scope 4

System Overview 5

Changes 5

Security Model and Privileged Roles 6

Low Severity 7

L-01 Gated ERC-1155 Validation Hook Does Not Utilize BlockNumberish 7

L-02 Missing Docstrings or Incomplete Docstrings 7

Notes & Additional Information 8

N-01 Constructor Call of BlockNumberish Can Be Removed 8

N-02 Unused Imports 8

Conclusion 9

Summary

Type	DeFi
Timeline	From 2025-12-22 To 2026-01-14
Languages	Solidity

Total Issues	4 (3 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	2 (1 resolved, 1 partially resolved)
Notes & Additional Information	2 (2 resolved)

Scope

OpenZeppelin performed an audit of the Uniswap/continuous-clearing-auction-internal repository, specifically the changes between commits 64f5212 and af19ecf.

In scope were the following files:

```
contracts
└── src
    ├── ContinuousClearingAuction.sol
    ├── StepStorage.sol
    ├── TokenCurrencyStorage.sol
    │
    ├── interfaces
    │   ├── IContinuousClearingAuction.sol
    │   ├── IStepStorage.sol
    │   └── ITokenCurrencyStorage.sol
    │
    └── external
        └── ILBPInitializer.sol
```

System Overview

The Uniswap Continuous Clearing Auction facilitates the launch of new tokens through a continuous clearing mechanism. It establishes a fair clearing price through continuous bidding and scheduled token releases. The central `Auction` contract may be deployed independently or through the `AuctionFactory`, which initializes parameters such as pricing rules, auction length, issuance phases, and validation procedures. The auction commences at a designated starting block and concludes at a specified ending block, with a set amount of tokens released per block.

Bidders place orders by specifying their maximum acceptable price. These bids align to discrete price ticks based on configurable tick spacing. Custom validation hooks can enforce additional constraints, such as whitelists or limits, prior to bid acceptance. Throughout the process, checkpoints capture the auction's state, including the clearing price, cumulative tokens distributed, and total funds collected. The clearing price represents the point where demand aligns with available supply, and it increases as new bids are received.

Bids classify as fully or partially filled depending on their relation to the current clearing price. Upon auction completion, participants can withdraw their bids to retrieve any unused funds or claim acquired tokens, with partial fills utilizing checkpoint data for proportional allocation. Following the auction, the system assesses whether it has achieved graduation by meeting a fundraising threshold. Successful auctions enable the withdrawal of raised currency and the recovery of unsold tokens. Unsuccessful auctions result in full refunds to bidders and the return of all tokens to the recipient. At the designated claim block, buyers may retrieve their allocated tokens to finalize distribution.

Changes

The `forceIterateOverTicks` function has been introduced, which allows for any user to iterate over ticks to reach an intermediate price so that a large amount of ticks can be crossed over multiple transactions. This function was added to circumvent a potential out-of-gas issue if too many ticks need to be crossed by to create the next checkpoint. The iteration over ticks does not change the final clearing price of the price, the amount the users have to bid, or how much the users have to pay for the tokens.

The `Blocknumberish` library was introduced and integrated into the continuous clearing auction to be compatible with layer 2 chains such as Arbitrum where `block.number` is not the L2 block number but rather the block number of the base chain.

Security Model and Privileged Roles

Throughout the changes to the in-scope codebase, the following assumptions were identified:

- The `forceIterateTicks` does not create any new checkpoints. Instead, the `checkpoint` function should be used for this purpose.
- The `newClearingPrice` after iterating over ticks may not correspond to the current correct price. It is only when a new checkpoint is created that the price is correctly updated.

Low Severity

L-01 Gated ERC-1155 Validation Hook Does Not Utilize `BlockNumberish`

The `validate` function of the `GatedERC1155ValidationHook` contract requires the `owner` and the `sender` of the bid hold at least one of the ERC-1155 token, and it is enforced until the `gateUntil` block number. However, the `gateUntil` block number is checked against `block.number` instead of utilizing `_getBlockNumberish()` function to fetch correct block number for Arbitrum One chain. This could be lead to inconsistent behavior of the validation hook on the Arbitrum One chain.

Consider utilizing the `_getBlockNumberish()` function to ensure that the hook is consistent across all the chains.

Update: *Resolved in pull request #12 at commit eb7c80b.*

L-02 Missing Docstrings or Incomplete Docstrings

Throughout the codebase, multiple instances of missing docstrings were identified:

- In `ContinuousClearingAuction.sol`, the `forceIterateOverTicks` function, not all return values are documented.
- In `ICheckpointStorage.sol`, the `checkpoints` function, not all return values are documented.
- In `IContinuousClearingAuction.sol`, return values of multiple functions are not documented.
- In `IStepStorage.sol`, the `step` function, not all return values are documented.

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Partially Resolved in pull request #115 at commit 787ae39.*

Notes & Additional Information

N-01 Constructor Call of `BlockNumberish` Can Be Removed

In the constructor of the `ContinuousClearingAuction` contract, the `BlockNumberish()` constructor is called. However, this call is not necessary and can be removed. This is because the `BlockNumberish` contract does not have any constructor and the child contract do not need to explicitly call the parent contract's constructor if it does not require any arguments.

Consider removing constructor call of `BlockNumberish` to improve code quality and avoid confusion.

Update: *Resolved in pull request #17 at commit 4f0b6dd.*

N-02 Unused Imports

Within `CheckpointStorage.sol`, multiple instances of unused imports were identified:

- The import `import {Bid, BidLib} from './libraries/BidLib.sol';`.
- The import `import {Checkpoint, CheckpointLib} from './libraries/CheckpointLib.sol';`.
- The import `import {FixedPoint96} from './libraries/FixedPoint96.sol';`.
- The import `import {ValueX7, ValueX7Lib} from './libraries/ValueX7Lib.sol';`.

Consider removing unused imports to improve the overall clarity and readability of the codebase.

Update: *Resolved in pull request #17 at commit 125b163.*

Conclusion

The audited changes to the Continuous Clearing Auction mitigate potential out-of-gas errors that could arise if a checkpoint-adding function attempts to iterate over too many ticks. Most notably, a new function has been introduced that allows any user to choose the tick at which the iteration ends.

No medium- or higher-severity vulnerabilities were found. However, some low- and note-severity issues were identified that focus on small code improvements, additional checks, and documentation inaccuracies. The Uniswap Labs team is appreciated for providing a detailed walkthrough of the codebase and promptly answering any questions the audit team had.

The Uniswap Labs team is appreciated for providing a detailed walkthrough of the codebase and promptly answering any questions the audit team had about the codebase.