Ekubo Auctions Audit Report

Prepared by Riley Holterhus

February 18, 2026

Contents

1 Introduction

1.1 About Ekubo

Ekubo is a high-performance, gas-optimized AMM. It supports several configuration options for pools, including choices between concentrated and stableswap liquidity, hookable extensions, and parameters such as fees and tick spacing. Ekubo is deployed on both Starknet and EVM-based chains.

1.2 About the Auditor

Riley Holterhus is an independent security researcher who focuses on Solidity smart contracts. Other than conducting independent security reviews, he works as a Lead Security Researcher at Spearbit, and also searches for vulnerabilities in live codebases. Riley can be reached by email at rileyholterhus@gmail.com, by Telegram at @holterhus and on Twitter/X at @rileyholterhus.

1.3 Disclaimer

This report is intended to detail the identified vulnerabilities of the reviewed smart contracts and should not be construed as an endorsement or recommendation of any project, individual or entity. While the authors have made reasonable efforts to detect potential issues, the absence of any undetected vulnerabilities or issues cannot be guaranteed. Additionally, the security of the smart contracts may be affected by future changes or updates. By using the information in this report, you acknowledge that you are doing so at your own risk and that you should exercise your own judgment when implementing any recommendations or making decisions based on the findings. This report has been provided on an “as-is” basis and DOES NOT CONSTITUTE A GUARANTEE OR WARRANTY OF ANY FORM.

2 Audit Overview

2.1 Scope of Work

On February 16, 2026, Riley Holterhus conducted a manual security review of a subset of Ekubo's `evm-contracts`. The goal was to identify potential vulnerabilities and logic issues in the reviewed components.

The audit was performed on the codebase found in the `EkuboProtocol/evm-contracts` GitHub repository. The audit started on commit 1858186. Only a portion of the codebase was considered in scope, specifically `Auctions.sol` along with `auctionKey.sol` and `auctionConfig.sol`. These contracts utilize the existing `TWAMM` and `BoostedFees` extensions to allow anyone to create token auctions that sell a token over time via TWAMM and, upon completion, split the sale proceeds between the auction creator and the "graduation pool" - a `BoostedFees` pool that receives its share of the proceeds as liquidity incentives.

2.2 Summary of Findings

Each finding from the audit has been assigned a severity level of "Critical", "High", "Medium", "Low" or "Informational". These severities aim to capture the impact and likelihood of each potential issue.

In total, **3 findings** were identified, all being informational.

3 Findings

3.1 Informational Findings

3.1.1 `startBoost()` does not auto-initialize the pool unlike `sellByAuction()`

Description: The `CALL_TYPE_SELL_BY_AUCTION` logic auto-initializes the TWAMM launch pool if it hasn't been initialized already. The `CALL_TYPE_START_BOOST` logic doesn't have similar logic for the BoostedFees graduation pool. If the BoostedFees graduation pool is not already initialized, the `addIncentives()` call will ultimately revert. This isn't a big concern since the call simply reverts, and anyone can permissionlessly initialize the pool, but it is a UX asymmetry.

Recommendation: Consider adding auto-initialization logic in the `CALL_TYPE_START_BOOST` flow, similar to the pattern used in `CALL_TYPE_SELL_BY_AUCTION`.

Ekubo: Addressed in commit 258e1fd and commit dec2f02.

Auditor: Verified. The auto-initialize logic has been removed from the `CALL_TYPE_SELL_BY_AUCTION` logic in favor of separate `maybeInitializePool()` functions, similar to the `BasePositions` contract.

3.1.2 Clamped boost rate can still revert due to pre-existing incentives

Description: The `CALL_TYPE_START_BOOST` logic in the `Auctions` contract clamps `boostRate` to `MAX_ABS_VALUE_SALE_RATE_DELTA`:

```
boostRate = uint112(
    FixedPointMathLib.min((uint256(boostAmount) << 32) / duration, MAX_ABS_VALUE_SALE_RATE_DELTA)
);
```

`MAX_ABS_VALUE_SALE_RATE_DELTA` is the maximum rate change that can be associated with a given timestamp. However, note that if prior `startBoost()` calls or external users have already added incentives to the same pool with the same end timestamp, any nonzero existing rate at that timestamp reduces the remaining room that can be added, and adding `MAX_ABS_VALUE_SALE_RATE_DELTA` on top could still exceed the limit.

This isn't an issue in practice, since it's always possible to retry these failed calls with a smaller amount, or potentially wait for a different timestamp.

Recommendation: This finding has been provided for informational purposes only. No changes are needed.

Ekubo: Acknowledged. Note that the `MAX_ABS_VALUE_SALE_RATE_DELTA` clamping also doubles as a safeguard against `uint112` casting overflow, so simply removing it would require further changes.

Auditor: Acknowledged.

3.1.3 Boost duration is always strictly greater than `minBoostDuration`

Description: In the `Auctions` contract, the `CALL_TYPE_START_BOOST` handler computes the boost end time as follows:

```
uint256 minBoostEndTime = block.timestamp + auctionKey.config.minBoostDuration();
uint256 boostEndTimeRaw = nextValidTime({currentTime:block.timestamp, afterTime:minBoostEndTime});
```

Since `nextValidTime()` returns the next valid TWAMM timestamp strictly after `afterTime`, the actual boost duration is always strictly longer than `minBoostDuration`, even when `minBoostEndTime` is itself a valid time. This may not be obvious from the name: `minBoostDuration` is not an achievable minimum, but rather a strict lower bound that the actual boost duration will always exceed.

Recommendation: This finding has been provided for informational purposes only. No changes are needed.

Ekubo: Acknowledged.

Auditor: Acknowledged.