By
Quit

Contract Review
Issue date
1/28/2026

Overview

The following is a review of the TLRankedAuction contract, a fully on-chain ranked auction system supporting 2-512 ERC-721 tokens with automatic price discovery via a sorted doubly linked list.

Contracts in scope for this review include:

- `src/TLRankedAuction.sol`

Test contracts and mocks were reviewed to understand expected behavior but are not in scope.

This review is based on SHA `13af92d70952bb154dc466d8a19e93d465be8a53`, and aims to identify security vulnerabilities, opportunities for gas optimization, and general best-practice recommendations with regard to the contracts in scope. The review should not be considered an endorsement of the project, nor is it a guarantee of security.

Remediations were reviewed as of SHA `3850b068d083b731c00bff36789f8781d36876c5` and each findings' status was updated accordingly.

Findings

I-01: No auction cancellation mechanism after setup

Severity: Informational

Status: Resolved (`3850b068d083b731c00bff36789f8781d36876c5`)

Once `setupAuction()` transitions the contract to LIVE state, there is no mechanism for the owner to cancel or abort the auction. If configuration errors are discovered, the auction still must run to completion. Recommend considering a `cancelAuction()` function callable only before any bids are placed, which would transition back to `CONFIGURING` and allow prize token withdrawal.

I-02: Duplicate prize token entries possible when `tokenOwner` is the contract itself

Severity: Informational

Status: Resolved (`3850b068d083b731c00bff36789f8781d36876c5`)

`depositPrizeTokens()` accepts an arbitrary `tokenOwner` and does not prevent `tokenOwner == address(this)`. If the owner passes the contract itself as the token owner, the call can include token IDs already escrowed in the contract. Most ERC721 implementations allow self transfers, which would re-append the token ID to `prizeTokenIds`. This can result in duplicate token IDs in the prize list, causing later claims or withdrawals to

revert when the contract no longer owns the token. Recommend disallowing `tokenOwner == address(this)` in `depositPrizeTokens()` to prevent accidental or intentional duplication.

G-01: Unnecessary SafeCast usage
Severity: Gas Optimization
Status: Resolved (`3850b068d083b731c00bff36789f8781d36876c5`)
SafeCast is used in five locations where overflow is impossible given real-world constraints:

- L161: `SafeCast.toUint128(msg.value)`: uint128 max exceeds total ETH supply
- L204: `SafeCast.toUint128(uint256(bidAmount) + msg.value)`: once again, uint128 max fits this comfortably
- L250: `SafeCast.toUint64(newEnd - uint256(openAt))`: Duration bounded by `hardEndAt`. uint64 is not at risk of over or underflow.
- L755: `SafeCast.toUint128(tailBid + tailBid * CREATE_BID_BPS / BASIS)`: Result is 1.025 * tailBid; tailBid bounded by ETH supply
- L761: `SafeCast.toUint128(uint256(bidAmount) * INCREASE_BID_BPS / BASIS)`: Result is 0.5% of bidAmount, always smaller than the uint128 input

Recommend replacing SafeCast calls with direct casts to save gas.

G-02: Arithmetic operations safe for unchecked blocks
Severity: Gas Optimization
Status: Resolved (`3850b068d083b731c00bff36789f8781d36876c5`)
Several arithmetic operations can safely be made unchecked.

L166: `nextBidId++`: uint32 overflow requires billions of bids at minimum `START_BID` ETH
L426: `nextRank - 1`: nextRank initialized to 1, only incremented; always ≥1
L443: `nextRank++`: Bounded by `listSize ≤ NUM_TOKENS ≤ 512`
L507: `nextUnallocatedRank - 1`: Initialized to `listSize+1 ≥ 1`, only incremented
L508: `NUM_TOKENS - alreadyProcessed`: Function guards ensure `alreadyProcessed ≤ NUM_TOKENS`
L516: `nextUnallocatedRank++`: Bounded by `NUM_TOKENS ≤ 512`
L413: `uint256(listSize) * uint256(clearingPrice)`: 512 * ETH supply << uint256 max
L479, L517, L740: `rank - 1`: rank validated ≥ 1 before subtraction
L480: `storedBid.amount - clearingPrice`: Ranked bids always ≥ `clearingPrice` by auction logic
L536: `pendingProceeds -= amountToWithdraw`: `amountToWithdraw` capped to `pendingProceeds` on line 532

Recommend wrapping these operations in unchecked blocks to save gas per occurrence.

Summary

The `TLRankedAuction` contract is well constructed with careful attention to edge cases and security. The doubly linked list implementation correctly handles all boundary conditions including single-element lists, head/tail operations, and concurrent bid manipulation. CEI is properly followed and fail-safe mechanisms (`pendingRefunds`, `pendingNfts`) are in place to handle failed transfers.

It should be noted that thanks to proper CEI adherence, `nonReentrant` may not be necessary. Testing demonstrated that without the guard, a malicious bidder contract receiving a tail refund could atomically call `increaseBid()` mid-transaction, using the refunded ETH to boost another of their bids. While this doesn't corrupt state or steal funds, it enables an unintended "free" bid increase. If this is acceptable, then `ReentrancyGuard` and the `nonReentrant` modifier can be removed. *Note from team: “I decided to leave nonReentrant modifiers on there just as a belt + suspenders”*

All findings are informational, low severity, or gas optimizations. The contract is suitable for production deployment with the understanding that users trust the owner not to self-deal, the NFT contract must be legitimate, and final-second sniping is possible after the 2-hour extension cap.