OpenZeppelin | security

Uniswap TWAP Auction Audit

November 13, 2025

Table of Contents

Summary

Type	DeFi
Timeline	From 2025-10-20 To 2025-10-31
Languages	Solidity

Total Issues	32 (18 resolved, 3 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	11 (6 resolved, 2 partially resolved)
Notes & Additional Information	20 (11 resolved, 1 partially resolved)

Scope

OpenZeppelin audited the Uniswap/twap-auction repository at commit 93c0c78.

In scope were the following files:

```
src
├── Auction.sol
├── AuctionFactory.sol
├── AuctionStepStorage.sol
├── BidStorage.sol
├── CheckpointStorage.sol
├── PermitSingleForwarder.sol
├── TickStorage.sol
├── TokenCurrencyStorage.sol
├── interfaces
│   ├── IAuction.sol
│   ├── IAuctionFactory.sol
│   ├── IAuctionStepStorage.sol
│   ├── IBidStorage.sol
│   ├── ICheckpointStorage.sol
│   ├── IPermitSingleForwarder.sol
│   ├── ITickStorage.sol
│   ├── ITokenCurrencyStorage.sol
│   ├── IValidationHook.sol
│   └── external
│       ├── IDistributionContract.sol
│       ├── IDistributionStrategy.sol
│       └── IERC20Minimal.sol
└── libraries
    ├── AuctionStepLib.sol
    ├── BidLib.sol
    ├── CheckpointLib.sol
    ├── ConstantsLib.sol
    ├── CurrencyLibrary.sol
    ├── FixedPoint128.sol
    ├── FixedPoint96.sol
    ├── ValidationHookLib.sol
    └── ValueX7Lib.sol
```

After the first phase of the fix review, the protocol was modified, and the resulting changes were audited as part of the `v1.0.0-candidate` with a final commit `154fd18`. All resolutions and the final state of the audited codebase mentioned in this report are contained at that commit.

System Overview

The Uniswap TWAP Auction protocol has been designed to launch and distribute new tokens using a time-weighted average price (TWAP) mechanism. Its primary goal is to determine a fair clearing price for a token sale based on continuous bidding and supply issuance over time.

Architecture and Core Components

At a high level, the system operates as a modular, configurable auction platform that allows token projects to raise funds in a chosen currency (such as ETH or an ERC-20 token) while gradually releasing their token supply according to a predefined schedule. The core component of the system is the `Auction` contract, which can be deployed directly or through the `AuctionFactory`. The factory handles initialization and configuration by decoding parameters such as pricing rules, auction duration, supply release schedule, and validation logic.

Auction Lifecycle and Supply Issuance

The auction begins at a specified start block and proceeds until an end block, following a step-based issuance schedule that defines how many tokens are sold per block. Each step is encoded compactly in bytes to minimize on-chain storage costs and is read efficiently from external storage. The auction price is represented using Q96 fixed-point arithmetic, which ensures high-precision price calculations and prevents rounding errors in critical financial operations.

Bidding Mechanism

Participants join the auction by submitting bids. Each bid specifies the maximum price the bidder is willing to pay, expressed in Q96 format, along with the amount of currency they want to contribute. Bids are stored according to discrete price ticks determined by a configurable tick spacing, ensuring efficient price iteration and simplifying demand aggregation. Optional validation hooks can be configured to enforce custom bidding constraints such as allowlists or rate limits before bids are accepted.

Checkpoint Creation and Clearing-Price Discovery

Throughout the auction, the system continuously computes checkpoints, which are snapshots of the auction's state that record metrics such as the clearing price, cumulative tokens sold, and total currency raised. The clearing price represents the marginal price at which total demand equals the token supply being sold at that point. This price can only increase over time, ensuring monotonic price discovery as new bids arrive.

Bid Settlement and Exit Process

As the auction progresses, bids are dynamically categorized into fully filled or partially filled groups depending on whether their maximum price remains above or equal to the current clearing price. When the auction concludes, users can exit their bids to reclaim any unspent funds or claim the tokens they purchased. Fully filled bids can be exited directly, while partially filled bids use checkpoint hints to efficiently compute their pro-rata allocation.

Graduation and Fund Management

After the auction ends, the system determines whether it has graduated (i.e., a minimum amount of currency has been successfully raised). Graduated auctions allow the fund recipient to withdraw the raised currency using the `sweepCurrency` function, while unsold tokens can be recovered through `sweepUnsoldTokens`. If the auction fails to graduate, all bids are refunded, and all unsold tokens are returned to the designated recipient.

Token Claiming and Finalization

Once the claim block is reached, participants can claim their purchased tokens, finalizing the distribution. All accounting is performed in terms of currency instead of tokens, improving precision and reducing computation costs throughout the process.

Security Model and Trust Assumption

During the audit, the following trust assumptions were identified:

- The final clearing price of the auction is highly susceptible to manipulation near the end and should be used with extreme caution in any context which assumes that it represents the fair market price.
- The `Auction` contract relies on Permit2 to transfer currency from the user to itself. The Permit2 contract is trusted to behave according to its specification.
- During auction creation, the funds and tokens recipients parameters must be set correctly, as any raised funds or unsold tokens could otherwise be lost.
- The validation hook is used to validate bids, and if it does not function correctly, it could disrupt the contract's logic and prevent bidding. Therefore, it should be configured carefully to ensure it works correctly.

Privileged Roles

The protocol does not implement any roles that have access to privileged functionality.

Medium Severity

M-01 Incorrect Value Overwrite In `AuctionFactory`

The `initializeDistribution` function of the `AuctionFactory` contract overwrites certain auction parameters with `msg.sender` if those parameters have been set as special constants. In the `getAuctionAddress` `view` function, the same parameters are overwritten again with `msg.sender`. However, in the context of the `getAuctionAddress` function, the caller of the initializer is specified with the `sender` parameter, and the caller of this `view` function is irrelevant. This discrepancy results in the derivation of the wrong contract address in the `getAuctionAddress` function when an overwrite is performed.

In the `getAuctionAddress` function, consider replacing all instances of `msg.sender` with `sender`.

Update: *Resolved in pull request #219.*

Low Severity

L-01 Excess Tokens Sent to Contract Become Locked

The `onTokensReceived` function verifies whether enough tokens were received by the `Auction` contract. However, the issue is that it is possible to send more tokens than the configured `TOTAL_SUPPLY`, which leads to a loss of surplus tokens as they become locked in the contract. The `sweepUnsoldTokens` function only allows sweeping up `TOTAL_SUPPLY`.

Consider adding logic to the `sweepUnsoldTokens` function to allow sweeping surplus tokens from the contract.

Update: *Acknowledged, will resolve. The Uniswap team stated:*

> *Acknowledged, we are tracking this as a documentation task from the last audit.*

L-02 Missing Docstrings

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

- In `CheckpointStorage.sol`, the `CheckpointStorage` abstract contract
- In `IBidStorage.sol`, the `IBidStorage` interface

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: *Resolved in pull request #260.*

L-03 Incomplete Docstrings

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

- In `IAuction.sol`, the `blockNumber` parameter of the `ClearingPriceUpdated` event is not documented.
- In `IAuction.sol`, not all return values of the `claimBlock` function are documented.
- In `IAuction.sol`, not all return values of the `validationHook` function are documented.
- In `IAuction.sol`, not all return values of the `currencyRaisedQ96_X7` function are documented.
- In `IAuction.sol`, not all return values of the `sumCurrencyDemandAboveClearingQ96` function are documented.
- In `IAuctionStepStorage.sol`, not all return values of the `step` function are documented.
- In `IBidStorage.sol`, not all return values of the `nextBidId` function are documented.
- In `IBidStorage.sol`, the `bidId` parameter and not all return values of the `bids` function are documented.
- In `ICheckpointStorage.sol`, the `blockNumber` parameter and not all return values of the `checkpoints` function are documented.
- In `ITickStorage.sol`, the `price` parameter and not all return values of the `ticks` function are documented.
- In `ITokenCurrencyStorage.sol`, not all return values of the `currency` function are documented.

- In `ITokenCurrencyStorage.sol`, not all return values of the `token` function are documented.
- In `ITokenCurrencyStorage.sol`, not all return values of the `totalSupply` function are documented.
- In `ITokenCurrencyStorage.sol`, not all return values of the `tokensRecipient` function are documented.
- In `ITokenCurrencyStorage.sol`, not all return values of the `fundsRecipient` function are documented.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Partially resolved in pull request #260. The Uniswap team stated:*

> *Fixed some, lean against adding unecessary return comments for getters in the interface.*

L-04 Lack of Multicall Feature

In the `IPermitSingleForwarder` interface, the comments justify the `payable` modifier on the `permit` function as a necessity for multicall operations. However, the `Auction` contract lacks any multicall functionality. Such functionality would be convenient for submitting or exiting multiple bids at once, or when permit-approving and submitting a bid.

Consider adding multicall functionality to the `Auction` contract. If no multicall support is intended, either consider updating the comments and removing the `payable` modifier from the `permit` function or completely getting rid of the permit forwarder contract.

Update: *Resolved in pull request #259. The Uniswap team stated:*

> We will remove the contract since we don't have multicall._

L-05 Inefficient Token Purchase Due to Remainder Loss in Bid Submission

When submitting a bid, the user specifies the amount of currency to be used for purchasing tokens. The remainder obtained by dividing the submitted amount by the clearing price is considered spent, but no tokens are received for it.

The following example illustrates a scenario in which the user overpays for tokens:

1. The clearing price is 1000.
2. The user submits a bid with an amount of 1999.
3. The user receives 1 token.
4. The user loses 999 units of currency, even though they could have received the same number of tokens by submitting 1000 instead.

This behavior may seem negligible in most cases, but it could become more significant when different tokens are used as the payment currency, or when the clearing price is high relative to the currency's precision. For example, when using tokens such as Gemini USD (which only supports two decimal places), this could lead to an actual loss of value in certain edge cases.

Consider either redesigning the logic so that users only pay the exact amount for the tokens and the remainder is refunded, or clearly documenting this behavior.

Update: *Acknowledged, will resolve. The Uniswap team stated:*

> *We believe that the best course of action would be to document this behavior, especially the risks around low prices + low decimal tokens.*

L-06 Unsafe ABI Encoding

The `ValidationHookLib` contract uses `abi.encodeWithSelector` which is not type-safe. This is error-prone and should be considered unsafe.

Consider replacing the use of unsafe ABI encoding with `abi.encodeCall` which checks whether the supplied values actually match the types expected by the called function and also avoids errors caused by typos.

Update: *Resolved at commit 66cde50. The Uniswap team stated:*

> *Fixed, a different issue recommended we do try-catch.*

L-07 Setting `lastFullyFilledCheckpointBlock` as the Last Checkpoint Always Reverts During Partial Exit

In the `exitPartiallyFilledBid` function of the `Auction` contract, the `lastFullyFilledCheckpointBlock` parameter allows hinting the contract to the correct

checkpoint to prevent looping through all the checkpoints. The `lastFullyFilledCheckpointBlock` cannot be the last checkpoint block, because in that case, the `exitBid` function should be used instead. However, the code inexplicably uses special logic to overwrite the `lastFullyFilledCheckpointBlock` to the last checkpoint if it is equal to the current block number. This logic is unnecessary and the overwrite flow will always lead to a reversion.

Consider updating the logic to not overwrite the `lastFullyFilledCheckpointBlock` parameter.

Update: *Resolved in pull request #237.*

L-08 Auction's Final Clearing Price Is Simple to Manipulate

The auction exposes the clearing price of its last checkpoint in the `clearingPrice` function of the `CheckpointStorage` contract. However, this value is only suitable for determining the maximum price to use when bidding. Any other usage of the function, especially after the conclusion of the auction, is highly risky as the value can be easily manipulated.

The clearing price at each checkpoint is determined based on the demand for the *remaining* token supply. The remaining supply gets lower and lower as the auction proceeds. As the remaining supply gets lower, it becomes cheaper to manipulate the clearing price. The following extreme scenario demonstrates the manipulation:

- **Auction steps**:
    1. 99.99999% supply at block 1
    2. 0.00001% supply at block 2
- **Current clearing price**: 1 (`2**96` in Q96)
- **Total supply**: 10 million
- **Currency demand**: 10 million
- **Current block/step**: 2
- **Attacker bid**: 10 (ten) currency at max price of 10 (`10*2**96` in Q96)

In the above scenario, the final clearing price will increase tenfold with a currency bid amount of just 10.

Consider thoroughly documenting the risks of depending on the final `clearingPrice`. If possible, never use `clearingPrice` for any purpose other than determining the maximum price when bidding.

Update: *Acknowledged, will resolve. The Uniswap team stated:*

> *We will document that the clearing price is subject to manipulation particularly for small supply values and towards the end of the auction.*

L-09 Incorrect Check in `MaxBidPriceLib` Library

Within the `maxBidPrice` function of the `MaxBidPriceLib` library, there is an incorrect comparison. The condition should use the less than or equal to operator (`<=`) instead of less than (`<`). Otherwise, passing a `totalSupply` value of `1 << 75` will result in a `maxBid` value of `2**160` instead of `type(uint160).max`. This causes an overflow in `uint256` when `2**160` value is shifted left by 96 bits.

Consider changing the `<` operator to `<=` in the comparison.

Update: *Resolved in pull request #252.*

L-10 Unhandled Overflow in Constructor

Within the constructor of `ContinuousClearingAuction`, if the provided `tickSpacing` value exceeds `MAX_BID_PRICE`, it will result in an unhandled overflow condition. Consider redesigning the logic to include a validation check that ensures the provided `tickSpacing` value does not exceed `MAX_BID_PRICE` before it is used in subsequent calculations.

Update: *Resolved in pull request #276.*

L-11 Missing Validation for Bid Amount in `submitBid` Flows

The `submitBid` logic currently lacks a check to ensure that the bid amount meets a minimum required value, even though a `BidAmountTooSmall` error is defined. This omission allows users to submit excessively small bids or even with the amount equal to zero, potentially causing unnecessary transaction spam and reducing the efficiency of the bidding system.

Consider adding a validation to ensure that the bid amount is at least sufficient to cover the cost of a single token at the current clearing price.

Update: *Partially resolved in pull request #276. Bids with a zero amount are rejected.*

Notes & Additional Information

N-01 Unnecessary Cast

Within the `TokenCurrencyStorage` contract, the `address(_currency)` cast is unnecessary.

To improve the overall clarity and maintainability of the codebase, consider removing any unnecessary casts.

Update: *Resolved in pull request #260.*

N-02 Unused Code

Throughout the codebase, multiple instances of unused code were identified:

- The `TOTAL_SUPPLY_Q96` immutable variable in `TokenCurrencyStorage` contract
- All `using` statements in `CheckpointLib` library
- All `using` statements except for Currency in `TokenCurrencyStorage` contract
- Some `using` statements in `CheckpointStorage` contract

Consider removing unused code to improve the clarity and maintainability of the codebase.

Update: *Resolved in pull request #260.*

N-03 Inconsistent Type Usage in `initializeDistribution`

The `initializeDistribution` function of the `AuctionFactory` contract accepts an amount parameter of type `uint256`, but then validates that it does not exceed the maximum value of a `uint128` container. Throughout the rest of the logic, it is used as a `uint128` through casting.

Consider changing the `amount` parameter type to `uint128` for consistency and clarity.

Update: *Acknowledged, not resolved. The Uniswap team stated:*

> *Won't fix. Will keep interface as uint256 in case other integrations require that.*

N-04 Missing Security Contact

Providing a specific security contact (such as an email address or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Consider adding a NatSpec comment containing a security contact to each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Resolved in pull request #274.*

N-05 Missing Check of Validation Contract

The constructor of the `Auction` contract is missing logic to verify the validation contract. Providing an address that contains no code or only implements a generic fallback function will still result in successful validation during submitting bids, even if the validation contract is incorrect.

Consider adding a check to ensure that the provided validation contract implements the expected interface.

Update: *Acknowledged, not resolved. The Uniswap team stated:*

> *Lean against fixing, I assume the recommendation is to use ERC-165 or other introspection.*

N-06 Missing Zero-Address Validation of `owner`

While submitting a bid through the one of `submitBid` functions, it is possible to specify the `owner` of the bid. The issue is that the logic does not validate whether the provided address is

a zero address, which might lead to a loss of funds. It would not be possible to retrieve the sent native currency for refunds or to claim the tokens.

Consider adding a zero-address check to ensure it has not been set by accident.

Update: *Resolved in pull request #260.*

N-07 Unused Imports

Throughout the codebase, multiple instances of unused imports were identified:

- The import of `IERC20Minimal` and `FixedPoint128` in `Auction.sol`.
- The import of `FixedPoint128` in `CheckpointStorage.sol`.
- The import of `BidLib` and `ValueX7` in `TickStorage.sol`.
- The import of `ValueX7` in `TokenCurrencyStorage.sol`.
- The import of `ValueX7` in `ITokenCurrencyStorage.sol`.
- The import `FixedPoint128`, `FixedPoint96` and `ValueX7` in `BidLib.sol`.
- The import of `FixedPoint128` in `CheckpointLib.sol`.

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

Update: *Partially resolved in pull request #260.*

N-08 Incorrect Documentation

Throughout the codebase, multiple instances of incorrect documentation were identified:

- The comment describing the Bid's `owner` parameter states that the owner is the only address allowed to exit bid, whereas the implemented logic allows anyone to trigger the `exitBid` or `exitPartiallyFilledBid` functions.
- The `isGraduated` logic considers auctions to be graduated when the amount of raised currency reaches or exceeds the configured `REQUIRED_CURRENCY_RAISED_Q96` parameter. However, the NatSpec for the `IAuction` interface incorrectly states that the graduated auction is considered graduated if the clearing price is greater than the floor price.
- The comment for the `currencyRaisedAtClearingPriceQ96_X7` parameter in the `Checkpoint` struct incorrectly describes the parameter as representing tokens sold, whereas it actually represents the amount of currency raised at the clearing price.

- The spacing around the NatSpec of for the `CheckpointStorage` contract is inconsistent.

Consider addressing the instances listed above to improve the clarity and correctness of the documentation.

Update: *Resolved in pull request #260 and pull request #276.*

N-09 Library Constants Visibility Not Explicitly Declared

Within `ConstantsLib.sol`, multiple instances of constants not having an explicitly declared visibility were identified.

For improved code clarity, consider explicitly declaring the visibility of such constants as `internal`.

Update: *Acknowledged, not resolved.*

N-10 Incorrect Solidity Version in Libraries Using Custom Errors

Throughout the codebase, several libraries that use custom errors are marked with an incorrect Solidity version. Note that custom errors were introduced in Solidity version `0.8.4`.

- `BidLib.sol` is using Solidity version `^0.8.0`.
- `CurrencyLibrary.sol` is using Solidity version `^0.8.0`.
- `ValidationHookLib.sol` is using Solidity version `^0.8.0`.

Consider updating the minimum Solidity version for the specified libraries to `^0.8.4`.

Update: *Resolved in pull request #260.*

N-11 Function Visibility Overly Permissive

Throughout the codebase, multiple instances of functions having overly permissive visibility were identified:

- The `currencyRaised` function in `Auction.sol` with `public` visibility could be limited to `external`.
- The `submitBid` function in `Auction.sol` with `public` visibility could be limited to `external`.
- The `getAuctionAddress` function in `AuctionFactory.sol` with `public` visibility could be limited to `external`.
- The `clearingPrice` function in `CheckpointStorage.sol` with `public` visibility could be limited to `external`.
- The `getTick` function in `TickStorage.sol` with `public` visibility could be limited to `external`.

To better convey the intended use of functions and to potentially realize some additional gas savings, consider changing a function's visibility to be only as permissive as required.

Update: *Resolved in pull request #260. The Uniswap team stated:*

> *Fixed, `getTick` was removed in another PR in favor of `ticks()` which is already external.*

N-12 Interpretation of Block Number Varies Across Networks

The protocol logic heavily relies on block numbers retrieved through `block.number`. On Arbitrum, `block.number` corresponds to the block number of the L1 (parent chain), which may lead to unexpected behavior.

If the protocol is intended to be deployed on Arbitrum, consider implementing logic to retrieve the block number using the appropriate precompile call.

Update: *Acknowledged, not resolved. The Uniswap team stated:*

> *Acknowledged, currently no plans to do so.*

N-13 Confusing Logic for Setting Permit2 Address

Within the `PermitSingleForwarder.sol` contract, there is a constant `PERMIT2` and an immutable `permit2`. The immutable value is set within the constructor and used throughout the code. The constant value is set to the correct permit2 address but is only used in the inheriting `Auction` contract to set the immutable variable to itself.

Consider removing the immutable `permit2` and the constructor from the `PermitSingleForwarder` contract and only using the constant `PERMIT2` throughout the codebase.

Update: *Resolved in pull request #259. The Uniswap team stated:*

> *Contract has been removed.*

N-14 Event Parameter Could Be Indexed

The `address token` parameter in the `AuctionCreated` event could benefit from being indexed.

Consider adding the `indexed` keyword to the event parameter.

Update: *Resolved in pull request #260.*

N-15 Incorrect Accounting for Tokens With Fees Used As Currency

The current auction implementation does not support tokens that impose transfer fees. Using such tokens as the auction currency will lead to incorrect accounting, as the amount received will be less than what the protocol accounts for.

Consider either adding support for fee-on-transfer tokens or clearly documenting that these types of tokens are not supported and must not be used as the auction currency.

Update: *Acknowledged, will resolve. The Uniswap team stated:*

> *Acknowledged, this will be added to documentation.*

N-16 Reentrancy Through Hook Validation

The bid-submission logic triggers a validation hook that executes an external call via `IValidationHook`. If the execution is hijacked, it is possible to re-enter the `submitBid` logic, which may confuse external integrations that rely on the `nextBidId` function.

The following scenario illustrates the issue:

1. The external contract calls `nextBidId`, and the value `10` is returned.
2. `submitBid` is triggered.
3. Execution is hijacked through the validation hook logic, which re-enters `submitBid` and creates a bid with ID `10`.
4. The original `submitBid` continues execution and creates a bid with ID `11`.
5. The external contract relies on the initial `nextBidId`, which was `10`.

Consider thoroughly documenting the risks of validation hook reentrancy to ensure that external integrations do not rely on the `nextBidId` function.

Update: *Acknowledged, will resolve. The Uniswap team stated:*

> *Acknowledged, will document this behavior.*

N-17 Redundant Override Specifiers

Explicitly specifying the overrides for interface functions is not necessary. Throughout the codebase, multiple instances of functions specifying an override for interface functions were identified:

- The `step` function of the `AuctionStepStorage` contract
- The `startBlock` function of the `AuctionStepStorage` contract
- The `endBlock` function of the `AuctionStepStorage` contract
- The `pointer` function of the `AuctionStepStorage` contract
- The `nextBidId` function of the `BidStorage` contract
- The `bids` function of the `BidStorage` contract
- The `currency` function of the `TokenCurrencyStorage` contract
- The `token` function of the `TokenCurrencyStorage` contract
- The `totalSupply` function of the `TokenCurrencyStorage` contract
- The `tokensRecipient` function of the `TokenCurrencyStorage` contract
- The `fundsRecipient` function of the `TokenCurrencyStorage` contract
- The `floorPrice` function of the `TickStorage` contract

- The `tickSpacing` function of the `TickStorage` contract
- The `nextActiveTickPrice` function of the `TickStorage` contract
- The `ticks` function of the `TickStorage` contract
- The `lastCheckpointedBlock` function of the `CheckpointStorage` contract
- The `checkpoints` function of the `CheckpointStorage` contract
- The `claimBlock` function of the `Auction` contract
- The `validationHook` function of the `Auction` contract
- The `currencyRaisedQ96_X7` function of the `Auction` contract
- The `sumCurrencyDemandAboveClearingQ96` function of the `Auction` contract

To simplify the function headers, consider removing the override specifier from above-listed instances.

Update: *Resolved in pull request #260.*

N-18 Inconsistent Library Names

Most libraries under `src/libraries` directory use the "Lib" suffix in their names. However, `CurrencyLibrary` uses the full "Library" suffix, while `FixedPoint96` and `FixedPoint128` do not use a suffix.

For consistency, consider updating all library names to use the "Lib" suffix.

Update: *Acknowledged, not resolved.*

N-19 Potential Improvements to `CurrencyLibrary`

There are three minor issues in the `transfer` function of `CurrencyLibrary`.

1. **Dependence on implicit execution order in the `and` statement**: For improved code clarity, this can be divided into separate statements.
2. **Returning success for non-existing tokens**: Although this is safe in the context of the `Auction` contract, the behavior should still be documented to prevent misuse of the library.
3. **Unnecessary** free memory cleanup: This operation is not necessary and can be safely removed to save gas.

Consider applying the suggested changes for improved code clarity and maintainability.

Update: *Acknowledged, not resolved. The Uniswap team stated:*

> *Acknowledged, won't fix as CurrencyLibrary was previously audited in Uni v4.*

N-20 Duplicate `onlyActiveAuction` Checks

The `Auction` contract defines two public `submitBid` functions. One accepts a `prevTickPrice` as a hint, while the other automatically uses the `FLOOR_PRICE` constant and delegates execution to the first function. Since both functions use the `onlyActiveAuction` modifier, the modifier is executed twice when a user calls the version without a hint. This leads to redundant checks and unnecessary overhead.

Consider removing the `onlyActiveAuction` modifier from the delegating function, as the check will already be enforced by the main `submitBid` function.

Update: *Resolved in pull request #260.*

Conclusion

The Uniswap TWAP Auction protocol has been developed to fairly distribute tokens and discover their market price by matching supply with participants' currency bids. The audit identified one medium-severity issue, along with several minor issues, leading to recommendations for improving code consistency and readability.

The Uniswap team is appreciated for being highly cooperative and providing clear explanations throughout the audit process.