Report

v. 1.0

Customer

Ekubo

Smart Contract Audit

TWAMM

27th April 2025

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	25.04.25	A. Zveryanskaya	Initial Draft
0.2	27.04.25	A. Zveryanskaya	Minor revision
1.0	27.04.25	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

Ekubo is the most powerful AMM ever, featuring 100x concentrated liquidity, extensibility, and highly optimized smart contracts to provide the best execution and LP returns.

After fixing the indicated issues the smart contracts should be re-audited.

3 Project scope

We were asked to review:

- Original Code

Files:

/			
	Orders.sol		
math/			
	twamm.sol	timeBitmap.sol	time.sol
libraries/			
	TWAMMLib.sol		
lens/			
	TWAMMDataFetcher.sol		
extensions/			
	TWAMM.sol		

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Recommendations** contain code style, best practices and other suggestions.

5 Our findings

We found 2 critical, 2 major, and a few less important issues.

6 Critical Issues

CVF-1 INFO

- **Category** Flaw
- **Source** Orders.sol

Description The “handleLockData” takes only 5 arguments, thus the “minRefund” value will be ignored.

Recommendation Implement “minRefund” check either inside the “handleLockData” function or outside it.

Client Comment *Fixed by removing the minRefund parameter altogether. If a caller wishes to implement a minimum refund, they can use checkDeadline via multicall.*

```
85 abi.encode(bytes1(0xdd), recipient, id, orderKey, -int256(uint256(
       ↪ saleRateDecrease)), minRefund)
```

CVF-2 INFO

- **Category** Unclear behavior
- **Source** Orders.sol

Description This value could be greater than orderKey.endTime - lastUpdateTime.

Recommendation Ensure, sale duration doesn’t exceed orderKey.endTime-lastUpdateTime.

```
141 uint32 saleDuration = uint32(
        FixedPointMathLib.min(
            FixedPointMathLib.min(secondsSinceLastUpdate,
               ↪ secondsSinceOrderStart), totalOrderDuration
        )
    );
```

7 Major Issues

CVF-3 INFO

- **Category** Unclear behavior
- **Source** timeBitmap.sol

Description There are no range checks for the arguments, which could lead to overflow and overlapping values.
Specifying valid argument ranges in the documentation comment above a function is usually not enough to ensure safety usage, as this comment isn't visible at places where the function is used. Proper way would be to either implement range check assertions inside the function, or clearly mark the function as unsafe (in its name), so this "unsafe" sign will be visible at all function usages, encouraging people to actually read the function comment carefully before touching the adjacent code.

Recommendation Implement proper range checks or clearly mark the function as unsafe.

Client Comment *The comment on line already indicates the word parameter must be less than 2**252. Still, improved comment to say index is expected to be less than 2**8.*

```
22 time := shl(4, add(mul(word, 256), index))
```

CVF-11 INFO

- **Category** Procedural
- **Source** TWAMM.sol

Description This TODO should be resolved.

Recommendation Consider using safe math or wider type.

```
381 // todo: what if params.saleRateDelta is type(int112).min?
_updateTime(poolId, params.orderKey.endTime, -params.saleRateDelta,
    ↪ isToken1, numOrdersChange);
```

8 Moderate Issues

CVF-4 INFO

- **Category** Overflow/Underflow
- **Source** twamm.sol

Description Overflow is possible here. There are explicit assumptions in the comments, but such assumptions are not good protection from overflow.

Client Comment *We already have dev comments indicating the assumptions about the input values in 3 out of 5 cases, and in the last case overflow is not actually possible. Added a comment for the missing one.*

```
15 saleRate := div(shl(32, amount), duration)
```

```
29 result := add(saleRate, saleRateDelta)
```

```
43 amount := shr(32, add(mul(saleRate, duration), mul(0xffffffff,
      ↪ roundUp)))
```

```
71 uint256 saleRatio = (saleRateToken1 << 128) / saleRateToken0;
```

```
126         int256 ePowExponent = int256(uint256(exp2(uint128(exponent))
              ↪ ) << 64);
```

CVF-6 INFO

- **Category** Overflow/Underflow
- **Source** TWAMM.sol

Description Overlow and underflow are possible here.
Adding a signed value to an unsigned one may involve desired technical overflow, when the signed value is negative. However, it could also lead to an undesired real overflow/underflow, when the correct result wouldn't fit into the destination type. In the client comment why real oveflow/underflow isn't possible one refers to assumptions about how this function is used, and what are posible argument values. While these assumptions could be true, they are not explicitly documented in the code, so it would be hard to use them. Also, relying on assumptions about how the function is used is in generally unsafe. The overflow protection isn't sufficient, as it only check that the result fits into 32 bits. However, overflow or underflow may easily result in a value fitting into 32 bits.

Recommendation Use safe arithmetics.

Client Comment *Overflow is how we do decrementing (note we are adding a signed integer to an unsigned integer), and overflow by incrementing is impossible because _updateTime is only called with numOrdersChange = sub(iszero(saleRate), iszero(saleRateNext)), i.e. we only add at most 1. In addition, we have an overflow check on the following line, even though it's unnecessary in the context of the TWAMM contract.*

```
240 let numOrdersNext := add(numOrders, numOrdersChange)
```

CVF-7 INFO

- **Category** Overflow/Underflow
- **Source** TWAMM.sol

Description Underflow is possible here.

Client Comment *Underflow is expected and safe. The global reward rate grows until it overflows just like with liquidity provider pool fees.*

```
325 uint256 purchasedAmount = computeRewardAmount(rewardRateInside -
    ↪ order.rewardRateSnapshot, saleRate);
```

CVF-8 INFO

- **Category** Overflow/Underflow
- **Source** TWAMM.sol

Description Overflow is possible when performing left shift.

Client Comment *computeRewardAmount returns a uint128, which we cast to a uint256. This cannot overflow*

```
338 sub(rewardRateInside, div(shl(128, purchasedAmount), saleRateNext)),
```

CVF-9 INFO

- **Category** Overflow/Underflow
- **Source** TWAMM.sol

Description Underflow is possible during subtraction.

Client Comment *Underflow is expected and safe. The global reward rate grows until it overflows just like with liquidity provider pool fees.*

```
338 sub(rewardRateInside, div(shl(128, purchasedAmount), saleRateNext)),
```

CVF-10 INFO

- **Category** Overflow/Underflow
- **Source** TWAMM.sol

Description Overflow is possible here.

Recommendation Use safe conversion.

Client Comment *It's not possible because we use addSaleRateDelta to get saleRateNext which reverts if the result does not fit into 112 bits. In addition, amountSold cannot exceed uint112 because it is computed from sum of (sale rate * order duration) » 32 , which is limited to 112 bits since saleRate is a uint112 and order duration must be less than 2**32*

```
351 uint112(saleRateNext),
```

```
354     amountSold
```

CVF-12 INFO

- **Category** Suboptimal
- **Source** TWAMMDataFetcher.sol

Description This call could be optimized.

Recommendation Instead of calculating step size on each iteration, calculate the initial step size before the loop and then increase it when necessary.

Client Comment *Data fetcher is called off-chain and does not need optimization.*

```
25 t = nextValidTime(currentTime, t);
```

CVF-13 INFO

- **Category** Documentation
- **Source** TWAMM.sol

Description Despite the comment, actual code may store either (1, 1) or (0, 0).

Recommendation Clarify or fix the comment.

```
249 // write the poolRewardRatesBefore[poolId][time] = (1,1)
```

```
262     sstore(slot0, iszero(numOrders))
        sstore(add(slot0, 1), iszero(numOrders))
```

CVF-14 INFO

- **Category** Overflow/Underflow
- **Source** Orders.sol

Description Here an implicit underflow check is used to enforce an important business-level constraint, which is a bad practice.

Recommendation Add an explicit ”require” statement to ensure ”orderKey.endTime“ isn’t before “realStart”.

```
65 saleRate = uint112(computeSaleRate(amount, uint32(orderKey.endTime -
   ↪  realStart)));
```

9 Recommendations

CVF-15 INFO

- **Category** Suboptimal
- **Source** time.sol

Description Specifying a particulr compiler version makes it harder migrating to newer versions.

Recommendation Specify as "^0.8.0" or as "^0.8.28" if there is something special regarding this particular versions. Also relevant for: timeBitmap.sol, twamm.sol, TWAMMLib.sol, TWAMMDataFetcher.sol, TWAMM.sol, Orders.sol.

Client Comment *Acknowledged.*

```
2 pragma solidity =0.8.28;
```

CVF-16 INFO

- **Category** Unclear behavior
- **Source** time.sol

Description We didn't review these files.

Client Comment *Acknowledged.*

```
4 import {LibBit} from "solady/utils/LibBit.sol";
  import {FixedPointMathLib} from "solady/utils/FixedPointMathLib.sol"
    ↪ ;
```

CVF-17 INFO

- **Category** Documentation
- **Source** time.sol

Description The semantics of the "time" argument is unclear.

Recommendation Explain in the documentation comment.

```
16 function computeStepSize(uint256 currentTime, uint256 time) pure
    ↪ returns (uint256 stepSize) {
```

CVF-22 INFO

- **Category** Unclear behavior
- **Source** timeBitmap.sol

Description We didn't review this file.

Client Comment *Acknowledged.*

```
5 import {FixedPointMathLib} from "solady/utils/FixedPointMathLib.sol"
    ↪ ;
```

CVF-25 INFO

- **Category** Suboptimal
- **Source** timeBitmap.sol

Recommendation This could be simplified as: if (nextTime > untilTime)

```
70 if (nextTime - fromTime > untilTime - fromTime) {
```

CVF-26 INFO

- **Category** Unclear behavior
- **Source** twamm.sol

Description We didn't review these files.

Client Comment *Acknowledged.*

```
6 import {exp2} from "./exp2.sol";
  import {FixedPointMathLib} from "solady/utils/FixedPointMathLib.sol"
    ↪ ;
```

CVF-27 INFO

- **Category** Documentation
- **Source** twamm.sol

Description Despite the comment, the actual result seems to be in .128 format.

Recommendation Fix the comment.

```
54 // Computes the quantity `c = (sqrtSaleRatio - sqrtRatio) / (
      ↪ sqrtSaleRatio + sqrtRatio)` as a signed 64.64 number
```

```
58         FixedPointMathLib.dist(sqrtRatio, sqrtSaleRatio), (1 << 128)
             ↪ , sqrtRatio + sqrtSaleRatio
```

CVF-28 INFO

- **Category** Suboptimal
- **Source** twamm.sol

Recommendation This could be simplified as: let sign = sub(shl(1, gt(sqrtSleRatio, sqrtRatio)), 1) c := mul(sign, unsigned)

```
61 let negativeMult := sub(0, lt(sqrtSaleRatio, sqrtRatio))
```

```
63 c := add(mul(negativeMult, unsigned), mul(iszero(negativeMult),
      ↪ unsigned))
```

CVF-29 INFO

- **Category** Suboptimal
- **Source** twamm.sol

Recommendation Conversion to “uint256” is redundant, as “saleRatetoken0” is already “uint256”.

```
114 uint256 sqrtSaleRateWithoutFee = FixedPointMathLib.sqrt(uint256(
       ↪ saleRateToken0) * saleRateToken1);
```

CVF-30 INFO

- **Category** Suboptimal
- **Source** TWAMMLib.sol

Recommendation Binary ANDs here are redundant, as Solidity compiler allows garbage in unused bits.

Client Comment *That assumes the abicoder will do ANDs before returning the narrow typed values. Is this true?*

```
26 lastVirtualOrderExecutionTime := and(s, 0xffffffff)
   saleRateToken0 := and(shr(32, s), 0xffffffffffffffffffffffffffffffff)
```

CVF-31 INFO

- **Category** Unclear behavior
- **Source** TWAMMDataFetcher.sol

Description We didn't review these files.

Client Comment *This contract is only called off-chain and was not supposed to be included in the audit. Performance is not important for an off-chain lens contract.*

```
14 import {LibBit} from "solady/utils/LibBit.sol";
   import {FixedPointMathLib} from "solady/utils/FixedPointMathLib.sol"
      ↪ ;
```

CVF-32 INFO

- **Category** Readability
- **Source** TWAMMDataFetcher.sol

Description Declaring top-level functions and types in a file named after a contract makes it harder navigating through code.

Recommendation Move the top-level declarations into the contract or move them into a separate file.

Client Comment *This contract is only called off-chain and was not supposed to be included in the audit. Performance is not important for an off-chain lens contract.*

```
18 function getAllValidFutureTimes(uint256 currentTime) pure returns (
   ↪ uint256[] memory times) {
```

```
36 struct TimeSaleRateInfo {
```

```
42 struct PoolState {
```

CVF-33 INFO

- **Category** Suboptimal
- **Source** TWAMMDataFetcher.sol

Description The value “poolKey.toPoolId()” is calculated several times.

Recommendation Calculate once and reuse.

Client Comment *This contract is only called off-chain and was not supposed to be included in the audit. Performance is not important for an off-chain lens contract.*

```
64 (SqrtRatio sqrtRatio, int32 tick, uint128 liquidity) = core.
   ↪ poolState(poolKey.toPoolId());
```

```
66     twamm.poolState(poolKey.toPoolId());
```

```
72 bytes32 poolId = poolKey.toPoolId();
```

CVF-34 INFO

- **Category** Suboptimal
- **Source** TWAMMDataFetcher.sol

Description The value “mload(timeInfoSlots)” is calculated on every iteration.

Recommendation Calculate once before the loop.

Client Comment *This contract is only called off-chain and was not supposed to be included in the audit. Performance is not important for an off-chain lens contract.*

```
78 for { let i := 0 } lt(i, mload(timeInfoSlots)) { i := add(i, 1) } {
```

CVF-35 INFO

- **Category** Suboptimal
- **Source** TWAMMDataFetcher.sol

Recommendation This addition could be moved out of the loop.

Client Comment *This contract is only called off-chain and was not supposed to be included in the audit. Performance is not important for an off-chain lens contract.*

```
79 let offset := mul(add(i, 1), 32)
```

CVF-36 INFO

- **Category** Suboptimal
- **Source** TWAMMDataFetcher.sol

Recommendation It would be more efficient to use left shift instead of multiplication here.

Client Comment *This contract is only called off-chain and was not supposed to be included in the audit. Performance is not important for an off-chain lens contract.*

```
79 let offset := mul(add(i, 1), 32)
```

CVF-37 INFO

- **Category** Suboptimal
- **Source** TWAMMDataFetcher.sol

Description Converting a counter into a memory offset on each iteration is inefficient.

Recommendation Optimize like this: let ptr := add (timeInfoSlots, 32); let endPtr := add (ptr, shl (5, mload (timeInfoSlots))) for {} lt (ptr, endPtr) {ptr:= add (ptr, 32)} { mstore (0, mload (ptr)) }

Client Comment *This contract is only called off-chain and was not supposed to be included in the audit. Performance is not important for an off-chain lens contract.*

```
78 for { let i := 0 } lt(i, mload(timeInfoSlots)) { i := add(i, 1) } {
       let offset := mul(add(i, 1), 32)
80     mstore(0, mload(add(allValidTimes, offset)))
```

CVF-38 INFO

- **Category** Suboptimal
- **Source** TWAMMDataFetcher.sol

Recommendation Here “require” should be used instead of “assert”.

Client Comment *This contract is only called off-chain and was not supposed to be included in the audit. Performance is not important for an off-chain lens contract.*

```
88 assert(success);
```

CVF-39 INFO

- **Category** Suboptimal
- **Source** TWAMMDataFetcher.sol

Description Converting a counter into an offset on each iteration is inefficient.

Recommendation Calcualte the initial offset before the loop and increase it by 32 on each iteration.

Client Comment *This contract is only called off-chain and was not supposed to be included in the audit. Performance is not important for an off-chain lens contract.*

```
98 let value := mload(add(result, mul(add(i, 1), 32)))
```

CVF-40 INFO

- **Category** Readability
- **Source** TWAMMDataFetcher.sol

Recommendation The “add(i, 1)” addition could be moved outside the loop.

Client Comment *This contract is only called off-chain and was not supposed to be included in the audit. Performance is not important for an off-chain lens contract.*

```
98 let value := mload(add(result, mul(add(i, 1), 32)))
```

CVF-41 INFO

- **Category** Suboptimal
- **Source** TWAMMDataFetcher.sol

Recommendation Left should would be more efficient that multiplication.

Client Comment *This contract is only called off-chain and was not supposed to be included in the audit. Performance is not important for an off-chain lens contract.*

```
98 let value := mload(add(result, mul(add(i, 1), 32)))
```

CVF-42 INFO

- **Category** Unclear behavior
- **Source** TWAMM.sol

Description We didn’t review this file.

Client Comment *Acknowledged.*

```
17 import {FixedPointMathLib} from "solady/utils/FixedPointMathLib.sol"
   ↪ ;
```

CVF-43 INFO

- **Category** Suboptimal
- **Source** TWAMM.sol

Description Declaring top-level functions and types in a file named after a contract makes it harder navigating through code.

Recommendation Move the top-level declarations into the contract of move them into a separate file.

Client Comment *Acknowledged.*

25
```
function twammCallPoints() pure returns (CallPoints memory) {
```

40
```
struct OrderKey {
```

49
```
function toOrderId(OrderKey memory orderKey) pure returns (bytes32
    ↪ id) {
```

55
```
struct UpdateSaleRateParams {
```

61
```
struct CollectProceedsParams {
```

66
```
function orderKeyToPoolKey(OrderKey memory orderKey, address twamm)
    ↪ pure returns (PoolKey memory poolKey) {
```

CVF-44 INFO

- **Category** Bad datatype
- **Source** TWAMM.sol

Recommendation The type for these fields should be “IERC20”.

Client Comment *Acknowledged.*

41
```
address sellToken;
address buyToken;
```

CVF-45 INFO

- **Category** Suboptimal
- **Source** TWAMM.sol

Recommendation This TODO should be either resolved or removed.

```
43 // todo: these could take up as few as 32+64+64=160 bits
```

CVF-46 INFO

- **Category** Bad datatype
- **Source** TWAMM.sol

Recommendation The type for the “twamm” argument should be more specific.

Client Comment *Acknowledged.*

```
66 function orderKeyToPoolKey(OrderKey memory orderKey, address twamm)
   ↪ pure returns (PoolKey memory poolKey) {
```

CVF-47 INFO

- **Category** Documentation
- **Source** TWAMM.sol

Recommendation Should be “96 bytes”, not bits.

```
84 // move free memory pointer forward 96 bits
```

CVF-49 INFO

- **Category** Suboptimal
- **Source** TWAMM.sol

Recommendation The “owner” and “orderKey” parameters should be indexed.

Client Comment *Acknowledged.*

```
96 event OrderUpdated(address owner, bytes32 salt, OrderKey orderKey,
   ↪ int112 saleRateDelta);
event OrderProceedsWithdrawn(address owner, bytes32 salt, OrderKey
   ↪ orderKey, uint128 amount);
```

CVF-50 INFO

- **Category** Suboptimal
- **Source** TWAMM.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

Client Comment *Acknowledged. Arguments in these cases would not be useful in debugging and callers should not rely on try-catch patterns.*

```
99  error TimeNumOrdersOverflow();
100 error TickSpacingMustBeMaximum();
    error OrderAlreadyEnded();
    error InvalidTimestamps();
    error MustCollectProceedsBeforeCanceling();
    error MaxSaleRateDeltaPerTime();
```

CVF-51 INFO

- **Category** Suboptimal
- **Source** TWAMM.sol

Recommendation It would be more efficient to merge these mappings into a single mapping whose keys are pool IDs and values are structs encapsulating values of the original mappings.

Client Comment *Acknowledged.*

```
133 mapping(bytes32 poolId => PoolState) internal poolState;
    mapping(bytes32 poolId => mapping(uint256 word => Bitmap bitmap))
       ↪ internal poolInitializedTimesBitmap;
    mapping(bytes32 poolId => mapping(uint256 time => TimeInfo))
       ↪ internal poolTimeInfos;
```

```
138 mapping(bytes32 poolId => FeesPerLiquidity) internal poolRewardRates
       ↪ ;
    mapping(bytes32 poolId => mapping(uint256 time => FeesPerLiquidity))
       ↪  internal poolRewardRatesBefore;
```

```
146 mapping(bytes32 poolId => bool) poolInitialized;
```

CVF-52 INFO

- **Category** Suboptimal
- **Source** TWAMM.sol

Recommendation It would be more efficient to merge these mappings into a single mapping whose keys are pool IDs and times, and values are structs encapsulating the values of the original mappings.

Client Comment *Acknowledged.*

```
135 mapping(bytes32 poolId => mapping(uint256 time => TimeInfo))
      ↪ internal poolTimeInfos;
```

```
139 mapping(bytes32 poolId => mapping(uint256 time => FeesPerLiquidity))
      ↪  internal poolRewardRatesBefore;
```

CVF-53 INFO

- **Category** Suboptimal
- **Source** TWAMM.sol

Recommendation It is a good practice to put a comment into an empty slot to explain why the block is empty.

Client Comment *Acknowledged.*

```
148 constructor(ICore core) BaseExtension(core) BaseForwardee(core) {}
```

CVF-55 INFO

- **Category** Suboptimal
- **Source** TWAMM.sol

Recommendation This would produce shorter bytecode: shr (32, numOrdersNext)

Client Comment *Acknowledged. Does not change gas costs after optimizer.*

```
242 if gt(numOrdersNext, 0xffffffff) {
```

CVF-56 INFO

- **Category** Suboptimal
- **Source** TWAMM.sol

Recommendation This expression could be optimized as: xor(iszero(numOrder), iszero(numOrdersNext))

Client Comment *Acknowledged. Fixed and saved 3 gas*

```
247 flip := iszero(eq(iszero(numOrders), iszero(numOrdersNext)))
```

CVF-57 INFO

- **Category** Suboptimal
- **Source** TWAMM.sol

Description The first slot of “data” is decoded twice.

Recommendation Refactor to decode once.

Client Comment *It’s not trivial to do this and stay within solidity.*

```
294 uint256 callType = abi.decode(data, (uint256));
```

```
297     (, UpdateSaleRateParams memory params) = abi.decode(data, (
        ↪ uint256, UpdateSaleRateParams));
```

```
436     (, CollectProceedsParams memory params) = abi.decode(data, (
        ↪ uint256, CollectProceedsParams));
```

CVF-58 INFO

- **Category** Suboptimal
- **Source** TWAMM.sol

Recommendation Conversions to “address” are redundant, as pool key tokens are already “address”.

```
410 core.load(address(poolKey.token0), address(poolKey.token1), bytes32
    ↪ (0), 0, amountAbs);
```

```
413 core.load(address(poolKey.token0), address(poolKey.token1), bytes32
    ↪ (0), amountAbs, 0);
```

```
423     address(this), address(poolKey.token0), address(poolKey.token1),
        ↪  bytes32(0), 0, amountAbs
```

```
427     address(this), address(poolKey.token0), address(poolKey.token1),
        ↪  bytes32(0), amountAbs, 0
```

CVF-59 INFO

- **Category** Suboptimal
- **Source** TWAMM.sol

Recommendation Conversion to “int256” is redundant, as compiler would do it anyway.

```
416 amountDelta += int256(int128(fee));
```

CVF-60 INFO

- **Category** Documentation
- **Source** TWAMM.sol

Recommendation This comment is confusing and should be removed or translated into plain English.

```
489 // or(or(and(timestamp(), 0xffffffff), shl(32, saleRateToken0)), shl
    ↪ (144, saleRateToken1))
```

CVF-62 INFO

- **Category** Suboptimal
- **Source** TWAMM.sol

Recommendation It is safe here to copy the returned data into the memory starting at zero address.

Client Comment *According to the solidity compiler documentation, it is not recommended from within a memory-safe block.*

```
663 if iszero(call(gas(), target, 0, o, 100, 0, 0)) {
        returndatacopy(o, 0, returndatasize())
```

CVF-63 INFO

- **Category** Unclear behavior
- **Source** Orders.sol

Description We didn't review these files.

Client Comment *Acknowledged.*

```
4 import {ERC721} from "solady/tokens/ERC721.sol";
```

```
17 import {FixedPointMathLib} from "solady/utils/FixedPointMathLib.sol"
       ↪ ;
```

CVF-64 INFO

- **Category** Suboptimal
- **Source** Orders.sol

Recommendation These errors could be made more helpful by adding certain parameters into them.

Client Comment *Acknowledged.*

```
25 error InvalidDuration();
   error OrderAlreadyEnded();
   error MaxSaleRateExceeded();
   error MinimumRefund();
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting