Report

v. 1.0

Customer

Zaha Studio

Smart Contract Audit

TWAMM Hook

23rd January 2025

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	23.01.25	A. Zveryanskaya	Initial Draft
0.2	23.01.25	A. Zveryanskaya	Minor revision
1.0	23.01.25	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.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

/			
	TWAMM.sol	ITWAMM.sol	
libraries/			
	OrderPool.sol	PoolGetters.sol	TransferHelper.sol
	TwammMath.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 1 major, and a few less important issues. All identified Major issues have been fixed.

Fixed 1 out of 1 issues

6 Major Issues

CVF-1 FIXED

- **Category** Flaw
- **Source** TWAMM.sol

Description There is no check to ensure the order expiration timestamp is in the future.

Recommendation Add such a check.

Client Comment *The order expiration timestamp is checked on line 191.*

```
230 if (orderKey.expiration % expirationInterval != 0) {
```

7 Moderate Issues

CVF-2 INFO

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

Description This function doesn't check that the expiration time is in the past and that there were no missed intervals.

Recommendation Either add corresponding checks or clearly state in a comment that the caller is responsible for expiration consistency.

Client Comment *Both of those things are checked prior to calling that function, and additionally this function is only responsible from bringing the state up to the interval, when there's an expiration at the interval. Checks are present before the function is ever called.*

```
20 function advanceToInterval(State storage self, uint256 expiration,
      ↪ uint256 earningsFactor) internal {
```

CVF-3 INFO

- **Category** Suboptimal
- **Source** PoolGetters.sol

Description Some functions in this library duplicate functionality of the Uniswal v4 StateLibrary library.

Recommendation Use functions from StateLibrary where possible.

Client Comment *The library does not reimplement any logic in the 'StateLibrary' lib, in fact it uses the 'StateLibrary' lib internally.*

```
12 library PoolGetters {
```

CVF-4 INFO

- **Category** Readability
- **Source** TransferHelper.sol

Description Relying on low-level compiler behavior is a bad practice, as it make code harder to read and more error prone.

Recommendation Perform call in a separate statement.

Client Comment *'TransferHelper' is used verbatim from the Uniswap v4 codebase.*

```
34 // Counterintuitively, this call must be positioned second to the or
      ↪ () call in the
   // surrounding and() call or else returndatasize() will be zero
      ↪ during the computation.
```

CVF-5 FIXED

- **Category** Suboptimal
- **Source** TwammMath.sol

Description Here a hybrid of fixed point and floating point number formats is used. Such hybrid format is inefficient and confusing.

Recommendation Use floating point or fixed point number, but not hybrids of them.

```
49 bytes16 sqrtSellRatioX96Bytes = sellRateBytes1.div(sellRateBytes0).
      ↪ sqrt().mul(Q96);
```

```
59 bytes16 newSqrtPriceBytesX96 = calculateNewSqrtPrice(priceParams).
      ↪ mul(Q96);
```

```
127 bytes16 sqrtSellRatioX96 = sellRate1Bytes.div(sellRate0Bytes).sqrt()
      ↪ .mul(Q96);
```

CVF-6 FIXED

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

Description There is no range check for the "_expirationInterval" argument.

Recommendation Add a check to ensure the value is not zero.

```
54 constructor(IPoolManager _manager, uint256 _expirationInterval)
   ↪ BaseHook(_manager) {
```

CVF-7 INFO

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

Description The expiration timestamp calculated here may not be a multiple of expirationInterval.

Recommendation Add an explicit check to ensure that expiration timestamp is a multiple of expirationInterval.

Client Comment *Line 230 checks if the expiration is on the interval or not.*

```
188 orderKey = OrderKey(msg.sender, (currentTimestampAtInterval +
    ↪ duration).toUint160(), zeroForOne);
```

CVF-8 INFO

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

Description Here a number of token less than "amountIn" could be transferred and the caller won't know the exact transferred amount.

Recommendation Either return the actual transferred amount or require "amountIn" to be a multiple of "duration".

Client Comment *This is an optimization to avoid transferring more than necessary. The number of tokens will still be fairly close. It's acceptable to have a small discrepancy here. This same calculation will happen on the FE anyway, so we'll make sure it's always correct.*

```
202 .safeTransferFrom(msg.sender, address(this), sellRate * duration);
```

CVF-9 INFO

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

Description It seems that this condition cannot be true, as the loop above could be terminated either when nextExpirationTimestamp > currentTimestampAtInterval or when _hasOutstandingOrder(self) is false. However, when nextExpirationTimestamp > currentTimestampAtInterval, then it is not possible that prevTimestamp < currentTimestampAtInterval, as prevTimestamp = nextExpirationTimestamp - expirationInterval, and all timestamps are multiple of expirationInterval.

Recommendation Remove this conditional statement.

Client Comment *As you can see in coverage, this condition is in fact hit. When you ask? Essentially, the loop you see runs over when there's available sellRate in either directions, so this loop can actually end without ever hitting the break so the latter condition can be true that way. The first part of the condition will also be true when prevTimestamp was never updated in the loop, aka when sellRateEnding during the entire loop was zero (which just means the sellRate modification happens in the future and we haven't been there yet).*

```
453 if (prevTimestamp < currentTimestampAtInterval &&
    ↪ _hasOutstandingOrders(self)) {
```

8 Minor Issues

CVF-10 FIXED

- **Category** Procedural
- **Source** OrderPool.sol

Recommendation Consider specifying as "^0.8.0" unless there is something special regarding this particular version. Also relevant for: PoolGetters.sol, TransferHelper.sol, TWAMM.sol.

Client Comment *Will be pinning to '^0.8.26' by the end of the audit. Right now they just use the min solidity version that includes support for all uses opcodes or features.*

```
2 pragma solidity ^0.8.15;
```

CVF-11 FIXED

- **Category** Procedural
- **Source** PoolGetters.sol

Description This compiler version requirement is inconsistent with other files in the same code base.

Recommendation Use consistent version requirements.

Client Comment *Will be pinning to '^0.8.26' by the end of the audit. Right now they just use the min solidity version that includes support for all uses opcodes or features.*

```
2 pragma solidity ^0.8.19;
```

CVF-12 INFO

- **Category** Suboptimal
- **Source** PoolGetters.sol

Recommendation This could be simplified as: l := shr (128, value)

```
29 l := shr(128, and(value, shl(128, sub(shl(128, 1), 1))))
```

CVF-13 INFO

- **Category** Suboptimal
- **Source** PoolGetters.sol

Recommendation Teh expression “shl(128, sub(shl(128, 1), 1))” is actually constant and could be precomputed.

```
29 l := shr(128, and(value, shl(128, sub(shl(128, 1), 1))))
```

CVF-14 INFO

- **Category** Suboptimal
- **Source** PoolGetters.sol

Recommendation This could be simplified as: type(uint256).max » 255 - bitPos

```
67 uint256 mask = (1 << bitPos) - 1 + (1 << bitPos);
```

CVF-15 INFO

- **Category** Suboptimal
- **Source** TwammMath.sol

Description A value is multiplied by “Q96” and then divided by the same value.

Recommendation Reuse the original value.

Client Comment *Storage optimization, to avoid storing one extra value. Arguably can be done in a different way.*

```
49 bytes16 sqrtSellRatioX96Bytes = sellRateBytes1.div(sellRateBytes0).
   ↪ sqrt().mul(Q96);
```

```
52     sqrtSellRatio: sqrtSellRatioX96Bytes.div(Q96),
```

CVF-16 FIXED

- **Category** Suboptimal
- **Source** TwammMath.sol

Recommendation The value “int256(2).fromUInt()” should be precomputed.

Client Comment *Acceptable.*

```
132 bytes16 denominator = uint256(2).fromUInt().mul(sqrtSellRate);
```

CVF-17 FIXED

- **Category** Procedural
- **Source** ITWAMM.sol

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

Client Comment *Will improve the errors.*

```
14 error PoolWithNativeNotSupported();
   error InvalidTargetTimestamp();
```

CVF-18 FIXED

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

Description This compiler version requirement is inconsistent with other files in the same code base.

Recommendation Use consistent version requirements.

Client Comment *Will be pinning to ‘^0.8.26‘ by the end of the audit. Right now they just use the min solidity version that includes support for all uses opcodes or features.*

```
2 pragma solidity ^0.8.26;
```

CVF-19 FIXED

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

Description Unlike other Uniswap v4 files, this one is imported from local code base.

Recommendation Import from Uniswap repository.

```
4 import {BaseHook} from "v4-periphery/src/base/hooks/BaseHook.sol";
```

CVF-20 INFO

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

Recommendation This line should be surrounded with an “unchecked” block to allow maxSwapAmount=type(int256).min

Client Comment *Agreed. Will not change however, we do not want to go as far as ‘type(int256).min‘ and the strong intentional typing is preferred.*

```
155 uint256 maxToSwap = maxSwapAmount > 0 ? uint256(maxSwapAmount) :
    ↪ uint256(-maxSwapAmount);
```

CVF-21 INFO

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

Recommendation The “toint256” call is redundant here, as “maxToSwap” was just converted from “int256”. Just do an unsafe conversion and then unchecked negation to allow the type(int256).min. value.

Client Comment *Agreed. Will not change however, we do not want to go as far as ‘type(int256).min‘ and the strong intentional typing is preferred.*

```
162 IPoolManager.SwapParams(zeroForOne, -maxToSwap.toInt256(),
    ↪ sqrtPriceLimitX96);
```

CVF-22 FIXED

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

Recommendation These checks should be performed earlier.

Client Comment *Agreed.*

294
```
if (orderKey.owner != msg.sender) {
    revert MustBeOwner(orderKey.owner, msg.sender);
}
if (order.sellRate == 0) {
    revert OrderDoesNotExist(orderKey);
}
```

CVF-23 INFO

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

Recommendation Brackets around multiplication are redundant.

Client Comment *Agreed. Will not change however, just a preference.*

303
```
buyTokensOwed = ((earningsFactorLast - order.earningsFactorLast) *
    ↪ order.sellRate) >> FixedPoint96.RESOLUTION;
```

CVF-24 INFO

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

Recommendation It would be more efficient to do: unchecked { uint256(-int256(delta.amount0())) }

Client Comment *Agreed. Will not change.*

355
```
key.currency0.settle(poolManager, address(this), uint256(uint128(-
    ↪ delta.amount0())), false);
```

358
```
key.currency1.take(poolManager, address(this), uint256(uint128(delta
    ↪ .amount1())), false);
```

362
```
key.currency1.settle(poolManager, address(this), uint256(uint128(-
    ↪ delta.amount1())), false);
```

365
```
key.currency0.take(poolManager, address(this), uint256(uint128(delta
    ↪ .amount0())), false);
```

CVF-25 FIXED

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

Description The variable name is confusing, as the target timestamp is not always the current time.

Recommendation Rename to “targetTimestampAtInterval”.

Client Comment *Agreed. Will change.*

```
390 uint256 currentTimestampAtInterval = _getIntervalTime(
       ↪ targetTimestamp);
```

CVF-26 FIXED

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

Description The expression “orderPool0For1.sellRateCurrent != 0” is calculated twice.

Recommendation Calculate once and reuse.

Client Comment *Acceptable.*

```
416     if (orderPool0For1.sellRateCurrent != 0 && orderPool1For0.
           ↪ sellRateCurrent != 0) {
```

```
436                 orderPool0For1.sellRateCurrent != 0
```

```
454 if (orderPool0For1.sellRateCurrent != 0 && orderPool1For0.
       ↪ sellRateCurrent != 0) {
```

```
474             orderPool0For1.sellRateCurrent != 0
```

CVF-27 INFO

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

Recommendation This commented out line should be uncommented or removed.

```
446 // console2.log("hasOutstandingOrders", _hasOutstandingOrders(self))
       ↪ ;
```

CVF-28 INFO

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

Recommendation According to the comment, this useless branch should be removed.

534
```
    } else {
        // @review This is now useless since timestamps are always
           ↪ on interval.
        self.orderPool0For1.advanceToCurrentTime(earningsFactorPool0
           ↪ );
        self.orderPool1For0.advanceToCurrentTime(earningsFactorPool1
           ↪ );
    }
```

612
```
} else {
    // @review This is now useless since timestamps are always on
       ↪ interval.
    orderPool.advanceToCurrentTime(accruedEarningsFactor);
}
```

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