Report

v. 1.0

Customer

Ekubo

Smart Contract Audit

Ekubo Protocol

1st April 2025

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	24.03.25	A. Zveryanskaya	Initial Draft
0.2	31.03.25	A. Zveryanskaya	Minor revision
1.0	1.04.25	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.

The Ekubo protocol vision is to provide a balance between the best swap execution and liquidity provider returns. The contracts are relentlessly optimized to be able to provide the most capital efficient liquidity ever at the lowest cost.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

/			
	Positions.sol	Core.sol	BaseURLTokenURI Generator.sol
	Router.sol		
types/			
	sqrtRatio.sol	positionKey.sol	position.sol
	poolKey.sol	feesPer Liquidity.sol	callPoints.sol
math/			
	ticks.sol	tickBitmap.sol	swap.sol
	sqrtRatio.sol	liquidity.sol	isPriceIncreasing.sol
	fee.sol	delta.sol	constants.sol
	bitmap.sol		
libraries/			
	ExposedStorageLib.sol	CoreLib.sol	
lens/			
	TokenDataFetcher.sol	Quote DataFetcher.sol	PriceFetcher.sol
	CoreDataFetcher.sol		
interfaces/			
	ITokenURIGenerator.sol	IFlash Accountant.sol	IExposedStorage.sol
	ICore.sol		

extensions/			
	Oracle.sol		
base/			
	UsesCore.sol	SlippageChecker.sol	Permittable.sol
	PayableMulticallable.sol	FlashAccountant.sol	ExposedStorage.sol
	BaseLocker.sol	BaseForwardee.sol	BaseExtension.sol

Then we reviewed the diff containing changes and fixes, as well as the new functionality, which includes the following files:

/			
	Oracle.sol	FlashAccountant.sol	ExposedStorageLib.sol
	BaseExtension.sol	callPoints.sol	MintableNFT.sol
	OracleLib.sol	Positions.sol	PriceFetcher.sol
	QuoteDataFetcher.sol	SlippageChecker.sol	

The fixes were reviewed under the CVF.Fix-2 numbering.

We did not include the non-security optimization issues to the report as some of the findings were deemed confidential.

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 critical, 6 major, and a few less important issues. All identified Critical and Major issues have been fixed.

Fixed 7 out of 7 issues

6 Critical Issues

CVF-1 FIXED

- **Category** Flaw
- **Source** FlashAccountant.sol

Description The actual size of packed data is calldatasize + 32.

```
79 // Call the original caller with the packed data
80 let success := call(gas(), caller(), 0, free, calldatasize(), 0, 0)
```

7 Major Issues

CVF-12 FIXED

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

Description Overflow is possible here.

Recommendation Perform an overflow check or clearly market the function as unsafe.

```
24 tick := mul(sub(rawIndex, 89421695), tickSpacing)
```

CVF-14 FIXED

- **Category** Flaw
- **Source** CoreLib.sol

Description The value returned here may be inaccurate, as “core.unsafeRead” may return garbage in some cases.

```
24  registered = uint256(core.unsafeRead(key)) != 0;
```

```
35  amountCollected = uint256(core.unsafeRead(key));
```

```
97  savedBalance = uint128(uint256(core.unsafeRead(key)) >> 128);
```

```
114 savedBalance0 = uint128(uint256(core.unsafeRead(key)) >> 128);
    savedBalance1 = uint128(uint256(core.unsafeRead(key)));
```

CVF-15 FIXED

- **Category** Flaw
- **Source** ExposedStorageLib.sol

Description These functions may silently return garbage in case the underlying call failed. Even worse, such a behavior could be triggered by the caller by providing not enough gas for the inner call.

Recommendation Check values returned by the “call” opcodes.

```
7  function unsafeRead(IExposedStorage target, bytes32 slot) internal
   ↪ view returns (bytes32 result) {
```

```
18 function unsafeReadTransient(IExposedStorage target, bytes32 slot)
   ↪ internal view returns (bytes32 result) {
```

CVF-20 FIXED

- **Category** Documentation
- **Source** QuoteDataFetcher.sol

Description This formula looks odd.

Recommendation Fix or clearly explain it.

```
52 int256(uint256(minTickSpacings)) * int256(uint256(poolKeys[i].
   ↪ tickSpacing())) * 256;
```

CVF-23 FIXED

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

Description The actual calldata length is “len+4”, as “len” itself is not passed.

```
64 if iszero(call(gas(), target, 0, result, add(len, 36), 0, 0)) {
```

```
97 if call(gas(), target, 0, result, add(len, 36), 0, 0) {
```

CVF-24 FIXED

- **Category** Documentation
- **Source** BaseLocker.sol

Description The selector signature doesn't match the actual argument count and types.

Recommendation Fix the selector of the arguments list, or clearly explain what is going on here.

Client Comment *The selector cannot be easily changed, since the extradata is just forwarded to the callback. Comment added.*

```
120 // selector of pay(address)
    mstore(free, shl(224, 0x0c11dedd))
    mstore(add(free, 4), token)
    mstore(add(free, 36), from)
    mstore(add(free, 68), amount)
```

8 Moderate Issues

CVF-3-9 INFO

- **Category** Procedural

Description This code assumes no garbage in unused bits of narrow types, which is not guaranteed by Solidity compiler.

Source: sqrtRatio.sol

```
45          gt(and(sqrtRatio, not(BIT_MASK)), TWO_POW_62_MINUS_ONE),
```

```
47          and(iszero(lt(sqrtRatio, MIN_SQRT_RATIO_RAW)), iszero(gt(
               ↪ sqrtRatio, MAX_SQRT_RATIO_RAW)))
```

```
92  r := shl(add(2, shr(89, and(sqrtRatio, BIT_MASK))), and(sqrtRatio,
       ↪ not(BIT_MASK)))
```

```
124 r := iszero(a)
```

```
130 r := xor(a, mul(xor(a, b), gt(b, a)))
```

```
136 r := xor(a, mul(xor(a, b), slt(b, a)))
```

Source: feesPerLiquidity.sol

```
25  mstore(result, div(shl(128, amount0), liquidity))
    mstore(add(result, 32), div(shl(128, amount1), liquidity))
```

Source: poolKey.sol

```
43  c := add(add(shl(96, _extension), shl(32, _fee)), _tickSpacing)
```

Source: ticks.sol

```
265 tickHigh = int32((logBaseTickSizeX128) >> 128);
```

```
267 tickLow = int32((logBaseTickSizeX128) >> 128);
```

Source: bitmap.sol

```
12  result := xor(bitmap, shl(index, 1))
```

```
18  yes := and(shr(index, bitmap), 1)
```

```
27     masked := and(bitmap, sub(shl(add(index, 1), 1), 1))
```

Source: fee.sol

```
8  result := shr(64, add(mul(amount, fee), 0xffffffffffffffff))
```

```
18 let v := shl(64, afterFee)
   let d := sub(0x10000000000000000, fee)
```

Source: isPriceIncreasing.sol

```
6  increasing := xor(isToken1, slt(amount, 0))
```

The Ethereum Virtual Machine operates on 256-bit words. As a consequence, all data within the EVM, irrespective of its declared type in Solidity, is ultimately stored in these 256-bit slots.

When Solidity types smaller than 256 bits, such as 'uint64', 'bytes16', or 'address', are stored, their values occupy the lower bits corresponding to their defined size. The remaining higher-order bits within the 256-bit word are not guaranteed to be automatically set to zero by the EVM or the Solidity compiler in all situations, particularly when these variables are accessed within inline assembly blocks.

While the Soldity compiler often takes care to clean these higher-order bits before performing high-level operations, this behavior is not assured within the context of inline assembly. Solidity's memory management system involves a "free memory pointer" and scratch space for temporary allocations. Memory locations might have been used in previous operations and their contents are not necessarily zeroed out.

This aspect of the EVM's memory model further reinforces the risk of relying on an assumption that unused bits will have a default value of zero when accessed in assembly.

Several locations in the codebase use assembly to perform bit operations on values smaller than 256 bits without properly masking them. This violates Solidity's documentation warning:

If you access variables of a type that spans less than 256 bits (for example 'uint64', 'address', or 'bytes16'), you cannot make any assumptions about bits not part of the encoding of the type. Especially, do not assume them to be zero.
Source: docs.soliditylang.org.

This issue is particularly concerning in functions that:

1. Read packed state variables from storage
2. Process these values without masking
3. Use them in calculations that affect the protocol's economic behavior

The most severe instances are:

1. In `CoreLib.sol`, when extracting liquidity from packed storage:

```
bytes32 p = core.unsafeRead(key);
assembly ("memory-safe") {
                // ...
    liquidity := shr(128, p)
}
```

1. In `Core.sol`'s `accumulateAsFees` function:

```
let liquidity := shr(128, sload(keccak256(0, 64)))
// ...
let v := div(shl(128, amount0), liquidity)
```

1. In swap event logging code:

```
mstore(add(o, 52), or(shl(128, delta0), and(delta1, 0
    ↪ xffffffffffffffffffffffffffffffff)))
```

Garbage bits can enter the system through several pathways:

1. **Storage slot packing**: The contract packs multiple values into single storage slots. For example, `sqrtRatio`, `tick`, and `liquidity` are all packed into a single slot. If one of these values is initially set with garbage bits outside its intended range, those bits will persist.
2. **Uninitialized memory**: When working with assembly, memory regions might not be fully initialized, allowing garbage bits to be read.
3. **Previous storage values**: If a storage slot previously held different values and is not completely overwritten, residual bits may remain.
4. **Integer cast overflow**: When larger integers are cast to smaller types in Solidity, the higher-order bits are truncated. However, in assembly, this truncation does not happen automatically unless explicitly masked.

This vulnerability can potentially lead to:

1. **Incorrect fee calculations**: In `accumulateAsFees`, corrupted liquidity values could lead to incorrect fee accounting, potentially cheating LPs of their rightful earnings.
2. **Corrupted protocol state**: As incorrect values propagate through calculations, they can corrupt the overall state of pools.
3. **Misleading event data**: Off-chain applications relying on emitted events may make incorrect decisions based on corrupted data.

Recommendation Add explicit masking when extracting or manipulating smaller types in assembly. For instance:

```
// Before
liquidity := shr(128, p)

// After
liquidity := and(shr(128, p), 0xffffffffffffffffffffffffffffffff)
```

Client Comment *Acknowledged. Clearing values costs gas. We do not clear values when the value is never expected to have dirty upper bits. Notably we use abicoder v2 which prevents the contract from accepting user inputs to external functions that do not conform to the specified types.*

CVF-11 INFO

- **Category** Procedural
- **Source** tickBitmap.sol

Description This code assumes no garbage in unused bits of narrow types, which is not guaranteed by Solidity compiler. See warning here: https://docs.solidity-lang.org/en/v0.8.28/assembly.html#access-to-external-variables-functions-and-libraries

Recommendation Properly clear the values before use.

Client Comment *Acknowledged. Clearing values costs gas. We do not clear values when the value is never expected to have dirty upper bits. Notably we use abicoder v2 which prevents the contract from accepting user inputs to external functions that do not conform to the specified types.*

```
14 let rawIndex := add(sub(sdiv(tick, tickSpacing), slt(smod(tick,
     ↪ tickSpacing), 0)), 89421695)
```

CVF-21 INFO

- **Category** Procedural
- **Source** QuoteDataFetcher.sol

Description This code assumes no garbage in unused bits of narrow types, which is not guaranteed by Solidity compiler. See warning here: https://docs.soliditylang.org/en/v0.8.28/assembly.html#access-to-external-variables-functions-and-libraries

Recommendation Properly clear the values before use.

Client Comment *Acknowledged. Clearing values costs gas. We do not clear values when the value is never expected to have dirty upper bits. Notably we use abicoder v2 which prevents the contract from accepting user inputs to external functions that do not conform to the specified types.*

```
101 v := or(shl(128, tick), and(liquidityDelta, 0
    ↪ xffffffffffffffffffffffffffffffff))
```

CVF-25 INFO

- **Category** Procedural
- **Source** FlashAccountant.sol

Description It is a bad practice to rely on callers for safety purposes.

Recommendation Just add an explicit range check for “debtChange”.

Client Comment *Acknowledged. We cannot introduce assertions without increasing gas costs. This is an issue with solidity.*

```
39 // This must be enforced at the places it is called for this
    ↪ contract's safety
```

CVF-26 INFO

- **Category** Procedural
- **Source** FlashAccountant.sol

Description This code assumes no garbage in unused bits of narrow types, which is not guaranteed by Solidity compiler. See warning here: https://docs.soliditylang.org/en/v0.8.28/assembly.html#access-to-external-variables-functions-and-libraries

Recommendation Properly clear the values before use.

Client Comment *Acknowledged. Clearing values costs gas. We do not clear values when the value is never expected to have dirty upper bits. Notably we use abicoder v2 which prevents the contract from accepting user inputs to external functions that do not conform to the specified types.*

```
44 mstore(0, add(add(shl(160, id), token), _DEBT_HASH_OFFSET))
```

CVF-28 INFO

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

Description Overflow is possible when converting type.

Recommendation Use safe conversion.

Client Comment *Same as line 26. Acknowledged. We cannot introduce assertions without increasing gas costs. This is an issue with solidity.*

```
220 _accountDebt(id, token, int256(uint256(amount)));
```

CVF-29 INFO

- **Category** Procedural
- **Source** Positions.sol

Description This code assumes no garbage in unused bits of narrow types, which is not guaranteed by Solidity compiler. See warning here: https://docs.soliditylang.org/en/v0.8.28/assembly.html#access-to-external-variables-functions-and-libraries

Recommendation Properly clear the values before use.

Client Comment *Acknowledged. Clearing values costs gas. We do not clear values when the value is never expected to have dirty upper bits. Notably we use abicoder v2 which prevents the contract from accepting user inputs to external functions that do not conform to the specified types.*

```
57 mstore(free, minter)
```

CVF-30 INFO

- **Category** Procedural
- **Source** Core.sol

Description This code assumes no garbage in unused bits of narrow types, which is not guaranteed by Solidity compiler. See warning here: https://docs.soliditylang.org/en/v0.8.28/assembly.html#access-to-external-variables-functions-and-libraries

Recommendation Properly clear the values before use.

Client Comment *Acknowledged. Clearing values costs gas. We do not clear values when the value is never expected to have dirty upper bits. Notably we use abicoder v2 which prevents the contract from accepting user inputs to external functions that do not conform to the specified types.*

```
510 let v := div(shl(128, mload(add(result, 96))), liquidity)
```

CVF-31 INFO

- **Category** Procedural
- **Source** Core.sol

Description This code assumes no garbage in unused bits of narrow types, which is not guaranteed by Solidity compiler. See warning here: https://docs.soliditylang.org/en/v0.8.28/assembly.html#access-to-external-variables-functions-and-libraries

Recommendation Properly clear the values before use.

Client Comment *Acknowledged. Clearing values costs gas. We do not clear values when the value is never expected to have dirty upper bits. Notably we use abicoder v2 which prevents the contract from accepting user inputs to external functions that do not conform to the specified types.*

```
560 sstore(keccak256(0, 64), add(add(sqrtRatio, shl(96, and(tick, 0
    ↪ xffffffff))), shl(128, liquidity)))
```

CVF-43 FIXED

- **Category** Documentation
- **Source** bitmap.sol

Description This description is confusing, as there could be several such bits.

Recommendation Elaborate more.

```
22 // Returns the index of the bit that is set and less or equally
    ↪ significant to index, or 256 if no such bit exists.
```

```
33 // Returns the index of the next bit that is set and more or equally
    ↪  significant to index, or 256 if no such bit exists.
```

CVF-44 FIXED

- **Category** Documentation
- **Source** fee.sol

Description Actually, in the code, fee is a 64-bit value encoding a 0.64 number.

Recommendation Fix the comment.

```
4 // Returns the fee to charge based on the amount, which is the fee (
    ↪ a 0.128 number) times the
```

CVF-45 INFO

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

Description Overflow is possible when converting to “int256”.

Recommendation Use safe conversion.

Client Comment *Overflow is not possible because amount0Delta and amount1Delta return uint128 values.*

```
34 sign * int256(uint256(amount0Delta(sqrtRatioLower, sqrtRatioUpper,
      ↪ magnitude, isPositive)))
```

```
38 sign * int256(uint256(amount0Delta(sqrtRatio, sqrtRatioUpper,
      ↪ magnitude, isPositive)))
```

```
41 sign * int256(uint256(amount1Delta(sqrtRatioLower, sqrtRatio,
      ↪ magnitude, isPositive)))
```

```
45 sign * int256(uint256(amount1Delta(sqrtRatioLower, sqrtRatioUpper,
      ↪ magnitude, isPositive)))
```

CVF-46 FIXED

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

Description There are no range checks for the arguments.

Recommendation Implement proper checks or clearly marks the function as unsafe.

```
21 function bitmapWordAndIndexToTick(uint256 word, uint256 index,
      ↪ uint32 tickSpacing) pure returns (int32 tick) {
```

CVF-47 INFO

- **Category** Documentation
- **Source** constants.sol

Description This value is less than the maximum tick whose corresponding sqrt price fits into 192 bits.

Recommendation Explain, why this particular number was chosen.

Client Comment *Acknowledged. Prefer to leave this out of the code. We choose a tick spacing and min/max tick such that we can store all the initialized ticks in a single bitmap for max tick spacing and also so that the price cannot go outside of boundaries for that max tick spacing (MAX_TICK and MIN_TICK are multiples of MAX_TICK_SPACING). Understand we gave up some of our maximum supported price range, but those prices are not realistic for any token that is practically traded on an AMM.*

```
5 int32 constant MAX_TICK = 88722835;
```

CVF.Fix-2.1 INFO

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

Description This code assumes no garbage in unused bits of narrow types, which is not guaranteed by Solidity compiler. See warning here: https://docs.solidity-lang.org/en/v0.8.28/assembly.html#access-to-external-variables-functions-and-libraries

Recommendation Properly clear the values before use or hash only used bits.

Client Comment *Acknowledged. Clearing upper bits has additional cost that we avoid. The token variable here _only_ comes from an external poolKey argument, and with abicoder v2 we know this argument cannot include garbage upper bits.*

```
115 +mstore(0, token)
```

CVF.Fix-2.2 INFO

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

Description This code assumes no garbage in unused bits of narrow types, which is not guaranteed by Solidity compiler. See warning here: https://docs.soliditylang.org/en/v0.8.28/assembly.html#access-to-external-variables-functions-and-libraries

Recommendation Properly clear the values before use.

Client Comment *Acknowledged. Clearing upper bits has additional cost that we avoid. locker in this line comes from _requireLocker which loads the locker from transient storage and does clean the upper bits.*

```
117 +tstore(_CURRENT_LOCKER_SLOT, or(shl(160, add(id, 1)), to))
```

CVF.Fix-2.3 FIXED

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

Description The comment directly contradicts with the code, as the code indeed reverts in case the called contract fails.

Recommendation Fix the code.

```
7  +///      an implementation of ExposedStorage that will never fail.
   ↪ The methods in this library will not revert if the called
8  +///      contract fails for any reason.
```

```
15 +            if iszero(staticcall(gas(), target, 0, 36, 0, 32)) {
   ↪ revert(0, 0) }
```

```
33 +            if iszero(staticcall(gas(), target, o, 100, o, 96)) {
   ↪ revert(0, 0) }
```

```
46 +            if iszero(staticcall(gas(), target, 0, 36, 0, 32)) {
   ↪ revert(0, 0) }
```

CVF.Fix-2.4 FIXED

- **Category** Documentation
- **Source** Oracle.sol

Description This code looks unsafe, while actually it is not.

Recommendation Clearly explain why underflow is fine here.

```
228 +uint32 targetDiff = current - uint32(time);
```

```
237 +    if (current - midSnapshot.timestamp >= targetDiff) {
```

```
275 +uint32 timePassed = uint32(atTime) - snapshot.timestamp;
```

```
290 +        uint32 timestampDifference = next.timestamp - snapshot.
    ↪ timestamp;
```

9 Recommendations

CVF-57 INFO

- **Category** Procedural
- **Source** position.sol

Description Specifying a particular compiler version makes it harder upgrading to newer versions.

Recommendation Specify as "^0.8.0" or as "^0.8.28" if there is something special regarding this particular version. Also relevant for:sqrtRatio.sol, feesPerLiquidity.sol, positionKey.sol, poolKey.sol, callPoints.sol, delta.sol, ticks.sol, bitmap.sol, fee.sol, isPriceIncreasing.sol, liquidity.sol, sqrtRatio.sol, swap.sol, tickBitmap.sol, constants.sol, CoreLib.sol, ExposedStorageLib.sol, CoreDataFetcher.sol, TokenDataFetcher.sol, PriceFetcher.sol, QuoteDataFetcher.sol, ITokenURIGenerator.sol, ICore.sol, IExposedStorage.sol, Oracle.sol, SlippageChecker.sol, Permittable.sol, PayableMulticallable.sol, ExposedStorage.sol, BaseLocker.sol, BaseExtension.sol, UsesCore.sol, FlashAccountant.sol, BaseForwardee.sol, BaseURLTokenURIGenerator.sol, Positions.sol, Router.sol, Core.sol.

Client Comment *Acknowledged. Because of our assumptions about the compiler behavior, e.g. especially regarding the presence of dirty bits of narrow types, we prefer to fix a specific solidity compiler version.*

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

CVF-58 INFO

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

Description We didn't review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

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

CVF-59 FIXED

- **Category** Documentation
- **Source** position.sol

Description The semantics of the returned values is unclear.

Recommendation Give descriptive names to the returned values and/or explain in a documentation comment.

```
16 returns (uint128, uint128)
```

CVF-60 INFO

- **Category** Procedural
- **Source** position.sol

Description We didn't review this function.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

```
21 uint128(FixedPointMathLib.fullMulDivN(difference.value0, position.
      ↪ liquidity, 128)),
   uint128(FixedPointMathLib.fullMulDivN(difference.value1, position.
      ↪ liquidity, 128))
```

CVF-68 INFO

- **Category** Procedural
- **Source** poolKey.sol

Description Bitwise “and” operations are redundant, as Solidity compiler permits garbage in higher bits of narrow data types. See warning in this section: https://docs.soliditylang.org/en/v0.8.28/assembly.html#access-to-external-variables-functions-and-libraries

Client Comment *It’s not redundant because elsewhere in our code we assume that there are not dirty bits for these values. It would not be good for a function to return a uint32 that has dirty upper bits. I understand that this is consistent with your view that you should always clear dirty bits, but it’s cheaper to do the bitwise ”and” when producing the stack variable than it is to do upper bit clearing every time you use the narrow type.*

```
13 r := and(mload(add(64, pk)), 0xffffffff)
```

```
19 r := and(mload(add(60, pk)), 0xffffffffffffffff)
```

```
25 r := and(mload(add(52, pk)), 0
     ↪ xffffffffffffffffffffffffffffffffffffffff)
```

CVF-69 INFO

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

Description The type for these fields should be “iERC20”.

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
49 address token0;
50 address token1;
```

CVF-74 INFO

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

Description We didn't review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

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

CVF-76 FIXED

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

Description The name is confusing, as this function not only sorts rates, but also converts them into fixed-point format.

Recommendation Choose better name.

```
10 function sortSqrtRatios(SqrtRatio sqrtRatioA, SqrtRatio sqrtRatioB)
```

CVF-78 INFO

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

Description We didn't review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

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

CVF-81 INFO

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

Description We didn't review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

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

CVF-87 INFO

- **Category** Procedural
- **Source** liquidity.sol

Description We didn't review these files.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

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

CVF-88 INFO

- **Category** Readability
- **Source** liquidity.sol

Description Here, "uint256(-1)" would be much more readable than "type(uint256.max)".

Client Comment *That produces Error (9640): Explicit type conversion not allowed from "int_const -1" to "uint256".*

```
28 int256 sign = int256(FixedPointMathLib.ternary(isPositive, 1, type(
    ↪ uint256).max));
```

CVF-90 INFO

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

Description We didn't review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

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

CVF-93 INFO

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

Description We didn't review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

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

8 import {SafeCastLib} from "solady/utils/SafeCastLib.sol";
```

CVF-96 INFO

- **Category** Readability
- **Source** swap.sol

Description Should be "else if" for readability.

Client Comment *Acknowledged, not changing, this is purely code style.*

```
108 if (sqrtRatioNextFromAmount == sqrtRatio) {
```

CVF-99 INFO

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

Description We didn't review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

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

CVF-105 INFO

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

Description We didn't review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

```
10 import {EfficientHashLib} from "solady/utils/EfficientHashLib.sol";
```

CVF-106 INFO

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

Description The type for the "extension" argument should be "IExtension".

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
16 function isExtensionRegistered(ICore core, address extension)
    ↪ internal view returns (bool registered) {
```

CVF-107 INFO

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

Description The storage offsets used here should be named constants.

Client Comment *Acknowledged, will not change this though.*

```
20  mstore(32, 0)
31  mstore(32, 1)
46  mstore(32, 2)
67  mstore(32, 4)
93  mstore(32, 8)
110 mstore(32, 8)
126 mstore(32, 5)
```

CVF-108 INFO

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

Description The type for the token arguments should be “IERC20”.

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
27 function protocolFeesCollected(ICore core, address token) internal
   ↪ view returns (uint256 amountCollected) {
```

```
80 function savedBalances(ICore core, address owner, address token,
   ↪ bytes32 salt)
```

```
100 function savedBalances(ICore core, address owner, address token0,
   ↪ address token1, bytes32 salt)
```

```
153 function save(ICore core, address owner, address token, bytes32 salt
   ↪ , uint128 amount) internal {
```

```
157 function load(ICore core, address token, bytes32 salt, uint128
   ↪ amount) internal {
```

CVF-110 INFO

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

Description Despite the name, this function returns only one saved balance.

Recommendation Rename the function.

Client Comment *Acknowledged, will not change though.*

```
80 function savedBalances(ICore core, address owner, address token,
   ↪ bytes32 salt)
```

CVF-113 INFO

- **Category** Readability
- **Source** CoreDataFetcher.sol

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

Client Comment *The constructor line is required because of the base class constructor. I've never seen a comment on what is obviously a required line of code.*

```
15 constructor(ICore core) UsesCore(core) {}
```

CVF-114 INFO

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

Description The argument type should be "IExtension".

Client Comment *Acknowledged, will not change though.*

```
17 function isExtensionRegistered(address extension) external view
   ↪ returns (bool registered) {
```

CVF-115 INFO

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

Description The type for the "token" arguments should be "IERC20".

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
21 function protocolFeesCollected(address token) external view returns
   ↪ (uint256 amountCollected) {
```

```
47 function savedBalances(address owner, address token, bytes32 salt)
   ↪ external view returns (uint256 savedBalance) {
```

CVF-117 INFO

- **Category** Procedural
- **Source** TokenDataFetcher.sol

Description We didn't review these files.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

```
4 import {SafeTransferLib} from "solady/utils/SafeTransferLib.sol";
  import {DynamicArrayLib} from "solady/utils/DynamicArrayLib.sol";
```

```
7 import {IERC20} from "forge-std/interfaces/IERC20.sol";
```

CVF-118 INFO

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

Description The type for these fields should be "IERC20".

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
13 address token;
```

```
18 address token;
```

CVF-119 INFO

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

Description The type for the "tokens" argument should be "IERC20".

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
23 function getNonzeroBalancesAndAllowances(address owner, address[]
     ↪ memory tokens, address[] memory spenders)
```

CVF-124 INFO

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

Description We didn't review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

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

CVF-130 INFO

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

Description The type for the token arguments should be "IERC20".

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
44  function getEarliestSnapshotTimestamp(address token) private view
       ↪ returns (uint256) {
```

```
56  function getMaximumObservationPeriod(address token) private view
       ↪ returns (uint32) {
```

```
68  function getAveragesOverPeriod(address baseToken, address quoteToken
       ↪ , uint64 startTime, uint64 endTime)
```

```
108     address baseToken,
        address quoteToken,
```

```
165     address baseToken,
        address quoteToken,
```

```
196     address baseToken,
        address quoteToken,
```

CVF-131 INFO

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

Description We didn't review the "sqrt" function.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

```
101 uint128(FixedPointMathLib.sqrt(uint256(amountBase) * uint256(
    ↪ amountQuote))), quote.tick - base.tick
```

```
156     uint128(FixedPointMathLib.sqrt(uint256(amountBase) * uint256(
        ↪ amountQuote))),
```

CVF-135 INFO

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

Description The type for the "baseTokens" argument should be "IERC20[]".

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
221 function getOracleTokenAverages(uint64 observationPeriod, address[]
    ↪ memory baseTokens)
```

CVF-138 INFO

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

Description We didn't review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

```
12 import {DynamicArrayLib} from "solady/utils/DynamicArrayLib.sol";
```

CVF-140 INFO

- **Category** Readability
- **Source** QuoteDataFetcher.sol

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

Client Comment *Not for constructors, it's obviously empty because we need to call the base constructors.*

```
34 constructor(ICore core) UsesCore(core) {}
```

CVF-141 FIXED

- **Category** Documentation
- **Source** QuoteDataFetcher.sol

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

Recommendation Explain in a documentation comment.

```
36 function getQuoteData(PoolKey[] calldata poolKeys, uint32
   ↪ minTickSpacings)
```

CVF-147 INFO

- **Category** Readability
- **Source** ICore.sol

Description This interface should be moved into a separate file named "IExtension.sol".

Client Comment *Extensions are only used with Core, so we chose to colocate the interface.*

```
18 interface IExtension {
```

CVF-148 INFO

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

Description The type for the “locker” arguments should be “ILocker”.

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
22 function beforeUpdatePosition(address locker, PoolKey memory poolKey
   ↪ , UpdatePositionParameters memory params)
```

```
25     address locker,
```

```
33     address locker,
```

```
41     address locker,
```

```
51 function beforeCollectFees(address locker, PoolKey memory poolKey,
   ↪ bytes32 salt, Bounds memory bounds) external;
```

```
53     address locker,
```

CVF-149 INFO

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

Description Events are usually named via nouns, such as “ProtocolFeesWithdrawal” or “Extension”.

Client Comment *We have too many integrations in progress to change event names to match some unwritten convention.*

```
63 event ProtocolFeesWithdrawn(address recipient, address token,
      ↪ uint256 amount);
   event ExtensionRegistered(address extension);
   event PoolInitialized(bytes32 poolId, PoolKey poolKey, int32 tick,
      ↪ SqrtRatio sqrtRatio);
   event PositionFeesCollected(bytes32 poolId, PositionKey positionKey,
      ↪  uint128 amount0, uint128 amount1);
   event FeesAccumulated(bytes32 poolId, uint128 amount0, uint128
      ↪ amount1);
   event PositionUpdated(
```

CVF-151 INFO

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

Description The type for rhe “token” parameter should be “IERC20”.

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
63 event ProtocolFeesWithdrawn(address recipient, address token,
      ↪ uint256 amount);
```

CVF-152 INFO

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

Description The type for the “extension” parameter should be “IExtension”.

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
64 event ExtensionRegistered(address extension);
```

CVF-153 INFO

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

Description The type for the “locker” parameter should be “ILocker”.

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
69 address locker, bytes32 poolId, UpdatePositionParameters params,
      ↪ int128 delta0, int128 delta1
```

CVF-155 INFO

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

Description The type for token arguments should be “IERC20”.

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
86 function withdrawProtocolFees(address recipient, address token,
   ↪ uint256 amount) external;
```

```
105 function load(address token0, address token1, bytes32 salt, uint128
   ↪ amount0, uint128 amount1) external;
```

```
108 function save(address owner, address token0, address token1, bytes32
   ↪  salt, uint128 amount0, uint128 amount1)
```

CVF-156 INFO

- **Category** Readability
- **Source** IFlashAccountant.sol

Description This interface should be moved into a separate file named “ILocker.sol”.

Client Comment *This is only used in the context of IFlashAccountant so it makes sense to keep it here.*

```
4 interface ILocker {
```

CVF-157 INFO

- **Category** Readability
- **Source** IFlashAccountant.sol

Description This interface should be moved into a separate file named “IForwardee.sol”.

Client Comment *This is only used in the context of IFlashAccountant so it makes sense to keep it here.*

```
8 interface IForwardee {
```

CVF-158 INFO

- **Category** Readability
- **Source** IFlashAccountant.sol

Description This interface should be moved into a separate file named “IPayer.sol”.

Client Comment *This is only used in the context of IFlashAccountant so it makes sense to keep it here.*

```
12 interface IPayer {
```

CVF-160 INFO

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

Description The type for the “token” arguments should be “IERC20”.

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
44 function pay(address token) external returns (uint128 payment);
```

```
48 function withdraw(address token, address recipient, uint128 amount)
   ↪ external;
```

CVF-163 INFO

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

Description We didn’t review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

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

CVF-164 INFO

- **Category** Readability
- **Source** Oracle.sol

Description Defining a top-level function in a file named after a contract makes it harder navigating through code.

Recommendation Move the function into the contract of move it into a separate file.

Client Comment *Acknowledged. We will keep it here.*

```
15 function oracleCallPoints() pure returns (CallPoints memory) {
```

CVF-168 INFO

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

Description The type for the “token” arguments should be “IERC20”.

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
71  function _emitSnapshotEvent(address token, Snapshot memory snapshot)
    ↪  private {

86  function getPoolKey(address token) public view returns (PoolKey
    ↪ memory) {

110 function maybeInsertSnapshot(bytes32 poolId, address token) private
    ↪ {

187 function expandCapacity(address token, uint64 minCapacity) external
    ↪ returns (uint64 capacity) {

205 function _getSnapshotLogical(Counts memory c, address token, uint256
    ↪  logicalIndex)

218     address token,

251 function findPreviousSnapshot(address token, uint64 time)

264     address token,

304 function extrapolateSnapshot(address token, uint64 atTime)

321 function getExtrapolatedSnapshotsForSortedTimestamps(address token,
    ↪ uint64[] memory timestamps)
```

CVF-171 INFO

- **Category** Readability
- **Source** Oracle.sol

Description Unnamed arguments make code harder to read.

Recommendation Give names to the unnamed arguments.

Client Comment *Named unused arguments would cause compiler warnings, in this case it's clearer they are not used.*

```
149 function beforeInitializePool(address, PoolKey calldata key, int32)
    ↪ external override onlyCore {
```

```
166 function beforeUpdatePosition(address, PoolKey memory poolKey,
    ↪ UpdatePositionParameters memory params)
```

```
176 function beforeSwap(address, PoolKey memory poolKey, int128 amount,
    ↪ bool, SqrtRatio, uint256)
```

```
187 function expandCapacity(address token, uint64 minCapacity) external
    ↪ returns (uint64 capacity) {
```

CVF-172 FIXED

- **Category** Documentation
- **Source** Oracle.sol

Description It is unclear what exactly a logical index is.

Recommendation Elaborate more.

```
204 // Given a logical index [0, count), returns the snapshot from
    ↪ storage
```

CVF-174 INFO

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

Description We didn't review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

```
4 import {SafeTransferLib} from "solady/utils/SafeTransferLib.sol";
```

CVF-175 INFO

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

Description The type for the "token" parameters should be "IERC20".

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
12 error MinimumOutputNotReceived(address token, uint256 minimumOutput)
       ↪ ;
   error MaximumInputExceeded(address token, uint256 maximumInput);
```

CVF-176 INFO

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

Description The type for the “token” arguments should be “IERC20”.

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
15 function balanceKey(address token, address account) private pure
   ↪ returns (bytes32 key) {
```

```
24 function getRecordedBalance(address token, address account) private
   ↪ view returns (uint256 prev) {
```

```
31 function getBalance(address token, address account) private view
   ↪ returns (uint256 balance) {
```

```
39 function recordBalanceForSlippageCheck(address token) external
   ↪ payable {
```

```
51 function checkMinimumOutputReceived(address token, uint256
   ↪ minimumOutput) external payable {
```

```
61 function checkMaximumInputNotExceeded(address token, uint256
   ↪ maximumInput) external payable {
```

CVF-177 FIXED

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

Description The base key address should be a named constant.

```
19 //
20 0x2ea13d3f0340a613d1765d6e239004eca4cb7efa2e253d1e113c4d333b8db7c8
   ↪ == `cast keccak "SlippageChecker#balanceKey"`
key := add(keccak256(0, 64),
0x2ea13d3f0340a613d1765d6e239004eca4cb7efa2e253d1e113c4d333b8db7c8)
```

CVF-179 INFO

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

Description The type for the “token” argument should be “IERC20”.

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
11 function permit(address token, uint256 amount, uint256 deadline,
   ↪ uint8 v, bytes32 r, bytes32 s) external payable {
```

CVF-180 INFO

- **Category** Procedural
- **Source** Permittable.sol

Description We didn’t review this function.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

```
12 SafeTransferLib.permit2(token, msg.sender, address(this), amount,
   ↪ deadline, v, r, s);
```

CVF-182 INFO

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

Description We didn’t review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

```
4 import {Multicallable} from "solady/utils/Multicallable.sol";
```

CVF-183 INFO

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

Description We didn't review these functions.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

```
8 _multicallDirectReturn(_multicall(data));
```

CVF-186 INFO

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

Description We didn't review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

```
6 import {SafeTransferLib} from "solady/utils/SafeTransferLib.sol";
```

CVF-187 INFO

- **Category** Readability
- **Source** BaseLocker.sol

Description Unnamed argument make code harder to read.

Recommendation Give names to unnamed arguments.

Client Comment *That would trigger a solidity compiler warning since the variable is not used.*

```
32 function payCallback(uint256, address token) external {
```

CVF-188 INFO

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

Description The type for the “token” argument should be “IERC20”.

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
32  function payCallback(uint256, address token) external {
```

```
111 function pay(address from, address token, uint256 amount) internal {
```

```
168 function withdraw(address token, uint128 amount, address recipient)
      ↪ internal {
```

CVF-190 FIXED

- **Category** Readability
- **Source** BaseLocker.sol

Description This assignment should be moved into the conditional statement below.

```
112 address target = address(accountant);
```

CVF-193 INFO

- **Category** Readability
- **Source** BaseExtension.sol

Description Unnamed arguments made code harder to read.

Recommendation Give names to the arguments.

Client Comment *That would trigger a solidity compiler warning since they are not used in the base extension.*

```
20 function beforeInitializePool(address, PoolKey calldata, int32)
   ↪ external virtual {

24 function afterInitializePool(address, PoolKey calldata, int32,
   ↪ SqrtRatio) external virtual {

28 function beforeUpdatePosition(address, PoolKey memory,
   ↪ UpdatePositionParameters memory) external virtual {

32 function afterUpdatePosition(address, PoolKey memory,
   ↪ UpdatePositionParameters memory, int128, int128)

39 function beforeSwap(address, PoolKey memory, int128, bool, SqrtRatio
   ↪ , uint256) external virtual {

43 function afterSwap(address, PoolKey memory, int128, bool, SqrtRatio,
   ↪  uint256, int128, int128) external virtual {

47 function beforeCollectFees(address, PoolKey memory, bytes32, Bounds
   ↪ memory) external virtual {

51 function afterCollectFees(address, PoolKey memory, bytes32, Bounds
   ↪ memory, uint128, uint128) external virtual {
```

CVF-196 INFO

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

Description We didn't review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

```
6 import {SafeTransferLib} from "solady/utils/SafeTransferLib.sol";
```

CVF-199 INFO

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

Description The type for the "token" arguments should be "IERC20".

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
41  function _accountDebt(uint256 id, address token, int256 debtChange)
      ↪ internal {

142 function pay(address token) external returns (uint128 payment) {

217 function withdraw(address token, address recipient, uint128 amount)
      ↪ external {
```

CVF-206 INFO

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

Description We didn't review these files.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

```
4 import {Test} from "forge-std/Test.sol";
```

```
6 import {LibString} from "solady/utils/LibString.sol";
  import {Ownable} from "solady/auth/Ownable.sol";
```

CVF-207 INFO

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

Description The type for this variable should be "ITokenURIGenerator".

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
11 address public replacementContract;
```

CVF-208 INFO

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

Description These functions should emit some events.

Client Comment *Not important for how we use them.*

```
18 function setBaseURL(string memory _baseURL) external onlyOwner {
```

```
22 function setReplacementContract(address _replacementContract)
     ↪ external onlyOwner {
```

CVF-209 INFO

- **Category** Readability
- **Source** BaseURLTokenURIGenerator.sol

Description Should be “else return” for readability.

Client Comment *Contract is low importance.*

```
31 return string(abi.encodePacked(baseURL, LibString.toString(id)));
```

CVF-211 INFO

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

Description We didn’t review these files.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

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

```
20 import {SafeCastLib} from "solady/utils/SafeCastLib.sol";
```

CVF-212 INFO

- **Category** Readability
- **Source** Positions.sol

Description Imports from different directories and even from third-party libraries are intermixed.

Recommendation Group imports by source.

Client Comment *Sorting imports is an exercise in vanity. If anyone cared then forge fmt would do it https://github.com/foundry-rs/foundry/issues/3396*

```
4  import {ERC721} from "solady/tokens/ERC721.sol";
   import {BaseLocker} from "./base/BaseLocker.sol";
   import {UsesCore} from "./base/UsesCore.sol";
   import {ICore, UpdatePositionParameters} from "./interfaces/ICore.
      ↪ sol";
   import {CoreLib} from "./libraries/CoreLib.sol";
   import {PoolKey} from "./types/poolKey.sol";
10 import {PositionKey, Bounds} from "./types/positionKey.sol";
   import {FeesPerLiquidity} from "./types/feesPerLiquidity.sol";
   import {Position} from "./types/position.sol";
   import {tickToSqrtRatio} from "./math/ticks.sol";
   import {maxLiquidity, liquidityDeltaToAmountDelta} from "./math/
      ↪ liquidity.sol";
   import {PayableMulticallable} from "./base/PayableMulticallable.sol"
      ↪ ;
   import {Permittable} from "./base/Permittable.sol";
   import {SlippageChecker} from "./base/SlippageChecker.sol";
   import {ITokenURIGenerator} from "./interfaces/ITokenURIGenerator.
      ↪ sol";
   import {SqrtRatio} from "./types/sqrtRatio.sol";
20 import {SafeCastLib} from "solady/utils/SafeCastLib.sol";
```

CVF-214 INFO

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

Description The value “0xdd” should be a named constant.

Client Comment *Acknowledged. These changes are inconsequential so we will not make them.*

```
135     abi.decode(lock(abi.encode(bytes1(0xdd), msg.sender, id, poolKey
           ↪ , bounds, liquidity)), (uint128, uint128));
```

```
224 if (callType == 0xdd) {
```

CVF-215 INFO

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

Description The value “0xff” should be a named constant.

Client Comment *Acknowledged. These changes are inconsequential so we will not make them.*

```
165     lock(abi.encode(bytes1(0xff), id, poolKey, bounds, liquidity,
           ↪ recipient, withFees)), (uint128, uint128)
```

```
239 } else if (callType == 0xff) {
```

CVF-216 FIXED

- **Category** Documentation
- **Source** Positions.sol

Description Actually, a narrow range of possible token IDs allows anybody to re-mint the token.

Recommendation Rephrase the comment.

```
178 // The NFT ID can be re-minted by the original minter after it is
       ↪ burned
```

CVF-218 INFO

- **Category** Readability
- **Source** Router.sol

Description Imports from different directories and even from third-party libraries are intermixed.

Recommendation Group imports by source.

Client Comment *Sorting imports is an exercise in vanity. If anyone cared then forge fmt would do it https://github.com/foundry-rs/foundry/issues/3396*

```
4  import {PayableMulticallable} from "./base/PayableMulticallable.sol"
       ↪ ;
   import {BaseLocker} from "./base/BaseLocker.sol";
   import {UsesCore} from "./base/UsesCore.sol";
   import {ICore} from "./interfaces/ICore.sol";
   import {PoolKey} from "./types/poolKey.sol";
   import {NATIVE_TOKEN_ADDRESS} from "./math/constants.sol";
10 import {isPriceIncreasing} from "./math/isPriceIncreasing.sol";
   import {Permittable} from "./base/Permittable.sol";
   import {SlippageChecker} from "./base/SlippageChecker.sol";
   import {SqrtRatio, toSqrtRatio} from "./types/sqrtRatio.sol";
   import {MIN_SQRT_RATIO, MAX_SQRT_RATIO} from "./types/sqrtRatio.sol"
       ↪ ;
   import {FixedPointMathLib} from "solady/utils/FixedPointMathLib.sol"
       ↪ ;
   import {CoreLib} from "./libraries/CoreLib.sol";
```

CVF-219 INFO

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

Description We didn't review this file.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

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

CVF-221 INFO

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

Description The type for this field should be “IERC20”.

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
25 address token;
```

CVF-222 INFO

- **Category** Readability
- **Source** Router.sol

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

Client Comment *Not needed for constructor.*

```
46 constructor(ICore core) BaseLocker(core) UsesCore(core) {}
```

CVF-223 INFO

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

Description The value “0x00” should be a named constant.

Client Comment *Acknowledged. These changes are inconsequential so we will not make them.*

```
51 if (callType == bytes1(0x00)) {
```

```
230             bytes1(0x00),
```

CVF-224 INFO

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

Description The value 0x01 should be a named constant.

Client Comment *Acknowledged. These changes are inconsequential so we will not make them.*

```
105 } else if (callType == bytes1(0x01) || callType == bytes1(0x02)) {
```

```
110     if (callType == bytes1(0x01)) {
```

```
202     if (callType == bytes1(0x01)) {
```

```
286 result = abi.decode(lock(abi.encode(bytes1(0x01), msg.sender, s,
    ↪ calculatedAmountThreshold)), (Delta[]));
```

CVF-225 INFO

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

Description The value “0x02” should be a named constant.

Client Comment *Acknowledged. These changes are inconsequential so we will not make them.*

```
105 } else if (callType == bytes1(0x01) || callType == bytes1(0x02)) {
```

```
294 results = abi.decode(lock(abi.encode(bytes1(0x02), msg.sender, swaps
    ↪ , calculatedAmountThreshold)), (Delta[][]));
```

CVF-226 INFO

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

Description The value “0x03” should be a named constant.

Client Comment *Acknowledged. These changes are inconsequential so we will not make them.*

207
```
} else if (callType == bytes1(0x03)) {
```

304
```
    lockAndExpectRevert(abi.encode(bytes1(0x03), poolKey, isToken1,
       ↪ amount, sqrtRatioLimit, skipAhead));
```

CVF-228 INFO

- **Category** Readability
- **Source** Core.sol

Description Imports from different directories and even from third-party libraries are intermixed.

Recommendation Group imports by source.

Client Comment *Sorting imports is an exercise in vanity. If anyone cared then forge fmt would do it https://github.com/foundry-rs/foundry/issues/3396*

```
4  import {CallPoints, addressToCallPoints} from "./types/callPoints.
       ↪ sol";
   import {PoolKey} from "./types/poolKey.sol";
   import {PositionKey, Bounds} from "./types/positionKey.sol";
   import {FeesPerLiquidity, feesPerLiquidityFromAmounts} from "./types
       ↪ /feesPerLiquidity.sol";
   import {isPriceIncreasing, SqrtRatioLimitWrongDirection, SwapResult,
       ↪  swapResult} from "./math/swap.sol";
   import {Position} from "./types/position.sol";
10 import {Ownable} from "solady/auth/Ownable.sol";
   import {tickToSqrtRatio, sqrtRatioToTick} from "./math/ticks.sol";
   import {Bitmap} from "./math/bitmap.sol";
   import {
       shouldCallBeforeInitializePool,
       shouldCallAfterInitializePool,
       shouldCallBeforeUpdatePosition,
       shouldCallAfterUpdatePosition,
       shouldCallBeforeSwap,
       shouldCallAfterSwap,
20     shouldCallBeforeCollectFees,
       shouldCallAfterCollectFees
   } from "./types/callPoints.sol";
   import {FixedPointMathLib} from "solady/utils/FixedPointMathLib.sol"
       ↪ ;
   import {SafeTransferLib} from "solady/utils/SafeTransferLib.sol";
   import {SafeCastLib} from "solady/utils/SafeCastLib.sol";
   import {ExposedStorage} from "./base/ExposedStorage.sol";
   import {liquidityDeltaToAmountDelta, addLiquidityDelta,
       ↪ subLiquidityDelta} from "./math/liquidity.sol";
   import {computeFee} from "./math/fee.sol";
   import {findNextInitializedTick, findPrevInitializedTick, flipTick}
       ↪ from "./math/tickBitmap.sol";
30 import {ICore, UpdatePositionParameters, IExtension} from "./
       ↪ interfaces/ICore.sol";
   import {FlashAccountant} from "./base/FlashAccountant.sol";
   import {EfficientHashLib} from "solady/utils/EfficientHashLib.sol";
   ...
   import {MIN_SQRT_RATIO, MAX_SQRT_RATIO, SqrtRatio} from "./types/
       ↪ sqrtRatio.sol";
```

CVF-229 INFO

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

Description We didn't review these files.

Client Comment *Acknowledged. Solady functions are assumed to behave correctly as described.*

```
10 import {Ownable} from "solady/auth/Ownable.sol";
```

```
23 import {FixedPointMathLib} from "solady/utils/FixedPointMathLib.sol"
   ↪ ;
   import {SafeTransferLib} from "solady/utils/SafeTransferLib.sol";
   import {SafeCastLib} from "solady/utils/SafeCastLib.sol";
```

```
32 import {EfficientHashLib} from "solady/utils/EfficientHashLib.sol";
```

CVF-230 INFO

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

Description The key type for this mapping should be "IERC20".

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
57 mapping(address token => uint256 amountCollected) private
   ↪ protocolFeesCollected;
```

CVF-233 INFO

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

Description The type for the “token” arguments should be “IERC20”.

Client Comment *We prefer to use address because we do not want to introduce non-Ekubo dependencies to the interface files since solidity has nominal type system instead of structural (e.g. two different ERC20-compliant interfaces that are named differently but have exactly the same functions are not considered equivalent by solidity).*

```
73  function withdrawProtocolFees(address recipient, address token,
        ↪ uint256 amount) external onlyOwner {
```

```
138 function load(address token0, address token1, bytes32 salt, uint128
        ↪ amount0, uint128 amount1) public {
```

```
167 function save(address owner, address token0, address token1, bytes32
        ↪  salt, uint128 amount0, uint128 amount1)
```

```
280 function _maybeAccountDebtToken0(uint256 id, address token0, int256
        ↪ debtChange) private {
```

CVF-236 FIXED

- **Category** Documentation
- **Source** Core.sol

Description Should be “-delta0”.

```
337 // protocolFees0 is at most equal to delta0, so this will maximally
        ↪ reach 0 and no overflow/underflow check is needed
```

CVF.Fix-2.7 INFO

- **Category** Procedural
- **Source** OracleLib.sol

Description Specifying a particular compiler version makes it harder upgrading to newer versions.

Recommendation Specify as "^0.8.0" or as "^0.8.28" if there is something special regarding this particular version.

Client Comment *Acknowledged.*

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

CVF.Fix-2.8 INFO

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

Description The type for the "token" arguments should be "IERC20".

Client Comment *Acknowledged.*

```
8 +function getEarliestSnapshotTimestamp(Oracle oracle, address token)
    ↪  internal view returns (uint256) {
```

```
23 +function getMaximumObservationPeriod(Oracle oracle, address token)
    ↪ internal view returns (uint32) {
```

CVF.Fix-2.9 INFO

- **Category** Readability
- **Source** PriceFetcher.sol

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

Recommendation Move the function declaration into the contract or move it into a separate file.

Client Comment *Acknowledged.*

```
16 +function getTimestampsForPeriod(uint256 endTime, uint32
    ↪ numIntervals, uint32 period)
```

CVF.Fix-2.11 INFO

- **Category** Documentation
- **Source** IExposedStorage.sol

Description The phrase “slot after the function selector” sounds confusing.

Recommendation Rephrase.

Client Comment *Acknowledged.*

```
7  +// Loads each slot after the function selector from the contract's
       ↪ storage and returns all of them.
9  +// Loads each slot after the function selector from the contract's
       ↪ transient storage and returns all of them.
```

CVF.Fix-2.12 INFO

- **Category** Documentation
- **Source** IExposedStorage.sol

Description It is unclear how a function with no returned values can return something.

Recommendation Elaborate more.

Client Comment *Acknowledged. It’s quite clear from the 2 line implementation.*

```
7  +// Loads each slot after the function selector from the contract's
       ↪ storage and returns all of them.
9  +// Loads each slot after the function selector from the contract's
       ↪ transient storage and returns all of them.
```

CVF.Fix-2.13 FIXED

- **Category** Documentation
- **Source** Oracle.sol

Description The comment gives no clue regarding the role of the “index” argument.

Recommendation Clearly explain the “index” argument or give this argument a more descriptive name.

```
28 +/// @dev Because the snapshots array is circular, the index of the most recently written snapshot can be any value in [0,c.count) .
29 +///      To simplify the code, we operate on the logical indices, rather than the actual indices.
30 +///      For logical indices, the most recently written value is always at logicalIndex = c.count-1 and the earliest snapshot is always at logicalIndex = 0.
31 +function logicalIndexToStorageIndex(uint256 index, uint256 count, uint256 logicalIndex) pure returns (uint32) {
```

CVF.Fix-2.14 INFO

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

Description The type for the “token” argument should be “IERC20”.

Client Comment *Acknowledged.*

```
43 +error NoPreviousSnapshotExists(address token, uint256 time);
```

```
252 +function findPreviousSnapshot(address token, uint256 time)
```

```
308 +function extrapolateSnapshot(address token, uint256 atTime)
```

CVF.Fix-2.20 INFO

- **Category** Procedural
- **Source** MintableNFT.sol

Description Specifying a particular compiler version makes it harder upgrading to newer versions.

Recommendation Specify as “^0.8.0” or as “^0.8.28” if there is something special regarding this particular version.

Client Comment *Acknowledged*

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

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