Report

v. 3.0

Customer

Maverick

Smart Contract Audit

Maverick Protocol

13th October 2022

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	12.10.22	A. Zveryanskaya	Initial Draft
0.2	12.10.22	A. Zveryanskaya	Minor revision
1.0	13.10.22	A. Zveryanskaya	Release
1.1	13.10.22	A. Zveryanskaya	CVF-19,26,35,88 category downgraded
2.0	13.10.22	A. Zveryanskaya	Release
2.1	13.10.22	A. Zveryanskaya	CVF-26 Client comment added
3.0	13.10.22	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting 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.

Following a formal product description, Maverick Protocol offers a new infrastructure for decentralized finance, built to facilitate the most liquid markets for traders, liquidity providers, DAO treasuries, and developers, powered by a revolutionary Automated Market Maker (AMM).

3 Project scope

We were asked to review:

- Original Repository
- Fix Repository

Files:

interfaces/		
IAddLiquidityCallback.sol	IFactory.sol	IPool.sol
ISwapCallback.sol		
libraries/		
Bin.sol	BinMap.sol	BinMath.sol
Cast.sol	Delta.sol	Deployer.sol
Math.sol	Twa.sol	
models/		
Factory.sol	Passport.sol	Pool.sol
PoolInspector.sol		

4 Methodology

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

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best 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 and that 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 is 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 that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check 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.

5 Our findings

We found 2 major, and a few less important issues. All identified Major issues have been fixed.

	Active	Fixed
Major	0	2
Moderate	0	6
Minor	30	64

Fixed 72 out of 102 issues

6 Major Issues

CVF-53. FIXED

- **Category** Suboptimal
- **Source** Bin.sol

Description This recursion does not scale and may exceed the gas limit, thus making the liquidity removal impossible.

Client Comment *Agree. We added a mechanism to partially traverse the linked list so that you can remove using multiple calls to the contract.*

```
179 return removeLiquidity(bins[self.mergeId], bins, MAXID, delta);
```

CVF-73. FIXED

- **Category** Flaw
- **Source** Pool.sol

Description These calls revert on failed transfer, thus inability to transfer one token makes it impossible to claim fees in the other token.

Recommendation Consider implementing an ability to claim fees in one token even if the other token is broken.

Client Comment *Agree.*

```
269 SafeERC20.safeTransfer(IERC20(tokenA), recipient, Math.toScale(
    ↪ uint256(protocolFeeA), tokenADecimals));
```

```
276 SafeERC20.safeTransfer(IERC20(tokenB), recipient, Math.toScale(
    ↪ uint256(protocolFeeB), tokenBDecimals));
```

7 Moderate Issues

CVF-19. FIXED

- **Category** Suboptimal
- **Source** PoolInspector.sol

Recommendation It is possible to trim an in-memory array in place, without allocating a new array. Just decrease the array length using an assembly block.

Client Comment *Agree.*

```
27  bins = new BinInfo[](activeCounter);
    for (uint256 i = 0; i < activeCounter; i++) {
        bins[i] = tempBins[i];
30  }
```

```
57  bins = new Bin.Cache[](activeCounter);
    for (uint256 i = 0; i < activeCounter; i++) {
        bins[i] = tempBins[i];
60  }
```

CVF-25. FIXED

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

Description Overflow is possible when performing unsafe type conversions.

Recommendation Consider using safe conversions.

Client Comment *Agree; we adjusted the types so that the timestamp support goes to several millions years in the future. we added a requires to lookback to bound it. and the further twa is bounded by MAX_TICK which is only ~400k which is well under 32 bits. All of these casts should be okay.*

```
13 self.twa = self.lastTimestamp == 0 ? int96(value) : int96(getTwa(
      ↪ self));
   self.lastTimestamp = uint32(block.timestamp);
```

```
18 return int32(PRBMathSD59x18.floor(self.twa) / PRBMathSD59x18.SCALE);
```

```
24 int256 _timeDiff = int256(block.timestamp - self.lastTimestamp) *
      ↪ PRBMathSD59x18.SCALE;
   int256 _lookback = int256(int32(self.lookback)) * PRBMathSD59x18.
      ↪ SCALE;
```

CVF-50. FIXED

- **Category** Suboptimal
- **Source** Bin.sol

Description In case deltaBOptimal > _deltaB and _existingReserveB <= 0, the value of "deltaBOptimal" will not be adjusted.

Recommendation Consider setting to "_deltaB" or reverting in such a case.

Client Comment *Agree. Refactored this entire function.*

```
80 if (deltaBOptimal > _deltaB && _existingReserveB > 0) {
```

CVF-71. FIXED

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

Description Overflow is possible when converting to "int256".

Recommendation Consider using safe conversion.

Client Comment *Agree. Got rid of signed types*

```
238 amountIn = !delta.exactOutput ? amount : int256(Math.toScale(uint256
    ↪ (delta.deltaInErc), tokenAIn ? tokenADecimals : tokenBDecimals
    ↪ )) + 1;
    amountOut = delta.exactOutput ? amount : int256(Math.toScale(uint256
    ↪ (delta.deltaOutErc), tokenAIn ? tokenBDecimals :
    ↪ tokenADecimals)) - 1;
```

CVF-74. FIXED

- **Category** Suboptimal
- **Source** Pool.sol

Description This loop doesn't scale. In case of too many bins it could exceed the block gas limit. As bins could be created by any user, it is possible to artificially create enough bin to make this function non-operational.

Recommendation Consider maintaining the correct aggregated bin balances to make this function unnecessary.

Client Comment *Agree. Trim function not needed. removed from codebase.*

```
289 for (uint208 i = 1; i <= state.binCounter; i++) {
```

CVF-88. FIXED

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

Description It is not ensured that tokenA and tokenB are different.

Recommendation Consider adding such check. The AMM would work fine if the user wanted to make a pool with the same token as A and B.

Client Comment *Agree.*

```
47 address _tokenA,
   address _tokenB
```

8 Minor Issues

CVF-1. FIXED

- **Category** Procedural
- **Source** BinMap.sol

Recommendation Should be "^0.8.0" unless there is something special about this particular version. Also relevant for the next files: PoolInspector.sol, Deployer.sol, Twa.sol, Math.sol, Delta.sol, Cast.sol, BinMath.sol, Bin.sol, Pool.sol, Factory.sol, Passport.sol, IFactory.sol, ISwapCallback.sol, IPool.sol.

Client Comment *Agree.*

```
2 pragma solidity ^0.8.16;
```

CVF-2. INFO

- **Category** Procedural
- **Source** BinMap.sol

Description We didn't review this file.

Client Comment *Noted.*

```
3 import "prb-math/contracts/PRBMathSD59x18.sol";
```

CVF-3. FIXED

- **Category** Readability
- **Source** BinMap.sol

Description Defining top-level constants in a file named after a library names it harder to navigate through the code.

Recommendation Consider either moving the constant definition into the library or moving them into a separate file named "constants.sol".

Client Comment *Agree.*

```
5  uint8 constant NONE = 1;
   uint8 constant RIGHT = 2;
   uint8 constant LEFT = 4;
   uint8 constant BOTH = 8;
   uint8 constant KINDS_COUNT = 4;
10 uint16 constant WORD_SIZE = 256;
   int32 constant OFFSET_MASK = 255;
   uint256 constant BIT_MASK = type(uint256).max;
   int32 constant KINDS = 4;
   uint32 constant MASK = 15;
```

CVF-4. FIXED

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

Description The names of these constants are too generic. They give no clue regarding the role of these constants.

Recommendation Consider using more specific names.

Client Comment *Agree.*

```
5  uint8 constant NONE = 1;
   uint8 constant RIGHT = 2;
   uint8 constant LEFT = 4;
   uint8 constant BOTH = 8;
```

```
14 uint32 constant MASK = 15;
```

CVF-5. FIXED

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

Description The argument name is confusing, as it actually contains a bit index in a bitmap, rather than a tick. There are 4 bits per tick in a bit map.

Recommendation Consider renaming.

Client Comment *Agree.*

```
20 function getMapPointer(int32 tick) internal pure returns (uint8
      ↪ offset, int32 mapIndex) {
```

CVF-6. FIXED

- **Category** Suboptimal
- **Source** BinMap.sol

Recommendation The "& OFFSET_MASK" part is redundant, as conversion to "uint8" would anyway drop higher bits.

Client Comment *Agree.*

```
21 offset = uint8(uint32(tick & OFFSET_MASK));
```

CVF-7. FIXED

- **Category** Suboptimal
- **Source** BinMap.sol

Recommendation It would be more efficient to use the "|=" operator to avoid calculating the "binMap[mapIndex]" slot address twice.

Client Comment *Agree.*

```
30 uint256 subMap = binMap[mapIndex];
   binMap[mapIndex] = subMap | (1 << offset);
```

CVF-8. FIXED

- **Category** Suboptimal
- **Source** BinMap.sol

Recommendation It would be more efficient to use the "&=" operator to avoid calculating the "binMap[mapIndex]" slot address twice.

Client Comment *Agree.*

```
39 uint256 subMap = binMap[mapIndex];
40 binMap[mapIndex] = subMap & ~(1 << offset) & BIT_MASK;
```

CVF-9. FIXED

- **Category** Suboptimal
- **Source** BinMap.sol

Recommendation The "& BIT_MASK" part is redundant as it effectively does nothing.

Client Comment *Agree.*

```
40 binMap[mapIndex] = subMap & ~(1 << offset) & BIT_MASK;
```

CVF-10. INFO

- **Category** Suboptimal
- **Source** BinMap.sol

Description This function is overcomplicated and inefficient.

Recommendation Consider simplifying and optimizing.

Client Comment *Noted.*

```
49 function getKindsAtTickRange(
```

CVF-11. INFO

- **Category** Suboptimal
- **Source** BinMap.sol

Description The "counter" return value is redundant.

Recommendation Just return an array without trailing unused element. Note that it is possible to allocate a large array, fill ti partially, and then reduce the array size via assembly.

Client Comment *Noted.*

```
54 ) internal view returns (Active[] memory activeList, uint256 counter
     ↪ ) {
```

CVF-12. FIXED

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

Recommendation The value "~3" should be a named constant.

Client Comment *Agree.*

```
64 _bitIndex &= ~3;
```

CVF-13. FIXED

- **Category** Suboptimal
- **Source** BinMap.sol

Recommendation As a bit index always fits into 8 bits, 0xFC could be used instead of ~3.

Client Comment *Agree.*

```
64 _bitIndex &= ~3;
```

CVF-14. FIXED

- **Category** Readability
- **Source** BinMap.sol

Recommendation This could be simplified as: tick = pos » 2;

Client Comment *Agree.*

```
70 if (pos < 0) {
       pos += 1;
       tick = pos / KINDS;
       tick -= 1;
   } else {
       tick = pos / KINDS;
   }
```

CVF-15. INFO

- **Category** Suboptimal
- **Source** BinMap.sol

Recommendation It would be more efficient to implement two separate functions: one to search left and another to search right.

Client Comment *Disagree (adds to code size, which is at a premium).*

```
95 bool isRight
```

CVF-16. FIXED

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

Recommendation The value ”5” should be a named constant.

Client Comment *Agree.*

```
110 for (uint256 i; i < 5; i++) {
```

CVF-17. FIXED

- **Category** Procedural
- **Source** PoolInspector.sol

Recommendation This file is imported twice. Remove one import.

Client Comment *Agree.*

```
4 import "../libraries/Bin.sol";
  import "../libraries/Bin.sol";
```

CVF-18. FIXED

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

Recommendation The bin status values should be named constants.

Client Comment *Agree.*

```
22 if (bin.status == 1 || bin.status == 2) {
```

```
52 if (bin.lowerTick == tick && bin.status == 1) {
```

CVF-20. FIXED

- **Category** Documentation
- **Source** Deployer.sol

Description The number format of these arguments is unclear.

Recommendation Consider documenting.

Client Comment *Agree.*

```
7 int256 _fee,
```

```
11 int16 _protocolFeeRatio,
```

CVF-21. FIXED

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

Recommendation The type of these arguments should be "IERC20".

Client Comment *Agree.*

```
12 address _tokenA,
   address _tokenB,
```

CVF-22. INFO

- **Category** Suboptimal
- **Source** Deployer.sol

Description In ERC20 the "decimals" property is used by UI to render token amounts in a human-readable way. Using this property in smart contracts is discourages.

Recommendation Consider treating all token amounts as integers.

Client Comment *Noted. We scale by decimals to avoid having to support an even larger price range.*

```
14 uint8 _tokenADecimals,
   uint8 _tokenBDecimals,
```

CVF-23. INFO

- **Category** Suboptimal
- **Source** Deployer.sol

Description Deploying a separate instance of the "Pool" contract for each pair consumes lots of gas.

Recommendation Consider deploying a minimal proxy instead as described in ERC-1167.

Client Comment *Noted. Minimal proxies don't let you set immutable values. immutable parameters per pool make the common pool operations much cheaper. so a minimal proxy would reduce deploy costs, deploys do not happend very often and it would make the more common operations more expensive.*

```
19 new Pool{salt: keccak256(abi.encode(_fee, _tickSpacing, _tokenA,
     ↪ _tokenB))}(
```

CVF-24. INFO

- **Category** Procedural
- **Source** Twa.sol

Description We didn't review this file.

Client Comment *Noted.*

```
3 import "prb-math/contracts/PRBMathSD59x18.sol";
```

CVF-26. INFO

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

Description This function calculates something that differs from both, weighted moving average and exponential moving average, as "self.twa" could include values observed before the current loobback period, and these values will still affect the new TWA value.

Recommendation Consider calculating either correct TWA or correct exponential moving average.

Client Comment *Noted. This function is a continuous TWAP. You need to keep self.twa around from times before the window because that is the price up until the current timestamp. Assuming you have a lookback of 3600 seconds, if you observe a price 1 at time 100, then you observe another price of 3 at time 7000, the TWAP at time 7000 is still 1. As time goes up from 7000 to 10,600, the twap will increase from 1 to 3. And at time 10,600 it will be 3.*

```
20 function getTwa(Instance storage self) internal view returns (int256
   ↪ ) {
```

CVF-27. INFO

- **Category** Procedural
- **Source** Math.sol

Description We didn't review this file.

Client Comment *Noted.*

```
3 import "prb-math/contracts/PRBMath.sol";
```

CVF-28. INFO

- **Category** Readability
- **Source** Math.sol

Recommendation Should be ”else return”.

Client Comment *Disagree.*

```
25 return PRBMath.mulDivSigned(x, y, k);
```

CVF-29. FIXED

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

Recommendation The value 18 should be a named constant.

Client Comment *Agree.*

```
28 if (decimals == 18) {
```

```
30 } else if (decimals > 18) {
       return amount * (10**(decimals - 18));
```

```
33     return amount / (10**(18 - decimals));
```

```
37 if (decimals == 18) {
```

```
39 } else if (decimals > 18) {
40     return amount / (10**(decimals - 18));
```

```
42     return amount * (10**(18 - decimals));
```

CVF-30. FIXED

- **Category** Suboptimal
- **Source** Math.sol

Description Calculating the scale factors each time is suboptimal.

Recommendation Consider refactoring to precompute them.

Client Comment *Agree.*

```
31 return amount * (10**(decimals - 18));
```

```
33 return amount / (10**(18 - decimals));
```

```
40 return amount / (10**(decimals - 18));
```

```
42 return amount * (10**(18 - decimals));
```

CVF-31. FIXED

- **Category** Suboptimal
- **Source** Cast.sol

Recommendation This function could be simplified as: function toInt256 (uint256 x) internal pure returns (int256 y) { require ((y = int256 (x)) >= 0); }

Client Comment *Agree.*

```
 7 function toInt256(uint256 x) internal pure returns (int256) {
       require(x <= MAX_INT256);
       return int256(x);
10 }
```

CVF-32. FIXED

- **Category** Suboptimal
- **Source** Cast.sol

Recommendation This function could be simplified as: function toInt128 (int256 x) internal pure returns (int128 y) { require ((y = int128 (x)) == x); }

Client Comment *Agree.*

```
11 function toInt128(int256 x) internal pure returns (int128) {
       require(x <= MAX_INT128 && x >= MIN_INT128);
       return int128(x);
   }
```

CVF-33. INFO

- **Category** Procedural
- **Source** BinMath.sol

Description We didn't review these files.

Client Comment *Noted.*

```
3 import "prb-math/contracts/PRBMathUD60x18.sol";
  import "prb-math/contracts/PRBMathSD59x18.sol";
```

CVF-34. FIXED

- **Category** Readability
- **Source** BinMath.sol

Description Defining a top-level constant in a file named after a library makes it harder to navigate through the code.

Recommendation Consider either moving the constant definition into the library or moving it into a separate file named "constants.sol".

Client Comment *Agree.*

```
5 int256 constant MAX_TICK = 460540;
```

CVF-35. INFO

- **Category** Unclear Behaviour
- **Source** BinMath.sol

Description This maximum tick value means that the allowed price range is from 1e-10 to 1e10, which doesn't looks sufficient. For example, the price relation between the most expensive and the cheapest coins in Top-100 coins on CoinMarketCap now exceeds 1e10.

Recommendation Consider allowing a wider price range.

Client Comment *This is the range of sqrt price. The range of price is 1e-20, 1e20. we renamed the function to reflect that it is the sqrt price, not price that is being returned.*

```
5 int256 constant MAX_TICK = 460540;
```

CVF-36. FIXED

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

Description The return type ”uint16” looks odd here, as a result of this function always fits into 8 bits.

Recommendation Consider changing the return type to ”uint8” or to ”uint256”.

Client Comment *Agree.*

```
9 function msb(uint256 x) internal pure returns (uint16 result) {
```

CVF-37. INFO

- **Category** Readability
- **Source** BinMath.sol

Recommendation This check could be simplified as: if (x « 128 != 0) {

Client Comment *Noted.*

```
48 if (x & 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF > 0) {
```

CVF-38. INFO

- **Category** Readability
- **Source** BinMath.sol

Recommendation This check could be simplified as: if (x « 192 != 0) {

Client Comment *Noted.*

```
53 if (x & 0xFFFFFFFFFFFFFFFF > 0) {
```

CVF-39. INFO

- **Category** Readability
- **Source** BinMath.sol

Recommendation This check could be simplified as: if (x « 224 != 0) {

Client Comment *Noted.*

```
58 if (x & 0xFFFFFFFF > 0) {
```

CVF-40. INFO

- **Category** Readability
- **Source** BinMath.sol

Recommendation This check could be simplified as: if (x « 240 != 0) {

Client Comment *Noted.*

```
63 if (x & 0xFFFF > 0) {
```

CVF-41. INFO

- **Category** Readability
- **Source** BinMath.sol

Recommendation This check could be simplified as: if (x « 248 != 0) {

Client Comment *Noted.*

```
68 if (x & 0xFF > 0) {
```

CVF-42. INFO

- **Category** Readability
- **Source** BinMath.sol

Recommendation This check could be simplified as: if (x « 252 != 0) {

Client Comment *Noted.*

```
73 if (x & 0xF > 0) {
```

CVF-43. INFO

- **Category** Readability
- **Source** BinMath.sol

Recommendation This check could be simplified as: if (x « 254 != 0) {

Client Comment *Noted.*

```
78 if (x & 0x3 > 0) {
```

CVF-44. INFO

- **Category** Readability
- **Source** BinMath.sol

Recommendation This check could be simplified as: if (x « 255 != 0) {

Client Comment *Noted.*

```
83 if (x & 0x1 > 0) result -= 1;
```

CVF-45. FIXED

- **Category** Suboptimal
- **Source** BinMath.sol

Description These lines are redundant, as the maximum allowed tick value is 0x706FC.

Recommendation Consider removing these lines.

Client Comment *Agree.*

```
109 if (tick & 0x80000 != 0) ratio = (ratio * 0x48a1703920644d4030024fe)
    ↪  >> 128;
110 if (tick & 0x100000 != 0) ratio = (ratio * 0x149b34ee7b4532) >> 128;
```

CVF-46. INFO

- **Category** Procedural
- **Source** Bin.sol

Description We didn't review these files.

Client Comment *Noted.*

```
3 import "prb-math/contracts/PRBMathSD59x18.sol";
  import "prb-math/contracts/PRBMath.sol";
```

CVF-47. FIXED

- **Category** Readability
- **Source** Bin.sol

Description Defining top-level constants in a file named after a library makes it harder to navigate through the code.

Recommendation Consider either moving the constants into the library or moving them into a separate file named "constants.sol".

Client Comment *Agree.*

```
 9 uint256 constant MAXID = type(uint256).max;
10 uint8 constant BIN_STATUS_UNINITIALIZED = 0;
   uint8 constant BIN_STATUS_ACTIVE = 1;
   uint8 constant BIN_STATUS_MERGED = 2;
   uint8 constant BIN_STATUS_INACTIVE = 3;
```

CVF-48. FIXED

- **Category** Suboptimal
- **Source** Bin.sol

Recommendation This recursion could be rewritten as a loop to save gas.

Client Comment *Agree. Funciton removed.*

```
40 uint208 id = activeMergeId(bins[self.mergeId], bins);
```

CVF-49. FIXED

- **Category** Suboptimal
- **Source** Bin.sol

Description This recursion doesn't scale and could exceed the block gas limit.

Recommendation Consider refactoring to replace this recursion with a scalable algorithm.

Client Comment *Agree. Removeliquidity was refactored to not need this funciton anymore.*

```
40 uint208 id = activeMergeId(bins[self.mergeId], bins);
```

CVF-51. FIXED

- **Category** Suboptimal
- **Source** Bin.sol

Description This should be calculated only when delta.excess <= 0.

Client Comment *Agree.*

```
118 deltaOut = (Math.mulDiv(delta.deltaOutInternal, thisBinAmount,
       ↪ totalAmount) + 1).toInt128();
```

CVF-52. FIXED

- **Category** Suboptimal
- **Source** Bin.sol

Description The expression ”tokenAmount.toInt128()” is calculated twice.

Recommendation Consider calculating once and reusing.

Client Comment *Agree.*

```
177 self.balances[tokenId] -= tokenAmount.toInt128();
    self.totalSupply -= tokenAmount.toInt128();
```

CVF-54. INFO

- **Category** Procedural
- **Source** Pool.sol

Description We didn’t review these files.

Client Comment *Noted.*

```
6 import "prb-math/contracts/PRBMathSD59x18.sol";
```

CVF-55. FIXED

- **Category** Readability
- **Source** Pool.sol

Description Defining top-level constants in a file named after a contract makes it harder to navigate through the code.

Recommendation Consider either moving the constant definitions into the contract or moving them into a separate file named "constants.sol".

Client Comment *Agree.*

```
17 uint8 constant UNLOCKED = 1;
   uint8 constant LOCKED = 2;
   uint8 constant EMERGENCY_UNLOCKED = 3;
20 uint8 constant EMERGENCY_LOCKED = 4;
   uint256 constant ACTION_SET_PROTOCOL_FEES = 1;
   uint256 constant ACTION_CLAIM_PROTOCOL_FEES = 2;
   uint256 constant ACTION_TRIM = 3;
   uint256 constant ACTION_EMERGENCY_MODE = 911;
```

CVF-56. FIXED

- **Category** Documentation
- **Source** Pool.sol

Description The number format of this variable is unclear.

Recommendation Consider documenting.

Client Comment *Agree. Documented in constructor.*

```
33 int256 public immutable override fee;
```

CVF-57. FIXED

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

Recommendation The type of these variables should be "IERC20".

Client Comment *Agree.*

```
35 address public immutable override tokenA;
   address public immutable override tokenB;
```

CVF-58. FIXED

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

Recommendation The type of this variable should be "Passport" or a interface extracted from it.

Client Comment *Agree.*

```
37 address immutable passport;
```

CVF-59. INFO

- **Category** Suboptimal
- **Source** Pool.sol

Description In ERC20 the "decimals" property is used by UI to render token amounts in a human-friendly way. Using this property in smart contracts is discouraged.

Recommendation Consider treating all token amounts as integers.

Client Comment *Noted. We scale by decimals to avoid having to support an even larger price range.*

```
38 uint8 immutable tokenADecimals;
   uint8 immutable tokenBDecimals;
```

CVF-60. FIXED

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

Recommendation The type of this variable should be "IFactory".

Client Comment *Agree.*

```
40 address immutable factory;
```

CVF-61. FIXED

- **Category** Documentation
- **Source** Pool.sol

Description The semantics of the keys in this mapping is unclear.

Recommendation Consider documenting.

Client Comment *Agree.*

```
43 mapping(int32 => mapping(uint256 => uint208)) binPositions;
```

CVF-62. INFO

- **Category** Suboptimal
- **Source** Pool.sol

Recommendation Taking into account that the second key in this mapping is actually a bit kind that fits into 8 bits, it would be more efficient to merge the two keys together into a single key.

Client Comment *Noted. We tested this and it is more gas to have one key.*

```
43 mapping(int32 => mapping(uint256 => uint208)) binPositions;
```

CVF-63. INFO

- **Category** Procedural
- **Source** Pool.sol

Recommendation Events usually named via nouns, such as ”BinMerge”, ”BinMove”, etc.

Client Comment *Noted. We will keep names as is.*

```
58 event AddLiquidity(address indexed sender, uint256 indexed tokenId,
      ↪ BinDelta[] binDeltas);
   event RemoveLiquidity(address indexed sender, address indexed
      ↪ recipient, uint256 indexed tokenId, BinDelta[] binDeltas);
60 event BinMerged(uint208 indexed binId, int128 reserveA, int128
      ↪ reserveB, uint208 mergeId);
   event BinMoved(uint208 indexed binId, int128 previousTick, int128
      ↪ newTick);
   event ProtocolFeeCollected(int256 protocolFeeA, int256 protocolFeeB)
      ↪ ;
```

CVF-64. FIXED

- **Category** Documentation
- **Source** Pool.sol

Description The number format of this argument is unclear.

Recommendation Consider documenting.

Client Comment *Agree.*

```
64 int256 _fee,
```

```
68 int16 _protocolFeeRatio,
```

CVF-65. FIXED

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

Recommendation The type of these arguments should be ”IERC20”.

Client Comment *Agree.*

```
69 address _tokenA,
70 address _tokenB,
```

CVF-66. FIXED

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

Recommendation The type of this argument should be ”Passport” or an interface extracted from it.

Client Comment *Agree.*

```
73 address _passport
```

CVF-67. FIXED

- **Category** Suboptimal
- **Source** Pool.sol

Recommendation This logic would be much simpler is status would be a bit mask where one bit represents the ”unlocked” flag and another bit represents the ”emergency” flag.

Client Comment *Agree.*

153
```
require(currentState.status == UNLOCKED || currentState.status ==
    ↪ EMERGENCY_UNLOCKED);
if (currentState.status == UNLOCKED) {
    state.status = LOCKED;
} else if (currentState.status == EMERGENCY_UNLOCKED) {
    state.status = EMERGENCY_LOCKED;
}
```

191
```
if (state.status == LOCKED) {
    state.status = UNLOCKED;
} else if (state.status == EMERGENCY_LOCKED) {
    state.status = EMERGENCY_UNLOCKED;
}
```

303
```
require(currentState.status == UNLOCKED || currentState.status ==
    ↪ EMERGENCY_UNLOCKED);
if (currentState.status == UNLOCKED) {
    state.status = LOCKED;
} else if (currentState.status == EMERGENCY_UNLOCKED) {
    state.status = EMERGENCY_LOCKED;
}
```

322
```
if (state.status == LOCKED) {
    state.status = UNLOCKED;
} else if (state.status == EMERGENCY_LOCKED) {
    state.status = EMERGENCY_UNLOCKED;
}
```

CVF-68. FIXED

- **Category** Readability
- **Source** Pool.sol

Recommendation This should be refactored as: if (bin.mergeId != 0) { ... } else if (bin.reserveA == 0 && bin.reserveB == 0) { ... }

Client Comment *Agree. This is handled by the bin now*

```
168 if (bin.reserveA == 0 && bin.reserveB == 0 && (bin.mergeId == 0)) {
```

```
171 }
```

```
173 if (bin.mergeId != 0) {
```

```
182 }
```

CVF-69. FIXED

- **Category** Readability
- **Source** Pool.sol

Recommendation This should be refactored as: while (delta.excess > 0) { ... }

Client Comment *Agree.*

```
221 while (true) {
        if (delta.excess > 0) {
```

```
226     } else {
            break;
        }
    }
```

CVF-70. FIXED

- **Category** Readability
- **Source** Pool.sol

Recommendation Brackets around the product are redundant.

Client Comment *Agree.*

```
237 twa.updateValue((currentState.activeBin * PRBMathSD59x18.SCALE) + (
    ↪ delta.endSqrtPrice - sqrtLowerTickPrice).div(
    ↪ sqrtUpperTickPrice - sqrtLowerTickPrice));
```

CVF-72. FIXED

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

Recommendation These functions should emit some events.

Client Comment *Agree.*

```
266 function claimProtocolFees(address recipient) internal {
```

```
283 function setProtocolFee(int16 _protocolFeeRatio) internal {
```

```
286 function trim() internal {
```

CVF-75. FIXED

- **Category** Suboptimal
- **Source** Pool.sol

Description The expression ”bins[i]” is calculated twice.

Recommendation Consider calculating once and reusing.

Client Comment *Noted.Trim function was removed.*

```
291 reserveA += bins[i].reserveA;
    reserveB += bins[i].reserveB;
```

CVF-76. INFO

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

Recommendation The type of this generic argument should be ”bytes” to allow implementing admin actions with arbitrary set of arguments.

Client Comment *Noted. Tried this but took up too much space.*

```
299 int16 val,
```

CVF-77. INFO

- **Category** Suboptimal
- **Source** Pool.sol

Description This loop doesn't scale and could exceed the block gas limit.

Recommendation Consider implementing a scalable approach.

Client Comment *Noted. binCounter is as big as the number of bins swapped. for a large swap there is no more scalable option. if the swap is so large that this exceeds the gas limit, the trader should swap a smaller input amount.*

```
372 for (uint256 i; i <= temp.binCounter; i++) {
```

CVF-78. FIXED

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

Description The function name is confusing as the function not always creates a new bin but in some cases may return an existing bin.

Recommendation Consider renaming to "getOrCreateBin" or something like this.

Client Comment *Agree.*

```
426 function _newBin(
```

CVF-79. FIXED

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

Recommendation The value "4" should be a named constant.

Client Comment *Agree.*

```
470 for (uint256 i; i < 4; i++) {
```

```
484 for (uint256 i; i < 4; i++) {
```

CVF-80. FIXED

- **Category** Suboptimal
- **Source** Pool.sol

Recommendation Here "=" could be used instead of "+=" as the original values are guaranteed to be zero.

Client Comment *Agree.*

```
488 currentReserveA += bin.reserveA;
    currentReserveB += bin.reserveB;
```

CVF-81. FIXED

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

Recommendation The value "1e15" should be a named constant.

Client Comment *Agree.*

```
522 delta.deltaInBinInternal = Math.max(0, delta.deltaInErc - feeBasis.
       ↪ mul(int256(state.protocolFeeRatio) * 1e15) - 1);
```

```
576 delta.deltaInBinInternal = Math.max(0, delta.deltaInErc - feeBasis.
       ↪ mul(int256(state.protocolFeeRatio) * 1e15) - 1);
```

CVF-82. FIXED

- **Category** Documentation
- **Source** Factory.sol

Description The semantics of the keys in this mapping is unclear.

Recommendation Consider documenting.

Client Comment *Agree.*

```
8 mapping(int256 => mapping(uint256 => mapping(address => mapping(
     ↪ address => address)))) pools;
```

CVF-83. FIXED

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

Recommendation The type of the last two keys should be ”IERC20”.

Client Comment *Agree.*

```
8 mapping(int256 => mapping(uint256 => mapping(address => mapping(
    ↪ address => address)))) pools;
```

CVF-84. FIXED

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

Recommendation The value type for this mapping should be ”IPool”.

Client Comment *Agree.*

```
8 mapping(int256 => mapping(uint256 => mapping(address => mapping(
    ↪ address => address)))) pools;
```

CVF-85. FIXED

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

Recommendation These functions should emit some events.

Client Comment *Agree.*

```
25 function setProtocolFeeRatio(int16 _protocolFeeRatio) external {
```

```
30 function setOwner(address _owner) external {
```

CVF-86. FIXED

- **Category** Documentation
- **Source** Factory.sol

Description The number format of these values is unclear.

Recommendation Consider documenting.

Client Comment *Agree.*

```
35 int256 _fee,
```

```
44 int256 _fee,
```

CVF-87. FIXED

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

Description The type of these arguments should be "IERC20".

Client Comment *Agree.*

```
37 address _tokenA,
   address _tokenB
```

```
47 address _tokenA,
   address _tokenB
```

CVF-89. INFO

- **Category** Suboptimal
- **Source** Factory.sol

Description In ERC20 the "decimals" property is used by UI to render token amounts in a human-readable way. Using this property in smart contracts is discouraged.

Recommendation Consider treating all token amounts as integers.

Client Comment *By sclaing by decimals, we avoid having to support an even wider price range than 1e-20, 1e20.*

```
55 pool = Deployer.deploy(_fee, _tickSpacing, _activeBin, lookback,
      ↪ protocolFeeRatio, _tokenA, _tokenB, IERC20Metadata(_tokenA).
      ↪ decimals(), IERC20Metadata(_tokenB).decimals(), passport);
```

CVF-90. INFO

- **Category** Procedural
- **Source** Passport.sol

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

Client Comment *Noted.*

```
10 constructor() ERC721("Maverick␣Passport", "MPP") {}
```

CVF-91. FIXED

- **Category** Documentation
- **Source** IFactory.sol

Description The number format of these arguments is unclear.

Recommendation Consider documenting.

Client Comment *Agree.*

```
5  int256 _fee,
```

```
12 int256 fee,
```

CVF-92. FIXED

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

Recommendation The type of these arguments should be ”IERC20”.

Client Comment *Agree.*

```
8  address _tokenA,
   address _tokenB
```

```
14 address tokenA,
   address tokenB
```

CVF-93. FIXED

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

Recommendation The return type should be ”IPool”.

Client Comment *Agree.*

```
10 ) external returns (address);
```

```
16 ) external view returns (address);
```

CVF-94. FIXED

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

Recommendation The return type should be ”Passport” or an interface extracted from it.

Client Comment *Agree.*

```
18 function passport() external view returns (address);
```

CVF-95. FIXED

- **Category** Documentation
- **Source** IFactory.sol

Description The number format of the returned value is unclear.

Recommendation Consider documenting.

Client Comment *Agree.*

```
19 function protocolFeeRatio() external view returns (int16);
```

CVF-96. FIXED

- **Category** Documentation
- **Source** IFactory.sol

Description The semantics of the returned value is unclear.

Recommendation Consider documenting.

Client Comment *Moot. No longer a function.*

```
20 function lookback() external view returns (uint32);
```

CVF-97. INFO

- **Category** Readability
- **Source** ISwapCallback.sol

Description The swap direction is unclear from the arguments.

Recommendation Consider passing signed "amountA" and "amountB" arguments instead.

Client Comment *noted*

```
5 uint256 amountIn,
uint256 amountOut,
```

CVF-98. FIXED

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

Description The semantics of these fields and their valid values are unknown.

Recommendation Consider defining the valid values for them as named constants.

Client Comment *Agree.*

```
7 uint8 kind;
```

```
19 uint8 status;
```

CVF-99. FIXED

- **Category** Documentation
- **Source** IPool.sol

Description The number forma of this field is unclear.

Recommendation Consider documenting.

Client Comment *Agree.*

```
21 int16 protocolFeeRatio;
```

CVF-100. FIXED

- **Category** Documentation
- **Source** IPool.sol

Description The number format of the returned values is unclear.

Recommendation Consider documenting.

Client Comment *Agree.*

```
23 function fee() external view returns (int256);
   function tickSpacing() external view returns (uint256);
```

```
29 function getTwa() external view returns (Twa.Instance memory);
```

CVF-101. FIXED

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

Recommendation The type of the returned values should be ”IERC20”.

Client Comment *Agree.*

```
25 function tokenA() external view returns (address);
   function tokenB() external view returns (address);
```

CVF-102. FIXED

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

Recommendation This function looks should not be a part of the ”IPool” interface, as it is not supposed to be called by normal users.

Client Comment *Agree.*

```
50 function adminAction(
```

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