ABDK CONSULTING

SMART CONTRACT AUDIT

MUFFIN

Part 1: CORE

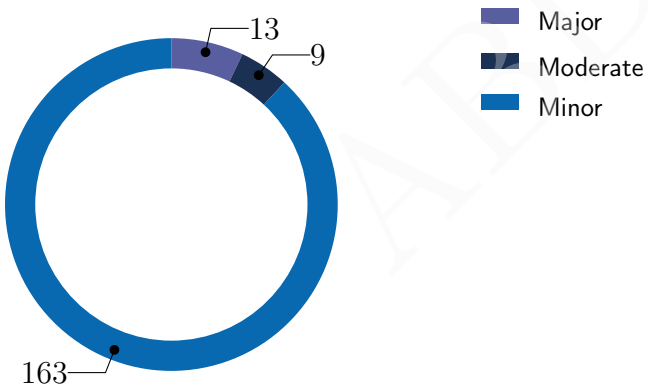
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
20th May 2022

We've been asked to review 32 files in a Github repository. We found 13 major and a few less important issues. All identified major issues have been fixed or otherwise addressed in collaboration with the client.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Fixed
CVF-2	Minor	Procedural	Info
CVF-3	Major	Suboptimal	Info
CVF-4	Minor	Documentation	Info
CVF-5	Minor	Bad datatype	Info
CVF-6	Minor	Suboptimal	Info
CVF-7	Minor	Unclear behavior	Fixed
CVF-8	Minor	Suboptimal	Fixed
CVF-9	Minor	Documentation	Info
CVF-10	Minor	Suboptimal	Fixed
CVF-11	Minor	Procedural	Fixed
CVF-12	Minor	Bad datatype	Info
CVF-13	Minor	Documentation	Fixed
CVF-14	Minor	Bad datatype	Fixed
CVF-15	Moderate	Unclear behavior	Info
CVF-16	Minor	Unclear behavior	Fixed
CVF-17	Minor	Suboptimal	Fixed
CVF-18	Minor	Procedural	Info
CVF-19	Minor	Suboptimal	Info
CVF-20	Minor	Suboptimal	Info
CVF-21	Minor	Overflow/Underflow	Info
CVF-22	Minor	Readability	Info
CVF-23	Minor	Unclear behavior	Info
CVF-24	Minor	Procedural	Info
CVF-25	Minor	Procedural	Fixed
CVF-26	Minor	Suboptimal	Info
CVF-27	Minor	Overflow/Underflow	Info

ID	Severity	Category	Status
CVF-28	Minor	Procedural	Fixed
CVF-29	Minor	Flaw	Info
CVF-30	Minor	Procedural	Info
CVF-31	Minor	Procedural	Fixed
CVF-32	Minor	Procedural	Fixed
CVF-33	Minor	Suboptimal	Fixed
CVF-34	Minor	Overflow/Underflow	Info
CVF-35	Major	Suboptimal	Fixed
CVF-36	Minor	Suboptimal	Fixed
CVF-37	Major	Procedural	Fixed
CVF-38	Minor	Suboptimal	Fixed
CVF-39	Minor	Suboptimal	Fixed
CVF-40	Moderate	Unclear behavior	Fixed
CVF-41	Major	Suboptimal	Info
CVF-42	Major	Suboptimal	Info
CVF-43	Minor	Suboptimal	Info
CVF-44	Minor	Unclear behavior	Info
CVF-45	Minor	Procedural	Info
CVF-46	Major	Procedural	Info
CVF-47	Minor	Bad datatype	Info
CVF-48	Minor	Suboptimal	Info
CVF-49	Minor	Unclear behavior	Info
CVF-50	Minor	Unclear behavior	Info
CVF-51	Minor	Overflow/Underflow	Fixed
CVF-52	Minor	Bad naming	Fixed
CVF-53	Minor	Unclear behavior	Info
CVF-54	Minor	Suboptimal	Info
CVF-55	Minor	Suboptimal	Info
CVF-56	Major	Documentation	Fixed
CVF-57	Minor	Documentation	Fixed

ID	Severity	Category	Status
CVF-58	Minor	Overflow/Underflow	Fixed
CVF-59	Major	Procedural	Info
CVF-60	Minor	Suboptimal	Fixed
CVF-61	Minor	Suboptimal	Info
CVF-62	Minor	Procedural	Info
CVF-63	Minor	Readability	Info
CVF-64	Minor	Bad datatype	Info
CVF-65	Minor	Bad datatype	Info
CVF-66	Minor	Bad datatype	Info
CVF-67	Major	Procedural	Fixed
CVF-68	Minor	Unclear behavior	Info
CVF-69	Minor	Suboptimal	Fixed
CVF-70	Minor	Bad datatype	Info
CVF-71	Minor	Procedural	Fixed
CVF-72	Minor	Bad naming	Fixed
CVF-73	Minor	Procedural	Fixed
CVF-74	Minor	Unclear behavior	Info
CVF-75	Minor	Suboptimal	Info
CVF-76	Minor	Documentation	Fixed
CVF-77	Minor	Suboptimal	Info
CVF-78	Minor	Suboptimal	Info
CVF-79	Minor	Suboptimal	Fixed
CVF-80	Minor	Procedural	Info
CVF-81	Minor	Readability	Info
CVF-82	Minor	Suboptimal	Fixed
CVF-83	Moderate	Unclear behavior	Info
CVF-84	Minor	Suboptimal	Info
CVF-85	Minor	Suboptimal	Fixed
CVF-86	Minor	Suboptimal	Fixed
CVF-87	Minor	Suboptimal	Fixed

ID	Severity	Category	Status
CVF-88	Minor	Suboptimal	Info
CVF-89	Minor	Unclear behavior	Info
CVF-90	Minor	Suboptimal	Info
CVF-91	Minor	Suboptimal	Info
CVF-92	Minor	Suboptimal	Fixed
CVF-93	Minor	Readability	Fixed
CVF-94	Minor	Readability	Fixed
CVF-95	Minor	Flaw	Fixed
CVF-96	Minor	Suboptimal	Fixed
CVF-97	Minor	Documentation	Fixed
CVF-98	Minor	Flaw	Info
CVF-99	Minor	Bad datatype	Info
CVF-100	Minor	Suboptimal	Info
CVF-101	Minor	Unclear behavior	Fixed
CVF-102	Major	Flaw	Fixed
CVF-103	Minor	Unclear behavior	Fixed
CVF-104	Minor	Bad naming	Fixed
CVF-105	Minor	Readability	Fixed
CVF-106	Minor	Suboptimal	Fixed
CVF-107	Minor	Documentation	Info
CVF-108	Minor	Suboptimal	Info
CVF-109	Minor	Overflow/Underflow	Info
CVF-110	Minor	Suboptimal	Info
CVF-111	Minor	Suboptimal	Info
CVF-112	Minor	Suboptimal	Info
CVF-113	Minor	Suboptimal	Info
CVF-114	Minor	Procedural	Fixed
CVF-115	Minor	Suboptimal	Fixed
CVF-116	Minor	Suboptimal	Fixed
CVF-117	Minor	Suboptimal	Fixed

ID	Severity	Category	Status
CVF-118	Moderate	Flaw	Fixed
CVF-119	Moderate	Overflow/Underflow	Info
CVF-120	Minor	Suboptimal	Info
CVF-121	Minor	Suboptimal	Info
CVF-122	Minor	Suboptimal	Info
CVF-123	Minor	Suboptimal	Info
CVF-124	Minor	Suboptimal	Info
CVF-125	Minor	Unclear behavior	Info
CVF-126	Minor	Suboptimal	Fixed
CVF-127	Minor	Suboptimal	Info
CVF-128	Minor	Documentation	Fixed
CVF-129	Minor	Documentation	Info
CVF-130	Minor	Suboptimal	Info
CVF-131	Minor	Documentation	Fixed
CVF-132	Minor	Suboptimal	Fixed
CVF-133	Minor	Bad datatype	Info
CVF-134	Minor	Suboptimal	Info
CVF-135	Moderate	Flaw	Fixed
CVF-136	Moderate	Flaw	Fixed
CVF-137	Minor	Suboptimal	Fixed
CVF-138	Minor	Suboptimal	Fixed
CVF-139	Minor	Flaw	Fixed
CVF-140	Minor	Suboptimal	Fixed
CVF-141	Minor	Suboptimal	Info
CVF-142	Minor	Suboptimal	Info
CVF-143	Moderate	Flaw	Fixed
CVF-144	Minor	Suboptimal	Fixed
CVF-145	Minor	Documentation	Info
CVF-146	Minor	Bad datatype	Info
CVF-147	Minor	Unclear behavior	Fixed

ID	Severity	Category	Status
CVF-148	Minor	Documentation	Info
CVF-149	Minor	Bad datatype	Info
CVF-150	Minor	Bad datatype	Info
CVF-151	Minor	Suboptimal	Fixed
CVF-152	Minor	Suboptimal	Info
CVF-153	Minor	Readability	Fixed
CVF-154	Minor	Readability	Fixed
CVF-155	Minor	Documentation	Info
CVF-156	Minor	Procedural	Fixed
CVF-157	Major	Flaw	Fixed
CVF-158	Minor	Documentation	Fixed
CVF-159	Minor	Suboptimal	Info
CVF-160	Minor	Suboptimal	Info
CVF-161	Minor	Unclear behavior	Info
CVF-162	Minor	Unclear behavior	Fixed
CVF-163	Minor	Documentation	Fixed
CVF-164	Moderate	Documentation	Info
CVF-165	Minor	Bad datatype	Info
CVF-166	Minor	Unclear behavior	Info
CVF-167	Minor	Bad datatype	Info
CVF-168	Minor	Unclear behavior	Fixed
CVF-169	Minor	Bad datatype	Info
CVF-170	Minor	Bad naming	Info
CVF-171	Major	Unclear behavior	Fixed
CVF-172	Minor	Suboptimal	Fixed
CVF-173	Minor	Suboptimal	Info
CVF-174	Minor	Bad datatype	Info
CVF-175	Minor	Bad datatype	Info
CVF-176	Minor	Bad datatype	Info
CVF-177	Minor	Bad naming	Info

ID	Severity	Category	Status
CVF-178	Major	Flaw	Info
CVF-179	Minor	Procedural	Info
CVF-180	Minor	Procedural	Info
CVF-181	Minor	Documentation	Info
CVF-182	Minor	Bad datatype	Info
CVF-183	Minor	Documentation	Info
CVF-184	Minor	Bad naming	Info
CVF-185	Minor	Bad datatype	Info

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	March 15, 2022	D. Khovratovich	Initial Draft
0.2	March 15, 2022	D. Khovratovich	Minor revision
1.0	March 15, 2022	D. Khovratovich	Release
1.1	May 15, 2022	D. Khovratovich	Customer comment check
1.2	May 15, 2022	D. Khovratovich	Files list and repo added
1.3	May 20, 2022	D. Khovratovich	Minor revision
2.0	May 20, 2022	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

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.
We have reviewed the contracts at repository:

- interfaces/common/IERC1271.sol
- interfaces/common/IERC20.sol
- interfaces/common/IERC721Descriptor.sol
- interfaces/common/IMulticall.sol
- interfaces/common/IWETH.sol
- interfaces/hub/positions/IMuffinHubPositions.sol
- interfaces/hub/positions/IMuffinHubPositionsActions.sol
- interfaces/hub/positions/IMuffinHubPositionsView.sol
- interfaces/hub/IMuffinHub.sol
- interfaces/hub/IMuffinHubActions.sol
- interfaces/hub/IMuffinHubBase.sol
- interfaces/hub/IMuffinHubEvents.sol
- interfaces/hub/IMuffinHubView.sol
- interfaces/IMuffinHubCallbacks.sol
- libraries/math/EMAMath.sol
- libraries/math/FullMath.sol
- libraries/math/Math.sol
- libraries/math/PoolMath.sol
- libraries/math/SwapMath.sol
- libraries/math/TickMath.sol
- libraries/utils/PathLib.sol
- libraries/utils/SafeTransferLib.sol
- libraries/Constants.sol
- libraries/Pools.sol

- libraries/Settlement.sol
- libraries/Positions.sol
- libraries/TickMaps.sol
- libraries/Ticks.sol
- libraries/Tiers.sol
- MuffinHub.sol
- MuffinHubBase.sol
- MuffinHubPositions.sol

The fixes were provided in a new commit.

2.1 About ABDK

ABDK Consulting, 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.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 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 and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **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.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Pools.sol

Recommendation Should be "^0.8.0". Also relevant for the next files: TickMath.sol, SwapMath.sol, SafeTransferLib.sol, PathLib.sol, Constants.sol, PoolMath.sol, Math.sol, FullMath.sol, EMAMath.sol, Settlement.sol, Positions.sol, TickMaps.sol, Ticks.sol, Tiers.sol.

Listing 1:

```
2 pragma solidity 0.8.10;
```

3.2 CVF-2

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Pools.sol

Description The same constant is defined in "SwapMath.sol".
Recommendation Consider defining in one place.
Client Comment Fixed-size array initialization requires the size constant to be defined in the same contract or library scope. This is why this constant is defined twice.

Listing 2:

```
30 uint256 internal constant MAX_TIERS = 6;
```

3.3 CVF-3

- **Severity** Major
- **Status** Info
- **Category** Suboptimal
- **Source** Pools.sol

Description Whether this amount is small or large depends on the token price and decimals number, which couldn't be predicted.
Recommendation Consider making this value configurable per pool or measure the tolerance as a fraction of a swap amount.
Client Comment This is a conjectural amount needed to account for the rounding error in the swap calculation. There's no need to make it larger and no room to make it smaller. Realistically, this amount is of small value in most common pairs. If it represents a large value for a token, we suggest users wrap the token first.

Listing 3:

```
31 int256 internal constant SWAP_AMOUNT_TOLERANCE = 100; //
    ↪ tolerance between desired and actual swap amounts
```

3.4 CVF-4

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** Pools.sol

Description The semantics of the keys in these mappings in unclear consider documenting.
Recommendation Alternatively consider giving more descriptive names to the mappings such as "tierToTickToSideToSettlement".
Client Comment Noted. For us, overly descriptive names make the code hard to read too.

Listing 4:

```
57 mapping(uint256 => TickMaps.TickMap) tickMaps;
   mapping(uint256 => mapping(int24 => Ticks.Tick)) ticks;
   mapping(uint256 => mapping(int24 => Settlement.Info[2]))
      ↪ settlements;
60 mapping(bytes32 => Positions.Position) positions;
```

3.5 CVF-5

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Pools.sol

Recommendation The type of these arguments should be "IERC20".
Client Comment Noted. We prefer fewer dependencies on other interfaces from our interfaces.

Listing 5:

```
75 address token0,
   address token1
```

3.6 CVF-6

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Pools.sol

Description This function doesn't ensure that token0 < token1, this it may use invalid pool IDs.
Recommendation Consider either requiring token0 < token1 or reordering them in case the order is wrong. Also, consider requiring that token0 != token1.
Client Comment The function (and also the library) is intended to be agnostic to anything about the tokens, including how the tokens should be ordered.

Listing 6:

```
78 poolId = keccak256(abi.encode(token0, token1));
```

3.7 CVF-7

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** Pools.sol

Description This function doesn't initialize "pool.unlocked" to true, so the caller would have to do this afterwards. This in unobvious and error-prone.
Recommendation Consider either unlocking the pool inside this function or adding a warning comment telling that this function leaves the pool locked.
Client Comment Added a warning comment.

Listing 7:

```
86 function initialize(
```

3.8 CVF-8

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Pools.sol

Description Assuming that certain arguments are checked saves a marginal amount of gas but makes the code more error-prone.
Recommendation Consider always checking arguments when such checks are cheap.

Listing 8:

```
90 uint8 tickSpacing, // assume checked
   uint8 protocolFee // assume checked
```

3.9 CVF-9

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** Pools.sol

Description The semantics of the returned amounts is unclear.
Recommendation Consider documenting and/or giving more descriptive names to the returned values.

Listing 9:

```
92 ) internal returns (uint256 amount0, uint256 amount1) {

108         uint256 amount0,
            uint256 amount1,

123 ) internal returns (uint256 amount0, uint256 amount1) {
```

3.10 CVF-10

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Pools.sol

Description Passing a zero value as "tickSpacing" would allow initializaing more than once.
Recommendation Consider explicitly requiring "tickSpacing" to be non-zero.

Listing 10:

```
93 require(pool.tickSpacing == 0); // ensure not initialized

97 pool.tickSpacing = tickSpacing;
```

3.11 CVF-11

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Pools.sol

Description So, the "min" value is inclusive and the "max" value is exclusive. This is inconsistent and error-prone.
Recommendation Consider making both values inclusive, as this is how "minimum" and "maximum" are usually understood.

Listing 11:

```
94 require(Constants.MIN_SQRT_P <= sqrtPrice && sqrtPrice <
     ↪ Constants.MAX_SQRT_P);
```

3.12 CVF-12

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Pools.sol

Recommendation These values should be named constants.
Client Comment Not going to change. These constants are only used here and also documented inline.

Listing 12:

```
95 require(sqrtGamma == 99850 || sqrtGamma == 99975); // mandatory
     ↪ 30bps or 5bps as initial fee
```

3.13 CVF-13

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Pools.sol

Description These functions lock the pool as a side effect. This is unobvious and error-prone.
Recommendation Consider either leaving pool locking to the caller or adding a waning comment telling that these functions leave the pool locked.
Client Comment Added warning comments.

Listing 13:

```
113 lock(pool);

282 lock(pool);

496 lock(pool);

798 lock(pool);
```

3.14 CVF-14

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Pools.sol

Recommendation This value should be a named constant.

Listing 14:

```
126 require(sqrtGamma <= 100000);

188 require(sqrtGamma <= 100000);
```

3.15 CVF-15

- **Severity** Moderate
- **Status** Info
- **Category** Unclear behavior
- **Source** Pools.sol

Description The min and the max ticks are actually valid ticks, just very unlikely to ever be reached.
Recommendation Consider using special values such as MIN_TICK - 1 and MAX_TICK+1 as indicators of no ticks above or below respectively.
Client Comment The "nextTickBelow" or "nextTickAbove" should be seen as the next tick (i.e. the next price point) at which the tier needs to pause during a swap (e.g. to cross tick and update active liquidity). Thus, here setting them to min and max tick makes sense because swap needs to stop at min or max tick too.

Listing 15:

```
135 nextTickBelow: Constants.MIN_TICK,
    nextTickAbove: Constants.MAX_TICK,
```

3.16 CVF-16

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** Pools.sol

Description Initializing the min and the max ticks in a special way assumes these ticks will never be cleared, which is not guaranteed.
Recommendation Consider using MIN_TICK - 1 and MAX_TICK + 1 as special ticks at the ends of the list, as such ticks guaranteed to never be used.
Client Comment Same comment as in CVF-40.

Listing 16:

```
142 // initialize min tick & max tick
```

3.17 CVF-17

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Pools.sol

Description These 80 and 64 bits shifts are confusing, as they hide 8-bits base liquidity shifts.
Recommendation Consider shifting the vase liquidity by 8 bits explicitly. Compiler is smart enough to calculate constant expressions at compile time.

Listing 17:

```
161 amount0 = UnsafeMath.ceilDiv(uint256(Constants.BASE_LIQUIDITY_D8
    ↪ ) << 80, sqrtPrice);
amount1 = UnsafeMath.ceilDiv(uint256(Constants.BASE_LIQUIDITY_D8
    ↪ ) * sqrtPrice, 1 << 64);
```

3.18 CVF-18

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Pools.sol

Recommendation These functions should log some events.
Client Comment Events are logged outside these functions in hub contract.

Listing 18:

```
169 function setPoolParameters(

180 function setTierParameters(
```

3.19 CVF-19

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Pools.sol

Description There is no range check for this argument .
Recommendation Consider adding an appropriate check.
Client Comment No range check is correct. Every possible value in uin8 is valid.

Listing 19:

```
172 uint8 protocolFee
```

3.20 CVF-20

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Pools.sol

Description There are no range checks for the elements of this array.
Recommendation Consider adding appropriate checks.
Client Comment No range check is correct. Every possible value in uin8 is valid.

Listing 20:

```
184 uint8 limitOrderTickSpacingMultiplier
```

3.21 CVF-21

- **Severity** Minor
- **Status** Info
- **Category** Overflow/Underflow
- **Source** Pools.sol

Description Overflow is possible here.
Recommendation Consider using safe conversion.
Client Comment Overflow is expected.

Listing 21:

```
205 uint32 timestamp = uint32(block.timestamp);
```

3.22 CVF-22

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** Pools.sol

Recommendation Consider initializing to 0 for readability.

Listing 22:

```
211 uint256 sumL;
    int256 sumLTick; // sum of liquidity * tick (Q24 * UQ128)
```

3.23 CVF-23

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** Pools.sol

Description Here the whole "Tier" structure is read into the memory, while only two fields from it are actually used.
Recommendation Consider reading just these two fields.
Client Comment This function is called during swap, and swap will load the whole tier array into memory anyway. In any case, we favour our code to save gas for swap.

Listing 23:

```
214 Tiers.Tier memory tier = tiers[i];
```

3.24 CVF-24

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Pools.sol

Recommendation Consider calculating this value with sub-tick precision.
Client Comment Noted. We think the current precision is fine.

Listing 24:

```
218 tickCum += int56((sumLTick * int256(uint256(secs))) / int256(
    ↪ sumL));
```

3.25 CVF-25

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Pools.sol

Description The same constant is defined in "SwapMath.sol"
Recommendation Consider defining in one place.

Listing 25:

```
239 int256 private constant REJECTED = type(int256).max; //
    ↪ represents the tier is rejected for the swap
```

3.26 CVF-26

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Pools.sol

Recommendation Using the "type(int256).min" value as a special one would be better, as it would make the number or positive and negative non-special values to be the same.
Client Comment Agree, but not refactoring it at this point since we may have to check the related maths again.

Listing 26:

```
239 int256 private constant REJECTED = type(int256).max; //
      ↪ represents the tier is rejected for the swap
```

3.27 CVF-27

- **Severity** Minor
- **Status** Info
- **Category** Overflow/Underflow
- **Source** Pools.sol

Description There can be no overflow in this block unless there can be more than 255 tiers (shifts do not overflow), so it is probably redundant.

Listing 27:

```
285 unchecked {
        if (amtDesired == 0 || amtDesired == REJECTED) revert
          ↪ InvalidAmount();
        if (tierChoices > 0x3F || tierChoices & ((1 << tiers.length)
          ↪  - 1) == 0) revert InvalidTierChoices();
    }
```

3.28 CVF-28

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Pools.sol

Recommendation The value "0x3F" should be derived from the "MAX_TIERS" constant.

Listing 28:

```
287 if (tierChoices > 0x3F || tierChoices & ((1 << tiers.length) -
      ↪ 1) == 0) revert InvalidTierChoices();
```

3.29 CVF-29

- **Severity** Minor
- **Status** Info
- **Category** Flaw
- **Source** Pools.sol

Description This check allows tiers above "tier.length" but below "MAX_TIERS" to be chosen.
Recommendation Consider forbidding this.
Client Comment Not a flaw. This is allowed.

Listing 29:

```
287 if (tierChoices > 0x3F || tierChoices & ((1 << tiers.length) -
    ↪ 1) == 0) revert InvalidTierChoices();
```

3.30 CVF-30

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Pools.sol

Description The expression "amtDesired > 0" is calculated twice.
Recommendation Consider calculating once and reusing.
Client Comment No changes due to being "stack too deep".

Listing 30:

```
293 zeroForOne: isToken0 == (amtDesired > 0),
    exactIn: amtDesired > 0,
```

3.31 CVF-31

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Pools.sol

Description The expression "amtDesired - amountA" is calculated several times.
Recommendation Consider calculating once and reusing.

Listing 31:

```
305     ? SwapMath.calcTierAmtsIn(tiers, isToken0, amtDesired -
          ↪ amountA, tierChoices)
        : SwapMath.calcTierAmtsOut(tiers, isToken0, amtDesired -
          ↪ amountA, tierChoices);

316 int256 amtRemaining = amtDesired - amountA;
```

3.32 CVF-32

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Pools.sol

Description The expression "tierChoices & ((1 « tiers.length) - 1)" is calculated on every loop iteration.
Recommendation Should be calculated once before the loop.

Listing 32:

```
320 cache.priceBoundReached == tierChoices & ((1 << tiers.length) -
    ↪ 1)
```

3.33 CVF-33

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Pools.sol

Description This function should return early in case the "priceBoundReached" bit is set for the tier.

Listing 33:

```
329 function _swapStep(
```

3.34 CVF-34

- **Severity** Minor
- **Status** Info
- **Category** Overflow/Underflow
- **Source** Pools.sol

Description Overflow is possible here.
Recommendation Consider using safe addition.
Client Comment Overflow is acceptable and realistically rare.

Listing 34:

```
365 cache.protocolFeeAmt += protocolFeeAmt;
```

3.35 CVF-35

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** Pools.sol

Recommendation It would be more efficient to just remove the tiers that reached price boundary from "tierChoices" rather then to collect them in a separate bit mask. This way such tiers will not be considered on future iterations.

Listing 35:

```
383 cache.priceBoundReached |= 1 << tierId;
```

3.36 CVF-36

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Pools.sol

Recommendation Consider using an expression instead of a hardcoded value. Compiler is smart enough to calculate constant expressions at compile time.

Listing 36:

```
429 bool noOverflow = amtIn < (1 << 215); // 256 - 41 bits
```

3.37 CVF-37

- **Severity** Major
- **Status** Fixed
- **Category** Procedural
- **Source** Pools.sol

Description These constants depend on MAX_TIERS.
Recommendation Consider deriving them from it.

Listing 37:

```
429 bool noOverflow = amtIn < (1 << 215); // 256 - 41 bits

450                     (noOverflow ? (state.amountIn << 41) / amtIn
                        ↪  : state.amountIn / ((amtIn >> 41) +
                        ↪ 1)) <<
                    (uint256(i) * 42);
```

3.38 CVF-38

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Pools.sol

Description The "tickLower >= constants.MAX_TICK" subexpression is redundant as it may never be true once both "tickLower >= tickUpper" and "tickUpper > Constants.MAX_TICK" are false.

Listing 38:

```
464 (tickLower >= tickUpper) ||
(Constants.MIN_TICK > tickLower || tickLower >= Constants.
    ↪ MAX_TICK) ||
(Constants.MIN_TICK >= tickUpper || tickUpper > Constants.
    ↪ MAX_TICK)
```

3.39 CVF-39

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Pools.sol

Description The "Constants.MIN_TICK >= tickUpper" subexpression is redundant as it may never be true once both "tickLower >= tickUpper" and "Constants.MIN_TICk > tickLower" are false.

Listing 39:

```
464 (tickLower >= tickUpper) ||
(Constants.MIN_TICK > tickLower || tickLower >= Constants.
    ↪ MAX_TICK) ||
(Constants.MIN_TICK >= tickUpper || tickUpper > Constants.
    ↪ MAX_TICK)
```

3.40 CVF-40

- **Severity** Moderate
- **Status** Fixed
- **Category** Unclear behavior
- **Source** Pools.sol

Description Here a special tick, i.e. MIN_TICK or MAX_TICK, could in theory be deleted.
Recommendation Consider preventing this in some way or explaining why thsi is impossible in practice.
Client Comment Ticks are cleared when both "tick.liquidityLowerD8" and "tick.liquidityUpperD8" become zero. Since every pool has an unremovable base liquidity ranging from min to max tick, the concern boils down to whether "minTick.liquidityLowerD8" and "maxTick.liquidityUpperD8" would deplete to zero, which is not possible unless having bugs. And practically, the room for unknown bugs is small since the updates of these values are just simple arithemetic, even without division.
In any case, code is updated to revert when clearing min or max tick, as an extra safety measure.

Listing 40:

```
617 delete ticks[tick];
```

3.41 CVF-41

- **Severity** Major
- **Status** Info
- **Category** Suboptimal
- **Source** Pools.sol

Description As "limitOrderType" is "uint8" there are many more possible values that are not limit orders.
Recommendation Consider checking for known limit order position types instead.
Client Comment Not fixing because practically in the code there're only three possible values for "limitOrderType", even though it's an "uint8". Those values are defined in Positions.sol.

Listing 41:

```
680 if (position.limitOrderType != Positions.NOT_LIMIT_ORDER) {

734 if (position.limitOrderType != Positions.NOT_LIMIT_ORDER) {
```

3.42 CVF-42

- **Severity** Major
- **Status** Info
- **Category** Suboptimal
- **Source** Pools.sol

Description In case the new limit order type is the same as the current one, this function does lots of unnecessary work just to leave the position type unchanged .
Recommendation Consider doing nothing or reverting in such a case.
Client Comment Agree it does unnecessary work, but we still want to check if position is settled or has invalid tick range and then raise error for that. Turning it to a no-op will skip these checks. Reverting is also not considered as it breaks the idempotency of the function.

Listing 42:

```
709 /// @dev It first unsets position from being a limit order (if
    ↪ it is), then set position to a new limit order type
```

3.43 CVF-43

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Pools.sol

Description This check relies on particular values of limit order type constants. Using an enum type for "limitOrderType" would make this unnecessary.
Client Comment Have tested enum but it exceessively performs runtime check everytime it's used. The runtime check bytecode is duplicated too many times to the extent that the optimizer runs need to be tuned down, which is a significant cost. Therefore, so we decided not to change it.

Listing 43:

```
720 require(limitOrderType <= Positions.ONE_FOR_ZERO);
```

3.44 CVF-44

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** Pools.sol

Description While token amounts taken from removing liquidity are all converted to the desired side, the fee amounts are not converted, so if one placed a limit order to convert from tokenA to tokenB, he will still collect some amount of tokenA as fees, which seems odd.
Recommendation Consider converting fees as well.
Client Comment No changes. We should let users do the swapping themselves.

Listing 44:

```
794 uint256 feeAmtOut0,
    uint256 feeAmtOut1
```

3.45 CVF-45

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** MuffinHub.sol

Recommendation Should be "^0.8.0" unless there is something special about this version. Also revelant for the next files: MuffinHubPositions.sol, MuffinHubBase.sol.
Client Comment No changes as we don't want others to compile the contracts in other versions.

Listing 45:

```
2 pragma solidity 0.8.10;
```

3.46 CVF-46

- **Severity** Major
- **Status** Info
- **Category** Procedural
- **Source** MuffinHub.sol

Description The way how these functions are currently implemented doesn't allow using "withdraw" inside a deposit callback.
Recommendation Consider implementing such possibility in some way.
Client Comment The contract is expected to not support "withdraw" inside "deposit" callback. The only signifcant downside is user cannot perform flashloan in this way, which we think it's acceptable.

Listing 46:

```
36 function deposit(

52 function withdraw(
```

3.47 CVF-47

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** MuffinHub.sol

Recommendation The type of the token arguments should be "IERC20".
Client Comment Noted. Prefer fewer dependencies on other interfaces from our interfaces.

Listing 47:

```
39  address token,

55  address token,

69  address token0,
70  address token1,

90  address token0,
    address token1,

110 address tokenIn,
    address tokenOut,

174 address tokenIn,
    address tokenOut,

210 address tokenIn,
    address tokenOut,
```

3.48 CVF-48

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MuffinHub.sol

Recommendation Consider doing nothing if the amount is 0.
Client Comment User should not call these functions if their "amount" input is 0.

Listing 48:

```
40  uint256 amount,

56  uint256 amount
```

3.49 CVF-49

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** MuffinHub.sol

Description The account's balance is increased by "amount" even if callback actually transferred more tokens.
Recommendation Consider removing the "amount" argument and depositing as many tokens as the callback actually transferred.
Client Comment Decided to keep it and treat the argument as a value to check against the actual change in token balance after the callback. This is to avoid user's loss of fund due to any unaware decrease in token balance during deposit callback.

Listing 49:

```
47 accounts[token][getAccHash(recipient, recipientAccRefId)] +=
    ↪ amount;
```

3.50 CVF-50

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** MuffinHub.sol

Description This function doesn't check token lock, thus it could be called inside a deposit callback, while the outcome of such call would most probably be not what the caller wants.
Recommendation Consider either supporting using "withdraw" inside a deposit callback, or checking that the token is not locked inside "withdraw".
Client Comment Added a warning comment instead.

Listing 50:

```
52 function withdraw(
```

3.51 CVF-51

- **Severity** Minor
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** MuffinHub.sol

Description Here compiler enforced underflow check is used to verify a business-level constraint. This is a bad practice, as it makes the code harder to read.
Recommendation Consider adding an explicit "require" statement.

Listing 51:

```
58 accounts[token][getAccHash(msg.sender, senderAccRefId)] -=
     ↪ amount;

79 accounts[token0][getAccHash(msg.sender, senderAccRefId)] -=
     ↪ amount0;
80 accounts[token1][getAccHash(msg.sender, senderAccRefId)] -=
     ↪ amount1;

97 accounts[token0][getAccHash(msg.sender, senderAccRefId)] -=
     ↪ amount0;
   accounts[token1][getAccHash(msg.sender, senderAccRefId)] -=
     ↪ amount1;
```

3.52 CVF-52

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** MuffinHub.sol

Description This function should return the ID of the created pool.

Listing 52:

```
68 function createPool(
```

3.53 CVF-53

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** MuffinHub.sol

Description A pool can be created by anyone, and the creator may choose a value for "sqrtGamma", while only the governance may change this value after a pool was create. This looks odd.
Recommendation Consider creating all pools with the default gamma value specified by the governance. Alternatively, consider allowing only the governance to create pools.

Listing 53:

```
71 uint24 sqrtGamma,
```

3.54 CVF-54

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MuffinHub.sol

Description The fact that token addresses should be properly ordered is an internal specific of this particular implementation.
Recommendation Consider reordering the tokens if the order is incorrect, rather than reverting.
Client Comment Noted. Prefer callers doing it by themselves.

Listing 54:

```
75 if (token0 >= token1 || token0 == address(0)) revert
   ↪ InvalidTokenOrder();
```

3.55 CVF-55

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MuffinHub.sol

Description The pool locks himself but needs an external call to gets unlocked. This looks odd and error-prone.
Recommendation Consider locking and unlocking in the same place.
Client Comment "Unlock" needs to be done after token transfer, so it must be in hub contract. Putting "lock" into Pools library seems odd but it makes the lock's logic crystal clear. This sacrifices code readability but it mininizes the chance of forgetting to lock pool — which can be catastrophic.

Listing 55:

```
84  pool.unlock();

101 pool.unlock();

122 pool.unlock();
```

3.56 CVF-56

- **Severity** Major
- **Status** Fixed
- **Category** Documentation
- **Source** MuffinHub.sol

Description Whether this extra cost is small or not depends on the value of a token base unit that is hard to predict. For some tokens 100 base units could have a significant value.
Client Comment The 100 base units are practically of small value in most common token pairs. Comments are updated anyway for clarity.

Listing 56:

```
140 // for the previous swap. The extra token is not refunded and
    ↪ thus results in a very small extra cost.
```

3.57 CVF-57

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** MuffinHub.sol

Description It is unclear what a positive and a negative value of this argument mean.
Recommendation Consider explaining in a documentation comment.

Listing 57:

```
177 int256 amountDesired,
```

3.58 CVF-58

- **Severity** Minor
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** MuffinHub.sol

Description Overflow is possible here.

Listing 58:

```
202 if (protocolFeeAmt != 0) tokens[tokenIn].protocolFeeAmt +=
    ↪ uint248(protocolFeeAmt);
```

3.59 CVF-59

- **Severity** Major
- **Status** Info
- **Category** Procedural
- **Source** MuffinHub.sol

Description In case the callback transferred more tokens than needed, extra tokens are not refunded.
Recommendation Consider depositing them.
Client Comment Not changing. The callback already has an argument to specify the amount needed, so there's no reason users would still send more. Also, adding the "deposit" functionality into "swap" complicates things.

Listing 59:

```
234 checkBalanceAndUnlock(tokenIn, balanceBefore + amountIn);
```

3.60 CVF-60

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** MuffinHub.sol

Description The expression "pools[poolId]" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 60:

```
247 return (pools[poolId].tickSpacing, pools[poolId].protocolFee);
```

3.61 CVF-61

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MuffinHub.sol

Description There is no range check for this argument.
Recommendation Consider adding an appropriate check.
Client Comment Not changing. Users should be aware of the supported tick range.

Listing 61:

```
265 int24 tick
```

3.62 CVF-62

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** MuffinHub.sol

Description Blindly forwarding all unrecognized requests to anther contract is a bad practice. Better approach would be to declare all the entry points explicitly in the main contract and move implementations of some functions into libraries deployed separately. This way compiler will be able to generate full API description from the main contract.
Client Comment Agree, but the suggested alternative significantly increases contract bytecode size, to the extent that we need either move more functions to the secondary contract or tune down the optimizer runs. Both ways add non-negligible gas costs, so we don't consider.

Listing 62:

```
301 fallback() external {
```

3.63 CVF-63

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** MuffinHub.sol

Description Most of the logic of this assembly block could be written in plain Solidity. Only the revert and return logic in the end actually requires assembly.
Recommendation Consider using as little assembly as possible.
Client Comment The code is from openzepplin and was audited beforehand, so we prefer not to "reinvent the wheel".

Listing 63:

```
303 assembly {
```

3.64 CVF-64

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** MuffinHubBase.sol

Recommendation The type of these fields should be "IERC20".
Client Comment Noted. Prefer fewer dependencies on other interfaces from our interfaces.

Listing 64:

```
18 address token0;
   address token1;
```

3.65 CVF-65

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** MuffinHubBase.sol

Recommendation If the variables are mutable they are not default otherwise they should be declared as constants.
Client Comment Not agree. They are the default parameters for new pools and these defaults are allowed to be updated as well.

Listing 65:

```
25 uint8 internal defaultTickSpacing = 200;

27 uint8 internal defaultProtocolFee = 0;
```

3.66 CVF-66

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** MuffinHubBase.sol

Recommendation The type of the "token" arguments should be "IERC20".
Client Comment Noted. Prefer fewer dependencies on other interfaces from our interfaces.

Listing 66:

```
39 function getBalance(address token) private view returns (uint256
   ↪ ) {

46 function getBalanceAndLock(address token) internal returns (
   ↪ uint256) {

53 function checkBalanceAndUnlock(address token, uint256
   ↪ balanceMinimum) internal {
```

3.67 CVF-67

- **Severity** Major
- **Status** Fixed
- **Category** Procedural
- **Source** MuffinHubBase.sol

Description In case the returned data length is greater than 32 bytes, extra bytes are silently ignored.
Recommendation Consider reverting in such a case.

Listing 67:

```
41 if (!success || data.length < 32) revert FailedBalanceOf();
```

3.68 CVF-68

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** MuffinHubBase.sol

Recommendation Note that this method does not prevent reentering the contract with another token.
Client Comment Reentering the contract is allowed.

Listing 68:

```
45 /// @dev "Lock" the token so the token cannot be used as input
   ↪ token again until unlocked
```

3.69 CVF-69

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** MuffinHubBase.sol

Description The expression "token[token]" is calculated twice.

Listing 69:

```
47 require(tokens[token].locked != 1); // 1 means locked
tokens[token].locked = 1;
```

3.70 CVF-70

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** MuffinHubBase.sol

Recommendation There should be names constants for the values of the "locked" field.
Client Comment The constants are not reused elsewhere so we prefer keeping them unnamed. Documented inline to clarify the meaning of the values.

Listing 70:

```
47 require(tokens[token].locked != 1); // 1 means locked
   tokens[token].locked = 1;

55 tokens[token].locked = 2;
```

3.71 CVF-71

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** IERC20.sol

Recommendation Should be "^0.8.0" as compatibility with future major releases cannot be guaranteed. Also relevant for the next files: IMuffinHubView.sol, IMuffinHubPositions.sol, IMuffinHubActions.sol, IMuffinHubPositionsView.sol, IMuffinHubCallbacks.sol, IMuffinHub.sol, IMuffinHubPositionsActions.sol, IMuffinHubEvents.sol, IMuffinHubBase.sol, IWETH.sol, IMulticall.sol, IERC721Descriptor.sol.

Listing 71:

```
2 pragma solidity >=0.8.0;
```

3.72 CVF-72

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** IERC20.sol

Description This interface declares only a subset of the ERC-20 API.
Recommendation Consider renaming to "IERC20Minimal" or something like this to emphasize that this interface is incomplete.

Listing 72:

```
4 interface IERC20 {
```

3.73 CVF-73

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** TickMath.sol

Description These variables are also declared in Constants.sol .
Recommendation Consider using a single source.

Listing 73:

```
 8 int24 internal constant MIN_TICK = -776363;
   int24 internal constant MAX_TICK = 776363;
10 uint128 internal constant MIN_SQRT_P = 65539;
   uint128 internal constant MAX_SQRT_P =
     ↪ 340271175397327323250730767849398346765;
```

3.74 CVF-74

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** TickMath.sol

Description It is unclear how all these magic numbers were calculated. They are different from what Uniswap V3 uses. BTW, Uniswap shared a link to the explanation regarding where their constants came from: github.com/Uniswap/v3-core/issues/500
Client Comment In short, for each tick index, we calculate its sqaure root price and then convert it back to an approximated tick index with 128 fraction bits. We then compute the difference between the actual tick index and the approximated value, i.e. the approximation errors. And these magic numbers are the maximum approximation errors in specific tick ranges.

Listing 74:

```
113 int24 tickUpper = int24((r +
      ↪ 17996007701288367970265332090599899137) >> 128);
    int24 tickLower = int24(
        r < -23015440253774670196347843960637302805014528 ? (r -
          ↪ 985771436367297374661640326341208309 77) >> 128 :
        r < -16209792915355900927080351812001940051381 4528 ? (r -
          ↪ 24359334487912856315448148570931 3) >> 128 :
        r >> 128
    );
```

3.75 CVF-75

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** SwapMath.sol

Recommendation Using the "type(int256).min" value as a special one would be better, as it would make the number or positive and negative non-special values to be the same.
Client Comment Agree, but not refactoring it at this point since we may have to check the related maths again.

Listing 75:

```
14 int256 private constant REJECTED = type(int256).max;
```

3.76 CVF-76

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** SwapMath.sol

Description The exact semantics of these functions is unclear.
Recommendation Consider documenting.

Listing 76:

```
19 function calcTierAmtsIn(

74 function calcTierAmtsOut(
```

3.77 CVF-77

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** SwapMath.sol

Recommendation The final amounts linearly depend on the "liquidity" value, so instead of multiplying every individual "lsg" and "res" element by liquidity, consider multiplying the calculated amount at the very end of the function. This would require dividing the amount by the liquidity before adding to the denominator.

Listing 77:

```
42 num += (lsg[i] = UnsafeMath.ceilDiv(liquidity * 1e5, sqrtGamma))
       ↪ ;
   denom += (res[i] = isToken0
       ? UnsafeMath.ceilDiv(liquidity * Q72 * 1e10, uint256(t.
          ↪ sqrtPrice) * sqrtGamma * sqrtGamma)
       : UnsafeMath.ceilDiv(liquidity * t.sqrtPrice, (Q72 *
          ↪ sqrtGamma * sqrtGamma) / 1e10));

92 }
   // calculate numerator and denominator of sqrt lamdba (lagrange
       ↪ multiplier)
   Tiers.Tier memory t = tiers[i];
   uint256 liquidity = uint256(t.liquidity);
   num += (lsg[i] = (liquidity * 1e5) / t.sqrtGamma);
   denom += int256(res[i] = isToken0 ? (liquidity << 72) / t.
       ↪ sqrtPrice : (liquidity * t.sqrtPrice) >> 72);
```

3.78 CVF-78

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** SwapMath.sol

Recommendation The values "Q72 * 1e10" and "Q72 / 1e10" could be precomputed.
Client Comment The compiler is smart enough to precompute Q72*1e10. But pre-computing Q72/1e10 losses some precisions, so we don't do it.

Listing 78:

```
44 ? UnsafeMath.ceilDiv(liquidity * Q72 * 1e10, uint256(t.sqrtPrice
       ↪ ) * sqrtGamma * sqrtGamma)
   : UnsafeMath.ceilDiv(liquidity * t.sqrtPrice, (Q72 * sqrtGamma *
       ↪  sqrtGamma) / 1e10));
```

3.79 CVF-79

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** SwapMath.sol

Description The expression "denom * num" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 79:

```
53 bool wontOverflow = ((denom * num) / denom == num) && (denom *
   ↪ num <= uint256(type(int256).max));
```

3.80 CVF-80

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** SwapMath.sol

Recommendation The "i++" should be moved into the brackets after "for".
Client Comment It's not in that bracket because there's a condition where "i" should not increment.

Listing 80:

```
54 for (uint256 i; i < tiers.length; ) {

68     i++;

105 for (uint256 i; i < tiers.length; ) {

115    i++;
```

3.81 CVF-81

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** SwapMath.sol

Description The "mulDiv" function is able to efficiently handle the case when the multiplication result fits into 256 bits.
Recommendation Consider always calling "mulDiv" for code simplicity.
Client Comment Not considering as it adds non-negligible gas costs.

Listing 81:

```
57      (amts[i] = wontOverflow
            ? int256((denom * lsg[i]) / num).sub(int256(res[i]))
            : FullMath.mulDiv(denom, lsg[i], num).toInt256().sub(
              ↪ int256(res[i]))) < 0

180 ? int256((uint256(amtA) * gamma) / 1e10)

196     ? UnsafeMath.ceilDiv(uint256(amtInExclFee) * 1e10, gamma).
          ↪ toInt256()

230 ? UnsafeMath.ceilDiv(uint256(amtInExclFee) * 1e10, gamma).
    ↪ toInt256()
```

3.82 CVF-82

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** SwapMath.sol

Description the subexpression ".sub(int256(res[i])" is coded twice.
Recommendation Consider placing after the ternary operator to avoid code duplication.

Listing 82:

```
58 ? int256((denom * lsg[i]) / num).sub(int256(res[i]))
   : FullMath.mulDiv(denom, lsg[i], num).toInt256().sub(int256(res[
   ↪ i]))) < 0
```

3.83 CVF-83

- **Severity** Moderate
- **Status** Info
- **Category** Unclear behavior
- **Source** SwapMath.sol

Description These changes could make the previously rejected tiers valid again, making the whole logic unpredictable.
Recommendation Consider just reverting in case any of the tiers chosen by the caller ought to be rejected.
Client Comment The caller chooses the tiers they are "willing" to swap at. It does not mean the swap must happen in all of the chosen tiers.

Listing 83:

```
62  num -= lsg[i];
    denom -= res[i];

109 num -= lsg[i];
110 denom -= int256(res[i]);
```

3.84 CVF-84

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** SwapMath.sol

Description This restarts the loop after the first rejected tier, which could lead to O(n^2) complexity.
Recommendation Consider finishing the loop anyway rejected as many tiers as possible, and then restart if any tiers were rejected.
Client Comment Rejecting a tier can possibliy lead to another unrejected tier being needed to reject. We're unsure whether the suggested way is faster in the end, so we keep it unchanged at this point until more research is done.

Listing 84:

```
64  i = 0;

111 i = 0;
```

3.85 CVF-85

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** SwapMath.sol

Recommendation In case "amtA" is really large, it would be fine to first divide by 1e10 and then multiply by gamma. No need for the "mulDiv" function.

Listing 85:

```
181 : int256(FullMath.mulDiv(uint256(amtA), gamma, 1e10));
```

3.86 CVF-86

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** SwapMath.sol

Recommendation The final conversion to "int256" could be performed once after the ternary operator.

Listing 86:

```
196     ? UnsafeMath.ceilDiv(uint256(amtInExclFee) * 1e10, gamma).
          ↪ toInt256()
        : FullMath.mulDivRoundingUp(uint256(amtInExclFee), 1e10,
          ↪ gamma).toInt256();

230 ? UnsafeMath.ceilDiv(uint256(amtInExclFee) * 1e10, gamma).
      ↪ toInt256()
    : FullMath.mulDivRoundingUp(uint256(amtInExclFee), 1e10, gamma).
      ↪ toInt256();
```

3.87 CVF-87

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** SwapMath.sol

Recommendation In case "amtInExclFee" is really large, it would be fine to first divide by gamma and then multiply by 1e10. No need for the "mulDiv" function.

Listing 87:

```
197     : FullMath.mulDivRoundingUp(uint256(amtInExclFee), 1e10,
          ↪ gamma).toInt256();

231 : FullMath.mulDivRoundingUp(uint256(amtInExclFee), 1e10, gamma).
      ↪ toInt256();
```

3.88 CVF-88

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** SafeTransferLib.sol

Recommendation The code would be smaller if "0x23b872dd" would be stored here. This would require the offsets of all other arguments to be increased by 28.
Client Comment Not changing. It costs one more "add" in the "call" below.

Listing 88:

```
48 mstore(freeMemoryPointer, 0
    ↪ x23b872dd0000000000000000000000000000000000000000000000000000000000000000
    ↪ ) // Begin with the function selector.
```

3.89 CVF-89

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** SafeTransferLib.sol

Description Masking is redundant as the argument being masked are already addresses. Is there any way how an attacker may pass a malformed address here that doesn't fit into 160 bits?
Client Comment Consulted the library author but in general we don't have a clear idea how. But to play safe, just keep this part unchanged.

Listing 89:

```
49 mstore(add(freeMemoryPointer, 4), and(from, 0
    ↪ xffffffffffffffffffffffffffffffffffffffff)) // Mask and
    ↪ append the "from" argument.
50 mstore(add(freeMemoryPointer, 36), and(to, 0
    ↪ xffffffffffffffffffffffffffffffffffffffff)) // Mask and
    ↪ append the "to" argument.

74 mstore(add(freeMemoryPointer, 4), and(to, 0
    ↪ xffffffffffffffffffffffffffffffffffffffff)) // Mask and
    ↪ append the "to" argument.

98 mstore(add(freeMemoryPointer, 4), and(to, 0
    ↪ xffffffffffffffffffffffffffffffffffffffff)) // Mask and
    ↪ append the "to" argument.
```

3.90 CVF-90

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** SafeTransferLib.sol

Recommendation The code would be smaller if "0xa9059cbb" would be stored here. This would require the offsets of all other arguments to be increased by 28.
Client Comment Not changing. It costs one more "add" in the "call" below.

Listing 90:

```
73 mstore(freeMemoryPointer, 0
    ↪ xa9059cbb0000000000000000000000000000000000000000000000000000000000000000
    ↪ ) // Begin with the function selector.
```

3.91 CVF-91

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** SafeTransferLib.sol

Recommendation The code would be smaller if "0x095ea7b3" would be stored here. This would require the offsets of all other arguments to be increased by 28.
Client Comment Not changing. It costs one more "add" in the "call" below.

Listing 91:

```
97 mstore(freeMemoryPointer, 0
    ↪ x095ea7b3000000000000000000000000000000000000000000000000000000000000000000
    ↪ ) // Begin with the function selector.
```

3.92 CVF-92

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** SafeTransferLib.sol

Description This variable is redundant, as "returndatasize" is cheaper than "dup".

Listing 92:

```
116 let returnDataSize := returndatasize()
```

3.93 CVF-93

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** PathLib.sol

Recommendation Consider initializing this constant like this: ADDR_UINT8_BYTES = ADDR_BYTES + 1;

Listing 93:

```
6 uint256 internal constant ADDR_UINT8_BYTES = 21;
```

3.94 CVF-94

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** PathLib.sol

Recommendation Consider initializing this constant via an expression, rather than a precomputed value: PATH_MAX_BYTES = ADDR_UINT8_BYTES * 256 - 1;

Listing 94:

```
7 uint256 internal constant PATH_MAX_BYTES = 5396; // 256 pools (i
    ↪ .e. 21 * 256 + 20 = 5396 bytes)
```

3.95 CVF-95

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** PathLib.sol

Description This function is unsafe as it silently returns a meaningless value on an invalid path.
Recommendation Consider ether adding a path validity check or a warning comment about function unsafety.

Listing 95:

```
15 function hopCount(bytes memory path) internal pure returns (
    ↪ uint256) {
```

3.96 CVF-96

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** PathLib.sol

Recommendation Subtraction "ADDR_BYTES" doesn't affect the result for a valid path, and this function is never used for a potentially invalid path, so consider removing the subtraction.

Listing 96:

```
17 return (path.length - ADDR_BYTES) / ADDR_UINT8_BYTES;
```

3.97 CVF-97

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** PathLib.sol

Description These functions are unsafe as they don't check the path length.
Recommendation Consider either adding appropriate checks or adding warning comments about unsafety of the functions.

Listing 97:

```
21 function decodePool(

43 function tokensInOut(bytes memory path, bool exactIn) internal
   ↪ pure returns (address tokenIn, address tokenOut) {

51 function _readAddressAt(bytes memory data, uint256 offset)
   ↪ internal pure returns (address addr) {

57 function _readUint8At(bytes memory data, uint256 offset)
   ↪ internal pure returns (uint8 value) {
```

3.98 CVF-98

- **Severity** Minor
- **Status** Info
- **Category** Flaw
- **Source** MuffinHubPositions.sol

Description There is no explicit check that the pool is initialized.
Recommendation Consider adding such check.
Client Comment Unintialized pool is locked by default.

Listing 98:

```
29  (Pools.Pool storage pool, bytes32 poolId) = pools.getPoolAndId(
      ↪ params.token0, params.token1);

77  (Pools.Pool storage pool, bytes32 poolId) = pools.getPoolAndId(
      ↪ params.token0, params.token1);

116 (Pools.Pool storage pool, bytes32 poolId) = pools.getPoolAndId(
      ↪ token0, token1);

131 (Pools.Pool storage pool, bytes32 poolId) = pools.getPoolAndId(
      ↪ params.token0, params.token1);
```

3.99 CVF-99

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** MuffinHubPositions.sol

Recommendation The type of the token arguments should be "IERC20".
Client Comment Noted. Prefer fewer dependencies on other interfaces from our interfaces.

Listing 99:

```
108     address token0,
        address token1,

288 function collectProtocolFee(address token, address recipient)
      ↪ external onlyGovernance {
```

3.100 CVF-100

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** MuffinHubPositions.sol

Description This event is emitted even if nothing actually changed.
Client Comment Not going to change.

Listing 100:

```
257 emit GovernanceUpdated(_governance);
```

3.101 CVF-101

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** MuffinHubPositions.sol

Recommendation This function should emit some event.

Listing 101:

```
261 function setDefaultParameters(uint8 tickSpacing, uint8
    ↪ protocolFee) external onlyGovernance {
```

3.102 CVF-102

- **Severity** Major
- **Status** Fixed
- **Category** Flaw
- **Source** MuffinHubPositions.sol

Description There are no range checks for the arguments.
Recommendation Consider adding appropriate checks.

Listing 102:

```
261 function setDefaultParameters(uint8 tickSpacing, uint8
    ↪ protocolFee) external onlyGovernance {
```

3.103 CVF-103

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** MuffinHubPositions.sol

Recommendation This function should return the collected amount.

Listing 103:

```
288 function collectProtocolFee(address token, address recipient)
    ↪ external onlyGovernance {
```

3.104 CVF-104

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** PoolMath.sol

Description Using the "uint" type with its alias "uint256" in the same file makes code harder to read.
Recommendation Consider using type names in a consistent way.

Listing 104:

```
 8 using Math for uint256;

30         uint num = uint(liquidity) * (sqrtP0 - sqrtP1);
```

3.105 CVF-105

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** PoolMath.sol

Recommendation Should be "amt0" instead of "amt1".

Listing 105:

```
20 /// if sqrtP0 < sqrtP1 (price goes up):      => amt1 is output
   ↪ => round down
```

3.106 CVF-106

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** PoolMath.sol

Description The term "down" here is confusing, as amount is negative and rounded towards zero, i.e. up.
Recommendation Consider using "away from zero" and "towards zero" instead of "up" and "down".

Listing 106:

```
20 /// if sqrtP0 < sqrtP1 (price goes up):      => amt1 is output
   ↪ => round down

45 /// if sqrtP0 > sqrtP1 (price goes down):    => amt1 is output
   ↪ => round down
```

3.107 CVF-107

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** PoolMath.sol

Description This expression is too complicated and hard to read.
Recommendation Consider using conditional statements instead of ternary operators, and adding some comments.
Client Comment Won't change.

Listing 107:

```
32 amt0 = Math.toInt256(
       num < Q184
           ? (priceUp ? (num << 72) / denom : UnsafeMath.ceilDiv(
              ↪ num << 72, denom))
           : (priceUp ? FullMath.mulDiv(num, Q72, denom) : FullMath
              ↪ .mulDivRoundingUp(num, Q72, denom))
```

3.108 CVF-108

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** PoolMath.sol

Description The "mulDiv" function is able to efficiently handle the case when the multiplication fits into 256 bits.
Recommendation Consider always using 'mulDiv' to simplify the code.
Client Comment Won't change. It adds non-negligible gas costs.

Listing 108:

```
34 ? (priceUp ? (num << 72) / denom : UnsafeMath.ceilDiv(num << 72,
    ↪  denom))

87     ? uint128(UnsafeMath.ceilDiv(num << 72, denom)) // denom > 0

100 ? UnsafeMath.ceilDiv(num << 72, denom).toUint128()

124 ? UnsafeMath.ceilDiv(absAmt1 * Q72, liquidity)

131 ? (uint(amt1) * Q72) / liquidity
```

3.109 CVF-109

- **Severity** Minor
- **Status** Info
- **Category** Overflow/Underflow
- **Source** PoolMath.sol

Description Overflow is possible here.
Recommendation Consider using safe inversion.
Client Comment Disagree. "amt0" and "amt1" are never type(int256).min here.

Listing 109:

```
37 if (priceUp) amt0 *= -1;

58 if (priceDown) amt1 *= -1;
```

3.110 CVF-110

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** PoolMath.sol

Recommendation Consider implementing "ceilRShift" function that shifts a number right "rounding" up.
Client Comment Won't change. Gas savings are minimal.

Listing 110:

```
57 amt1 = (priceDown ? num >> 72 : UnsafeMath.ceilDiv(num, Q72)).
    ↪ toInt256();
```

3.111 CVF-111

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** PoolMath.sol

Description Precision degradation is possible when L « $\sqrt{}$P0.
Recommendation Consider calculating as: 1 / (1 / $\sqrt{}$P0 + Δx / L) Note, that the denominator could be calculated as a 0.256 binary fixed-point number. 2^256 / x could be calculated as 1 + (~x + 1) / x.
Client Comment We don't think there'll be a precision degradation.

Listing 111:

```
64 ///                 = L/(L/√P0 + Δx)                otherwise
```

3.112 CVF-112

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** PoolMath.sol

Description This formula is used only for non-negative amounts, and for a negative amount, the function just reverts in case of overflow. .
Recommendation Consider using this formula for negative amounts as well.
Client Comment If Δx is negative and Δx $\sqrt{}$P0 overflows, then (L/$\sqrt{}$P0 + Δx) must be negative. This value being negative is invalid, so we revert it.

Listing 112:

```
64 ///                  = L/(L/√P0 + Δx)                  otherwise
```

3.113 CVF-113

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** PoolMath.sol

Description The expression "product / absAmt0 == sqrtP0" is calculated in both branches.
Recommendation Consider calculating once before the conditional statement.
Client Comment Won't change. It incurs more gas costs.

Listing 113:

```
83 if ((product / absAmt0 == sqrtP0) && ((denom = liquidityX72 +
   ↪ product) >= liquidityX72)) {

95 require(product / absAmt0 == sqrtP0);
```

3.114 CVF-114

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Math.sol

Recommendation This library should be defined in a separate file named "UnsafeMath.sol".

Listing 114:

```
4 library UnsafeMath {
```

3.115 CVF-115

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Math.sol

Recommendation This could be simplified as: uint256 s = x + uint256(int256(y)); // Overflow is fine here assert ((z = uint96 (s)) == s);

Listing 115:

```
17 int256 s = int256(uint256(x)) + int256(y);
assert(s >= 0 && s <= int256(uint256(type(uint96).max)));
z = uint96(uint256(s));
```

3.116 CVF-116

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Math.sol

Description The explicit conversion of "y" to "int256" is redundant, as compiler may do this automatically.

Listing 116:

```
17 int256 s = int256(uint256(x)) + int256(y);
```

3.117 CVF-117

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Math.sol

Recommendation This could be simplified as: uint256 s = x + uint256(int256(y)); // Overflow is fine here assert ((z = uint128 (s)) == s);

Listing 117:

```
26 int256 s = int256(uint256(x)) + y; // won't overflow
assert(s >= 0 && s <= int256(uint256(type(uint128).max)));
z = uint128(uint256(s));
```

3.118 CVF-118

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** Math.sol

Description The formulas for "z" in the comment and in the code are different.
Recommendation Consider fixing the incorrect one.
Client Comment Fixed. The comment was wrong.

Listing 118:

```
36 /// @dev Compute z = max(x - y, 0) and r = x - z

40         else r = y - x;
```

3.119 CVF-119

- **Severity** Moderate
- **Status** Info
- **Category** Overflow/Underflow
- **Source** Math.sol

Description Underflow is possible here.
Client Comment Underflow should be impossible. "y" must be larger than "x" in this branch.

Listing 119:

```
40 else r = y - x;
```

3.120 CVF-120

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Math.sol

Recommendation This could be simplified as: assert ((z = uint96 (x)) == x);
Client Comment Tested. Does not save gas.

Listing 120:

```
52 assert(x <= type(uint96).max);
   z = uint96(x);
```

3.121 CVF-121

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Math.sol

Recommendation This could be simplified as: assert ((z = int256 (x)) >= 0);
Client Comment Tested. Does not save gas.

Listing 121:

```
57 assert(x <= uint256(type(int256).max));
   z = int256(x);
```

3.122 CVF-122

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Math.sol

Recommendation "This could be simplified as: assert ((z = int96 (x)) >= 0);"
Client Comment Tested. Does not save gas.

Listing 122:

```
62 assert(x <= uint96(type(int96).max));
   z = int96(x);
```

3.123 CVF-123

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** FullMath.sol

Recommendation This is equivalent to: return prod0 / denominator;
Client Comment Noted. Prefer to keep this part untouched.

Listing 123:

```
42 require(denominator > 0);
   assembly {
       result := div(prod0, denominator)
   }
   return result;
```

3.124 CVF-124

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** FullMath.sol

Recommendation These checks could be merged to a single "remoninator > prod1" check placed before the "if (prod1 == 0) conditional statement.
Client Comment Noted. Prefer to keep this part untouched.

Listing 124:

```
42      require(denominator > 0);

51  require(denominator > prod1);
```

3.125 CVF-125

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** FullMath.sol

Description According to a comment above, twos may never be zero, so this comment is misleading.
Client Comment Noted. Prefer to keep this part untouched.

Listing 125:

```
89  // If twos is zero, then it becomes one
```

3.126 CVF-126

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** FullMath.sol

Description The only operation inside this "unchecked" block, that could be checked, is "result++" which is actually checked manually for overflow.
Recommendation Consider removing this "unchecked" block as well as the manual overflow check below.

Listing 126:

```
132 unchecked {
```

3.127 CVF-127

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** FullMath.sol

Description The expression "mulmod(a, b, denominator)" was already calculated inside the "mulDiv" call above, and is calculated here again.
Recommendation Consider refactoring the code to avoid double evaluation of the same value.
Client Comment Noted. Prefer to keep this part untouched.

Listing 127:

```
133 result = mulDiv(a, b, denominator);
    if (mulmod(a, b, denominator) > 0) {
```

3.128 CVF-128

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** EMAMath.sol

Description It is not obvious why this approximation does work.
Recommendation Consider explaining in a comment.

Listing 128:

```
58 d20 = (r * r) >> 192; // approximation
```

3.129 CVF-129

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IMuffinHubView.sol

Description The number format of the "protocolFee" return values is unclear.
Recommendation Consider documenting.

Listing 129:

```
 9 function getDefaultParameters() external view returns (uint8
     ↪ tickSpacing, uint8 protocolFee);

11 function getPoolParameters(bytes32 poolId) external view returns
     ↪ (uint8 tickSpacing, uint8 protocolFee);
```

3.130 CVF-130

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IMuffinHubView.sol

Recommendation Consider calculating and returning these values with subtick precision.
Client Comment Noted. We think the current precision is fine.

Listing 130:

```
40 int24 tickEma20,
   int24 tickEma40,
```

3.131 CVF-131

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Settlement.sol

Description It is unclear, what the "false" value means.
Recommendation Consider explaining.

Listing 131:

```
42 * @param isAdd          True if the change is additive
```

3.132 CVF-132

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Settlement.sol

Description The "Settlement." prefix is redundant inside the "Settlement" library.

Listing 132:

```
48 mapping(int24 => Settlement.Info[2]) storage settlements,
```

3.133 CVF-133

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Settlement.sol

Recommendation The type of these arguments should be a enum.
Client Comment No changes because enum is too costly in terms of gas and bytecode size.

Listing 133:

```
52 uint8 limitOrderType,

95 uint8 limitOrderType,
```

3.134 CVF-134

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Settlement.sol

Recommendation This argument wouldn't be necessary if the "liquidityDeltaD8" argument would be signed.
Client Comment No changes because there're places in the code where "liquidityDeltaD8" is larger than type(int96).max.

Listing 134:

```
54 bool isAdd,
```

3.135 CVF-135

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** Settlement.sol

Description As "limitOrderType" is "uint8", there are much more than one possible invalid value for it.
Recommendation Consider checking for valid values instead.

Listing 135:

```
57 assert(limitOrderType != Positions.NOT_LIMIT_ORDER);
```

3.136 CVF-136

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** Settlement.sol

Description Here all "limitOrderType" values other than "NOT_LIMIT_ORDER" and "ZERO_FOR_ONE".

Listing 136:

```
59 Info storage settlement = limitOrderType == Positions.
   ↪ ZERO_FOR_ONE
60     ? settlements[tickUpper][1]
       : settlements[tickLower][0];
```

3.137 CVF-137

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Settlement.sol

Description Both returned values, that are read from the storage here, were potentially read already.
Recommendation Consider reusing the already read values.

Listing 137:

```
86 return (settlement.nextSnapshotId, settlement.tickSpacing);
```

3.138 CVF-138

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Settlement.sol

Description "liquidityDeltaD8" is compared with zero twice.
Recommendation Consider comparing once and reusing the result.

Listing 138:

```
106 uint96(liquidityDeltaD8 < 0 ? -liquidityDeltaD8 :
    ↪ liquidityDeltaD8),
liquidityDeltaD8 > 0,
```

3.139 CVF-139

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** Settlement.sol

Description The function doesn't ensure that settlements are initialized.
Recommendation Consider adding explicit checks.

Listing 139:

```
138 settlement = settlements[tickEnd][0];

147 settlement = settlements[tickEnd][1];
```

3.140 CVF-140

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Settlement.sol

Description Double conversion is redundant here. Just convert to "int16".

Listing 140:

```
139 tickStart = tickEnd + int24(uint24(settlement.tickSpacing));
```

3.141 CVF-141

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Settlement.sol

Description The "settlement.nextSnapshotId" value is read from the storage twice.
Recommendation Consider reading once and reusing.
Client Comment Not changing.

Listing 141:

```
158     settlement.snapshots[settlement.nextSnapshotId] = Snapshot(

166 settlement.nextSnapshotId++;
```

3.142 CVF-142

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Settlement.sol

Description These blocks are very similar.
Recommendation Consider extracting to a function to avoid code duplication.

Listing 142:

```
172 int24 below = start.nextBelow;
    int24 above = start.nextAbove;
    ticks[below].nextAbove = above;
    ticks[above].nextBelow = below;
    delete ticks[tickStart];
    tickMap.unset(tickStart);

182 int24 below = end.nextBelow;
    int24 above = end.nextAbove;
    ticks[below].nextAbove = above;
    ticks[above].nextBelow = below;
    delete ticks[tickEnd];
    tickMap.unset(tickEnd);

    // since the tier just crossed tickEnd, we can safely set tier's
       ↪ next ticks in this way
190 tier.nextTickBelow = below;
    tier.nextTickAbove = above;
```

3.143 CVF-143

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** Settlement.sol

Description If either "tickStart" or "tickEnd" is the min or the max tick, this will delete the min/max tick configuration effectively breaking the ticks list.
Recommendation Consider adding explicit checks to avoid such situation. Alternatively, add special ticks above max and below min that could never be used and thus could never be deleted.

Listing 143:

```
176 delete ticks[tickStart];

186 delete ticks[tickEnd];
```

3.144 CVF-144

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Settlement.sol

Description As "position.limitOrderType" is "uint8" there are many values other than "ZERO_FOR_ONE" and "ONE_FOR_ZERO".
Recommendation Consider checking for valid values instead.

Listing 144:

```
210 if (position.limitOrderType != Positions.NOT_LIMIT_ORDER) {
```

3.145 CVF-145

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IMuffinHubPositions.sol

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

Listing 145:

```
14 {}
```

3.146 CVF-146

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IMuffinHubActions.sol

Recommendation The type of the token arguments should be "IERC20".
Client Comment Noted. Prefer fewer dependencies on other interfaces from our interfaces.

Listing 146:

```
14 address token,

27 address token,

38 address token0,
   address token1,

51 address token0,
   address token1,

69 address tokenIn,
70 address tokenOut,
```

3.147 CVF-147

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** IMuffinHubActions.sol

Recommendation This function should return the ID of the created pool.

Listing 147:

```
37 function createPool(
```

3.148 CVF-148

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IMuffinHubPositionsView.sol

Description The semantics of the returns array is unclear.
Recommendation Consider documenting.

Listing 148:

```
47 function getLimitOrderTickSpacingMultipliers(bytes32 poolId)
     ↪ external view returns (uint8[6] memory);
```

3.149 CVF-149

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IMuffinHubCallbacks.sol

Recommendation The type of the token arguments should be "IERC20".
Client Comment Noted. We prefer fewer dependencies on other interfaces from our interfaces.

Listing 149:

```
 6 address token,

12 address token0,
   address token1,

20 address tokenIn,
   address tokenOut,
```

3.150 CVF-150

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Positions.sol

Recommendation The type of this field should be a enum.

Listing 150:

```
12 uint8 limitOrderType;
```

3.151 CVF-151

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Positions.sol

Description This assignment is the same for all three branches.
Recommendation Consider doing it once after the conditional statement.

Listing 151:

```
70 self.liquidityD8 = liquidityD8New;

77 self.liquidityD8 = liquidityD8New;

84 self.liquidityD8 = liquidityD8New;
```

3.152 CVF-152

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Positions.sol

Description The brackets around multiplications are redundant, as they don't affect operator precedence in this case.
Client Comment Noted. The formating is done by prettier.

Listing 152:

```
75 feeGrowthInside0 -= uint80((uint256(liquidityD8) *
     ↪ feeGrowthDelta0) / liquidityD8New);
   feeGrowthInside1 -= uint80((uint256(liquidityD8) *
     ↪ feeGrowthDelta1) / liquidityD8New);

82 feeAmtOut0 = (uint256(uint96(-liquidityDeltaD8)) *
     ↪ feeGrowthDelta0) >> 56;
   feeAmtOut1 = (uint256(uint96(-liquidityDeltaD8)) *
     ↪ feeGrowthDelta1) >> 56;
```

3.153 CVF-153

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** Positions.sol

Recommendation These assignments could be merged like this: self.feeGrowthInside0Last = feeGrowthInside0 - uint80(uint256(liquidityD8) * feeGrowthDelta0 / liquidityD8New);

Listing 153:

```
75 feeGrowthInside0 -= uint80((uint256(liquidityD8) *
     ↪ feeGrowthDelta0) / liquidityD8New);

78 self.feeGrowthInside0Last = feeGrowthInside0;
```

3.154 CVF-154

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** Positions.sol

Recommendation These assignments could be merged like this: self.feeGrowthInside1Last = feeGrowthInside1 - uint80(uint256(liquidityD8) * feeGrowthDelta1 / liquidityD8New);

Listing 154:

```
76 feeGrowthInside1 -= uint80((uint256(liquidityD8) *
     ↪ feeGrowthDelta1) / liquidityD8New);

79 self.feeGrowthInside1Last = feeGrowthInside1;
```

3.155 CVF-155

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IMuffinHub.sol

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

Listing 155:

```
9 interface IMuffinHub is IMuffinHubBase, IMuffinHubEvents,
    ↪ IMuffinHubActions, IMuffinHubView {}
```

3.156 CVF-156

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** TickMaps.sol

Recommendation Consider naming in camelCase, i.e. "blockMap".

Listing 156:

```
8 uint256 blockmap; //                    stores which blocks are
    ↪ initialized
```

3.157 CVF-157

- **Severity** Major
- **Status** Fixed
- **Category** Flaw
- **Source** TickMaps.sol

Description It is not checked that the division here leaves no remainder.
Recommendation Consider adding such check.
Client Comment Fixed by removing the whole "MIN_TICK_SPACING" constraint in the code. It had been hardcoded as "1" originally.

Listing 157:

```
25 compressed = uint256(int256((tick - Constants.MIN_TICK) /
    ↪ Constants.MIN_TICK_SPACING));
```

3.158 CVF-158

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** TickMaps.sol

Description This function doesn't do any overflow checks, assuming that the provided "compression" value is correct.
Recommendation If this is fine, consider mentioning this fact in the documentation comment. Otherwise, consider adding appropriate overflow checks.

Listing 158:

```
33 function _decompress(uint256 compressed) internal pure returns (
    ↪ int24 tick) {
```

3.159 CVF-159

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** TickMaps.sol

Description Here the value "self.words[wordIdx]" that was just written into the storage is read from the storage again.
Recommendation Consider caching the written value in a local variable and reusing.
Client Comment Tested. The compiler has done this optimization for the original code. The new code even costs more gas.

Listing 159:

```
50 self.words[wordIdx] &= ~(1 << (compressed & 0xFF));
   if (self.words[wordIdx] == 0) {
```

3.160 CVF-160

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** TickMaps.sol

Description Here the value "self.blocks[blockIdx]" that was just written in the storage is read from the storage again.
Recommendation Consider caching the written value in a local variable and reusing.
Client Comment Same as CVF-159.

Listing 160:

```
52 self.blocks[blockIdx] &= ~(1 << (wordIdx & 0xFF));
   if (self.blocks[blockIdx] == 0) {
```

3.161 CVF-161

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** TickMaps.sol

Description This function panics in case there are no set bits below the given tick. It is an impossible situation? If this situation is possible, consider reverting in such a case to save gas.
Client Comment Yes, this is expected to be impossible.

Listing 161:

```
79 assert(blockmap != 0);
```

3.162 CVF-162

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** Ticks.sol

Recommendation Consider elaborating a bit more regarding what exactly "outside" means here.

Listing 162:

```
23 uint80 feeGrowthOutside0;
   uint80 feeGrowthOutside1;
   uint96 secondsPerLiquidityOutside;
```

3.163 CVF-163

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Tiers.sol

Description The number format of this value is unclear.
Recommendation Consider documenting.

Listing 163:

```
8 uint24 sqrtGamma;
```

3.164 CVF-164

- **Severity** Moderate
- **Status** Info
- **Category** Documentation
- **Source** Tiers.sol

Description This function is asymmetric. In case tickNew == self.tick, it updates the "self.nextTickBelow" value but doesn't update "self.nextTickAbove" value.
Recommendation If this behavior is desired, consider explaining it in a comment. Otherwise consider implementing a symmetric behavior.
Client Comment Such asymmetric behaviour is desired.

Listing 164:

```
17 function updateNextTick(Tier storage self, int24 tickNew)
     ↪ internal {
```

3.165 CVF-165

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IMuffinHubPositionsActions.sol

Recommendation The type of these fields should be "IERC20".
Client Comment Noted. Prefer fewer dependencies on other interfaces from our interfaces.

Listing 165:

```
17 address token0;
   address token1;

46 address token0;
   address token1;
```

3.166 CVF-166

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** IMuffinHubPositionsActions.sol

Recommendation There should be named constants for the valid order type values.

Listing 166:

```
96 /// @param limitOrderType   Direction of limit order (0: N/A; 1:
   ↪  zero for one; 2: one for zero)
```

3.167 CVF-167

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IMuffinHubPositionsActions.sol

Recommendation The type of this argument should be a enum.

Listing 167:

```
104 uint8 limitOrderType
```

3.168 CVF-168

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** IMuffinHubPositionsActions.sol

Recommendation This function should return the fee amounts collected.

Listing 168:

```
137 function collectProtocolFee(address token, address recipient)
    ↪ external;
```

3.169 CVF-169

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IMuffinHubEvents.sol

Recommendation The type of the "token", "token0", and "token1" arguments should be "IERC20".
Client Comment Noted. Prefer fewer dependencies on other interfaces from our interfaces.

Listing 169:

```
  6 event Deposit(address indexed recipient, uint256 indexed
    ↪ recipientAccRefId, address indexed token, uint256 amount);

  9 event Withdraw(address indexed recipient, uint256 indexed
    ↪ senderAccRefId, address indexed token, uint256 amount);

 12 event PoolCreated(address indexed token0, address indexed token1
    ↪ );

 26 event CollectProtocol(address indexed recipient, address indexed
    ↪  token, uint256 amount);
```

3.170 CVF-170

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IMuffinHubEvents.sol

Recommendation Events are usually named via nouns, such as "Withdrawal", "NewPool", "TierUpdate" etc.

Listing 170:

```
 9 event Withdraw(address indexed recipient, uint256 indexed
     ↪ senderAccRefId, address indexed token, uint256 amount);

12 event PoolCreated(address indexed token0, address indexed token1
     ↪ );

15 event UpdateTier(

23 event UpdatePool(bytes32 indexed poolId, uint8 tickSpacing,
     ↪ uint8 protocolFee);

26 event CollectProtocol(address indexed recipient, address indexed
     ↪  token, uint256 amount);

29 event GovernanceUpdated(address governance);

64 event CollectSettled(

79 event SetLimitOrderType(
```

3.171 CVF-171

- **Severity** Major
- **Status** Fixed
- **Category** Unclear behavior
- **Source** IMuffinHubEvents.sol

Recommendation This event should contain the "poolId" parameter.

Listing 171:

```
12 event PoolCreated(address indexed token0, address indexed token1
     ↪ );
```

3.172 CVF-172

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** IMuffinHubEvents.sol

Recommendation The event parameter should be indexed.

Listing 172:

```
29 event GovernanceUpdated(address governance);
```

3.173 CVF-173

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IMuffinHubEvents.sol

Description The terms "sender" and "recipient" in case of a swap are confusing.
Recommendation Consider using something like "initiator" and "beneficiary" instead.

Listing 173:

```
94 address indexed sender,
   address indexed recipient,
```

3.174 CVF-174

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IMuffinHubBase.sol

Recommendation The type of the "token" argument should be "IERC20".
Client Comment Noted. Prefer fewer dependencies on other interfaces from our interfaces.

Listing 174:

```
12 function accounts(address token, bytes32 accHash) external view
     ↪ returns (uint256 balance);

18 function tokens(address token) external view returns (uint8
     ↪ locked, uint248 protocolFeeAmt);
```

3.175 CVF-175

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IMuffinHubBase.sol

Recommendation The type of the "locked" return value should be "bool".

Listing 175:

```
18 function tokens(address token) external view returns (uint8
     ↪ locked, uint248 protocolFeeAmt);
```

3.176 CVF-176

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IMuffinHubBase.sol

Recommendation The type of the return values should be "IERC20".
Client Comment Noted. Prefer fewer dependencies on other interfaces from our interfaces.

Listing 176:

```
24 function underlyings(bytes32 poolId) external view returns (
     ↪ address token0, address token1);
```

3.177 CVF-177

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IWETH.sol

Description This interface defines only a subset of WETH API.
Recommendation Consider renaming to "IWETHLike" or something like this to emphasize this fact.

Listing 177:

```
4 interface IWETH {
```

3.178 CVF-178

- **Severity** Major
- **Status** Info
- **Category** Flaw
- **Source** IMulticall.sol

Description There is no way to tell how the ether sent along with a "multicall" invocation should be split among the individual calls.
Recommendation Consider either making this function non-payable or making it to accept another array of ether amounts for each individual call. The sum of the values inside this array should be equal to "msg.value".
Client Comment Not a flaw. The suggestiom might be good, but refactoring is too much at this point. Currently, the contract who inherits "Multicall" is expected to be a temporary ETH custody and have functions to manage the transfer of ETH inside the contract. Our peripheral contracts are implemented like this, but are out of scope in this round of audit.

Listing 178:

```
8 /// @dev The `msg.value` should not be trusted for any method
    ↪ callable from multicall.
```

3.179 CVF-179

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** IERC1271.sol

Description This version requirement is inconsistent with version requirements for other files in this code base.
Recommendation Consider using a consistent version requirement, i.e. "^0.8.0".

Listing 179:

```
2 pragma solidity >=0.5.0;
```

3.180 CVF-180

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** IERC1271.sol

Recommendation Should be "^0.5.0 || ^0.6.0 || ^0.7.0 | ^0.8.0" (or better just "^0.8.0") as compatibility with future major releases cannot be guaranteed.

Listing 180:

```
2 pragma solidity >=0.5.0;
```

3.181 CVF-181

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IERC1271.sol

Description It is unclear what the function should do in case the signature is not valid.
Recommendation Consider explaining.

Listing 181:

```
9 /// @dev MUST return the bytes4 magic value 0x1626ba7e when
    ↪ function passes.
```

3.182 CVF-182

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IERC1271.sol

Recommendation The magic value should be a named constant defined in this interface.

Listing 182:

```
14 /// @return magicValue The bytes4 magic value 0x1626ba7e
```

3.183 CVF-183

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IERC1271.sol

Description This phrase sounds like the function always returns the magic value, while it only returns the magic value in case a signature check passed.
Recommendation Consider rephrasing.

Listing 183:

```
14 /// @return magicValue The bytes4 magic value 0x1626ba7e
```

3.184 CVF-184

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IERC721Descriptor.sol

Description The name is confusing as one could thing that this interface is defined by the ERC-721 standard, while this is actually not the case.
Recommendation Consider renaming to something like INFTDescriptor.

Listing 184:

```
4 interface IERC721Descriptor {
```

3.185 CVF-185

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IERC721Descriptor.sol

Recommendation The type of the "token" argument should be "IERC721".
Client Comment Noted. We prefer fewer dependencies on other interfaces from our interfaces.

Listing 185:

```
5 function tokenURI(address token, uint256 tokenId) external view
    ↪ returns (string memory);
```