ABDK CONSULTING

SMART CONTRACT AUDIT

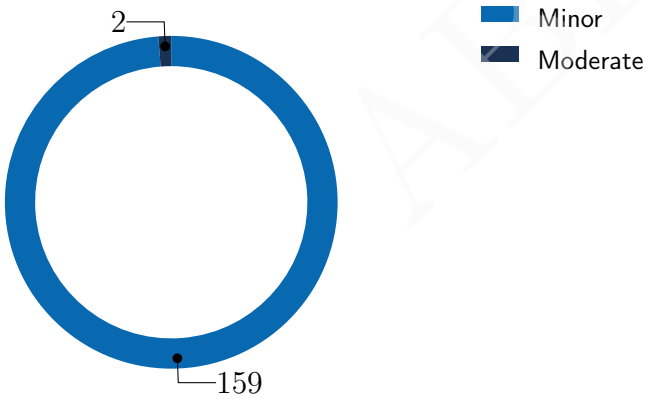
Uniswap V3

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
23rd March 2021

We've been asked to review Uniswap V3 smart contracts given in separate files in the Uniswap GitHub repo. We found no critical bugs, but have discovered a few moderate issues. Most of them were eventually downgraded to minor ones after the discussion with the authors about the protocol usecases and starting parameters.



Findings

ID	Severity	Subject	Status
CVF-1	Minor	Improper Solidity version	Info
CVF-2	Minor	Improper type	Info
CVF-3	Minor	Bad naming	Info
CVF-4	Minor	Magic number	Info
CVF-5	Minor	Improper approach	Info
CVF-6	Minor	Improper approach	Info
CVF-7	Minor	Improper approach	Info
CVF-8	Minor	Redundant check	Info
CVF-9	Minor	Improper approach	Info
CVF-10	Minor	Redundant code	Info
CVF-11	Minor	Improper approach	Info
CVF-12	Minor	Magic number	Info
CVF-13	Minor	Named constant missing	Info
CVF-14	Minor	Improper Solidity version	Info
CVF-15	Minor	Unspecific types	Info
CVF-16	Minor	Unspecific types	Info
CVF-17	Minor	Unspecific types	Info
CVF-18	Minor	Improper Solidity version	Info
CVF-19	Minor	Redundant code	Info
CVF-20	Minor	Complicated code	Info
CVF-21	Minor	Complicated code	Info
CVF-22	Minor	Unclear meaning	Info
CVF-23	Minor	Complicated code	Info
CVF-24	Minor	Improper approach	Info
CVF-25	Minor	Precision degradation possibility	Info
CVF-26	Minor	Improper approach	Info
CVF-27	Minor	Bad naming	Info

ID	Severity	Subject	Status
CVF-28	Minor	Overflow possibility	Info
CVF-29	Minor	Bad naming	Info
CVF-30	Minor	Improper approach	Info
CVF-31	Minor	Improper approach	Info
CVF-32	Minor	Improper approach	Info
CVF-33	Minor	Improper approach	Info
CVF-34	Minor	Complicated code	Info
CVF-35	Minor	Confusing description	Fixed
CVF-36	Moderate	Subefficient check	Info
CVF-37	Minor	Improper approach	Info
CVF-38	Minor	Comment missing	Fixed
CVF-39	Minor	Bad naming	Info
CVF-40	Minor	Improper approach	Info
CVF-41	Minor	Improper approach	Info
CVF-42	Minor	Complicated code	Info
CVF-43	Minor	Incorrect description	Info
CVF-44	Minor	Check missing	Info
CVF-45	Minor	Improper approach	Info
CVF-46	Minor	Improper approach	Info
CVF-47	Minor	Improper approach	Info
CVF-48	Minor	Improper approach	Info
CVF-49	Minor	Complicated code	Info
CVF-50	Minor	Improper approach	Info
CVF-51	Minor	Check missing	Info
CVF-52	Minor	Complicated code	Info
CVF-53	Minor	Improper approach	Info
CVF-54	Minor	Improper approach	Info
CVF-55	Minor	Complicated code	Info
CVF-56	Minor	Improper approach	Info
CVF-57	Minor	Inconsistent comment	Fixed

ID	Severity	Subject	Status
CVF-58	Minor	Redundant rounding	Info
CVF-59	Minor	Improper approach	Info
CVF-60	Minor	Bad naming	Info
CVF-61	Minor	Redundant code	Info
CVF-62	Minor	Improper approach	Info
CVF-63	Minor	Improper approach	Info
CVF-64	Minor	Improper approach	Info
CVF-65	Minor	Incorrect compiler version	Info
CVF-66	Minor	Improper approach	Info
CVF-67	Minor	Bad naming	Info
CVF-68	Minor	Overflow	Info
CVF-69	Minor	Check missing	Info
CVF-70	Minor	Improper approach	Info
CVF-71	Minor	Unclear function purpose	Info
CVF-72	Minor	Redundant function call	Info
CVF-73	Minor	Additional comment	Fixed
CVF-74	Minor	Check missing	Info
CVF-75	Minor	Improper approach	Info
CVF-76	Minor	Comment missing	Fixed
CVF-77	Minor	Check missing	Info
CVF-78	Minor	Improper approach	Info
CVF-79	Minor	Complicated code	Info
CVF-80	Minor	Improper approach	Info
CVF-81	Minor	Complicated code	Info
CVF-82	Minor	Improper Solidity version	Info
CVF-83	Minor	Redundant library	Info
CVF-84	Minor	Complicated code	Info
CVF-85	Minor	Complicated code	Info
CVF-86	Minor	Complicated code	Info
CVF-87	Minor	Improper Solidity version	Info

ID	Severity	Subject	Status
CVF-88	Minor	Complicated code	Info
CVF-89	Minor	Improper Solidity version	Info
CVF-90	Minor	Redundant library	Info
CVF-91	Minor	Complicated code	Info
CVF-92	Minor	Additional comment	Fixed
CVF-93	Minor	Bad naming	Info
CVF-94	Minor	Bad naming	Info
CVF-95	Minor	Redundant parameter	Info
CVF-96	Minor	Bad naming	Info
CVF-97	Minor	Unspecific types	Info
CVF-98	Minor	Redundant indexing	Info
CVF-99	Minor	Missed indexing	Info
CVF-100	Minor	Bad naming	Info
CVF-101	Minor	Redundant indexing	Info
CVF-102	Minor	Confusing comment	Fixed
CVF-103	Minor	Unspecific types	Info
CVF-104	Minor	Bad naming	Info
CVF-105	Minor	Improper Solidity version	Info
CVF-106	Minor	Improper approach	Info
CVF-107	Minor	Check missing	Info
CVF-108	Minor	Redundant code	Info
CVF-109	Minor	Improper approach	Info
CVF-110	Moderate	Overflow	Fixed
CVF-111	Minor	Additional comment	Fixed
CVF-112	Minor	Improper approach	Info
CVF-113	Minor	Redundant parameter	Info
CVF-114	Minor	Confusing comment	Fixed
CVF-115	Minor	Redundant parameters	Info
CVF-116	Minor	Redundant parameter	Info
CVF-117	Minor	Improper comment	Info

ID	Severity	Subject	Status
CVF-118	Minor	Bad naming	Info
CVF-119	Minor	Unclear description and improper function placement	Info
CVF-120	Minor	Bad naming	Info
CVF-121	Minor	Additional comment	Fixed
CVF-122	Minor	Bad naming	Info
CVF-123	Minor	Bad naming	Info
CVF-124	Minor	Additional comment	Fixed
CVF-125	Minor	Improper approach	Info
CVF-126	Minor	Bad naming	Info
CVF-127	Minor	Additional comment	Fixed
CVF-128	Minor	Additional comment	Fixed
CVF-129	Minor	Bad naming	Info
CVF-130	Minor	Confusing name	Info
CVF-131	Minor	Improper approach	Info
CVF-132	Minor	Bad naming	Info
CVF-133	Minor	Bad naming	Info
CVF-134	Minor	Typo	Info
CVF-135	Minor	Bad naming	Info
CVF-136	Minor	Bad naming	Info
CVF-137	Minor	Bad naming	Info
CVF-138	Minor	Improper approach	Info
CVF-139	Minor	Redundant parameter	Info
CVF-140	Minor	Redundant indexing	Info
CVF-141	Minor	Bad naming	Info
CVF-142	Minor	Improper approach	Info
CVF-143	Minor	Bad naming	Info
CVF-144	Minor	Typo	Info
CVF-145	Minor	Redundant parameter	Info
CVF-146	Minor	Confusing name	Info

ID	Severity	Subject	Status
CVF-147	Minor	Redundant parameter	Info
CVF-148	Minor	Redundant parameter	Info
CVF-149	Minor	Inconsistent formatting	Info
CVF-150	Minor	Additional comment	Info
CVF-151	Minor	Unclear comment	Info
CVF-152	Minor	Bad naming	Info
CVF-153	Minor	Redundant code	Info
CVF-154	Minor	Additional comment	Fixed
CVF-155	Minor	Bad naming	Info
CVF-156	Minor	Comment missing	Info
CVF-157	Minor	Bad naming	Info
CVF-158	Minor	Improper datatype	Info
CVF-159	Minor	Improper datatype	Info
CVF-160	Minor	Improper approach	Info
CVF-161	Minor	Improper approach	Info

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	Mar. 19, 2021	D. Khovratovich	Initial Draft
1.0	Mar. 23, 2021	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 audited the Uniswap Github repository with tag v1.0.0-beta.3. Concretely, the following files were audited:

- interfaces/callback/IUniswapV3FlashCallback.sol;
- interfaces/callback/IUniswapV3MintCallback.sol;
- interfaces/callback/IUniswapV3SwapCallback.sol;
- interfaces/pool/IUniswapV3PoolActions.sol;
- interfaces/pool/IUniswapV3PoolDerivedState.sol;
- interfaces/pool/IUniswapV3PoolEvents.sol;
- interfaces/pool/IUniswapV3PoolImmutables.sol;
- interfaces/pool/IUniswapV3PoolOwnerActions.sol;
- interfaces/pool/IUniswapV3PoolState.sol;
- interfaces/IERC20Minimal.sol;
- interfaces/IUniswapV3Factory.sol;
- interfaces/IUniswapV3Pool.sol;
- interfaces/IUniswapV3PoolDeployer.sol;
- libraries/BitMath.sol;
- libraries/FixedPoint128.sol;
- libraries/FixedPoint96.sol;
- libraries/FullMath.sol;
- libraries/LiquidityMath.sol;
- libraries/LowGasSafeMath.sol;
- libraries/Oracle.sol;
- libraries/Position.sol;
- libraries/SafeCast.sol;
- libraries/SecondsOutside.sol;

- libraries/SqrtPriceMath.sol;
- libraries/SwapMath.sol;
- libraries/Tick.sol;
- libraries/TickBitmap.sol;
- libraries/TickMath.sol;
- libraries/TransferHelper.sol;
- libraries/UnsafeMath.sol;
- NoDelegateCall.sol;
- UniswapV3Factory.sol;
- UniswapV3Pool.sol;
- UniswapV3PoolDeployer.sol.

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 About Customer

Uniswap is a decentralized trading platform, which has accumulated assets totalling more than 100 billion USD in equivalent. Currently the V2 version of Uniswap is in use. This audit covers the V3 version files.

2.3 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.

2.4 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and 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 Improper Solidity version

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** UniswapV3Factory.sol

Recommendation Should be "ô.7.0" according to common best practice, unless there is something special about this particular version.
Client Comment We do not want others to compile with other versions of solidity (or really at all, should use build artifacts).

Listing 1: Improper Solidity version

```
2 solidity =0.7.6;
```

3.2 CVF-2 Improper type

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** UniswapV3Factory.sol

Description Keys and values could use more specific types, such as IERC20 for the keys, and 'IUniswapV3Pool' for the values.
Client Comment We avoid creating dependencies from interfaces.

Listing 2: Improper type

```
20 mapping(address => mapping(address => mapping(uint24 => address))
   ↪ )) public override getPool;
```

3.3 CVF-3 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** UniswapV3Factory.sol

Description The name is confusing. It would be fine for a getter function but not for a property.
Recommendation Consider renaming to just "pool" or to something like "poolByTokenPair"
Client Comment Noted.

Listing 3: Bad naming

```
20 mapping(address => mapping(address => mapping(uint24 => address))
   ↪ )) public override getPool;
```

3.4 CVF-4 Magic number

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** UniswapV3Factory.sol

Recommendation These numbers should be made named constants.
Client Comment These constants should not be treated any differently from other fee levels.

Listing 4: Magic number

```
26 feeAmountTickSpacing[500] = 10;
   emit FeeAmountEnabled(500, 10);
   feeAmountTickSpacing[3000] = 60;
   emit FeeAmountEnabled(3000, 60);
30 feeAmountTickSpacing[10000] = 200;
   emit FeeAmountEnabled(10000, 200);
```

3.5 CVF-5 Improper approach

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** UniswapV3Factory.sol

Description Once enabled, a fee amount cannot be disabled or modified. Hardcoding some fee amounts in a constructor makes the contract less flexible.
Recommendation Consider either passing initial fee amounts and corresponding tick spacings as constructor parameters, or just not enabling any fee amounts in the constructor at all.
Client Comment Contract is only deployed a single time, so flexibility is not required.

Listing 5: Improper approach

```
26 feeAmountTickSpacing[500] = 10;
   emit FeeAmountEnabled(500, 10);
   feeAmountTickSpacing[3000] = 60;
   emit FeeAmountEnabled(3000, 60);
30 feeAmountTickSpacing[10000] = 200;
   emit FeeAmountEnabled(10000, 200);
```

3.6 CVF-6 Improper approach

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** UniswapV3Factory.sol

Description Here some fees are enabled in a simplified ways, bypassing checks normally performed by "enableFeeAmount" function. Also, such approach doesn't guarantee consistency between enabled fees and corresponding emitted events.
Recommendation Consider moving the logic of "enableFeeAmount", except for access control checks, into an internal function and calling this internal function here, and from the public "enableFeeAmount" function.
Client Comment Consistency is manually checked.

Listing 6: Improper approach

```
26 feeAmountTickSpacing[500] = 10;
   emit FeeAmountEnabled(500, 10);
   feeAmountTickSpacing[3000] = 60;
   emit FeeAmountEnabled(3000, 60);
30 feeAmountTickSpacing[10000] = 200;
   emit FeeAmountEnabled(10000, 200);
```

3.7 CVF-7 Improper approach

- **Severity** Minor
- **Status** Info
- **Category** Flaw
- **Source** UniswapV3Factory.sol

Description This code assigns 'token0' and 'token1' even when tokenA and tokenB are already in the proper order.
Recommendation Consider rewriting like this: if (tokenA > token B) (tokenA, tokenB) = (tokenB, tokenA);
Client Comment token0 and token1 are explicitly named as such to indicate order, whereas tokenA and tokenB are in no specific order.

Listing 7: Improper approach

```
41 (address token0, address token1) = tokenA < tokenB ? (tokenA,
   ↪ tokenB) : (tokenB, tokenA);
```

3.8 CVF-8 Redundant check

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** UniswapV3Factory.sol

Description This check looks redundant. It is anyway possible to pass an address that doesn't actually refers to a token smart contract.
Client Comment Noted.

Listing 8: Redundant check

```
42 require(token0 != address(0));
```

3.9 CVF-9 Improper approach

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** UniswapV3Factory.sol

Description This uses twice as much storage while offering only a marginal gas saving.
Recommendation Consider saving once and reordering addresses in getter (this way, 'getPool' should become a getter function rather than a property).
Client Comment Runtime vs. creation cost, where runtime is prioritized. Already documented on L48.

Listing 9: Improper approach

```
49 getPool[token1][token0][fee] = pool;
```

3.10 CVF-10 Redundant code

- **Severity** Minor
- **Status** Info
- **Category** Flaw
- **Source** UniswapV3Factory.sol

Description This code is executed even if the owner is not changed.
Client Comment Noted. Function result is idempotent.

Listing 10: Redundant code

```
56 emit OwnerChanged(owner, _owner);
   owner = _owner;
```

3.11 CVF-11 Improper approach

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** UniswapV3Factory.sol

Description There seems to be no way to disable a particular fee by setting spacing to 0. Probably not an issue.
Client Comment This is correct and a deliberate product choice–Uniswap factory owner should never be able to prevent pool creation.

Listing 11:

```
61 function enableFeeAmount(uint24 fee, int24 tickSpacing) public
   ↪ override {
```

3.12 CVF-12 Magic number

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** UniswapV3Factory.sol

Recommendation This should be a named constant being equal to 1e6, which occurs in other contracts.
Client Comment Noted.

Listing 12: Magic number

```
63 require(fee < 1000000);
```

3.13 CVF-13 Named constant missing

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** UniswapV3Factory.sol

Recommendation There should be a named constant for the value 16384, probably defined in the "TickMath" library. The value of the constant could probably be derived from the values of the 'MIN_TICK' and 'MAX_TICK' constants.
Client Comment Noted–it is documented inline and should not be used by other contracts.

Listing 13: Named constant missing

```
64 // tick spacing is capped at 16384 to prevent the situation
   ↪ where tickSpacing is so large that

67 require(tickSpacing > 0 && tickSpacing < 16384);
```

3.14 CVF-14 Improper Solidity version

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** UniswapV3PoolDeployer.sol

Recommendation Should be "ô.7.0" according to a common best practice, unless there is something special about this particular version.
Client Comment We do not want others to compile with other versions of solidity (or really at all, should use build artifacts).

Listing 14: Improper Solidity version

```
2 solidity =0.7.6;
```

3.15 CVF-15 Unspecific types

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** UniswapV3PoolDeployer.sol

Recommendation The types of these fields could be made more specific, namely "IUniswapV3Factory" for factory" and "IERC20" for "token0" and "token1".
Client Comment Noted, simply avoiding dependencies on other interfaces from our interfaces.

Listing 15: Unspecific types

```
10 address factory;
   address token0;
   address token1;
```

3.16 CVF-16 Unspecific types

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** UniswapV3PoolDeployer.sol

Recommendation The types of these arguments could be made more specific, namely "IUniswapV3Factory" for factory" and "IERC20" for "token0" and "token1".
Client Comment Noted, simply avoiding dependencies on other interfaces from our interfaces.

Listing 16: Unspecific types

```
23 address factory,
   address token0,
   address token1,
```

3.17 CVF-17 Unspecific types

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** UniswapV3PoolDeployer.sol

Recommendation The type of the returned value could be made more specific, namely "IUniswapV3Pool".
Client Comment Noted, simply avoiding dependencies on other interfaces from our interfaces.

Listing 17: Unspecific types

```
28 ) internal returns (address pool) {
```

3.18 CVF-18 Improper Solidity version

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

Recommendation Should be ĵ0.7.0 according to a common best practice, unless there is something special with this particular version.
Client Comment We do not want others to compile with other versions of solidity (or really at all, should use build artifacts).

Listing 18: Improper Solidity version

```
2 solidity =0.7.6;
```

3.19 CVF-19 Redundant code

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

Recommendation It would be more elegant to pass amount as a signed number to make "add" unnecessary.
Client Comment Noted, refactoring too major at this point.

Listing 19: Redundant code

```
25 /// @param amount How much of token0 to add or remove from
   ↪ virtual reserves
/// @param add Whether to add or remove the amount of token0
```

3.20 CVF-20 Complicated code

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

Description Each of these two functions actually implements two functions selected by the "add" parameters, which is usually specified in a calling code as a compile-time constant.
Recommendation Splitting these functions would make code simpler and more efficient.
Client Comment Noted, refactoring too major at this point.

Listing 20: Complicated code

```
28 function getNextSqrtPriceFromAmount0RoundingUp(

32     bool add

68 function getNextSqrtPriceFromAmount1RoundingDown(

72     bool add
```

3.21 CVF-21 Complicated code

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

Description These two conditional operations could be combined as:if ((product = amount * sqrtPX96) / amount == sqrtPX96 && (denominator = numerator1 + product) >= numerator1).
Client Comment Noted.

Listing 21: Complicated code

```
39 uint256 product;
40 if ((product = amount * sqrtPX96) / amount == sqrtPX96) {
       uint256 denominator = numerator1 + product;
       if (denominator >= numerator1)
```

3.22 CVF-22 Unclear meaning

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

Description What "lossless" means here? The division seems not to be precise.
Recommendation Fixed via https://github.com/Uniswap/uniswap-v3-core/commit/7445e61f17a7c1233bcabcdc920f0473793f6d78
Client Comment Fixed via https://github.com/Uniswap/uniswap-v3-core/commit/7445e61f17a7c1233bcabcdc920f0473793f6d78.

Listing 22: Unclear meaning

```
62 /// The formula we compute is lossless: sqrtPX96 +- amount /
   ↪ liquidity
```

3.23 CVF-23 Complicated code

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

Recommendation With b=2**96, the fullmul part of 'mulDiv' could be calculated via shifts.
Client Comment Noted, tried here and savings were minimal https://github.com/Uniswap/uniswap-v3-core/pull/435.

Listing 23: Complicated code

```
81 : FullMath.mulDiv(amount, FixedPoint96.Q96, liquidity)

90 : FullMath.mulDivRoundingUp(amount, FixedPoint96.Q96, liquidity)
```

3.24 CVF-24 Improper approach

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

Description Why not '>=' here? If 'sqrtPX96 == quote', there will be no underflow in subtraction.
Client Comment The price of 0 is not considered valid.

Listing 24: Improper approach

```
93 require(sqrtPX96 > quotient);
```

3.25 CVF-25 Precision degradation possibility

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

Description Double divisions here may lead to precision degradation.
Recommendation Consider using single division at least in cases when 'sqrtRatioAX96 *sqrtRatioBX96' fits into 256 bits.
Client Comment We do not believe precision is degraded by two divisions, see echidna tests here https://github.com/Uniswap/uniswap-v3-core/blob/8c58ae09ecbe1dedbe7aebce2ff0a2697c42f2ce/contracts/test/SqrtPriceMathEchidnaTest.sol#L10 L178.

Listing 25: Precision degradation possibility

```
166 ? UnsafeMath.divRoundingUp(
        FullMath.mulDivRoundingUp(numerator1, numerator2,
            ↪ sqrtRatioBX96),
        sqrtRatioAX96
    )
170 : FullMath.mulDiv(numerator1, numerator2, sqrtRatioBX96) /
    ↪ sqrtRatioAX96;
```

3.26 CVF-26 Improper approach

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

Recommendation For the case when denominator is the 2**96 constant, division part inside 'mulDiv' could be replaces with shifts.
Client Comment Noted, tried here and savings were minimal https://github.com/Uniswap/uniswap-v3-core/pull/435.

Listing 26: Improper approach

```
190 ? FullMath.mulDivRoundingUp(liquidity, sqrtRatioBX96 -
    ↪ sqrtRatioAX96, FixedPoint96.Q96)
    : FullMath.mulDiv(liquidity, sqrtRatioBX96 - sqrtRatioAX96,
    ↪ FixedPoint96.Q96);
```

3.27 CVF-27 Bad naming

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

Recommendation Better name would be "liquidityDelta".
Client Comment Adding some info, but will not change naming https://github.com/Uniswap/uniswap-v3-core/pull/441.

Listing 27: Bad naming

```
197 /// @param liquidity The change in liquidity

202     int128 liquidity

213 /// @param liquidity The change in liquidity

218     int128 liquidity
```

3.28 CVF-28 Overflow possibility

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

Description In case the call to getAmount?Delta will return 2$\hat{}$255, the function "toInt256()" will revert, while in fact, -2$\hat{}$255 could be represented as uint256, so phantom overflow is possible here. Probably not an issue.
Client Comment Noted. Will just cause an earlier than expected overflow error by max 1 unit.

Listing 28: Overflow possibility

```
206 ? -getAmount0Delta(sqrtRatioAX96, sqrtRatioBX96, uint128(-
    ↪ liquidity), false).toInt256()

222 ? -getAmount1Delta(sqrtRatioAX96, sqrtRatioBX96, uint128(-
    ↪ liquidity), false).toInt256()
```

3.29 CVF-29 Bad naming

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

Description The structure name is too generic.
Recommendation Consider renaming to "TickInfo" or "TickState".
Client Comment ‘Info‘ was chosen because it is nested within the library Tick.

Listing 29: Bad naming

```
17 struct Info {
```

3.30 CVF-30 Improper approach

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

Recommendation The 'minTick' could be calculated as TickMath.MIN_TICK - TickMath.MIN_TICK % tickSpacing.
Client Comment Noted.

Listing 30: Improper approach

```
34 int24 minTick = (TickMath.MIN_TICK / tickSpacing) * tickSpacing;
```

3.31 CVF-31 Improper approach

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

Recommendation Taking into account that ticks range is symmetric, 'maxTick' could be calculated as '-minTick'.
Client Comment Noted.

Listing 31: Improper approach

```
35 int24 maxTick = (TickMath.MAX_TICK / tickSpacing) * tickSpacing;
```

3.32 CVF-32 Improper approach

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

Recommendation The 'numTicks' could be calculated as TickMath.MAX_TICK / tickSpacing - TickMath.MIN_TICK / tickSpacing + 1, or, taking into account that ticks range is symmetric, as TickMath.MAX_TICK / tickSpacing * 2 + 1.
Client Comment Noted.

Listing 32: Improper approach

```
36 uint24 numTicks = uint24((maxTick - minTick) / tickSpacing) + 1;
```

3.33 CVF-33 Improper approach

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

Recommendation Consider wrapping the mapping into a struct with descriptive name.
Client Comment This has a non-negligible gas cost to it.

Listing 33: Improper approach

```
50  mapping(int24 => Tick.Info) storage self,
97  mapping(int24 => Tick.Info) storage self,
145 mapping(int24 => Tick.Info) storage self,
```

3.34 CVF-34 Complicated code

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

Recommendation By flipping the branches of this condition statement, the final formulas below could be simplified to feeGrowthAbove... - feeGrowthBelow....
Client Comment Noted.

Listing 34: Complicated code

```
74 if (tickCurrent < tickUpper) {
       feeGrowthAbove0X128 = upper.feeGrowthOutside0X128;
       feeGrowthAbove1X128 = upper.feeGrowthOutside1X128;
   } else {
       feeGrowthAbove0X128 = feeGrowthGlobal0X128 - upper.
           ↪ feeGrowthOutside0X128;
       feeGrowthAbove1X128 = feeGrowthGlobal1X128 - upper.
           ↪ feeGrowthOutside1X128;
80 }
```

3.35 CVF-35 Confusing description

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

Recommendation Consider adding more detail about what are upper and lower ticks.
Client Comment Fixed in https://github.com/Uniswap/uniswap-v3-core/pull/441.

Listing 35: Confusing description

```
93 /// @param upper A bool representing whether or not the call
       ↪ represents the upper, or lower tick
```

3.36 CVF-36 Subefficient check

- **Severity** Moderate
- **Status** Info
- **Category** Flaw
- **Source** Tick.sol

Description This check doesn't guarantee, that the amount of liquidity allocated for a single tick will not exceed max liquidity. For example, one may inject maxLiquidity into the position from tick #1 to tick #5, and somebody else could some more liquidity into the positions from tick #2 to tick #4. Thus for tick #3, the total liquidity will exceed maxLiquidity.
Client Comment The current liquidity of the pool is allowed to exceed max liquidity; however the pool liquidity may not exceed its uint128 container, which is the purpose of the max liquidity per tick constraint.

Listing 36: Subefficient check

```
111 require(liquidityGrossAfter <= maxLiquidity, 'LO');
```

3.37 CVF-37 Improper approach

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

Description A single operation probably is not worth a separate function call.
Client Comment Noted.

Listing 37: Improper approach

```
134 function clear(mapping(int24 => Tick.Info) storage self, int24
    ↪ tick) internal {
      delete self[tick];
    }
```

3.38 CVF-38 Comment missing

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

Recommendation Consider adding more details about how to interpret the values stored in the mapping. For the ticks above the current tick, the value is the number of seconds this tick was not above the current tick. For the ticks not above the current tick the value is current time minus the value of seconds this tick was not above the current tick.
Client Comment Fixed in https://github.com/Uniswap/uniswap-v3-core/pull/441.

Listing 38: Comment missing

```
5 @notice Contains methods for working with a mapping from tick to
    ↪ 32 bit timestamp values, specifically seconds
spent outside the tick.
```

3.39 CVF-39 Bad naming

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

Description The name is a bit misleading as the library also contains the function 'secondsInside'. Maybe just 'seconds' would be a better name.
Client Comment Noted–Seconds may be too generic.

Listing 39: Bad naming

```
8 SecondsOutside {
```

3.40 CVF-40 Improper approach

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

Description Passing 'tickSpasing' everywhere just wastes gas.
Recommendation Consider passing tick number already divided by tick spacing.
Client Comment Noted, and applicable only to SecondsOutside, not TickBitmap.

Listing 40: Improper approach

```
14  function position(int24 tick, int24 tickSpacing) private pure
      ↪ returns (int24 wordPos, uint8 shift) {

34      int24 tickSpacing,

50      int24 tickSpacing

65      int24 tickSpacing,

83      int24 tickSpacing

103     int24 tickSpacing,
```

3.41 CVF-41 Improper approach

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

Recommendation (compressed & 0x07)' would make logic for negative ticks easier to understand.
Client Comment Definitely agree, did not consider this option. Noted.

Listing 41: Improper approach

```
20  shift = uint8(compressed % 8) * 32;
```

3.42 CVF-42 Complicated code

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

Recommendation The code could be simplified by flipping the branches in conditional statement and removing "time -" return statement.
Client Comment Noted.

Listing 42: Complicated code

```
108 if (tickCurrent >= tickLower) {
        secondsBelow = get(self, tickLower, tickSpacing);
110 } else {
        secondsBelow = time - get(self, tickLower, tickSpacing);
    }

122 return time - secondsBelow - secondsAbove;
```

3.43 CVF-43 Incorrect description

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

Recommendation These descriptions are identical. Probably, the second description is incorrect.
Client Comment Fixed via https://github.com/Uniswap/uniswap-v3-core/commit/7445e61f17a7c1233bcabcdc920f0473793f6d78.

Listing 43: Incorrect description

```
18 /// @return amountIn The amount to be swapped in, of either
     ↪ token0 or token1, based on the direction of the swap
   /// @return amountOut The amount to be swapped in, of either
     ↪ token0 or token1, based on the direction of the swap
```

3.44 CVF-44 Check missing

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

Description Underflow is possible here in case feePips > 1e6.
Recommendation Consider adding explicit range check for 'feePips'.
Client Comment The 'feePips' is guaranteed to be l.t. '1e6' for any pool created by the factory due to checks in UniswapV3Factory##enableFeeAmount and immutability in UniswapV3Pool. Further checks are avoided to save gas.

Listing 44: Check missing

```
41 uint256 amountRemainingLessFee = FullMath.mulDiv(uint256(
     ↪ amountRemaining), 1e6 - feePips, 1e6);

95 feeAmount = FullMath.mulDivRoundingUp(amountIn, feePips, 1e6 -
     ↪ feePips);
```

3.45 CVF-45 Improper approach

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

Description The value of "amountIn" here is calculated with rounding up, so it could be greater than the true value. Thus the condition could be evaluated to false, even when amountRemainingLessFee is actually > the true value of amountIn, thus the expression on the "else" branch could in theory evaluate to a value that crosses the target ratio.
Client Comment We don't agree with the description. Upon further discussion with Mikhail, it seems like the interpretation of rounding up/down was that it could be off by more than 1 unit, when our echidna tests and verification indicate that it can only be different by a maximum of 1.

Listing 45: Improper approach

```
45 if (amountRemainingLessFee >= amountIn) sqrtRatioNextX96 =
     ↪ sqrtRatioTargetX96;
   else
       sqrtRatioNextX96 = SqrtPriceMath.getNextSqrtPriceFromInput(
           sqrtRatioCurrentX96,
           liquidity,
50         amountRemainingLessFee,
           zeroForOne
       );
```

3.46 CVF-46 Improper approach

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

Description When the target ratio > the current ratio, the 'zeroForOne' flag will be false, so the function 'getNextSqrtPriceFromOutput' will round up. Thus, the returned value could be greater than the true value, and thus, in theory, could be greater than the target ratio, which we are not allowed to cross. In case the target ratio <= the current ratio, 'getNestSqrtPrice-FromOutput' will round down, and thus again may cross the target ratio.
Client Comment We don't agree with the description. Upon further discussion with Mikhail, it seems like the interpretation of rounding up/down was that it could be off by more than 1 unit, when our echidna tests and verification indicate that it can only be different by a maximum of 1.

Listing 46: Improper approach

```
59 sqrtRatioNextX96 = SqrtPriceMath.getNextSqrtPriceFromOutput(
60     sqrtRatioCurrentX96,
       liquidity,
       uint256(-amountRemaining),
       zeroForOne
);
```

3.47 CVF-47 Improper approach

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

Description For the case when 'exectIn' flag is true, the "amountIn"should be capped via require statement, or it should be clearly explained why such capping is not necessary.
Client Comment This is checked with echidna tests that have been also passed through Mythx. We only check in the exact out case because the computations up to this line only guarantee a minimum of the desired amount out. This check is required in the exact output case so we do not send more than is desired by the user (the extra amount out is burned by the pool).

Listing 47: Improper approach

```
86 // cap the output amount to not exceed the remaining output
   ↪ amount
if (!exactIn && amountOut > uint256(-amountRemaining)) {
    amountOut = uint256(-amountRemaining);
}
```

3.48 CVF-48 Improper approach

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

Recommendation Should be 'if (exectIn && !max)'.
Client Comment Noted.

Listing 48: Improper approach

```
91 if (exactIn && sqrtRatioNextX96 != sqrtRatioTargetX96) {
```

3.49 CVF-49 Complicated code

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

Recommendation tick & 0xFF' would make the logic easier to understand.
Client Comment Noted.

Listing 49: Complicated code

```
16 bitPos = uint8(tick % 256);
```

3.50 CVF-50 Improper approach

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

Recommendation Passing ticks already divided by 'tickSpacing' would make code more efficient.
Client Comment TickBitmap should operate on underlying ticks because of nextInitializedTick.

Listing 50: Improper approach

```
26 int24 tickSpacing

45 int24 tickSpacing,
```

3.51 CVF-51 Check missing

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

Description It is not checked, that tick % tickSpacing == 0.
Recommendation Consider adding such check.
Client Comment It should not be checked, because then the caller would have to do this adjustment.

Listing 51: Check missing

```
44 int24 tick,

48 int24 compressed = tick / tickSpacing;
```

3.52 CVF-52 Complicated code

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

Recommendation This could be calculated as (1 « (uint(bitPos) + 1)) - 1.
Client Comment Noted.

Listing 52: Complicated code

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

3.53 CVF-53 Improper approach

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

Recommendation Instead of masking, you may do a shift: self[wordPos] « (255 - bitPos).
Client Comment Noted.

Listing 53: Improper approach

```
55 uint256 masked = self[wordPos] & mask;
```

3.54 CVF-54 Improper approach

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

Recommendation The multiplication by 'tickSpacing' should be done once, after the ternary operator.
Client Comment Noted.

Listing 54: Improper approach

```
61 ? (compressed - int24(bitPos - BitMath.mostSignificantBit(masked
   ↪ ))) * tickSpacing
   : (compressed - int24(bitPos)) * tickSpacing;

74 ? (compressed + 1 + int24(BitMath.leastSignificantBit(masked) -
   ↪ bitPos)) * tickSpacing
   : (compressed + 1 + int24(type(uint8).max - bitPos)) *
   ↪ tickSpacing;
```

3.55 CVF-55 Complicated code

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

Recommendation Instead of masking, you may do a shift: self[wordPos] » bitPos.
Client Comment Noted.

Listing 55: Complicated code

```
68 uint256 masked = self[wordPos] & mask;
```

3.56 CVF-56 Improper approach

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

Description Setting MAX_TICK this way could lead to an overflow in case MIN_TICK = -2̂23.
Recommendation Consider defining a position MAX_TICK and then derive MIN_TICK from it.
Client Comment Not possible, the constant MIN_TICK is not equal to type(int24).min.

Listing 56: Improper approach

```
11 int24 internal constant MAX_TICK = -MIN_TICK;
```

3.57 CVF-57 Inconsistent comment

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

Description The documentation comment above says that the function throws in case tick > MAX_TICK, but is also throws in case tick < -MAX_TICK. This implicitly assumes that MIN_TICK = -MAX_TICK.
Recommendation Consider making the documentation consistent with the code, and changing the code to require tick >= MIN_TICK, rather then tick >= =MAX_TICK.
Client Comment Code is equivalent and will not be changed, Comment fixed in https://github.com/Uniswap/uniswap-v3-core/pull/441.

Listing 57: Inconsistent comment

```
24 require(absTick <= uint256(MAX_TICK), 'T');
```

3.58 CVF-58 Redundant rounding

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

Description Rounding up here doesn't make much sense here, as the ratio itself is only an approximation or the true value and is not guaranteed to always round up. What is needed for the consistency with getTickAtSqrtRatio, is the relative precision of getSqrtRatioAtTick to be not worse than 0.00005%. As the minimum tick is about 2^-64, there are still about 32 bits of precision, so the worse relative precision is about 0.000000023%, thus good enough.
Client Comment The implementation of TickMath.getRatioAtTick originally returned Q128 numbers, which was adjusted to return Q96 numbers, and it is possible in this adjustment that the truncated precision causes the ratio to exist within a lower tick.

Listing 58: Redundant rounding

```
51 // we round up in the division so getTickAtSqrtRatio of the
   ↪ output price is always consistent
   sqrtPriceX96 = uint160((ratio >> 32) + (ratio % (1 << 32) == 0 ?
   ↪  0 : 1));
```

3.59 CVF-59 Improper approach

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

Recommendation This could be done in a single line like this:
require ((z = x + uint128(y)) == int256 (x) + int256 (y));
Client Comment Noted, this looks very complicated.

Listing 59: Improper approach

```
9  if (y < 0) {
10     require((z = x - uint128(-y)) < x, 'LS');
   } else {
       require((z = x + uint128(y)) >= x, 'LA');
   }
```

3.60 CVF-60 Bad naming

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

Description The name is too generic.
Recommendation Consider renaming to "PositionInfo".
Client Comment Noted, name was chosen because it is already in the context of library Position and is referenced as Position.Info.

Listing 60: Bad naming

```
13 struct Info {
```

3.61 CVF-61 Redundant code

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

Recommendation Probably there is no need to hash here as the input is 208 bits only.
Client Comment Noted.

Listing 61: Redundant code

```
36 position = self[keccak256(abi.encodePacked(owner, tickLower,
     ↪ tickUpper))];
```

3.62 CVF-62 Improper approach

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

Recommendation For denominator 2^128, muldiv could be done cheaper as a full mul and then shift.
Client Comment Noted.

Listing 62: Improper approach

```
66 FixedPoint128.Q128

74 FixedPoint128.Q128
```

3.63 CVF-63 Improper approach

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

Recommendation It would probably be cheaper to assign all fields at once: self = Info ({...});.
Client Comment I think the compiler translates this to individual field assignment regardless.

Listing 63: Improper approach

```
79 if (liquidityDelta != 0) self.liquidity = liquidityNext;
80 self.feeGrowthInside0LastX128 = feeGrowthInside0X128;
   self.feeGrowthInside1LastX128 = feeGrowthInside1X128;

84     self.feesOwed0 += feesOwed0;
       self.feesOwed1 += feesOwed1;
```

3.64 CVF-64 Improper approach

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

Recommendation These assignments should be made only if liquidityNext > 0.
Client Comment This cannot work that way, we expect the fee growth inside is always updated after a mint or burn in periphery as we use it to calculate fees owed to portions of a shared position.

Listing 64: Improper approach

```
80 self.feeGrowthInside0LastX128 = feeGrowthInside0X128;
   self.feeGrowthInside1LastX128 = feeGrowthInside1X128;
```

3.65 CVF-65 Incorrect compiler version

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

Description Most of other libraries desire compiler version 0.5.0+, but this one wants 0.6.0+.
Recommendation Consider using consistent compiler version requirements across the code.
Client Comment Most likely it requires a feature from 0.6.0.

Listing 65: Incorrect complier version

```
2 solidity >=0.6.0;
```

3.66 CVF-66 Improper approach

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

Description Branching is quite expensive. The same logic could be implemented cheaper as (x - 1) / y + 1, however, this will not work for x == 0. Also, efficient assembly implementation without branches is possible: z := add (div (x, y), gt (mod (x, y), 0))
Client Comment Noted.

Listing 66: Improper approach

```
9  function divRoundingUp(uint256 x, uint256 y) internal pure
       ↪ returns (uint256 z) {
10     // addition is safe because (type(uint256).max / 1) + (type(
           ↪ uint256).max % 1 > 0 ? 1 : 0) == type(uint256).max
       z = (x / y) + (x % y > 0 ? 1 : 0);
   }
```

3.67 CVF-67 Bad naming

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

Description The name is too generic.
Recommendation Consider giving this function some some specific name, such as "updateObservation".
Client Comment Naming within libraries is deliberately scoped.

Listing 67: Bad naming

```
30 function transform(
```

3.68 CVF-68 Overflow

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

Description Overflow is possible here.
Recommendation Consider using safe operations.
Client Comment Overflow is expected at most once per type(uint32).max seconds.

Listing 68: Overflow

```
40 tickCumulative: last.tickCumulative + int56(tick) * delta,
   liquidityCumulative: last.liquidityCumulative + uint160(
      ↪ liquidity) * delta,
```

3.69 CVF-69 Check missing

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

Description It is not checked, that 'last.initialized' is true.
Recommendation Consider adding explicit check or implement a separate logic for the case when last.initialized is false.
Client Comment Function is private and so this check must happen at callsite.

Listing 69: Check missing

```
42 initialized: true
```

3.70 CVF-70 Improper approach

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

Description It would be more flexible to wrap the array of 65536 observations into a struct, so the internal structure of this struct could be changed without changing the calling code. Also, the value 65536 should be made a named constant.
Client Comment Wrapping with structs has non-negligible costs.

Listing 70: Improper approach

```
51  function initialize(Observation[65535] storage self, uint32 time
    ↪ )

73      Observation[65535] storage self,

103     Observation[65535] storage self,

147     Observation[65535] storage self,

192     Observation[65535] storage self,

239     Observation[65535] storage self,

288     Observation[65535] storage self,
```

3.71 CVF-71 Unclear function purpose

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

Description This function always return (1, 1). Is it necessary to return any values at all?
Client Comment Simplifies the shared caller code between tests and production.

Listing 71: Unclear function purpose

```
53  returns (uint16 cardinality, uint16 cardinalityNext)
```

3.72 CVF-72 Redundant function call

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

Recommendation This function probably should not be called twice but there is no mechanism that would prevent that.
Client Comment The mechanism is the responsibility of the caller.

Listing 72: Redundant function call

```
51 function initialize(Observation[65535] storage self, uint32 time
   ↪ )
```

3.73 CVF-73 Additional comment

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

Recommendation Probably it should be added that the cardinality should be tracked externally as well.
Client Comment Fixed in https://github.com/Uniswap/uniswap-v3-core/pull/441.

Listing 73: Additional comment

```
60 /// @dev Writable at most once per block. Index represents the
   ↪ most recently written element, and must be tracked
   ↪ externally.
```

3.74 CVF-74 Check missing

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

Description There are no range checks for these parameters.
Recommendation Consider adding explicit checks that index < cardinality.
Client Comment We would like to avoid wasting gas by doing redundant checks.

Listing 74: Check missing

```
 74 uint16 index,

 78 uint16 cardinality,

150 uint16 index,
    uint16 cardinality

196 uint16 index,

198 uint16 cardinality

243 uint16 index,

245 uint16 cardinality

292 uint16 index,
```

3.75 CVF-75 Improper approach

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

Recommendation This function can be implemented in assembly as: r := iszero (xor (xor (gt (a, t), gt (b, t)), gt (a, b)))
Client Comment Noted. Looks very complicated although gas savings may be significant.

Listing 75: Improper approach

```
122 function lte(
```

3.76 CVF-76 Comment missing

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

Recommendation It should be noted here, 'beforeOrAt' and 'atOrAfter' are the same observations, or adjacent ones.
Client Comment Fixed in https://github.com/Uniswap/uniswap-v3-core/pull/441.

Listing 76: Comment missing

```
136 /// @notice Fetches the observations beforeOrAt and atOrAfter a
    ↪ target, i.e. where [beforeOrAt, atOrAfter] is satisfied
```

3.77 CVF-77 Check missing

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

Recommendation These constraints should be checked explicitly.
Client Comment If this comment means there should be a require at the end, that is the purpose of the unit tests/echidna tests.

Listing 77: Check missing

```
138 /// boundaries: older than the most recent observation and
    ↪ younger, or the same age as, the oldest observation
```

3.78 CVF-78 Improper approach

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

Description In case there is an observation whose timestamp exactly matches target, the behavior is not deterministic. The function may return this observation twice, or may return this observation and the previous one, or this and the next one.
Recommendation Consider making the behavior deterministic.
Client Comment Noted. This is a private function and as long as the result of the public function is correct, it is not important for this one to be deterministic.

Listing 78: Improper approach

```
144 /// @return beforeOrAt The observation recorded before, or at,
    ↪ the target
/// @return atOrAfter The observation recorded at, or after, the
    ↪  target
```

3.79 CVF-79 Complicated code

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

Recommendation The whole structures are unpacked into the memory here, while only certain fields from them are needed. This could probably be optimized.
Client Comment In either case it is only a single SLOAD.

Listing 79: Complicated code

```
159 beforeOrAt = self[i % cardinality];

167 atOrAfter = self[(i + 1) % cardinality];
```

3.80 CVF-80 Improper approach

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

Recommendation This check could be avoided, but checking before the loop that self[l] is initialized, and setting l to zero is it is not.
Client Comment Noted.

Listing 80: Improper approach

```
162 if (!beforeOrAt.initialized) {
```

3.81 CVF-81 Complicated code

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

Description Calculating an average rate and them applying it to the part of the interval looks suboptimal and could lean to precision degradation.
Recommendation Simpler way would be to just calculate weighted average of the cumulative values based on how close the target time is to the ends of the interval.
Client Comment It cannot lead to precision degradation because the difference is always divisible by the number of seconds elapsed and the other way is equivalent (verified by echidna test https://github.com/Uniswap/uniswap-v3-core/blob/4cf345e572691e0a74f7ede73823c1ecd5fa5bd0/contracts/test/OracleEchidnaTest.sol#L93-L109).

Listing 81: Complicated code

```
269 uint128 liquidityDerived =
270     uint128((atOrAfter.liquidityCumulative - beforeOrAt.
        ↪ liquidityCumulative) / delta);
at = transform(beforeOrAt, target, tickDerived, liquidityDerived
   ↪ );
```

3.82 CVF-82 Improper Solidity version

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** FixedPoint96.sol

Description Most of the files require Solidity 0.5.0+,while this file wants 0.4.0+.
Recommendation Consider making compiler version requirements consistent.
Client Comment Library pragmas were chosen based on features used since the libraries are shared.

Listing 82: Improper Solidity version

```
2 solidity >=0.4.0;
```

3.83 CVF-83 Redundant library

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** FixedPoint96.sol

Description This library doesn't define any functions, but just a couple of constants. Is it really necessary?
Client Comment Noted.

Listing 83: Redundant library

```
7 FixedPoint96 {
```

3.84 CVF-84 Complicated code

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** FixedPoint96.sol

Recommendation "1 « uint(RESOLUTION)" would be more readable.
Client Comment Noted.

Listing 84: Complicated code

```
9 uint256 internal constant Q96 = 0x1000000000000000000000000;
```

3.85 CVF-85 Complicated code

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

Description The return variable is not initialized explicitly, which makes code harder to read.
Client Comment Noted.

Listing 85: Complicated code

```
13 function mostSignificantBit(uint256 x) internal pure returns (
   ↪ uint8 r) {
```

3.86 CVF-86 Complicated code

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

Description The usages of type(...).max here look like hacks and doesn't work for the cases under 8 bits. Expressions like 2**128 - 1 would be more readable.
Client Comment Noted. Cannot guarantee optimizer will remove subtraction.

Listing 86: Complicated code

```
57 if (x & type(uint128).max > 0) {

62 if (x & type(uint64).max > 0) {

67 if (x & type(uint32).max > 0) {

72 if (x & type(uint16).max > 0) {

77 if (x & type(uint8).max > 0) {
```

3.87 CVF-87 Improper Solidity version

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

Description Solidity 0.8.x made SafeMath obsolete.
Recommendation Consider migrating to it.
Client Comment Migration to solidity 0.8.x was considered.

Listing 87: Improper Solidity version

```
2 solidity >=0.7.0;
```

3.88 CVF-88 Complicated code

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

Recommendation Putting the left side of "==" into brackets would make code more readable.
Client Comment Agreed but formatting is done by prettier.

Listing 88: Complicated code

```
28 require((z = x + y) >= x == (y >= 0));

34 require((z = x - y) <= x == (y >= 0));
```

3.89 CVF-89 Improper Solidity version

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** FixedPoint128.sol

Description Most of the files require Solidity 0.5.0+,while this file wants 0.4.0+.
Recommendation Consider making compiler version requirements consistent.
Client Comment Library pragmas were chosen based on features used since the libraries are shared.

Listing 89: Improper Solidity version

```
2 solidity >=0.4.0;
```

3.90 CVF-90 Redundant library

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** FixedPoint128.sol

Description This library doesn't define any functions, but just a single constant. Is it really necessary?
Client Comment Noted.

Listing 90: Redundant library

```
6 FixedPoint128 {
```

3.91 CVF-91 Complicated code

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** FixedPoint128.sol

Recommendation "1 « 128" would be more readable.
Client Comment Noted.

Listing 91: Complicated code

```
7 uint256 internal constant Q128 = 0
    ↪ x100000000000000000000000000000000;
```

3.92 CVF-92 Additional comment

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IUniswapV3PoolDeployer.sol

Recommendation It would be good to mention here that pool contracts are created via CREATE2 opcode. Otherwise, this comment is confusing.
Client Comment https://github.com/Uniswap/uniswap-v3-core/pull/441.

Listing 92: Additional comment

```
6 @dev This is used to remove all constructor arguments from the
    ↪ pool enabling pool addresses to be computed cheaply
```

3.93 CVF-93 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3PoolDeployer.sol

Description The function name is too generic.
Recommendation Consider making it more specific, like "poolParameters".
Client Comment Noted, but naming was chosen to be specific to the contract.

Listing 93: Bad naming

```
16 function parameters()
```

3.94 CVF-94 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3Factory.sol

Recommendation Events are usually named via nouns, such as "OwnerChange".
Client Comment We chose to consistently name our events as past tense.

Listing 94: Bad naming

```
10 event OwnerChanged(address indexed oldOwner, address indexed
    ↪ newOwner);
```

3.95 CVF-95 Redundant parameter

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3Factory.sol

Recommendation The "oldOwner" parameter is redundant as it could be derived from the previous event.
Client Comment Noted.

Listing 95: Redundant parameter

```
10 event OwnerChanged(address indexed oldOwner, address indexed
    ↪ newOwner);
```

3.96 CVF-96 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3Factory.sol

Recommendation Events are usually named via nouns, such as "PoolCreation" or "NewPool".
Client Comment We chose to consistently name our events as past tense.

Listing 96: Bad naming

```
18 event PoolCreated(
```

3.97 CVF-97 Unspecific types

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IUniswapV3Factory.sol

Recommendation The types of these parameters could be made more specific, such as IERC20 for "toekn0" and "token1", and "IUniswapV3Pool" for pool.
Client Comment We avoid creating dependencies from interfaces.

Listing 97: Unspecific types

```
19 address indexed token0,
20 address indexed token1,

23 address pool
```

3.98 CVF-98 Redundant indexing

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3Factory.sol

Recommendation This parameter should probably not be indexed.
Client Comment Disagree, it is useful to figure out all the pools created with a given fee.

Listing 98: Redundant indexing

```
21 uint24 indexed fee,
```

3.99 CVF-99 Missed indexing

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3Factory.sol

Recommendation This parameter should be indexed.
Client Comment Disagree, only a single event will ever have this value.

Listing 99: Missed indexing

```
23 address pool
```

3.100 CVF-100 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3Factory.sol

Recommendation Events are usually named via nouns, such as just "FeeAmount" or "NewFeeAmount".
Client Comment We chose to consistently name our events as past tense.

Listing 100: Bad naming

```
29 event FeeAmountEnabled(uint24 indexed fee, int24 indexed
   ↪ tickSpacing);
```

3.101 CVF-101 Redundant indexing

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3Factory.sol

Recommendation The "tickSpacing" parameter should probably not be indexed.
Client Comment Noted.

Listing 101: Redundant indexing

```
29 event FeeAmountEnabled(uint24 indexed fee, int24 indexed
   ↪ tickSpacing);
```

3.102 CVF-102 Confusing comment

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IUniswapV3Factory.sol

Description It is confusing that 'token0'/'token1' notation is used in parallel with "tokanA'/'tokenB' notation.
Recommendation Consider using only one of these notations.
Client Comment fixed in https://github.com/Uniswap/uniswap-v3-core/pull/438.

Listing 102: Confusing comment

```
42 /// @dev tokenA and tokenB may be passed in either token0/token1
   ↪  or token1/token0 order

54 /// @dev tokenA and tokenB may be passed in either order: token0
   ↪ /token1 or token1/token0. tickSpacing is retrieved
```

3.103 CVF-103 Unspecific types

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IUniswapV3Factory.sol

Description The types of the parameters and returned value could be made more specific. IERC20 for "tokenA" and "tokenB", and IUniswapV3Pool for "pool".
Client Comment We avoid creating dependencies from interfaces.

Listing 103: Unspecific types

```
45      address tokenA,
        address tokenB,

48  ) external view returns (address pool);

59      address tokenA,
60      address tokenB,

62  ) external returns (address pool);
```

3.104 CVF-104 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3Factory.sol

Recommendation Whether or not to prefix names of function parameters with underscore ('_') depends on personal preference, but consider using consistent naming schema across the code.
Client Comment Noted, although it seems like the wrong line is commented?

Listing 104: Bad naming

```
67  function setOwner(address _owner) external;
```

3.105 CVF-105 Improper Solidity version

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

Description Most of the files require Solidity 0.5.0+,while this file wants 0.4.0+.
Recommendation Consider making compiler version requirements consistent.
Client Comment Library pragmas were chosen based on features used since the libraries are shared.

Listing 105: Improper Solidity version

```
2 solidity >=0.4.0;
```

3.106 CVF-106 Improper approach

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

Description It looks weird to use assembly for simple division.
Recommendation Consider using plain division here.
Client Comment Noted.

Listing 106: Improper approach

```
36     result := div(prod0, denominator)

67 denominator := div(denominator, twos)

72 prod0 := div(prod0, twos)
```

3.107 CVF-107 Check missing

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

Description The code below looks like it is always executed, while it is executed only when prod1 1= 0.
Recommendation Consider putting it explicitly into “else” branch.
Client Comment Noted.

Listing 107: Check missing

```
39 }
```

3.108 CVF-108 Redundant code

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

Recommendation The "mulmod" function is available in Solidity. No need for assembly here.
Client Comment Noted.

Listing 108: Redundant code

```
52 assembly {
       remainder := mulmod(a, b, denominator)
   }
```

3.109 CVF-109 Improper approach

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

Description The 'mulmod(a, b, denominator)' is probably already calculated inside the muldiv.
Recommendation Consider reusing it somehow.
Client Comment Noted, could not figure out a way to reuse it without introducing significantly more bytecode.

Listing 109: Improper approach

```
118 return mulDiv(a, b, denominator) + (mulmod(a, b, denominator) >
      ↪ 0 ? 1 : 0);
```

3.110 CVF-110 Overflow

- **Severity** Moderate
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** FullMath.sol

Description Overflow is possible here. For example: a = 535006138814359, b = 432862656469423142931042426214547535783388063929571229938474969, denominator = 2.
Client Comment Fixed in https://github.com/Uniswap/uniswap-v3-core/pull/430.

Listing 110: Overflow

```
118 return mulDiv(a, b, denominator) + (mulmod(a, b, denominator) >
      ↪ 0 ? 1 : 0);
```

3.111 CVF-111 Additional comment

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** IUniswapV3PoolActions.sol

Description It is unclear what are "token0" and "token1" here. Are they toeken prices or token reserve amounts of what?
Recommendation Consider clarifying.
Client Comment Fixed in https://github.com/Uniswap/uniswap-v3-core/pull/441.

Listing 111: Additional comment

```
8 /// @dev Price is represented as a sqrt(token1/token0) Q64.96
    ↪ value
```

3.112 CVF-112 Improper approach

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3PoolActions.sol

Description Despite the comment before the interface, this function doesn't look like a permissionless one.
Recommendation Consider moving its declaration to some other place.
Client Comment It is indeed permissionless.

Listing 112: Improper approach

```
10 function initialize(uint160 sqrtPriceX96) external;
```

3.113 CVF-113 Redundant parameter

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3PoolActions.sol

Description This parameter looks redundant. Why not just to send minted token to the msg.sender?
Client Comment msg.sender may not wish to receive the minted liquidity.

Listing 113: Redundant parameter

```
24 address recipient,
```

3.114 CVF-114 Confusing comment

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IUniswapV3PoolActions.sol

Recommendation Word "fees" here is confusing, as fees are what users pay to the protocol. Better words would be "dividends" or "profits".
Client Comment Fixed in https://github.com/Uniswap/uniswap-v3-core/pull/441.

Listing 114: Confusing comment

```
31 /// @notice Collects fees owed to a position
```

3.115 CVF-115 Redundant parameters

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** IUniswapV3PoolActions.sol

Description These parameters look redundant. Why one would want to collect only a part of the fees?
Client Comment Tokens may be locked due to failure in token contract.

Listing 115: Redundant parameters

```
46 uint128 amount0Requested,
   uint128 amount1Requested
```

3.116 CVF-116 Redundant parameter

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3PoolActions.sol

Description This parameter looks redundant. Why not just to send tokens to the msg.sender?
Client Comment msg.sender may not wish to receive the tokens.

Listing 116: Redundant parameter

```
60 address recipient,
```

3.117 CVF-117 Improper comment

- **Severity** Minor
- **Status** Info
- **Category** Flaw
- **Source** IUniswapV3PoolActions.sol

Description So this parameter limits not the swap price, but the spot price after the swap, and actual swap price will be a bit better, but the user cannot precisely price how much better the swap price will be. For user it would be more convenient to specify the max input/min output rather then max/min price after the swap.
Client Comment The user specifies max or min inputs/outputs in periphery. This enables the user to limit how much gas to expend in the swap in addition to limited the maximum spot price the user pays, which is useful in arbitrage scenarios.

Listing 117: Improper comment

```
71 /// @param sqrtPriceLimitX96 The Q64.96 sqrt price limit. If
      ↪ zero for one, the price cannot be less than this
   /// value after the swap. If one for zero, the price cannot be
      ↪ greater than this value after the swap
```

3.118 CVF-118 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3PoolActions.sol

Recommendation The function name is confusing, as it doesn't have "loan" in it. Better name would be "flashLoan".
Client Comment Noted. Naming is deliberate.

Listing 118: Bad naming

```
92 function flash(
```

3.119 CVF-119 Unclear description and improper function placement

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IUniswapV3PoolActions.sol

Description This description is unclear and refers to implementation specific stuff like "observationCardinalityNext". Also, as function "observe" is not a part of this interface, it seems odd for this function to be here.
Recommendation Consider moving function outside of this interface.
Client Comment Noted, will not change.

Listing 119: Unclear description and improper function placement

```
99  /// @notice Increase the maximum number of price and liquidity
      ↪ observations that this pool will store
100 /// @dev This method is no-op if the pool already has an
      ↪ observationCardinalityNext greater than or equal to
    /// the input observationCardinalityNext.
    /// @param observationCardinalityNext The desired minimum number
      ↪  of observations for the pool to store
```

3.120 CVF-120 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3FlashCallback.sol

Description The interface name is confusing, as it doesn't have word "loan" in it, and "flash" may refer not only to loans.
Client Comment Noted. Naming is deliberate.

Listing 120: Bad naming

```
6 IUniswapV3FlashCallback {
```

3.121 CVF-121 Additional comment

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IUniswapV3FlashCallback.sol

Description It is unclear how exactly the sender should repay the debt.
Recommendation Consider adding more details regarding this.
Client Comment Fixed in https://github.com/Uniswap/uniswap-v3-core/pull/441.

Listing 121: Additional comment

```
7 /// @notice Called after transferring tokens to the `msg.sender
    ↪ `, allows the sender to perform any actions and then
  /// repay the flash transaction.
```

3.122 CVF-122 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3FlashCallback.sol

Description The function name is a bit cumbersome. Usually, callback functions are named like "onFlashLoan".
Client Comment Noted. Naming is deliberate.

Listing 122: Bad naming

```
13 function uniswapV3FlashCallback(
```

3.123 CVF-123 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3PoolState.sol

Description The particular slot number and the fact the all these values are actually stored in the same slot are implementation details and should not be reflected in the interface.
Recommendation Consider renaming the function and removing the reference to the slot number from the documentation comment.
Client Comment Agree, except gathering them all by slot makes it possible to query them in a single call, and they have no relation besides the fact that they are in a single storage slot.

Listing 123: Bad naming

```
8  /// @notice The 0th storage slot in the pool stores many values,
       ↪ and is exposed as a single method to save gas

20 function slot0()
```

3.124 CVF-124 Additional comment

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IUniswapV3PoolState.sol

Description This field actually contain two 4-bit values.
Recommendation Consider explaining this in the documentation comment as well as the semantics of these values.
Client Comment Fixed in https://github.com/Uniswap/uniswap-v3-core/pull/441.

Listing 124: Additional comment

```
18 /// feeProtocol The fees collected by the protocol for the pool,

29         uint8 feeProtocol,
```

3.125 CVF-125 Improper approach

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3PoolState.sol

Description The function not only loads the value stored in slot #0, but also unpacks all the values from it, while the caller may need only some of the values.
Recommendation Consider returning the stored value as is without unpacking, and provide a separate utility function for unpacking.
Client Comment User can do this in assembly.

Listing 125: Improper approach

```
24 uint160 sqrtPriceX96,
   int24 tick,
   uint16 observationIndex,
   uint16 observationCardinality,
   uint16 observationCardinalityNext,
   uint8 feeProtocol,
30 bool unlocked
```

3.126 CVF-126 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3PoolState.sol

Description The name is confusing, as it suggests that the function returns many ticks, while it returns only one.
Recommendation Consider renaming to "tick" or "tickInfo".
Client Comment Noted.

Listing 126: Bad naming

```
59 function ticks(int24 tick)
```

3.127 CVF-127 Additional comment

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IUniswapV3PoolState.sol

Description This description is confusing. It seems to be related to some deep implementation details.
Recommendation Consider adding more details here about how to use these bitmap values.
Client Comment Fixed in https://github.com/Uniswap/uniswap-v3-core/pull/441.

Listing 127: Additional comment

```
69 /// @notice Returns 256 packed tick initialized boolean values
70 /// @param wordPosition the index of the word in the bitmap to
     ↪ fetch. The initialized booleans are packed into words
   /// based on the tick and the pool's tick spacing
```

3.128 CVF-128 Additional comment

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IUniswapV3PoolState.sol

Description This description is confusing. It seems to be related to some deep implementation details.
Recommendation Consider adding more details here about how to use these seconds outside values.
Client Comment Fixed in https://github.com/Uniswap/uniswap-v3-core/pull/441.

Listing 128: Additional comment

```
74 /// @notice Returns 8 packed tick seconds outside values
   /// @param wordPosition The index of the word in the map to
     ↪ fetch. The seconds outside 32 bit values are packed into
   /// words based on the tick and the pool's tick spacing
```

3.129 CVF-129 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3PoolState.sol

Description The name if confusing, as it suggests that multiple positions will be returned, while actually, only one positions is returned.
Recommendation Consider renaming into "position" or "positionInfo".
Client Comment Noted.

Listing 129: Bad naming

```
86 function positions(bytes32 key)
```

3.130 CVF-130 Confusing name

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IUniswapV3PoolState.sol

Description The name is confusing as it suggests that the function returns multiple observations, while actually it returns only one.
Recommendation Consider renaming into "observation".
Client Comment Noted, will not fix.

Listing 130: Confusing name

```
107 function observations(uint256 index)
```

3.131 CVF-131 Improper approach

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

Recommendation This could be implemented as require ((z = int256 (y)) >= 0);
Client Comment Noted.

Listing 131: Improper approach

```
22 require(y < 2**255);
   z = int256(y);
```

3.132 CVF-132 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3PoolOwnerActions.sol

Description The name is confusing, as one could think this function sets a protocol named "fee protocol", while it actually sets a fee named "protocol fee".
Recommendation Consider renaming.
Client Comment Noted.

Listing 132: Bad naming

```
10 function setFeeProtocol(uint8 feeProtocol0, uint8 feeProtocol1)
     ↪ external;
```

3.133 CVF-133 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3PoolOwnerActions.sol

Description The name is very confusing, as this function actually collects a fee (namely the protocol fee), rather then a protocol.
Recommendation Consider renaming.
Client Comment Noted.

Listing 133: Bad naming

```
18 function collectProtocol(
```

3.134 CVF-134 Typo

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IUniswapV3PoolImmutables.sol

Recommendation The return type should be "IUniswapV3Factory".
Client Comment We disagree with the recommendation.

Listing 134: Typo

```
 7 /// @notice The contract that deployed the pool, which must
     ↪ adhere to the IUniswapV3Factory interface
/// @return The contract address
function factory() external view returns (address);
```

3.135 CVF-135 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IUniswapV3PoolImmutables.sol

Recommendation The return types should be "IERC20Minimal".
Client Comment We avoid creating dependencies from interfaces.

Listing 135: Bad naming

```
13 function token0() external view returns (address);

17 function token1() external view returns (address);
```

3.136 CVF-136 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3PoolImmutables.sol

Recommendation As there is also variable protocol fee, consider giving a more specific name to this function, such as "poolFee".
Client Comment Noted.

Listing 136: Bad naming

```
21 function fee() external view returns (uint24);
```

3.137 CVF-137 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3PoolImmutables.sol

Description The name is confusing, as one could think that it refers to the spacing between adjacent ticks.
Recommendation Consider choosing better name, "tickFactor" may be.
Client Comment Noted, but it could be described as the spacing between adjacent ticks.

Listing 137: Bad naming

```
28 function tickSpacing() external view returns (int24);
```

3.138 CVF-138 Improper approach

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3PoolEvents.sol

Description Usually, an interface contains declarations of related data types, functions, and events. Putting actions, observers, immutables, and events into different interfaces seems odd.
Client Comment You only import the interfaces you want.

Listing 138: Improper approach

```
4 @title Events emitted by a pool
  @notice Contains all events emitted by the pool
  IUniswapV3PoolEvents {
```

3.139 CVF-139 Redundant parameter

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** IUniswapV3PoolEvents.sol

Description These parameters look redundant. How could it be used?
Client Comment Noted. What do you mean redundant? They are used in indexing e.g. via the graph.

Listing 139: Redundant parameter

```
22 address sender,

40 address recipient,

58 address recipient,
```

3.140 CVF-140 Redundant indexing

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** IUniswapV3PoolEvents.sol

Description Is is really necessary to index these parameters? Ticks are more like vales rather then identifiers of enums.
Client Comment Yes, for the interface these are part of the position identifier.

Listing 140: Redundant indexing

```
24  int24 indexed tickLower,
    int24 indexed tickUpper,

41  int24 indexed tickLower,
    int24 indexed tickUpper,

59  int24 indexed tickLower,
60  int24 indexed tickUpper,
```

3.141 CVF-141 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3PoolEvents.sol

Description The name is a bit confusing. There are three "amount" parameters in the event, but this one is not qualified.
Recommendation Consider renaming to something like "mintAmount" or "liquidityAmount".
Client Comment Noted, it is implicit in the event name.

Listing 141: Bad naming

```
26  uint128 amount,
```

3.142 CVF-142 Improper approach

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3PoolEvents.sol

Description A mixture of types uint128, int256, and uint256 is used for token amounts in this interface. This is confusing and error-prone.
Recommendation Consider using int256 everywhere in public API, while underlying implementations may still use other types for efficiently.
Client Comment Noted. We agree it is confusing. Cannot be refactored at this point, though may be altered in periphery.

Listing 142: Improper approach

```
26      uint128 amount,
        uint256 amount0,
        uint256 amount1

43      uint128 amount0,
        uint128 amount1

61      uint128 amount,
        uint256 amount0,
        uint256 amount1

76      int256 amount0,
        int256 amount1,

92      uint256 amount0,
        uint256 amount1,
        uint256 paid0,
        uint256 paid1

120 event CollectProtocol(address indexed sender, address indexed
    ↪ recipient, uint128 amount0, uint128 amount1);
```

3.143 CVF-143 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3PoolEvents.sol

Description The name is a bit confusing. There are three "amount" parameters in the event, but this one is not qualified.
Recommendation Consider renaming to something like "burnAmount" or "liquidityAmount".
Client Comment Noted, it is implicit in the event name.

Listing 143: Bad naming

```
61 uint128 amount,
```

3.144 CVF-144 Typo

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IUniswapV3PoolEvents.sol

Recommendation Should be "delta", not "Delta".
Client Comment Fixed via https://github.com/Uniswap/uniswap-v3-core/commit/7445e61f17a7c1233bcabcdc920f0473793f6d78

Listing 144: Typo

```
70 /// @param amount1 The Delta of the token1 balance of the pool
```

3.145 CVF-145 Redundant parameter

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3PoolEvents.sol

Description These parameters look redundant. Swap is not the only operation that may change the current price and tick. Also, not every swap actually changes them.
Recommendation Consider emitting a separate event every time the current price is changed, regardless of what operation triggered the price change.
Client Comment Price changes only by swap and initialize.

Listing 145: Redundant parameter

```
78 uint160 sqrtPriceX96,
   int24 tick
```

3.146 CVF-146 Confusing name

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3PoolEvents.sol

Description The names are confusing. Better names would be "amountLoanedN", and "amountRepaidN".
Client Comment Noted.

Listing 146: Confusing name

```
92 uint256 amount0,
   uint256 amount1,
   uint256 paid0,
   uint256 paid1
```

3.147 CVF-147 Redundant parameter

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3PoolEvents.sol

Recommendation This parameter is redundant, the old value could be derived from the previous event.
Client Comment Noted, this is typical for all events.

Listing 147: Redundant parameter

```
104 uint16 observationCardinalityNextOld,
```

3.148 CVF-148 Redundant parameter

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3PoolEvents.sol

Recommendation These parameters are redundant and could be derived from the previous event.
Client Comment Noted, this is typical for all events.

Listing 148: Redundant parameter

```
109 /// @param feeProtocol0Old The previous value of the token0
    ↪ protocol fee
110 /// @param feeProtocol1Old The previous value of the token1
    ↪ protocol fee
```

3.149 CVF-149 Inconsistent formatting

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3PoolEvents.sol

Description Other events in this interface are formatted one parameter per line, but not these two.
Recommendation Consider using consistent formatting.
Client Comment We use prettier for formatting.

Listing 149: Inconsistent formatting

```
113 event SetFeeProtocol(uint8 feeProtocol0Old, uint8
    ↪ feeProtocol1Old, uint8 feeProtocol0New, uint8
    ↪ feeProtocol1New);

120 event CollectProtocol(address indexed sender, address indexed
    ↪ recipient, uint128 amount0, uint128 amount1);
```

3.150 CVF-150 Additional comment

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IUniswapV3PoolEvents.sol

Description The encoding of the fees is unclear.
Recommendation Consider adding more details about how to interpret these uint8 values.
Client Comment Protocol fee is clearly defined in other contracts.

Listing 150: Additional comment

```
113 event SetFeeProtocol(uint8 feeProtocol0Old, uint8
    ↪ feeProtocol1Old, uint8 feeProtocol0New, uint8
    ↪ feeProtocol1New);
```

3.151 CVF-151 Unclear comment

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** IUniswapV3PoolEvents.sol

Description Is it really necessary, to have both, sender and recipient addresses in the event as indexed parameters?
Client Comment Yes, for the interface.

Listing 151: Unclear comment

```
116 /// @param sender The address that collects the protocol fees
    /// @param recipient The address that receives the collected
       ↪ protocol fees
```

3.152 CVF-152 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3SwapCallback.sol

Description The function name is a bit cumbersome. Usually, callback functions are named like "onSwap".
Client Comment Noted, deliberately named to be relatively unique.

Listing 152: Bad naming

```
15 function uniswapV3SwapCallback(
```

3.153 CVF-153 Redundant code

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3SwapCallback.sol

Description Among these two values only one it unknown to the sender, as the other was specified as the "amountSpecified" parameter of the "swap" function.
Recommendation Consider leaving only one parameter whose meaning would depend on whether "amountSpecified" value was positive or negative.
Client Comment Noted. Too large a refactor.

Listing 153: Redundant code

```
16 int256 amount0Delta,
   int256 amount1Delta,
```

3.154 CVF-154 Additional comment

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** IUniswapV3MintCallback.sol

Description It is unclear how exactly the sender should pay the tokens.
Recommendation Consider adding more details regarding this.
Client Comment Fixed in https://github.com/Uniswap/uniswap-v3-core/pull/441.

Listing 154: Additional comment

```
7 /// @notice Called on `msg.sender` after making updates to a
    ↪ position. Allows the sender to pay the tokens
/// due for the minted liquidity.
```

3.155 CVF-155 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IUniswapV3MintCallback.sol

Recommendation The function name is a bit cumbersome. Usually, callback functions are named like "onMint".
Client Comment Deliberately cumbersome so that it does not clash with any existing function signatures.

Listing 155: Bad naming

```
13 function uniswapV3MintCallback(
```

3.156 CVF-156 Comment missing

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3Pool.sol

Recommendation It is a good practice to put a comment into empty block describing why it is empty.
Client Comment Noted.

Listing 156: Comment missing

```
22
```

3.157 CVF-157 Bad naming

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IERC20Minimal.sol

Description The names of event parameters here differ from the names in ERC-20 standard. Node, that even parameter names are part of the public API and are visible through Web3 API.
Client Comment Noted. This does not affect smart contracts.

Listing 157: Bad naming

```
45 event Transfer(address indexed from, address indexed to, uint256
   ↪  value);

51 event Approval(address indexed owner, address indexed spender,
   ↪ uint256 value);
```

3.158 CVF-158 Improper datatype

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IUniswapV3PoolDerivedState.sol

Description Why this function returns uint32? In Solidity, the standard data type for timestamps and time intervals is uint256.
Client Comment For packing, and accumulator values where seconds * value must fit within 256 bits.

Listing 158: Improper datatype

```
16 function secondsInside(int24 tickLower, int24 tickUpper)
   ↪ external view returns (uint32);
```

3.159 CVF-159 Improper datatype

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** IUniswapV3PoolDerivedState.sol

Description Why uint32 is used here? In Solidity, the standard data type for timestamps and time intervals is uint256.
Client Comment For packing, and accumulator values where seconds * value must fit within 256 bits.

Listing 159: Improper datatype

```
28 function observe(uint32[] calldata secondsAgos)
```

3.160 CVF-160 Improper approach

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3PoolDerivedState.sol

Description Why to pass an array of arguments? One can just call the function several times.
Client Comment The common case is 2 calls, which is roughly equivalent, and the case of more than 2 calls is significantly cheaper.

Listing 160: Improper approach

```
28 function observe(uint32[] calldata secondsAgos)
```

3.161 CVF-161 Improper approach

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** IUniswapV3PoolDerivedState.sol

Recommendation Packing each cumulative tick together with corresponding cumulative liquidity into a single 256-bit word would make calls to this function more efficient.
Client Comment Noted, but has a UX cost as well.

Listing 161: Improper approach

```
31 returns (int56[] memory tickCumulatives, uint160[] memory
   ↪ liquidityCumulatives);
```