ABDK CONSULTING

SMART CONTRACT AUDIT

Uniswap V3

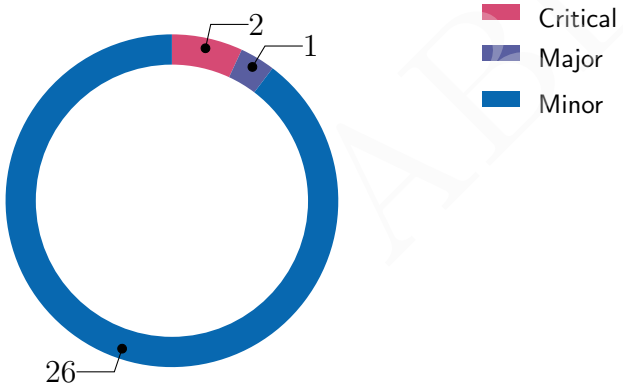
Peripheral. Part 1

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
26th April 2021

We've been asked to review Uniswap V3 smart contracts given in separate files in the Uniswap GitHub repo. We found two critical bugs, which were fixed later, and one another major but non-critical flaw.



Findings

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

ID	Severity	Category	Status
CVF-28	Minor	Suboptimal	Opened
CVF-29	Minor	Suboptimal	Opened

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	Apr. 25, 2021	D. Khovratovich	Initial Draft
0.2	Apr. 25, 2021	D. Khovratovich	Minor revision
1.0	Apr. 26, 2021	D. Khovratovich	Release
1.1	May o4, 2021	D. Khovratovich	Fixes update

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:

- NonfungiblePositionManager.sol;
- SwapRouter.sol.

Fixes were applied in these two commits.

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** Opened
- **Category** Suboptimal
- **Source** SwapRouter.sol

Recommendation Should be "ô.7.0" according to a common best practice.

Listing 1:

```
2 solidity =0.7.6;
```

3.2 CVF-2

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** SwapRouter.sol

Description We didn't review these files.

Listing 2:

```
9  './interfaces/ISwapRouter.sol';
10 './base/PeripheryImmutableState.sol';
   './base/PeripheryValidation.sol';
   './base/PeripheryPayments.sol';
   './base/Multicall.sol';
   './base/SelfPermit.sol';
   './libraries/Path.sol';
   './libraries/PoolAddress.sol';
   './libraries/CallbackValidation.sol';
   './interfaces/external/IWETH9.sol';
```

3.3 CVF-3

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** SwapRouter.sol

Recommendation The type of this argument should be "IUniswapV3Factory", the type of this argument should be IWETH9.

Listing 3:

```
40 constructor(address _factory, address _WETH9)
     ↪ PeripheryImmutableState(_factory, _WETH9) {}
```

3.4 CVF-4

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SwapRouter.sol

Description The comment is confusing, as the in/out amounts are not swapped, but rather made equal. Also, this assignment is redundant, just use "tokenOut" instead of "tokenIn" in the next line.

Listing 4:

```
79 tokenIn = tokenOut; // swap in/out because exact output swaps
   ↪ are reversed
```

3.5 CVF-5

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SwapRouter.sol

Recommendation If the "MIN_SQRT_RATIO" and "MAX_SQRT_RATIO" contracts contain the minimum and the maximum valid sqrt price, then adding/subtracting one from them seems redundant.

Listing 5:

```
102 ? (zeroForOne ? TickMath.MIN_SQRT_RATIO + 1 : TickMath.
    ↪ MAX_SQRT_RATIO - 1)

178 ? (zeroForOne ? TickMath.MIN_SQRT_RATIO + 1 : TickMath.
    ↪ MAX_SQRT_RATIO - 1)
```

3.6 CVF-6

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** SwapRouter.sol

Description This line is confusing, as it is not obvious that the path is different on every iteration.
Recommendation Probably, renaming the variable to something like "stillHasMultiplePools" would help.

Listing 6:

```
136 bool hasMultiplePools = params.path.hasMultiplePools();
```

3.7 CVF-7

- **Severity** Critical
- **Status** Opened
- **Category** Flaw
- **Source** SwapRouter.sol

Recommendation This should be address(this) for all but the first swap.

Listing 7:

```
145 payer: msg.sender
```

3.8 CVF-8

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SwapRouter.sol

Recommendation Why is it necessary to reset? In case the value is non-zero, reset is just waste of gas. Also, it is possible to add a flag to the "SwapCallbackData" structure, that would indicate that the last swap is the only one, and thus there is no need to cache the input amount.

Listing 8:

```
209 // has to be reset even though we don't use it in the single hop
    ↪ case
210 amountInCached = DEFAULT_AMOUNT_IN_CACHED;
```

3.9 CVF-9

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** SwapRouter.sol

Recommendation Why is it necessary to reset the "amountInCached" storage variable? In case this variable is non-zero, the set looks like waste of gas.

Listing 9:

```
230 amountInCached = DEFAULT_AMOUNT_IN_CACHED;
```

3.10 CVF-10

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** NonfungiblePositionManager.sol

Recommendation Should be "̂0.7.0" according to a common best practice.

Listing 10:

```
2 solidity =0.7.6;
```

3.11 CVF-11

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** NonfungiblePositionManager.sol

Description We didn't review these files.

Listing 11:

```
9  './interfaces/INonfungiblePositionManager.sol';
10 './interfaces/INonfungibleTokenPositionDescriptor.sol';
   './libraries/PositionKey.sol';
   './libraries/PoolAddress.sol';
   './base/LiquidityManagement.sol';
   './base/PeripheryImmutableState.sol';
   './base/Multicall.sol';
   './base/ERC721Permit.sol';
   './base/PeripheryValidation.sol';
   './base/SelfPermit.sol';
```

3.12 CVF-12

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** NonfungiblePositionManager.sol

Description Underscore ('_') prefix is commonly used for local variable to distinguish them from storage variables, however here this prefix is used for storage variable. This could confuse people.
Recommendation Consider following the common practice.

Listing 12:

```
53 mapping(address => uint80) private _poolIds;

56 mapping(uint80 => PoolAddress.PoolKey) private _poolIdToPoolKey;

59 mapping(uint256 => Position) private _positions;

67 address private immutable _tokenDescriptor;
```

3.13 CVF-13

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** NonfungiblePositionManager.sol

Recommendation Why the type is uint176? It should be uint256 to minimize gas costs.

Listing 13:

```
62 uint176 private _nextId = 1;
```

3.14 CVF-14

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** NonfungiblePositionManager.sol

Recommendation The type of this storage variable should be "INonfungibleTokenPosition-Descriptor".

Listing 14:

```
67 address private immutable _tokenDescriptor;
```

3.15 CVF-15

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** NonfungiblePositionManager.sol

Recommendation The type of this argument should be "IUniswapV3Factory".

Listing 15:

```
70 address _factory,
```

3.16 CVF-16

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** NonfungiblePositionManager.sol

Recommendation The type of this argument should be IWETH9.

Listing 16:

```
71 address _WETH9,
```

3.17 CVF-17

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** NonfungiblePositionManager.sol

Recommendation The type of this argument should be "INonfungibleTokenPositionDescriptor".

Listing 17:

```
72 address _tokenDescriptor_
```

3.18 CVF-18

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** NonfungiblePositionManager.sol

Description The name is confusing, as one could think that this functions deals with multiple positions, while it actually returns only one position.
Recommendation Consider renaming to "position", "getPosition", "getTokenPosition", or something like this.

Listing 18:

```
78 function positions(uint256 tokenId)
```

3.19 CVF-19

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** NonfungiblePositionManager.sol

Recommendation This utility function doesn't do anything with the non-fungible positions. It should not be in this smart contract.

Listing 19:

```
117 function createAndInitializePoolIfNecessary(
```

3.20 CVF-20

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** NonfungiblePositionManager.sol

Recommendation The type of the returned value should be "IUniswapV3Pool".

Listing 20:

```
122 ) external payable override returns (address pool) {
```

3.21 CVF-21

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NonfungiblePositionManager.sol

Recommendation This code duplication could be avoided by moving the pool initialization to the very end of the function, and returning preliminary in case the initialization is not necessary: pool = factory.getPool(...); if (pool == 0) pool = factory.createPool (...); else if (/* pool is initialized /*) return; pool.initialize (...);

Listing 21:

```
127 IUniswapV3Pool(pool).initialize(sqrtPriceX96);

131     IUniswapV3Pool(pool).initialize(sqrtPriceX96);
```

3.22 CVF-22

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** NonfungiblePositionManager.sol

Description Overflow is possible here in theory. Probably not an issue.

Listing 22:

```
140     _poolIds[pool] = (poolId = _nextPoolId++);

174 _mint(params.recipient, (tokenId = _nextId++));

400 return uint256(_positions[tokenId].nonce++);
```

3.23 CVF-23

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** NonfungiblePositionManager.sol

Description A function named "mint" emits an event named "IncreaseLiquidity". This is confusing and makes it harder to map emitted events with the code.
Recommendation Consider renaming the event.

Listing 23:

```
199 emit IncreaseLiquidity(tokenId, liquidity, amount0, amount1);
```

3.24 CVF-24

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** NonfungiblePositionManager.sol

Description There is no explicit check for whether the token with such Id does exist.
Recommendation Consider adding the check.

Listing 24:

```
234 Position storage position = _positions[tokenId];

297 Position storage position = _positions[tokenId];

343 Position storage position = _positions[tokenId];

393 Position storage position = _positions[tokenId];

400 return uint256(_positions[tokenId].nonce++);
```

3.25 CVF-25

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NonfungiblePositionManager.sol

Recommendation These calculations may be done much more efficiently as fullMull + shift, taking into account that the denominators are powers of two known at the compile time.

Listing 25:

```
260 FullMath.mulDiv(
263     FixedPoint128.Q128
267 FullMath.mulDiv(
270     FixedPoint128.Q128
312     FullMath.mulDiv(
315         FixedPoint128.Q128
321     FullMath.mulDiv(
324         FixedPoint128.Q128
358     FullMath.mulDiv(
361         FixedPoint128.Q128
365     FullMath.mulDiv(
368         FixedPoint128.Q128
```

3.26 CVF-26

- **Severity** Critical
- **Status** Opened
- **Category** Flaw
- **Source** NonfungiblePositionManager.sol

Recommendation It is not checked that the value of the "liquidity" argument doesn't exceed the current liquidity of the token. Combined with unsafe subtraction in the end of the function, this allows a token holder to steal liquidity from the owners of those other tokens whose "tickLower" and "tickUpper" are the same.

Listing 26:

```
284 uint128 liquidity,

330 position.liquidity -= liquidity;
```

3.27 CVF-27

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NonfungiblePositionManager.sol

Recommendation This check is redundant. It saves gas on very unlikely case when someone would try to collect zero amount, but makes more expensive all the normal invocations of the "collect" function.

Listing 27:

```
342 require(amount0Max > 0 || amount1Max > 0);
```

3.28 CVF-28

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NonfungiblePositionManager.sol

Recommendation The type cast is redundant, as uint96 could be implicitly converted to uint256.

Listing 28:

```
400 return uint256(_positions[tokenId].nonce++);
```

3.29 CVF-29

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** NonfungiblePositionManager.sol

Recommendation This event is logged even if ‘to‘ was already approved.

Listing 29:

```
413 emit Approval(ownerOf(tokenId), to, tokenId);
```