ABDK CONSULTING

SMART CONTRACT AUDIT

Muffin

Part 2: Periphery

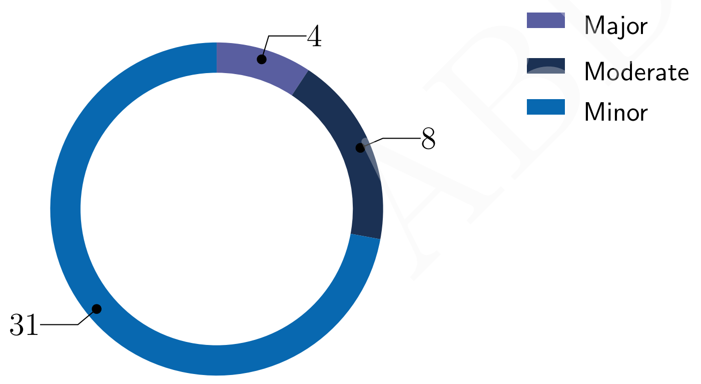
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
22nd April 2022

We've been asked to review the 8 files in a Github repository. We found 4 major, and a few less important issues. All identified major issues have been fixed or otherwise addressed in collaboration with the client.



Findings

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

ID	Severity	Category	Status
CVF-28	Minor	Suboptimal	Skipped
CVF-29	Minor	Suboptimal	Fixed
CVF-30	Moderate	Unclear behavior	Fixed
CVF-31	Minor	Bad datatype	Skipped
CVF-32	Minor	Bad datatype	Skipped
CVF-33	Minor	Bad datatype	Skipped
CVF-34	Minor	Suboptimal	Fixed
CVF-35	Minor	Bad datatype	Skipped
CVF-36	Minor	Suboptimal	Skipped
CVF-37	Major	Flaw	Replied
CVF-38	Moderate	Flaw	Replied
CVF-39	Moderate	Flaw	Replied
CVF-40	Moderate	Flaw	Replied
CVF-41	Minor	Procedural	Skipped
CVF-42	Minor	Suboptimal	Skipped
CVF-43	Minor	Flaw	Skipped

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	April 19, 2022	D. Khovratovich	Initial Draft
0.2	April 22, 2022	D. Khovratovich	Minor revision
1.0	April 22, 2022	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.
We have reviewed the contracts at the next tag:

- periphery/base/ERC721.sol
- periphery/base/ERC721Extended.sol
- periphery/base/ManagerBase.sol
- periphery/base/Multicall.sol
- periphery/base/SelfPermit.sol
- periphery/Manager.sol
- periphery/base/PositionManager.sol
- periphery/base/SwapManager.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 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
- **Category** Procedural
- **Status** Skipped
- **Source** ERC721Extended.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.
Recommendation Consider specifying as "^0.8.0". Also relevant for the naxt files: SelfPermit.sol, Manager.sol, SwapManager.sol, PositionManager.sol, Multicall.sol, ManagerBase.sol.

Listing 1:

```
pragma solidity 0.8.10;
```

3.2 CVF-2

- **Severity** Minor
- **Category** Bad datatype
- **Status** Skipped
- **Source** ERC721Extended.sol

Recommendation The type of this variable should be "IERC721Descriptor".

Listing 2:

```
address public tokenDescriptor;
```

3.3 CVF-3

- **Severity** Minor
- **Category** Suboptimal
- **Status** Skipped
- **Source** ERC721Extended.sol

Description Storing nonces per token rather than per signer looks weird.
Recommendation Consider using per-signer nonces.

Listing 3:

```
mapping(uint256 => uint256) public nonces;
```

3.4 CVF-4

- **Severity** Minor
- **Category** Bad datatype
- **Status** Skipped
- **Source** ERC721Extended.sol

Recommendation The argument type should be "IERC721Descriptor".

Listing 4:

```
function setTokenDescriptor(address descriptor) external {
```

3.5 CVF-5

- **Severity** Minor
- **Category** Suboptimal
- **Status** Skipped
- **Source** ERC721Extended.sol

Recommendation One way to optimize this function is to calculate the domain separator in constructor and store it together with the chain ID in immutable variables. When domain separator is needed, check whether the chain ID is still the same, and if it is, use the stored domain separator, otherwise calculate domain separator.

Listing 5:

```
function DOMAIN_SEPARATOR() public view returns (bytes32
    ↪ domainSeperator) {
```

3.6 CVF-6

- **Severity** Moderate
- **Category** Suboptimal
- **Status** Skipped
- **Source** ERC721Extended.sol

Description ERC-1271 signatures may have arbitrary length, while this implementation allows only signatures of 65 bytes long.
Recommendation Consider refactoring the code to support signatures of arbitrary length.

Listing 6:

```
require(IERC1271(owner).isValidSignature(digest, abi.
    ↪ encodePacked(r, s, v)) == 0x1626ba7e, "Unauthorized");
```

3.7 CVF-7

- **Severity** Minor
- **Category** Procedural
- **Status** Replied
- **Source** ERC721Extended.sol

Description The order of v,r,s in the encoding is different to the more traditional ecrecover order.
Recommendation Consider using the same order for uniformity.
Client Comment The reference implementation in EIP-1271 is using the order of (r, s, v). We're following that. https://eips.ethereum.org/EIPS/eip-1271#reference-implementation

Listing 7:

```
76 require(IERC1271(owner).isValidSignature(digest, abi.
   ↪ encodePacked(r, s, v)) == 0x1626ba7e, "Unauthorized");
```

3.8 CVF-8

- **Severity** Minor
- **Category** Bad datatype
- **Status** Skipped
- **Source** SelfPermit.sol

Recommendation The type of this argument should be "IERC20".

Listing 8:

```
14 address token,

36 address token,
```

3.9 CVF-9

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

Description This should be done via a normal Solidity function call.
Recommendation Just define an interface with the "permit" function signature, cast “token” to this interface, and call the “permit” function as usual.

Listing 9:

```
    (bool success, ) = token.call(
        abi.encodeWithSelector(0xd505accf, msg.sender, address(
            ↪ this), value, deadline, v, r, s)
    );
    require(success, "PERMIT_FAILED");

        abi.encodeWithSelector(0x8fcbaf0c, msg.sender, address(
            ↪ this), nonce, expiry, true, v, r, s)
    );
    require(success, "PERMIT_FAILED");
}
```

3.10 CVF-10

- **Severity** Minor
- **Category** Bad datatype
- **Status** Skipped
- **Source** Manager.sol

Recommendation The types of the arguments should be “IMuffinHub” and “IWETH” respectively.

Listing 10:

```
constructor(address _hub, address _WETH9) ManagerBase(_hub,
    ↪ _WETH9) {}
```

3.11 CVF-11

- **Severity** Minor
- **Category** Procedural
- **Status** Skipped
- **Source** Manager.sol

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

Listing 11:

```
constructor(address _hub, address _WETH9) ManagerBase(_hub,
    ↪ _WETH9) {}
```

3.12 CVF-12

- **Severity** Minor
- **Category** Bad datatype
- **Status** Skipped
- **Source** SwapManager.sol

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

Listing 12:

```
address tokenIn,
address tokenOut,

address tokenIn,
address tokenOut,

address tokenIn,
address tokenOut,
```

3.13 CVF-13

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

Recommendation A more elegant way to silence unused arguments warnings is to just remove (comment out) names of unused arguments from the function signature.

Listing 13:

```
tokenOut; // shhh
amountOut; // shhh
```

3.14 CVF-14

- **Severity** Minor
- **Category** Bad datatype
- **Status** Skipped
- **Source** PositionManager.sol

Recommendation The type of these fields should be “iERC20”.

Listing 14:

```
address token0;
address token1;

address token0;
address token1;
```

3.15 CVF-15

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

Description Despite the comment, this is not an array but rather a mapping.
Recommendation Consider fixing the comment.

Listing 15:

```
/// @notice Array of pools represented by its underlying token
   ↪ pair (pairId => Pair)
mapping(uint40 => Pair) public pairs;
```

3.16 CVF-16

- **Severity** Minor
- **Category** Documentation
- **Status** Skipped
- **Source** PositionManager.sol

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

Listing 16:

```
constructor() ERC721Extended("Deliswap Position", "DELI-POS") {}
```

3.17 CVF-17

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

Description The doesn't take into account the "_operatorApprovals" mapping.
Recommendation Consider checking against this mapping as well.

Listing 17:

```
require(msg.sender == positionsByTokenId[tokenId].owner || msg.
    ↪ sender == getApproved(tokenId), "NOT_APPROVED");
```

3.18 CVF-18

- **Severity** Minor
- **Category** Bad datatype
- **Status** Skipped
- **Source** PositionManager.sol

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

Listing 18:

```
function _cacheTokenPair(address token0, address token1)
    ↪ internal returns (uint40 pairId) {
```

3.19 CVF-19

- **Severity** Moderate
- **Category** Flaw
- **Status** Replied
- **Source** PositionManager.sol

Description Pool ID calculated here depends on the order of two tokens.
Recommendation Consider either adding a require statement to ensure that tokens are specifying in proper order, or add a logic to automatically reorder tokens in case their order is incorrect.
Client Comment Not agree. This is good to make periphery contracts agnostic to what a correct token ordering is, and delegate this logic and the validation responsibility to the hub contract.

Listing 19:

```
bytes32 poolId = keccak256(abi.encode(token0, token1));
```

3.20 CVF-20

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

Description The pool ID calculation logic is coded in several places.
Recommendation Consider extracting to a utility function.

Listing 20:

```
48  bytes32 poolId = keccak256(abi.encode(token0, token1));

71  (uint8 tickSpacing, ) = IMuffinHub(hub).getPoolParameters(
      ↪ keccak256(abi.encode(token0, token1)));

232     IMuffinHub(hub).getTier(keccak256(abi.encode(pair.token0,
          ↪ pair.token1)), info.tierId).sqrtPrice,

363         keccak256(abi.encode(pair.token0, pair.token1)),

404     keccak256(abi.encode(token0, token1)),
```

3.21 CVF-21

- **Severity** Minor
- **Status** Skipped
- **Category** Bad datatype
- **Source** PositionManager.sol

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

Listing 21:

```
66  address token0,
    address token1,

86  address token0,
    address token1,
```

3.22 CVF-22

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

Description The number formal of these values is unclear.
Recommendation Consider documenting.

Listing 22:

```
uint24 sqrtGamma,
uint128 sqrtPrice
```

3.23 CVF-23

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

Description These shifts by 80 and by 64 looks cryptic.
Recommendation Consider writing them as (72 + 8) and (72 - 8) respectively.

Listing 23:

```
deposit(msg.sender, token0, UnsafeMath.ceilDiv(uint256(Constants
  ↪ .BASE_LIQUIDITY_D8) << 80, sqrtPrice));
deposit(msg.sender, token1, UnsafeMath.ceilDiv(uint256(Constants
  ↪ .BASE_LIQUIDITY_D8) * sqrtPrice, 1 << 64));
```

3.24 CVF-24

- **Severity** Moderate
- **Category** Flaw
- **Status** Replied
- **Source** PositionManager.sol

Description There is no check that the tokens go in proper order.
Recommendation Consider either checking the token order or automatically reordering the tokens in case their order is incorrect.
Client Comment Not agree. This is good to make periphery contracts agnostic to what a correct token ordering is, and delegate this logic and the validation responsibility to the hub contract.

Listing 24:

```
pairId: _cacheTokenPair(params.token0, params.token1),

Pair(params.token0, params.token1),
```

3.25 CVF-25

- **Severity** Major
- **Category** Flaw
- **Status** Replied
- **Source** PositionManager.sol

Description This check doesn't take collected fees into account, thus transaction could be reverted even if the total collected amounts are enough.
Recommendation Consider taking collected fees into account in this check.
Client Comment Not a flaw. This check is for preventing price slippage, which only reflects on the position's underlying token amounts. Fee amounts should not be taken into account. In any case, comments are updated to explain "amount{0,1}Min" more clearly.

Listing 25:

```
316 require(amount0 >= params.amount0Min && amount1 >= params.
    ↪ amount1Min, "Price slippage");
```

3.26 CVF-26

- **Severity** Minor
- **Category** Bad datatype
- **Status** Skipped
- **Source** PositionManager.sol

Recommendation The type of these returned values should be “IERC20”.

Listing 26:

```
388 address token0,
    address token1,
```

3.27 CVF-27

- **Severity** Major
- **Category** Suboptimal
- **Status** Replied
- **Source** PositionManager.sol

Recommendation These custom functions wouldn't be necessary if position owner addresses would be stored separately (in a separate mapping) from position information. This would also make NFT transfers a bit cheaper and would allow using the standard ERC721 implementation from OpenZeppelin, with necessity to fork it.
Client Comment This is intended to store the position owner address together with the positon information in the same struct in one mapping. All together they fit into one slot, reducing gas usage on every position-related action.

Listing 27:

```
418 function _getOwner(uint256 tokenId) internal view override
    ↪ returns (address owner) {

423 function _setOwner(uint256 tokenId, address owner) internal
    ↪ override {
```

3.28 CVF-28

- **Severity** Minor
- **Category** Suboptimal
- **Status** Skipped
- **Source** Multicall.sol

Description The error data returned here doesn't include the operation index, so it is hard to tell which operation failed.
Recommendation Consider using a named error to wrap both, the index of the failed operation and the error data, returned from it.

Listing 28:

```
18 if (result.length < 68) revert();

22 revert(abi.decode(result, (string)));
```

3.29 CVF-29

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

Description This code unpacks a string error message from returned data, and then packs this message back, which is waste of gas.
Recommendation Consider just returning the result as is: assembly { revert(add(result, 0x20), mload (result)) }

Listing 29:

```
assembly {
    result := add(result, 0x04)
}
revert(abi.decode(result, (string)));
```

3.30 CVF-30

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

Description This screws up the length of "result" which could lead to undesired consequences.
Recommendation See the following answer for details: https://ethereum.stackexchange.com/a/110428

Listing 30:

```
result := add(result, 0x04)
```

3.31 CVF-31

- **Severity** Minor
- **Category** Bad datatype
- **Status** Skipped
- **Source** ManagerBase.sol

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

Listing 31:

```
address public immutable WETH9;
```

3.32 CVF-32

- **Severity** Minor
- **Category** Bad datatype
- **Status** Skipped
- **Source** ManagerBase.sol

Recommendation The type of this variable should be “IMuffinHub”.

Listing 32:

```
address public immutable hub;
```

3.33 CVF-33

- **Severity** Minor
- **Category** Bad datatype
- **Status** Skipped
- **Source** ManagerBase.sol

Recommendation The argument types should be “IMuffinHub” and “IWETH” respectively.

Listing 33:

```
constructor(address _hub, address _WETH9) {
```

3.34 CVF-34

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

Description The explicit conversion to “uint256” is redundant, as Solidity compiler is able to do it implicitly.

Listing 34:

```
accRefId = uint256(uint160(user));
```

3.35 CVF-35

- **Severity** Minor
- **Category** Bad datatype
- **Status** Skipped
- **Source** ManagerBase.sol

Recommendation The type of this argument should be "IERC20".

Listing 35:

```
28 address token,

51 address token,

64 address token,

76 address token,

90 address token,
```

3.36 CVF-36

- **Severity** Minor
- **Category** Suboptimal
- **Status** Skipped
- **Source** ManagerBase.sol

Description This doesn't cover situation when contract's ether balance is not enough, but there is some WETH at contract's balance and ether balance together with WETH balance would be enough.
Recommendation Consider handling such case.

Listing 36:

```
32 if (token == WETH9 && address(this).balance >= amount) {
```

3.37 CVF-37

- **Severity** Major
- **Category** Flaw
- **Status** Replied
- **Source** ManagerBase.sol

Description The returned value is ignored.
Recommendation Consider reverting in case "false" was returned.
Client Comment WETH9 will always revert the transaction if the transfer fails. Checking the return value should be redundant.

Listing 37:

```
35 IWETH(WETH9).transfer(hub, amount);
```

3.38 CVF-38

- **Severity** Moderate
- **Category** Flaw
- **Status** Replied
- **Source** ManagerBase.sol

Description This code pays with contract's own ether even when payer is not this contract.
Client Comment Not a flaw. This contract is not going to store any funds. It works as a temporary ETH/WETH custody in one atomic transaction for multicall users to wrap/unwrap ethers. User is expected to take away all ETH and WETH from the contract at the end of their transaction.
In addition, if users worry about their WETH in the contract being stolen by another malicioius contract in the middle of a multicall, the "unwrapWETH" function has an argument "amountMinimum" to ensure users can take away an expected amount of WETH at the end.

Listing 38:

```
IWETH(WETH9).transfer(hub, amount);
```

3.39 CVF-39

- **Severity** Moderate
- **Category** Flaw
- **Status** Replied
- **Source** ManagerBase.sol

Description This function is payable but doesn't use "msg.value". This makes it possible for anyone to take any unsolicited ether on contract's balance.
Recommendation Consider passing "msg.value" into the callback and use it there instead of the contracts's balance. Alternatively, consider checking, that either "msg.value" is zero, or "token" is WETH and "msg.value" equals to "amount".
Client Comment Not a flaw. All ethers in the contract is expected to belong to the caller. See CVF-38. In addition, "msg.value" should not be used inside a multicall.

Listing 39:

```
) public payable {
```

3.40 CVF-40

- **Severity** Moderate
- **Category** Flaw
- **Status** Replied
- **Source** ManagerBase.sol

Description These functions are callable by anyone, and could be used to take funds from the contract.
Recommendation Consider limiting access to these functions.
Client Comment Not a flaw. See CVF-38.

Listing 40:

```
function unwrapWETH(uint256 amountMinimum, address recipient)
    ↪ external payable {

function refundETH() external payable {
```

3.41 CVF-41

- **Severity** Minor
- **Category** Procedural
- **Status** Skipped
- **Source** ERC721.sol

Description Solidity doesn't support immutable string variables, however it is possible to pack a short string into a 256-bit word and store such word in an immutable variable.
Recommendation See the following gist for an example how this could be done efficiently: https://gist.github.com/3sGgpQ8H/567354534170905e047b299286697e19

Listing 41:

```
string private _name;

string private _symbol;
```

3.42 CVF-42

- **Severity** Minor
- **Category** Suboptimal
- **Status** Skipped
- **Source** ERC721.sol

Recommendation These functions wouldn't be necessary if the corresponding variables would be declared public and named appropriately.

Listing 42:

```
function name() public view virtual override returns (string
    ↪ memory) {

function symbol() public view virtual override returns (string
    ↪ memory) {
```

3.43 CVF-43

- **Severity** Minor
- **Category** Flaw
- **Status** Skipped
- **Source** ERC721.sol

Description The case when a callback reverts with empty data is not distinguishable from the case when it reverted with exactly this string message.
Recommendation Consider concatenating a constant prefix with the data returned by the callback like this: revert(string(abi.encodePacked("ERC721 receiver error: ", reason));

Listing 43:

```
385 revert("ERC721: transfer to non ERC721Receiver implementer");
```