Report

v. 1.0

Customer

Uniswap

Smart Contract Audit

UniswapX

17th July 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	12.07.23	A. Zveryanskaya	Initial Draft
0.2	17.07.23	A. Zveryanskaya	Minor revision
1.0	17.07.23	A. Zveryanskaya	Release

2 Introduction

All modifications to this document are prohibited. Violators will be prosecuted to the full extent of the U.S. law.

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) 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.

The Uniswap Protocol is an open-source protocol for providing liquidity and trading ERC20 tokens on Ethereum. It eliminates trusted intermediaries and unnecessary forms of rent extraction, allowing for safe, accessible, and efficient exchange activity. The protocol is non-upgradable and designed to be censorship resistant.

3 Project scope

We were asked to review:

- Original UniswapX Code

Then were asked to review some additional code:

- UniswapX update

And:

- Code with Fixes

UniswapX files:

base/			
	IPSFees.sol	ReactorEvents.sol	ReactorStructs.sol
external/			
	ISwapRouter02.sol	IUniV3SwapRouter.sol	
interfaces/			
	IReactor.sol	IReactorCallback.sol	IValidationCallback.sol
lens/			
	OrderQuoter.sol		
lib/			
	CurrencyLibrary.sol	DutchDecayLib.sol	DutchLimitOrderLib.sol
	ExclusiveDutchLimit OrderLib.sol	Exclusivity OverrideLib.sol	ExpectedBalanceLib.sol
	LimitOrderLib.so	Permit2Lib.sol	ResolvedOrderLib.sol
reactors/			
	BaseReactor.sol	DutchLimitOrder Reactor.sol	ExclusiveDutch LimitOrderReactor.sol
	LimitOrderReactor.sol		
sample-executors/			
	SwapRouter02 Executor.sol	UniswapV3Executor.sol	
sample-validation-contracts/			
	ExclusiveFiller Validation.sol		

UniswapX update:

base/			
	ProtocolFees.sol	ReactorEvents.sol	ReactorStructs.sol
external/			
	ISwapRouter02.sol	IUniV3SwapRouter.sol	
interfaces/			
	IProtocolFee Controller.sol	IReactor.sol	IReactorCallback.sol
	IValidationCallback.sol		
lens/			
	OrderQuoter.sol		
lib/			
	CurrencyLibrary.sol	DutchDecayLib.sol	DutchLimitOrderLib.sol
	ExclusiveDutch LimitOrderLib.sol	Exclusivity OverrideLib.sol	ExpectedBalanceLib.sol
	LimitOrderLib.sol	OrderInfoLib.sol	Permit2Lib.sol
	ResolvedOrderLib.sol		
reactors/			
	BaseReactor.sol	DutchLimitOrder Reactor.sol	ExclusiveDutchLimit OrderReactor.sol
	LimitOrderReactor.sol		
sample-executors/			
	SwapRouter02 Executor.sol		
sample-validation-contracts/			
	ExclusiveFiller Validation.sol		

4 Methodology

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

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

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 14 major, and a few less important issues.

	Active	Fixed
Major	7	7
Moderate	3	2
Minor	44	31

Fixed 40 out of 94 issues

6 Major Issues

CVF-1. FIXED

- **Category** Unclear behavior
- **Source** SwapRouter02Executor.sol

Description The returned value is ignored.

Recommendation Consider using the “safeApprove” function or checking the returned value.

```
74 +tokensToApprove[i].approve(address(swapRouter02), type(uint256).max
      ↪ );
```

CVF-2. FIXED

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

Description The comment is actually inaccurate, as the filler may use not only the ether sent along with the call, but also ether that was at the contract’s balance before the call.

Recommendation Consider either forbidding to use ether from the contract’s balance, or making this comment more accurate.

```
105 +// refund any remaining ETH to the filler. Only occurs when filler
       ↪ sends more ETH than required to
106 +// `execute()` or `executeBatch()`
```

CVF-3. INFO

- **Category** Suboptimal
- **Source** DutchLimitOrderLib.sol

Recommendation The memory address could be calculated incrementally, but adding 0×20 to the previous address. No need to do all these calculations on every iteration.

Client Comment *Gas was worse when doing this, likely due to newly required dup/swaps.*

```
84 +mstore(add(add(packedHashes, 0x20), mul(i, 0x20)), outputHash)
```

CVF-4. INFO

- **Category** Suboptimal
- **Source** LimitOrderLib.sol

Recommendation The memory address could be calculated incrementally by adding 0×20 to the previous address. No need to do all these calculations on every iteration.

Client Comment *Gas was worse when doing this, likely due to newly required dup/swaps.*

```
53 +mstore(add(add(packedHashes, 0x20), mul(i, 0x20)), outputHash)
```

CVF-5. FIXED

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

Description The function actually operates with a single order (that could have multiple outputs), so plural “orders” sounds odd.

Recommendation Consider either changing to “order” or changing the argument type to “ResolveOrder[]”.

```
8  +/// @notice Get fee outputs for the given orders
9  +/// @param orders The orders to get fee outputs for
10 +/// @return List of list of fee outputs to append for each provided
      ↪  order
11 +function getFeeOutputs(ResolvedOrder memory orders) external view
      ↪ returns (OutputToken[] memory);
```

CVF-6. INFO

- **Category** Suboptimal
- **Source** ProtocolFees.sol

Recommendation As each output amount could be used only once, the following optimization is possible: once an amount is used, replace the used element of “order.outputs” with “order.outputs[outputsLength - 1]” and reduce “outputLength” by one, then proceed to the next iteration without incrementing “j”. The would allow not iterating through already used outputs.

Client Comment *outputsLength is used later on, so we also need to store a new stack variable placeholder. This, plus the extra memory manipulation means the gas benefits is actually even worse for small output lengths which we expect to be the general case.*

```
70 +tokenValue += output.amount;
```

CVF-7. INFO

- **Category** Unclear behavior
- **Source** UniswapV3Executor.sol

Description This function doesn't allow specifying the amount to be claimed.

Recommendation Consider adding such ability, as there could be tokens with transfer limits or transfer fees.

```
63 63 function claimTokens(ERC20 token) external onlyOwner {
```

CVF-8. FIXED

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

Recommendation Consider throwing an error from the "validate" function when validation failed. This would make returning a value from the "validate" function redundant.

```
23 23 if (
24 24     resolvedOrder.info.validationContract != address(0)
25 25         && !IValidationCallback(resolvedOrder.info.
                 ↪ validationContract).validate(filler, resolvedOrder)
26 26 ) {
27 27     revert ValidationFailed();
28 28 }
```

CVF-9. INFO

- **Category** Suboptimal
- **Source** DutchDecayLib.sol

Description It seems that for all the usages, it is known a compile time, whether decay is positive or negating, so handing the different between these two cases at run time is suboptimal.

Recommendation Consider implementing separate functions for positive and negative decay.

Client Comment *Talked with Alice and we determined it is not worth to implement this because the gas savings are minimal and it makes the code less readable.*

```
16 16 /// @dev handles both positive and negative decay depending on
      ↪ startAmount and endAmount
```

CVF-10. FIXED

- **Category** Suboptimal
- **Source** DutchDecayLib.sol

Description The equality checks for amounts and times optimize rare case at the cost of making the most common case more expensive.

Recommendation Consider removing these optimizations.

```
28 28 } else if (endTime < block.timestamp || startAmount == endAmount ||
      ↪ startTime == endTime) {
```

CVF-11. FIXED

- **Category** Documentation
- **Source** ExclusivityOverrideLib.sol

Description Technically, scaling up by more than 100% is possible.

Recommendation Consider explaining why it is not allowed here.

```
41 41 // if override is greater than 100% we cannot scale outputs
```

CVF-12. FIXED

- **Category** Suboptimal
- **Source** DutchLimitOrderLib.sol

Description Filling an address array and then repacking it tightly is suboptimal.

Recommendation Consider filling a tightly packed array using assembly to avoid repacking.

```
87 87     for (uint256 i = 0; i < outputs.length; i++) {
88 88         outputHashes[i] = hash(outputs[i]);
89 89     }
```

```
91 91 return keccak256(abi.encodePacked(outputHashes));
```

CVF-13. INFO

- **Category** Suboptimal
- **Source** IPSFees.sol

Description While the mulDivDown function is very efficient in general case, for specific cases more efficient approaches do exist. For example, when the denominator is known at the compile time, its modular multiplicative inverse could be precomputed.

```
60  60  uint256 protocolFeeAmount = output.amount.mulDivDown(
        ↪ PROTOCOL_FEE_BPS, BPS);
```

CVF-14. INFO

- **Category** Suboptimal
- **Source** IPSFees.sol

Description As ”PROTOCOL_FEE_BPS” is a constant, there is no need to split fees between the protocol and the interface on every order.

Recommendation Consider accumulating raw (unsplit) fees and splitting only when fees are claimed by either the protocol of the interface.

```
60  60  uint256 protocolFeeAmount = output.amount.mulDivDown(
        ↪ PROTOCOL_FEE_BPS, BPS);
```

7 Moderate Issues

CVF-15. INFO

- **Category** Suboptimal
- **Source** ProtocolFees.sol

Recommendation The complexity of this function is O(n^2). It could be reduced to O(n ln n) by demanding both, order outputs and fee outputs to be sorted by token.

Client Comment *# fees and outputs are expected to always be small, so the extra integration complexity of requiring sorted not worth the small gain.*

```
33 +function _injectFees(ResolvedOrder memory order) internal view {
```

CVF-16. FIXED

- **Category** Flaw
- **Source** CurrencyLibrary.sol

Description There is no check to ensure that value was actually sent along with the current call.

Recommendation Consider adding such a check.

```
45 45 /// @dev if curency is ETH, the value must have been sent in the
      ↪ execute call and is transferred directly
```

CVF-17. INFO

- **Category** Suboptimal
- **Source** ExpectedBalanceLib.sol

Description This function has O(n^2) complexity.

Recommendation Consider optimizing it. One way to do this would be to collect all the outputs of all orders without deduplication, then sort the resulting array, and then deduplicate. This should reduce the complexity to about O(n ln n).

Client Comment *We have decided not to implement any optimizations as we have deemed them impractical.*

```
19 19 function getExpectedBalances(ResolvedOrder[] memory orders)
```

CVF-18. INFO

- **Category** Suboptimal
- **Source** Permit2Lib.sol

Description Using the same nonce for an order and for a Permit2 transfer could be limiting, as the token owner may want to also sign Permit2 transaction not related to any orders, and their nonces could conflict with the nonces of future orders.

Recommendation Consider using separate nonces with Permit2.

Client Comment *It is known that nonce overlap may occur if other apps use Permit2 signature transfers. Acceptable risk.*

```
18 18  nonce: order.info.nonce,
```

CVF-19. FIXED

- **Category** Suboptimal
- **Source** OrderQuoter.sol

Description All the elements of ”resolvedOrders” array except for the first one are silently ignored.

Recommendation Consider explicitly asserting that ”resolverOrders.length” is one.

```
48 48  bytes memory order = abi.encode(resolvedOrders[0]);
```

8 Minor Issues

CVF-20. FIXED

- **Category** Suboptimal
- **Source** ExclusiveFillerValidation.sol

Recommendation This error could be made more useful by adding a “filler” parameter to it.

```
8 +error NotExclusiveFiller();
```

CVF-21. FIXED

- **Category** Bad datatype
- **Source** SwapRouter02Executor.sol

Recommendation The type of the “_reactor” argument should be “IReactor”.

```
26 +constructor(address _whitelistedCaller, address _reactor, address
   ↪ _owner, ISwapRouter02 _swapRouter02)
```

CVF-22. FIXED

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

Recommendation The type of this constant should be “IReactorCallback”.

```
32 +address public constant DIRECT_FILL = address(1);
```

CVF-23. FIXED

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

Recommendation The type of the “_permit2” argument should be “Permit2”.

```
34 +constructor(address _permit2, address _protocolFeeOwner)
   ↪ ProtocolFees(_protocolFeeOwner) {
```

CVF-24. FIXED

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

Recommendation The type of the "_permit2" argument should be "Permit2".

```
21 +constructor(address _permit2, address _protocolFeeOwner)
      ↪ BaseReactor(_permit2, _protocolFeeOwner) {}
```

CVF-25. FIXED

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

Recommendation The type of the "_permit2" argument should be "Permit2".

```
29 +constructor(address _permit2, address _protocolFeeOwner)
      ↪ BaseReactor(_permit2, _protocolFeeOwner) {}
```

CVF-26. FIXED

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

Recommendation The type of the "_permit2" argument should be "Permit2".

```
15 +constructor(address _permit2, address _protocolFeeOwner)
      ↪ BaseReactor(_permit2, _protocolFeeOwner) {}
```

CVF-27. INFO

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

Recommendation The type of the "currency" argument should be "ERC20".

Client Comment *'currency' can be ETH as well, so this does not hold true*

```
51 +function transferFromDirectFiller(address currency, address
      ↪ recipient, uint256 amount, IAllowanceTransfer permit2)
```

CVF-28. INFO

- **Category** Procedural
- **Source** DutchLimitOrderLib.sol

Description We didn't review this file.

```
6 +import {ERC20} from "solmate/src/tokens/ERC20.sol";
```

CVF-29. FIXED

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

Description The function actually returnes just a list of outputs, not a list of lists.

Client Comment .

```
10 +/// @return List of list of fee outputs to append for each provided
   ↪  order
```

CVF-30. INFO

- **Category** Procedural
- **Source** ReactorStructs.sol

Description We didn't review this file.

```
6 +import {ERC20} from "solmate/src/tokens/ERC20.sol";
```

CVF-31. INFO

- **Category** Procedural
- **Source** ProtocolFees.sol

Description We didn't review these files.

```
4 +import {Owned} from "solmate/src/auth/Owned.sol";
5 +import {SafeTransferLib} from "solmate/src/utils/SafeTransferLib.
   ↪ sol";
6 +import {FixedPointMathLib} from "solmate/src/utils/
   ↪ FixedPointMathLib.sol";
7 +import {ERC20} from "solmate/src/tokens/ERC20.sol";
```

CVF-32. FIXED

- **Category** Suboptimal
- **Source** ProtocolFees.sol

Recommendation These errors could be made more useful by adding some parameters to them.

```
18 +error DuplicateFeeOutput();
19 +error FeeTooLarge();
20 +error InvalidFeeToken();
```

CVF-33. FIXED

- **Category** Procedural
- **Source** ProtocolFees.sol

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

```
28 +constructor(address _owner) Owned(_owner) {}
```

CVF-34. FIXED

- **Category** Suboptimal
- **Source** ProtocolFees.sol

Description This function should emit some event.

```
98 +function setProtocolFeeController(address _feeController) external
   ↪ onlyOwner {
```

CVF-35. FIXED

- **Category** Procedural
- **Source** ExclusiveFillerValidation.sol

Recommendation Consider specifying as "^0.8.0" unless there is something special about this particular version. Also relevant for: SwapRouter02Executor.sol, UniswapV3Executor.sol, BaseReactor.sol, DutchLimitOrderReactor.sol, ExclusiveDutchLimitOrderReactor.sol, LimitOrderReactor.sol, CurrencyLibrary.sol, ResolvedOrderLib.sol, ExpectedBalanceLib.sol, DutchDecayLib.sol, ExclusivityOverrideLib.sol, LimitOrderLib.sol, ExclusiveDutchLimitOrderLib.sol, DutchLimitOrderLib.sol, OrderQuoter.sol, IReactor.sol, IReactorCallback.sol, IValidationCallback.sol, ISwapRouter02.sol, IUniV3SwapRouter.sol, ReactorStructs.sol, IPSFees.sol, ReactorEvents.sol.

```
2 2 pragma solidity ^0.8.19;
```

CVF-36. FIXED

- **Category** Procedural
- **Source** ExclusiveFillerValidation.sol

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

Client Comment *Just deleted the constructor instead.*

```
8  8  constructor() {}
```

CVF-37. INFO

- **Category** Procedural
- **Source** SwapRouter02Executor.sol

Description We didn't review these files.

```
4  4  import {Owned} from "solmate/src/auth/Owned.sol";
5  5  import {SafeTransferLib} from "solmate/src/utils/SafeTransferLib.sol
       ↪ ";
6  6  import {ERC20} from "solmate/src/tokens/ERC20.sol";
7  7  import {WETH} from "solmate/src/tokens/WETH.sol";
```

CVF-38. FIXED

- **Category** Suboptimal
- **Source** SwapRouter02Executor.sol

Recommendation These errors could be made more useful by adding certain parameters to them.

Client Comment *Deleted "EtherSendFail" error as it was unused. No need to add param to "CallerNotWhitelisted".*

```
18 18  error CallerNotWhitelisted();
```

```
20 20  error EtherSendFail();
```

CVF-39. FIXED

- **Category** Bad datatype
- **Source** SwapRouter02Executor.sol

Recommendation The type of this variable should be "ISwapRouter02".

```
23 23  address private immutable swapRouter02;
```

CVF-40. FIXED

- **Category** Bad datatype
- **Source** SwapRouter02Executor.sol

Recommendation The type of the "_swapRouter02" should be "ISwapRouter02".

```
28  28  constructor(address _whitelistedCaller, address _reactor, address
        ↪ _owner, address _swapRouter02) Owned(_owner) {
```

CVF-41. FIXED

- **Category** Bad datatype
- **Source** SwapRouter02Executor.sol

Recommendation The type of the "tokensToApprove" argument should be "IERC20".

```
72  72  function multicall(address[] calldata tokensToApprove, bytes[]
        ↪ calldata multicallData) external onlyOwner {
```

CVF-42. FIXED

- **Category** Suboptimal
- **Source** SwapRouter02Executor.sol

Description This check looks odd, as even of the balance is not zero, it doesn't mean that the balance is sufficient.

Recommendation Consider removing this check.

```
84  84  if (balanceWETH == 0) revert InsufficientWETHBalance();
```

CVF-43. FIXED

- **Category** Procedural
- **Source** SwapRouter02Executor.sol

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

```
96  96  receive() external payable {}
```

CVF-44. INFO

- **Category** Procedural
- **Source** UniswapV3Executor.sol

Description We didn't review these files.

```
4 4  import {Owned} from "solmate/src/auth/Owned.sol";
5 5  import {SafeTransferLib} from "solmate/src/utils/SafeTransferLib.sol
       ↪ ";
6 6  import {ERC20} from "solmate/src/tokens/ERC20.sol";
```

CVF-45. INFO

- **Category** Bad datatype
- **Source** UniswapV3Executor.sol

Recommendation The type of this variable should be "IUniV3SwapRouter".

Client Comment *UniswapV3Executor has been deleted.*

```
19 19  address public immutable swapRouter;
```

CVF-46. INFO

- **Category** Bad datatype
- **Source** UniswapV3Executor.sol

Recommendation The type of the "_swapRouter" argument should be "IUniV3SwapRouter".

Client Comment *UniswapV3Executor has been deleted.*

```
22 22  constructor(address _reactor, address _swapRouter, address _owner)
         ↪ Owned(_owner) {
```

CVF-47. INFO

- **Category** Procedural
- **Source** BaseReactor.sol

Description We didn't review these files.

```
4 4  import {SafeTransferLib} from "solmate/src/utils/SafeTransferLib.sol
       ↪ ";
```

```
8 8  import {ERC20} from "solmate/src/tokens/ERC20.sol";
```

CVF-48. INFO

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

Recommendation The type of this variable should be ”Permit2”.

Client Comment *There is no IPermit2 interface because permit2 can be either ISignatureTransfer or IAllowanceTransfer. No action taken.*

```
31 31 address public immutable permit2;
```

CVF-49. INFO

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

Recommendation The type of the ”_permit2” argument should be ”Permit2”.

Client Comment *There is no IPermit2 interface because permit2 can be either ISignatureTransfer or IAllowanceTransfer. No action taken.*

```
34 34 constructor(address _permit2, uint256 _protocolFeeBps, address
      ↪ _protocolFeeRecipient)
```

CVF-50. FIXED

- **Category** Procedural
- **Source** BaseReactor.sol

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

Client Comment *Note that this receive function will be removed after Mark’s updated-rotocol-fees branch*

```
40 40 receive() external payable {}
```

CVF-51. FIXED

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

Recommendation The type of the ”fillContract” argument should be ”IReactorCallback”.

```
73 73 function _fill(ResolvedOrder[] memory orders, address fillContract,
      ↪ bytes calldata fillData) internal {
```

CVF-52. INFO

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

Recommendation The type of the ”_permit2” argument should be ”Permit2”.

Client Comment *There is no IPermit2 interface because permit2 can be either ISignatureTransfer or IAllowanceTransfer. No action taken.*

```
21 21 constructor(address _permit2, uint256 _protocolFeeBps, address
      ↪ _protocolFeeRecipient)
```

CVF-53. FIXED

- **Category** Procedural
- **Source** DutchLimitOrderReactor.sol

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

Client Comment *This is simply required to pass parameters to parent constructor. A comment is unnecessary IMO.*

```
23 23 {}
```

CVF-54. INFO

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

Recommendation The type of the ”_permit2” argument should be ”Permit2”.

Client Comment *There is no IPermit2 interface because permit2 can be either ISignatureTransfer or IAllowanceTransfer. No action taken.*

```
29 29 constructor(address _permit2, uint256 _protocolFeeBps, address
      ↪ _protocolFeeRecipient)
```

CVF-55. INFO

- **Category** Procedural
- **Source** ExclusiveDutchLimitOrderReactor.sol

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

Client Comment *This is simply required to pass parameters to parent constructor. A comment is unnecessary IMO.*

```
31 31 {}
```

CVF-56. INFO

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

Recommendation The type of the ”_permit2” argument should be ”Permit2”.

Client Comment *There is no IPermit2 interface because permit2 can be either ISignatureTransfer or IAllowanceTransfer. No action taken.*

```
15 15 constructor(address _permit2, uint256 _protocolFeeBps, address
      ↪ _protocolFeeRecipient)
```

CVF-57. INFO

- **Category** Procedural
- **Source** LimitOrderReactor.sol

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

Client Comment *This is simply required to pass parameters to parent constructor. A comment is unnecessary IMO.*

```
17 17 {}
```

CVF-58. INFO

- **Category** Procedural
- **Source** CurrencyLibrary.sol

Description We didn't review these files.

```
4 4  import {ERC20} from "solmate/src/tokens/ERC20.sol";
```

```
7 7  import {SafeTransferLib} from "solmate/src/utils/SafeTransferLib.sol
       ↪ ";
```

CVF-59. INFO

- **Category** Procedural
- **Source** CurrencyLibrary.sol

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

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

Client Comment *Keeping top level to avoid having to use longer variable references*

```
9 9  address constant NATIVE = 0x0000000000000000000000000000000000000000
       ↪ ;
```

CVF-60. INFO

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

Recommendation The type of the "currency" argument should be 'IERC20".

Client Comment *'currency' is not IERC20 because it can be native ETH as welll*

```
23 23  function balanceOf(address currency, address addr) internal view
         ↪ returns (uint256 balance) {
```

```
35 35  function transfer(address currency, address recipient, uint256
         ↪ amount) internal {
```

```
50 50  function transferFromDirectTaker(address currency, address recipient
         ↪ , uint256 amount, address permit2) internal {
```

```
62 62  function isNative(address currency) internal pure returns (bool) {
```

CVF-61. INFO

- **Category** Procedural
- **Source** ResolvedOrderLib.sol

Recommendation The term ”filler” is not conventional. The conventional term is ”taker”.

```
13 13  /// @param filler The filler of the order
```

CVF-62. INFO

- **Category** Procedural
- **Source** ExpectedBalanceLib.sol

Description Declaring a top-level structure in a file named after a library makes it harder to navigate through code.

Recommendation Consider either moving the structure declaration into the library, or moving it into a separate file.

Client Comment *Keeping top level to avoid having to use longer variable references*

```
7 7  struct ExpectedBalance {
```

CVF-63. INFO

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

Recommendation The type of this field should be ”IERC20”.

Client Comment *‘token‘ can be native ETH as well, so this is untrue.*

```
9 9  address token;
```

CVF-64. FIXED

- **Category** Suboptimal
- **Source** ExpectedBalanceLib.sol

Recommendation This error could be made more useful by adding certain parameters into it.

```
16 16  error InsufficientOutput();
```

CVF-65. INFO

- **Category** Procedural
- **Source** DutchDecayLib.sol

Description We didn't review this file.

```
6  6   import {FixedPointMathLib} from "solmate/src/utils/FixedPointMathLib
       ↪ .sol";
```

CVF-66. FIXED

- **Category** Suboptimal
- **Source** DutchDecayLib.sol

Recommendation Should be "<=" instead of "<".

```
28 28  } else if (endTime < block.timestamp || startAmount == endAmount ||
       ↪ startTime == endTime) {
```

CVF-67. FIXED

- **Category** Suboptimal
- **Source** DutchDecayLib.sol

Recommendation Should be ">=" instead of ">".

```
30 30  } else if (startTime > block.timestamp) {
```

CVF-68. INFO

- **Category** Procedural
- **Source** ExclusivityOverrideLib.sol

Description We didn't review this file.

```
4  4   import {FixedPointMathLib} from "solmate/src/utils/FixedPointMathLib
       ↪ .sol";
```

CVF-69. INFO

- **Category** Procedural
- **Source** LimitOrderLib.sol

Description Declaring a top-level structure in a file named after a library makes it harder to navigate through code.

Recommendation Consider either moving the structure declaration into the library or moving into a separate file.

Client Comment *Keeping top level to avoid having to use longer variable references*

```
7  7  struct LimitOrder {
```

CVF-70. INFO

- **Category** Procedural
- **Source** Permit2Lib.sol

Description We didn't review this file.

```
4  4  import {ERC20} from "solmate/src/tokens/ERC20.sol";
```

CVF-71. INFO

- **Category** Procedural
- **Source** ExclusiveDutchLimitOrderLib.sol

Description Declaring a top-level structure in a file named after a library makes it harder to navigate through code.

Recommendation Consider either moving the structure declaration into the library or moving it into a separate file.

Client Comment *Keeping top level to avoid having to use longer variable references.*

```
7  7  struct ExclusiveDutchLimitOrder {
```

CVF-72. INFO

- **Category** Suboptimal
- **Source** ExclusiveDutchLimitOrderLib.sol

Description This approach is error prone as it is easy to forget to update the hash code when changing the order fields.

Recommendation Consider doing some safety check here.

```
57 57  bytes.concat(
```

CVF-73. INFO

- **Category** Procedural
- **Source** DutchLimitOrderLib.sol

Description Declaring top-level structures in a file named after a library makes it harder to navigate through code.

Recommendation Consider either moving the structure definitions into the library or moving them into a separate file.

Client Comment *Keeping top level to avoid having to use longer variable references.*

```
7 7    struct DutchOutput {
21 21  struct DutchInput {
30 30  struct DutchLimitOrder {
```

CVF-74. INFO

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

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

Client Comment *Type must remain ”address” as token can be native ETH as well.*

```
9 9    address token;
23 23  address token;
```

CVF-75. INFO

- **Category** Suboptimal
- **Source** DutchLimitOrderLib.sol

Description This approach is error prone as it is easy to forget to update the hash code when changing the order fields.

Recommendation Consider doing some safety check here.

Client Comment *This is a limitation of forge. We have tests in our SDK to safety test this.*

```
99 99  abi.encode(
```

CVF-76. FIXED

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

Recommendation The return type should be ”IReactor”.

```
29 29  function getReactor(bytes memory order) public pure returns (address
          ↪  payable reactor) {
```

CVF-77. FIXED

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

Recommendation The type of the ”fillContract” arguments should be ”IReactorCallback”.

```
12 12  function execute(SignedOrder calldata order, address fillContract,
          ↪ bytes calldata fillData) external payable;
```

```
18 18  function executeBatch(SignedOrder[] calldata orders, address
          ↪ fillContract, bytes calldata fillData)
```

CVF-78. INFO

- **Category** Procedural
- **Source** IReactorCallback.sol

Recommendation The term ”filler” is not conventional. The conventional term is ”taker”.

```
10 10  /// @param filler The address who called execute on the reactor
```

CVF-79. INFO

- **Category** Procedural
- **Source** IValidationCallback.sol

Recommendation The term ”filler” is not conventional. The conventional term is ”taker”.

```
9  9    /// @param filler The filler of the order
```

CVF-80. INFO

- **Category** Procedural
- **Source** ISwapRouter02.sol

Description Declaring a top-level struct in a file named after an interface makes it harder to navigate through code.

Recommendation Consider either moving the struct definition into the interface or moving it into a separate file.

Client Comment *Keeping top level to avoid having to use longer variable references.*

```
4  4    struct ExactInputSingleParams {
```

CVF-81. INFO

- **Category** Bad datatype
- **Source** ISwapRouter02.sol

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

Client Comment *This struct is taken from Uniswap’s ISwapRouter interface and should not be changed.*

```
5  5    address tokenIn;
6  6    address tokenOut;
```

CVF-82. INFO

- **Category** Bad datatype
- **Source** ISwapRouter02.sol

Recommendation The return type should be ”IWETH9”.

Client Comment *This struct is taken from Uniswap’s ISwapRouter interface and should not be changed.*

```
22  22    function WETH9() external view returns (address);
```

CVF-83. INFO

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

Description The file name ”ReactorStructs.sol” is confusing, as one could think that a contract or interface or library named ”ReactorStructs” is defined in this file, while this is not the case.

Recommendation Consider renaming to ”reactor_structs.sol” or just ”types.sol”.

Client Comment *Keeping as is because other projects have used similar convention.*

```
1  1  // SPDX-License-Identifier: GPL-2.0-or-later
```

CVF-84. FIXED

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

Recommendation The type of this field should be ”IReactor”.

```
10 10 address reactor;
```

CVF-85. FIXED

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

Recommendation The type of this field should be ”IValidationCallback”.

```
19 19 address validationContract;
```

CVF-86. FIXED

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

Recommendation The type of this field should be ”IERC20”.

```
26 26 address token;
```

CVF-87. INFO

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

Recommendation The type of this field should be ”IERC20”.

Client Comment *Should not be changed as output token can be native ETH as well.*

```
34 34   address token;
```

CVF-88. INFO

- **Category** Procedural
- **Source** IPSFees.sol

Description We didn’t review these files.

```
4 4   import {SafeTransferLib} from "solmate/src/utils/SafeTransferLib.sol
          ↪ ";
5 5   import {FixedPointMathLib} from "solmate/src/utils/FixedPointMathLib
          ↪ .sol";
6 6   import {ERC20} from "solmate/src/tokens/ERC20.sol";
```

CVF-89. INFO

- **Category** Suboptimal
- **Source** IPSFees.sol

Recommendation These errors could be made more useful by adding some parameters into them.

```
16 16   error InvalidFee();
```

```
18 18   error UnauthorizedFeeRecipient();
19 19   error FailedToSendEther();
```

CVF-90. INFO

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

Recommendation The type of the first key should be ”IERC20”.

```
32 32   mapping(address => mapping(address => uint256)) public feesOwed;
```

CVF-91. INFO

- **Category** Suboptimal
- **Source** IPSFees.sol

Description The expression ”feesOwed[output.token]” is calculated twice.

Recommendation Consider calculating once and reusing.

```
68 68  feesOwed[output.token][PROTOCOL_FEE_RECIPIENT_STORED] +=
          ↪ protocolFeeAmount;

70 70  feesOwed[output.token][output.recipient] += interfaceFeeAmount;
```

CVF-92. INFO

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

Description This function should emit some event.

```
99 99  function setProtocolFeeRecipient(address _protocolFeeRecipient)
          ↪ external onlyProtocolFeeRecipient {
```

CVF-93. FIXED

- **Category** Procedural
- **Source** ReactorEvents.sol

Recommendation This contract could be turned into an interface.

```
7 7  contract ReactorEvents {
```

CVF-94. INFO

- **Category** Procedural
- **Source** ReactorEvents.sol

Recommendation The terms ”filler” and ”offerer” are not conventional. The conventional terms are ”taker” and ”maker”.

Client Comment *Will fix in another PR.*

```
10 10  /// @param filler The address which executed the fill

12 12  /// @param offerer The offerer of the filled order
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting