Uniswapx Security Review

Auditors

Noah Marconi, Lead Security Researcher

Kaden, Security Researcher

Shung, Associate Security Researcher

Report prepared by: Lucas Goiriz

October 5, 2024

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

The UniswapX Protocol is a non-custodial trading structure that leverages Dutch auctions. It integrates on-chain and off-chain liquidity, providing a seamless trading experience. This unique approach also shields swappers from MEV by incorporating them into the price enhancement process, leading to gas-less swaps. Trading on UniswapX involves creating signed orders that outline specific swap conditions. Fillers, or participants, then utilize various strategies to compete with one another and fulfill these orders.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of UniswapX according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 8 days in total, Uniswapx engaged with Spearbit to review the Uniswapx protocol. In this period of time a total of **12** issues were found.

Summary

Project Name	Uniswapx
Repository	Uniswapx
Commit	f77785...b0e358
Type of Project	DeFi, Dex
Audit Timeline	Jul 22nd to Jul 30th
Two week fix period	Jul 1 - Jul 9

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	3	1	2
Gas Optimizations	4	2	2
Informational	5	2	3
Total	**12**	**7**	**5**

5 Findings

5.1 Low Risk

5.1.1 Lack of chain ID in cosigner signature message

Severity: Low Risk

Context: PriorityOrderReactor.sol#L91, PriorityOrderLib.sol#L155-L157

Description: The cosigner signature message in the PriorityOrderLib does not include a domain separator or chain ID. While the `orderHash` includes the reactor address, the absence of the chain ID in the cosigner's signature message means that this signature could potentially be replayed in a fork scenario.

The `cosignerDigest` function simply concatenates the `orderHash` with the encoded `cosignerData`, without incorporating any chain-specific information:

```
return keccak256(abi.encodePacked(orderHash, abi.encode(order.cosignerData)));
```

This differs from the `orderHash` itself, which is verified by Permit2 and includes a domain separator with chain ID.

Recommendation:

1. Include the chain ID in the cosigner's signature message:

```
  function cosignerDigest(PriorityOrder memory order, bytes32 orderHash) internal view returns
↪   (bytes32) {
      return keccak256(abi.encodePacked(orderHash, block.chainid, abi.encode(order.cosignerData)));
  }
```

2. Alternatively, implement a full EIP-712 domain separator for the cosigner signature, which would include the chain ID along with other relevant information.

These changes would ensure that cosigner signatures are chain-specific, preventing potential replay attacks in fork scenarios.

UniswapX: Fixed in PR 260.

Spearbit: Fixed as recommended.

5.1.2 Input scaling quickly scales to 0

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: When scaling `input.amount`, values quickly scale to 0. See the pseudocode below:

```
priorityFee = tx.gasprice - block.basefee;
priorityFee -= baselinePriorityFeeWei;
scalingFactor = priorityFee * input.mpsPerPriorityFeeWei;
if (scalingFactor >= MPS) then 0 else (amount * (MPS - scalingFactor)) / MPS
```

UniswapX confirmed `mpsPerPriorityFeeWei` will typically be either `0` or a `1`. `MPS` is a constant of `1e7`.

In current conditions, a `priorityFee` less than `MPS` does offer some room to compete for arbitrage when filling an order.

If gas prices were to spike, 1e7 as a percentage of total gas price becomes smaller. Potentially leaving little room to compete for first position in a block.

Recommendation: Consider treating `mpsPerPriorityFeeWei` as the numerator in a fraction rather than 1 or 0. For example, `priorityFee.mulDiv(mpsPerPriorityFeeWeiNUMERATOR, mpsPerPriorityFeeWeiBASE)` allows for fine grained control over how much scaling occurring, even with higher base gas fees.

UniswapX: Acknowledged. Appreciate the suggestion - we are actively thinking about this and monitoring if we begin to see that we are either not getting enough granularity or that our transactions are not being included. I don't think we will take any immediate action but we would like to see preliminary results / analytics for what we have currently before deciding.

Spearbit: Acknowledged.

5.1.3 Zero value transfers may occur

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: User may include 0 values in their order (user error) and fillers may fill order with priority fees that scale to 0 (filler error).

- For tokens allowing 0 value transfers, gas may be wasted.
- For tokens restricting 0 value transfers, orders will revert.

Recommendation: May be handled by the filler's simulation. Consider if skipping 0 value transfers is desirable at the contract level.

UniswapX: Acknowledged.

Spearbit: Acknowledged.

5.2 Gas Optimization

5.2.1 Order looped through multiple times

Severity: Gas Optimization

Context: *(No context files were provided by the reviewer)*

Description: When an order is filled, there are multiple loops iterating over the order data:

- Resolving Order
  1. During order validation when `order.input.mpsPerPriorityFeeWei > 0`, each `PriorityOutput` is checked to confirm `order.outputs[i].mpsPerPriorityFeeWei == 0`.
  2. When scaling, each `PriorityOutput` is scaled.
- Preparing Order
  1. The mock fee controller iterates over each of the `OutputToken`s (assuming the production fee controller does the same).
  2. When injecting fees, each output is iterated over to copy to a new memory array, with fee outputs appended.
  3. When injecting fees, each `fee OutputToken` is looped over.
  4. During the `fee OutputToken` loop, each token iterates backwards (some savings) to confirm it is not a duplicate of a `fee OutputToken` from an earlier index.
  5. During the `fee OutputToken` loop, each `non-fee OutputToken` is iterated over, tallying the `tokenValue`. This value is later used to confirm the fee is not too large.
- Filling Order
  1. Filling the order loops through each `ResolvedOrder`, which includes the fee transfers.

Recommendation: Given the maturity and current deployed status of the codebase, an optimization of this sort is `NOT` recommended. Consider for future versions:

1. Iterating through the `calldata` array, and loading to memory in one, or close to one, loop. This would mean fusing one or more of the loops in `_resolve`, `_prepare`, `_fill`, and `transferFill` related functions and modifying the functions to be called within the loop body, rather than performing their own loops.
2. Appending fees to the `OutputToken` array requires duplication of the incoming array `OutputToken[] memory newOutputs = new OutputToken[](outputsLength + feeOutputsLength);`. Treating the two arrays separately (outputs and fee related outputs) can allow for modifying the outputs in-place rather than duplicating.
3. If the `FeeController` is able to return sorted tokens, a check against the previous token (revert if `feeOutputs[i].token <= feeOutputs[i - 1].token`) may be used for duplicate detection.

Above would require modifications to the array types (e.g. currently `PriorityInput` must be converted to `InputToken` when scaling) and the implementation logic and is not recommended for this current version of the protocol.

UniswapX: Acknowledged.

Spearbit: Acknowledged.

5.2.2 Short circuit input and output scaling in the case that they won't be scaled

Severity: Gas Optimization

Context: PriorityFeeLib.sol#L29, PriorityFeeLib.sol#L42

Description: In the input and output `PriorityFeeLib` `scale` functions, we scale the provided amount accordingly to a `priorityFee` based scaling mechanism:

```
amount: input.amount.mulDivDown((MPS - scalingFactor), MPS),
```

```
amount: output.amount.mulDivUp((MPS + (priorityFee * output.mpsPerPriorityFeeWei)), MPS),
```

In both cases, we apply this scaling logic regardless of whether the result will be any different. Since at most only one of the input or output will be scaled for each order, at least one of them will be scaled redundantly with each order.

Recommendation: Short circuit the scaling logic in the case that the `scalingFactor` or `output.mpsPerPriorityFeeWei` is 0:

```
- amount: input.amount.mulDivDown((MPS - scalingFactor), MPS),
+ amount: scalingFactor == 0 ? input.amount : input.amount.mulDivDown((MPS - scalingFactor), MPS),
```

```
- amount: output.amount.mulDivUp((MPS + (priorityFee * output.mpsPerPriorityFeeWei)), MPS),
+ amount: output.mpsPerPriorityFeeWei == 0 ?
+                 output.amount :
+                 output.amount.mulDivUp((MPS + (priorityFee * output.mpsPerPriorityFeeWei)), MPS),
```

UniswapX: Fixed in PR 263.

Spearbit: Fixed as recommended.

5.2.3 Typehashes can be precomputed off-chain

Severity: Gas Optimization

Context: PriorityOrderLib.sol#L52-L98

Description: In `PriorityOrderLib`, we compute several EIP-712 typehashes on-chain during contract deployment, e.g.:

```
bytes32 internal constant PRIORITY_INPUT_TOKEN_TYPE_HASH = keccak256(PRIORITY_INPUT_TOKEN_TYPE);
```

As defined in the EIP-712 spec:

> The typeHash is a constant for a given struct type and does not need to be runtime computed.

As such, we can precompute these typehashes off-chain and simply set the constants as the result.

Recommendation: Precompute EIP-712 typehashes off-chain and set the result as the constant. Include the computation as a comment to easily validate the correctness. Additionally, add tests which compute the typehashes and assert that they're identical to the precomputed hashes.

UniswapX: Acknowledged. Appreciate the suggestion but I'm leaning more towards keeping it as is to get parity with the other libraries in the repo.

Spearbit: Acknowledged.

5.2.4 Small gas savings when using unchecked

Severity: Gas Optimization

Context: *(No context files were provided by the reviewer)*

Description: This function returns early when `scalingFactor >= MPS` meaning the language level checked math is no longer needed.

Recommendation: Wrap `MPS - scalingFactor` in an unchecked block.

UniswapX: Fixed in PR 264.

Spearbit: Fix confirmed.

5.3 Informational

5.3.1 Redundant self-reference in PriorityOrderLib constants

Severity: Informational

Context: PriorityOrderLib.sol#L73-L75, PriorityOrderLib.sol#L93-L96

Description: The `PriorityOrderLib` library unnecessarily self-references to access its own constants within the same contract. This is redundant as these constants can be accessed directly within the library.

Recommendation: Remove the `PriorityOrderLib.` prefix when accessing constants within the PriorityOrderLib. For example, change `PriorityOrderLib.PRIORITY_INPUT_TOKEN_TYPE` to simply `PRIORITY_INPUT_TOKEN_TYPE`.

UniswapX: Fixed in PR 261.

Spearbit: Fixed as recommended.

5.3.2 `PriorityOutput` array hash can be computed without inline assembly

Severity: Informational

Context: PriorityOrderLib.sol#L119-L133

Description: `PriorityOutput` struct array is hashed using inline assembly.

```
/// @notice returns the hash of an output token struct
function hash(PriorityOutput[] memory outputs) private pure returns (bytes32) {
    unchecked {
        bytes memory packedHashes = new bytes(32 * outputs.length);

        for (uint256 i = 0; i < outputs.length; i++) {
            bytes32 outputHash = hash(outputs[i]);
            assembly {
                mstore(add(add(packedHashes, 0x20), mul(i, 0x20)), outputHash)
            }
        }

        return keccak256(packedHashes);
    }
}
```

Although the implementation is accurate, there is no need to resort to inline assembly. Using `abi.encodePacked` on a `bytes32` array would generate the same `packedHashes` value.

Recommendation: Consider the alternative below:

```
bytes32[] memory hashes = new bytes32[](outputs.length);
for (uint256 i = 0; i < outputs.length; i++) hashes[i] = hash(outputs[i]);
return keccak256(abi.encodePacked(hashes));
```

UniswapX: Acknowledged. Agree with suggestion, but would prefer to keep the code the same as the other order types / reactors within the repo.

Spearbit: Acknowledged.

5.3.3 Improper use of compound assignment operation

Severity: Informational

Context: PriorityOrderReactor.sol#L134

Description: In `PriorityOrderReactor._checkPermit2Nonce()`, there is a compound assignment operation.

```
uint256 flipped = bitmap ^= bit;
```

This operation assigns the result of `bitmap ^ bit` to both `bitmap` and to `flipped` variables. However, `bitmap` is only an intermediate variable used for calculation purposes, and updating its value is unnecessary.

Recommendation: Consider replacing `^=` with `^`.

UniswapX: Fixed in PR 262.

Spearbit: Fixed as recommended.

5.3.4 Lack of scaling on both input and output allowed

Severity: Informational

Context: PriorityOrderReactor.sol#L101-L107

Description: In `PriorityOrderReactor._validateOrder`, we validate that the `mpsPerPriorityFeeWei` isn't set on both the input and the output, however, it doesn't enforce that the `mpsPerPriorityFeeWei` is non-zero for at least one of them. Since the unscaled input/output amounts represent maximum/minimum amounts if they are to be scaled, in the case that no scaling is occurring, this can result in an order being created without the MEV tax being applied, resulting in a loss for the user. Since a missing `mpsPerPriorityFeeWei` for both the input and output is at best just an inefficient limit order, we never want this to occur.

Recommendation: Revert if both the input and output have a `mpsPerPriorityFeeWei` of 0.

UniswapX: Acknowledged. Appreciate the suggestion - generally agree but I think we will guard against this offchain in our SDKs.

Spearbit: Acknowledged.

5.3.5 Changes to transaction ordering mechanisms can break expectations of MEV tax

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The MEV tax mechanism introduced in the code under audit works by scaling the input or output in the favor of the order maker according to the `priorityFee` of the transaction. This logic depends upon the expectation that transactions are ordered based on the `priorityFee`, which is currently true for OP stack chains. This expectation is important because collusion between the filler and block builder or sequencer can result in an order being filled with a `priorityFee` of 0 ahead of other fill transactions, resulting in the filler getting the lowest possible cost by not paying the MEV tax.

Assuming that OP stack chain sequencers are trusted, we don't risk this issue currently, however, there is a draft spec to move to a PBS-style mechanism. Similarly, some OP stack chains, notably Celo, have signalled intent to move to a classic leader election mechanism. In the case that any chains operating the `PriorityOrderReactor` move to either of these mechanisms, the MEV tax can be avoided at the expense of order makers.

Recommendation: Consider implementing a switch to disable usage of the `PriorityOrderReactor` in the case that one of the chains which it's deployed to changes their transaction ordering mechanism.

UniswapX: Acknowledged. Agree with suggestion, I think we will lean towards an off chain solution of deprecating use of the reactor if we detect that a chain has changed their ordering policy.

Spearbit: Acknowledged.