Uniswap Foundation: Kyber Hook Security Review

Auditors

Alireza Arjmand, Lead Security Researcher

R0bert, Lead Security Researcher

Akshay Srivastav, Security Researcher

0xluk3, Associate Security Researcher

Report prepared by: Lucas Goiriz

October 20, 2025

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

Kyber Network is a multi-chain crypto trading and liquidity hub that connects liquidity from different sources to enable trades at the best rates.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Uniswap Foundation: Kyber Hook 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 2 days in total, Kyber engaged with Spearbit to review the kyber-exclusive-amm-sc protocol. In this period of time a total of **8** issues were found.

Summary

Project Name	Kyber
Repository	kyber-exclusive-amm-sc
Commit	323bfbb2
Type of Project	DeFi, Hooks
Audit Timeline	Oct 1st to Oct 3rd

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	2	0	2
Low Risk	0	0	0
Gas Optimizations	0	0	0
Informational	6	0	6
Total	**8**	**0**	**8**

5 Findings

5.1 Medium Risk

5.1.1 Signed swap digest lacks a domain separator

Severity: Medium Risk

Context: PancakeSwapInfinityKEMHook.sol#L124-L135, UniswapV4KEMHook.sol#L118-L129

Description: Both `UniswapV4KEMHook` and `PancakeSwapInfinityKEMHook` rebuild a quote digest by hashing:

```
keccak256(
  abi.encode(
    sender,
    key,
    params.zeroForOne,
    maxAmountIn,
    maxExchangeRate,
    exchangeRateDenom,
    nonce,
    expiryTime
  )
);
```

The tuple ties the authorization to the router (`sender`), the full `PoolKey` (which includes the hook address), trade direction, price and input caps, nonce and expiry. Crucially, no domain separator is folded in: chain ID, deployment salt, and contract identity outside key are absent. If the same hook instance (or the same `PoolKey`) is deployed on multiple networks, as CREATE3-based salt mining allows, an attacker can lift any valid signature+nonce from chain A and replay it on chain B. Because the digest matches, `SignatureChecker.isValidSignatureNow` succeeds and the swap executes without the signer's intention. That breaks the core guarantee that signed quotes are single-instance authorizations, allowing cross-chain replay swaps.

Recommendation: Introduce domain separation for the signed payload in both hooks. Adopt an EIP712 domain that at minimum commits to chainid.

Kyber: Acknowledged.

- The quote has a very short expiry time and only affects the EG amount, not poolFee.
- To avoid the operational costs and LP migration burden of redeployment across chains, we will implement chain-specific operator signing keys as a mitigation measure.

Spearbit: Acknowledged.

5.1.2 Quotes can be frontrun by replaying them through the router

Severity: Medium Risk

Context: PancakeSwapInfinityKEMHook.sol#L124-L135, UniswapV4KEMHook.sol#L118-L129

Description: Both hooks accept swaps when `SignatureChecker` validates `keccak256(abi.encode(sender, key, ..., nonce, expiryTime))`. The `sender` field is the router contract that called the pool manager. That restricts execution to Kyber's router, but not to any particular user. Because the router is public, anyone can forward the calldata and signature. If Alice broadcasts a signed swap, an MEV bot can copy the calldata, submit it first and the hook sees the same router address and quote terms: the attacker's swap succeeds, consuming the nonce. Alice's transaction then reverts at `_useUnorderedNonce` because the nonce is already marked as used. The attacker, this way, can invalidate the quote with a dust swap. Designers intended router-level exclusivity, yet end users receive no front-running protection, exposing every quote to mempool sniping.

Recommendation: Include the router's original caller in the signature. The original caller can be retrieved by the hook by calling `router.msgSender()` function. See Uniswap V4 hook guide on accessing msg.sender securely.

Kyber: Acknowledged.

- The `UniswapV4KEMHook` protects makers (LPs).
- Takers/traders receive front-running protections at the Aggregator contract level, as this is an exclusive liquidity source. Since the signed `sender` is the `Aggregator` contract.

Spearbit: Acknowledged.

5.2 Informational

5.2.1 Trust assumptions

Severity: Informational

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

Description: For the contracts to operate correctly, the following assumptions must hold:

- The hook owner is responsible for maintaining the `claimable` mapping as well as keeping the `quoteSigner` and `egRecipient` addresses up to date.
- The contract relies on a signature check, where signatures are issued by a backend controlled by the Kyber team. The trust assumptions regarding this backend are:
  - It must not sign multiple swaps with the same nonce unless the previous one has expired.
  - It must ensure that `egAmount` does not exceed a threshold that would render a swap unprofitable for the trader.

Kyber: Acknowledged.

- This is an exclusive hook implementation that serves two key purposes: the EG mechanism & the exclusive logic while maintaining the same risk profile for Liquidity Providers (LPs) as the base Uniswap V4 pool.
- The taker/trader benefits from protections at the Aggregator contract level.
- The EG sharing mechanism operates on a trust basis.

Spearbit: Acknowledged.

5.2.2 Rescue function name does not reflect handling of native assets

Severity: Informational

Context: Rescuable.sol#L14-L29

Description: The `rescueERC20s` function also transfers the chain's native asset if `token == address(0)`. While correct, the current name does not clearly reflect that it rescues both ERC20 tokens and the native coin.

Recommendation: Rename the function to something more descriptive to reflect its ability to handle both ERC20s and native assets.

Kyber: Acknowledged. It remains acceptable at this stage since the functionality serves only as a rescue mechanism for an optional flow, and the hook is not designed to hold any Native or ERC20 tokens. Any Native or ERC20 in the hook contract is sent by mistake.

Spearbit: Acknowledged.

5.2.3 Zero nonce remains reusable until signature expiry

Severity: Informational

Context: UniswapV4KEMHook.sol#L116

Description: Both hooks call `_useUnorderedNonce(nonce)` in `beforeSwap`. The helper ignores and skips any check on the zero nonce:

```
function _useUnorderedNonce(uint256 nonce) internal {
  // ignore nonce 0 for flexibility
  if (nonce == 0) return;

  uint256 wordPos = nonce >> 8;
  uint256 bitPos = uint8(nonce);

  uint256 bit = 1 << bitPos;
  uint256 flipped = nonces[wordPos] ^= bit;
  if (flipped & bit == 0) revert NonceAlreadyUsed(nonce);

  emit UseNonce(nonce);
}
```

so any quote signed with `nonce = 0` is never recorded as used. An attacker who sees such a payload can reuse the same calldata through the router as many times as the signature's `expiryTime` allows, defeating the "single-use" guarantee that unordered nonces are meant to provide. Because the nonce is router-scoped, a single leaked 0-nonce signature effectively authorises every router caller/user until it expires.

Recommendation: Consider making zero nonces invalid. Either add `require(nonce != 0)` before calling `_useUnorderedNonce`, or change `_useUnorderedNonce` to revert on zero.

Kyber: Acknowledged.

- This is intentional to increase system flexibility. Setting nonce to non-zero prevents replay attacks, while setting it to zero saves gas, so it's a deliberate trade-off.
- The hook protects LPs only; taker protection occurs at the Aggregator contract level.

Spearbit: Acknowledged.

5.2.4 `updateClaimable` allows duplicate arguments

Severity: Informational

Context: BaseKEMHook.sol#L52

Description: In `BaseKEMHook.sol`, in functions `updateClaimable` and further `_updateClaimable`, there is no duplicate check on the input array which allows duplicated accounts to be supplied to the function.

```
function _updateClaimable(address[] memory accounts, bool newStatus) internal {
  for (uint256 i = 0; i < accounts.length; i++) {
    claimable[accounts[i]] = newStatus;

    emit UpdateClaimable(accounts[i], newStatus);
  }
}
```

If an account is provided more than once in the input array by mistake, especially with different `newStatus` values, this won't be noticed by the contract leading to potentially undesired final status of such account.

Recommendation: Consider implementing a duplicate check for the input arguments. However, this may come with slightly increased gas cost of execution for such safeguard.

Kyber: Acknowledged.

- This is an operational function.
- Each call applies the same `newStatus`, and the system tracks state through emitted events and the claimable function.

Spearbit: Acknowledged.

5.2.5 Overly restrictive `maxAmountIn` check against `amountSpecified`

Severity: Informational

Context: UniswapV4KEMHook.sol#L111-L114

Description: In `exactIn` mode, `amountSpecified` serves as an upper bound for `amountIn`, not its actual value. Enforcing a check of `maxAmountIn` directly against `amountSpecified` is unnecessarily restrictive and can cause valid swaps to revert.

Recommendation: Perform the `maxAmountIn` validation against the actual `amountIn` used in the swap, which is available in the `afterSwap` hook. This ensures that the limit is enforced precisely and does not reject valid transactions.

Kyber: Acknowledged.

- Since this only affects extreme cases and also requires additional effort in the simulation system, we'll maintain the current implementation.
- In the off-chain simulation system, it always assumes the full `amountSpecified` is used for the swap, then the `maxAmountIn` is calculated based on `amountSpecified`, thus, comparing with this value makes it more consistent between on-chain and off-chain calculations.

Spearbit: Acknowledged.

5.2.6 Unsafe typecast and unchecked arithmetic operations

Severity: Informational

Context: PancakeSwapInfinityKEMHook.sol#L156-L166, PancakeSwapInfinityKEMHook.sol#L178, UniswapV4KEMHook.sol#L150-L160, UniswapV4KEMHook.sol#L174

Description: The `afterswap` functions of `UniswapV4KEMHook` & `PancakeSwapInfinityKEMHook` contracts perform unsafe typecasting of `int256` to `int128` which can possibly result in overflow of `int128` values.

Also in the first `unchecked` block of `afterswap` functions, if the `amountIn` is equal to `type(int128).min` then its negation will overflow and equals to `1`, which will make `maxAmountOut` very small and will claim most of the `amountOut` for the hook.

Both these cases are unintended and can be prevented by better coding practices.

Recommendation:

1. Consider using `Safecast` library for all explicit type conversions.
2. Either remove the `unchecked` block or explicitly handle the overflow/underflow scenarios.

Kyber: Acknowledged.

- If `egAmount` returned by `afterSwap` is less than the amount minted from UniswapV4KEMHook.sol#L167-L169, the transaction will revert.
- About `amountIn`. This is an extreme case that affects the taker, but still, they have the protection at the Aggregator contract level.
- We can also note that hook doesn't support `amountIn = type(int128).min`.

Spearbit: Acknowledged.