OpenZeppelin | security

# Uniswap V4 Periphery and Universal Router Audit

Uniswap

September 5, 2024

# Table of Contents

# Summary

| | |
|---|---|
| Type | DeFi |
| Timeline | From 2024-08-05<br>To 2024-08-23 |
| Languages | Solidity |

| | |
|---|---|
| Total Issues | 38 (25 resolved, 1 partially resolved) |
| Critical Severity Issues | 1 (1 resolved) |
| High Severity Issues | 1 (1 resolved) |
| Medium Severity Issues | 2 (2 resolved) |
| Low Severity Issues | 13 (11 resolved, 1 partially resolved) |
| Notes & Additional Information | 20 (9 resolved) |
| Client Reported Issues | 1 (1 resolved) |

# Scope

We audited the Uniswap/v4-periphery repository at commit df47aa9. In scope were the following files:

```
src
├── PositionManager.sol
├── V4Router.sol
├── base
│   ├── BaseActionsRouter.sol
│   ├── DeltaResolver.sol
│   ├── EIP712_v4.sol
│   ├── ERC721Permit_v4.sol
│   ├── ImmutableState.sol
│   ├── Multicall_v4.sol
│   ├── Notifier.sol
│   ├── Permit2Forwarder.sol
│   ├── PoolInitializer.sol
│   ├── ReentrancyLock.sol
│   ├── SafeCallback.sol
│   ├── UnorderedNonce.sol
│   └── hooks
│       └── BaseHook.sol
├── interfaces
│   ├── IEIP712_v4.sol
│   ├── IERC721Permit_v4.sol
│   ├── IMulticall_v4.sol
│   ├── INotifier.sol
│   ├── IPositionManager.sol
│   ├── IQuoter.sol
│   ├── ISubscriber.sol
│   ├── IV4Router.sol
│   └── external
│       └── IERC20PermitAllowed.sol
├── lens
│   ├── Quoter.sol
│   └── StateView.sol
└── libraries
    ├── ActionConstants.sol
    ├── Actions.sol
    ├── BipsLibrary.sol
    ├── CalldataDecoder.sol
    ├── ERC721PermitHash.sol
    ├── Locker.sol
    ├── PathKey.sol
    ├── PoolTicksCounter.sol
    ├── PositionConfig.sol
    ├── SafeCast.sol
    └── SlippageCheck.sol
```

We also audited the Uniswap/universal-router repository at commit 4ce107d. In scope were the following files, with a focus on the part related to v4 integration:

```
contracts
├── UniversalRouter.sol
├── base
│   ├── Callbacks.sol
│   ├── Dispatcher.sol
│   └── Lock.sol
├── libraries
│   └── Locker.sol
└── modules
    ├── MigratorImmutables.sol
    ├── V3ToV4Migrator.sol
    └── uniswap
        └── v4
            └── V4SwapRouter.sol
```

In addition, after the audit, we analyzed the changes added to `Uniswap/v4-periphery` repository in the `audit/feat-reverse-mapping` branch. The changes were aimed at improving interface standardization for ERC-721 and making on-chain integrations easier.

# System Overview

The Uniswap V4 Periphery acts as an abstraction layer built on top of the Uniswap V4 core to simplify interactions with the core. The Universal Router, on the other hand, is an ERC-20 and NFT swap router that provides users with greater flexibility for executing trades across multiple token types.

## Uniswap V4 Periphery

The Uniswap V4 Periphery consists of two main contracts: `PositionManager` and `V4Router`. The `PositionManager` is responsible for managing liquidity positions, while the `V4Router` provides the logic for swapping tokens with the support of native currency. In addition, the `Quoter` contract is implemented to allow off-chain simulations of swaps.

The `PositionManager` contract implements a suite of actions related to minting, burning, and modifying liquidity positions in v4 pools, and it tracks users' positions as ERC-721 NFT tokens within itself. It supports the execution of various v4 core actions, including `modifyLiquidity` with slippage checks, `settle`, `take`, `clear`, and `sync`. Additionally, the `PositionManager` facilitates token payments through permits using the `permit2` protocol. Beyond interacting with `PoolManager`, the `PositionManager` can be integrated with third-party protocols built on top of its v4 positions through a novel subscriber notification functionality. Each position can have one subscriber contract at a time, which receives notifications when a position's status changes (e.g., updates to position liquidity or ownership transfers). The subscriber contract cannot modify or transfer a subscribed position and users can always remove the subscriber, even in the case of a malicious subscriber contract.

The `V4Router` contract is an abstract contract that cannot be deployed on its own and must be inherited to provide external logic. Its functionality is utilized by the Universal Router. To enable swaps in v4 liquidity pools, the `unlockCallback` function of the V4Router provides a plethora of swap-related actions. These include single swaps with specified exact input/output amounts, multi-hop swaps with defined paths and amounts, and various token transfer and payment methods. Slippage checks are implemented to protect users from potential adverse tick manipulation. During a multi-hop swap, slippage is checked against the minimum amount out for the final swap instead of enforcing slippage checks on every individual swap.

## Universal Router

The Uniswap Universal Router, originally utilized by Uniswap V3 for flexible trading across multiple token types, has been upgraded to incorporate Uniswap V4 functionality. This enhancement introduces new commands that facilitate the migration of positions from Uniswap V3 to Uniswap V4 and enable token swaps, including for the native currency, within the Uniswap V4 protocol.

# Security Model and Trust Assumptions

The audit was conducted concurrently with other security firms and the findings were shared among the companies as they emerged. One such finding was discovered by a third-party security company during this audit and is reported under the "Client Reported Issues" section.

The following observations were made regarding the security model and trust assumptions of the audited codebase:

- Both the V4 Position Manager and Universal Router are non-upgradeable, permissionless, and ownerless contracts. The permission to spend users' tokens can be granted to these contracts via the `permit2` protocol. Users are advised to exercise caution when transferring tokens to either contract as any token balance left in the contracts can be removed immediately by any account.

- The Universal Router now interacts directly with both the V3 and V4 Position Managers. Users should exercise **extra caution** when granting approval for their positions in the V3 Position Manager during the migration process and should **never** grant approval for any V4 positions to the Universal Router. Otherwise, anyone could potentially remove the approved position's liquidity via the Universal Router.

- The V4 Position Manager notifies third-party contracts of any changes in position status through external calls. It is assumed that the owner or an approved account can unsubscribe the external contract at any time. To mitigate DoS attacks, only a portion of the gas is forwarded, up to a maximum of 1% of the block gas limit. However, this approach may still lead to a DoS condition, as the responsibility for removing the subscriber from the purchased or received position falls on the receiver, which could be costly depending on the blockchain.

# Critical Severity

## C-01 Accrued Fees Can Be Stolen

The `_increase` function of the `PositionManager` contract does not verify whether the caller is the owner of the specified `tokenId` or one of its approved accounts. As a result, anyone can increase the liquidity of any position. This presents a security risk as the returned `liquidityDelta` from the `PoolManager` includes the accrued fees of that position, allowing unauthorized users to collect the fees by increasing the liquidity by 0.

The following scenario demonstrates how the vulnerability could be exploited:

1. Joe accrues fees on a position with `tokenId` set to 10.
2. Eve triggers the `_increase` function with the liquidity set to 0 for `tokenId` 10.
3. The `PositionManager` calls `modifyLiquidity` on the `PoolManager`, resulting in a positive delta due to the fees accrued from the position.
4. Eve collects the positive delta from the `PoolManager` using the `take` function.

Consider restricting the ability to increase a position's liquidity to its owner or an approved account only.

***Update:*** *Resolved in pull request #290. The team stated:*

> *Fixed by adding `onlyIfApproved` to the `_increase()` handler function.*

# High Severity

## H-01 Slippage Checks Are Not Enforced When Fees Accrued Exceed Tokens Required for a Liquidity Deposit

When increasing the liquidity of a position, the `validateMaxInNegative` function in `SlippageCheck.sol` does not check slippage when `balanceDelta` is positive. `balanceDelta` is a combination of the tokens required to modify a liquidity position and the fees accrued to a liquidity position. When the fees accrued exceed the tokens required to increase the liquidity of a position, `balanceDelta` can be positive.

However, slippage checks should still be enforced on positive balance deltas. Any liquidity increase loses value as long as the current tick of the pool deviates from the correct price tick. Normally, slippage checks would prevent the tick from being manipulated by a frontrunner as manipulating the pool tick always increases the `amountIn` of one of the tokens. Yet, when the fees accrued exceed the tokens required for the liquidity deposit, it is impossible to set a slippage parameter when the price tick is manipulated adversely.

From an attacker's perspective, the fact that the liquidity deposit comes from fees makes no difference in a liquidity deposit sandwich attack. The liquidity depositor still loses value as their fees are converted to a liquidity position which is worth less than the pre-deposit value.

Consider checking the slippage on the amount of tokens required to modify a liquidity position without including the accrued fees.

***Update:*** *Resolved in pull request #285.*

# Medium Severity

## M-01 Universal Router Does Not Forward Native Tokens to V4 Position Manager

The Universal Router has implemented logic to facilitate the migration of positions from the Uniswap v3 Position Manager to the v4 Position Manager. However, the v4 position call does not forward native tokens despite the v4-core supporting native token pools. This will not allow any modification of liquidity positions that require native tokens in v4 pools through the router. In the context of migration, any wrapped native tokens from v3 pools may not be able to migrate directly to native tokens in v4 pools via the Universal Router.

Consider forwarding native tokens if the Universal Router is intended to support the aforementioned migration path.

***Update:*** *Resolved in pull request #379.*

## M-02 Slippage Check Can Be Bypassed With Unsafe Cast

When removing liquidity from the Position Manager, the `validateMinOut` function enforces slippage protection and ensures that users get at least the specified minimum amounts out. Even though it is likely that `liquidityDelta` will be positive for both tokens, given the introduction of delta-changing hooks, it is possible that the returned `liquidityDelta` is negative for one or both tokens. This scenario could happen, for example, when a custom hook penalizes liquidity removal and ends up requiring users to pay to withdraw.

In the `validateMinOut` function, the returned amount is cast to `uint128` before being compared to the user-specified values. When a negative `int128` value is cast to `uint128`, the returned amount may bypass the slippage check.

Consider modifying the core contracts to return values that exclude the delta caused by the hook and checking slippage against those values. Alternatively, consider allowing `int128` inputs for `amount0Min` and `amount1Min`. If safe cast is being used for the returned delta, consider ensuring that there are clear instructions for users to remove liquidity when their returned delta is negative.

***Update:*** *Resolved in pull request #285.*

# Low Severity

## L-01 ERC-721 Permit Signatures Cannot Be Revoked

The current implementation of `ERC721Permit_v4` does not include any functionality for invalidating existing signatures before the deadline. With the use of unordered nonce, a signed permit cannot be made invalid by increasing the nonce.

Consider adding an invalidation mechanism allowing a signed permit to be revoked.

***Update:*** *Resolved in pull request #304.*

## L-02 Incorrect Decoding of `bytes[] params` Calldata Length

When decoding the calldata for `bytes actions` and their respective parameters `bytes[] params`, the `decodeActionsRouterParams` function reads the lengths and offsets from the calldata. It then performs a check to ensure that the calldata in its entirety is not longer than the bytes that encode `actions` and `params` together. However, when performing this comparison, the calldata length of `bytes[] params` is decoded incorrectly to `params.length`, which is the array length of `bytes[] params`. This results in the `SliceOutOfBounds()` error being unreachable for the intended situation.

Consider counting each element of the `bytes[] params` array separately to estimate the total calldata length.

***Update:*** *Resolved in pull request #316.*

## L-03 Shadowed State Variable

The `subscriber` state variable is being shadowed in the inheriting contract's subscribe function. Although this does not cause any issues at this stage, it is a good practice to rename local variables that shadow any state variable to avoid any potential future mistakes and improve code clarity.

Consider renaming any local variables to prevent instances of shadowing.

***Update:*** *Resolved in pull request #287. The team stated:*

> *Fixed when we moved `subscribe()/unsubscribe()` to `Notifier.sol`*

## L-04 Checks-Effects-Interactions Pattern Not Followed for Unsubscribe Logic

The unsubscribe logic first makes an external call to the subscriber and then deletes the subscriber for the given token ID. This approach may lead to issues depending on the external integration, as the subscriber receiving the call to `notifyUnsubscribe` can still interact with other contracts while still being considered a valid subscriber for the specified `tokenId`.

Consider first deleting the subscriber for the specified `tokenId` and then executing the external call.

***Update:*** *Resolved in pull request #303.*

## L-05 Unreachable Branch in `_mapInputAmount`

When swapping via the `_swapExactInputSingle` function, one can specify the `amountIn` to be the contract balance via the `_mapInputAmount` function. However, the `_mapInputAmount` function takes a `uint128` value, while `ActionConstants.CONTRACT_BALANCE` is set to `1<<255`. This discrepancy makes it impossible to trigger the first `if` branch, preventing the direct use of the contract balance as an input amount.

Although the above scenario can be circumvented by first settling the amount into the open delta before swapping, nonetheless, consider setting `CONTRACT_BALANCE` to a value which is reachable with a `uint128` input.

***Update:*** *Resolved in pull request #286. The branch was removed as it was not being used anyway*

## L-06 Permit2 Signatures Could Be Front-Run to Temporarily Prevent Execution

The `Permit2Forwarder` contract allows setting the contract as a spender in Permit2. It implements two functions, `permit` and `permitBatch`, which are designed to be used within a multicall by the docstring. An attacker can front-run the multicall transaction that includes the `permit` or `permitBatch` calls and use the provided signature to approve the spender. In that case, the legitimate transaction will revert as it tries to use a signature that has already been used.

Consider wrapping the calls to Permit2 inside `try-catch` blocks to ensure that even if a call to Permit2 reverts, the multicall continues with the rest of the calls.

***Update:*** *Resolved in pull request #309.*

## L-07 Burning a Position Notifies the Subscriber But Does Not Unsubscribe Them

The burn operation on a position triggers the `_modifyLiquidity` logic, which, if a subscriber is set, will notify the subscriber via a call to `notifyModifyLiquidity`. The issue is that burn withdraws the entire amount of liquidity and burns the `tokenId`. However, the position still retains the subscriber. This could lead to issues depending on the external protocol integrations, as the external protocol might only be aware of the liquidity modification to 0 and not of the burning of the position.

Consider calling `unsubscribe` when a liquidity position is burned.

***Update:*** *Resolved in pull request #314, pull request #333.*

## L-08 Misleading Documentation

Throughout the codebase, multiple instances of misleading documentation were identified:

- The docstring of the decodeActionsRouterParams function says that it is equivalent to `abi.decode(params, (uint256[], bytes[]))`. Instead, it should be `abi.decode(params, (bytes, bytes[]))`.
- The docstring of the decodeSwapExactOutSingleParams function says that it is equivalent to `abi.decode(params, (IV4Router.ExactOutputSingleParams))`. Instead, it should be `abi.decode(params, (IV4Router.ExactInputSingleParams))`
- The docstring says that when specifiedAmount > 0, the swap is `exactInput`. Instead, it should be `exactOutput`.
- The docstring says that the output amount is cached for comparison in the swap callback. Since there are no callbacks from the v4-core swap function, it should be compared inside the `_swap` function.
- The `MINIMUM_VALID_RESPONSE_LENGTH` is set to 96 bytes and the docstring explains how these can be packed. However, when encoding a dynamic array of length 2, it will have extra slots reserved for array offset and array length, and the offset for each element in the array. Hence, the minimum valid response length with a length 2 array would be 192 bytes instead.

***Update:*** *Resolved in pull request #313.*

## L-09 Unsafe ABI Encoding

It is not an uncommon practice to use `abi.encodeWithSignature` or `abi.encodeWithSelector` to generate calldata for a low-level call. However, the first option is not typo-safe and the second option is not type-safe. The result is that both of these methods are error-prone and should be considered unsafe.

Within `Quoter.sol`, multiple uses of unsafe ABI encodings were identified:

- The use of `abi.encodeWithSelector` for the `_quoteExactInputSingle` function.
- The use of `abi.encodeWithSelector` for the `_quoteExactInput` function.
- The use of `abi.encodeWithSelector` for the `_quoteExactOutputSingle` function.
- The use of `abi.encodeWithSelector` for the `_quoteExactOutput` function.

Within `Dispatcher.sol`, the `abi.encodeWithSelector` is used to encode the call to `execute` function.

Consider replacing all the occurrences of unsafe ABI encodings with `abi.encodeCall` which checks whether the supplied values actually match the types expected by the called function and also avoids errors caused by typos.

***Update:*** *Resolved in pull request #308, pull request #382.*

## L-10 `try-catch` Block Can Still Revert

In the `_unsubscribe` function of `Notifier.sol`, the external call to the subscriber contract is wrapped in a `try-catch` block so that a user can always unsubscribe safely even with a malicious subscriber. However, in Solidity, `try-catch` blocks can still revert when the call is to an address with no contract code.

For instance, consider a chain where `selfdestruct` has not been deprecated. A malicious contract could accept subscriptions and implement the `ISubscriber` interface and also have a `selfdestruct` function. After it has enough subscribers, the admin can `selfdestruct` the contract. Having said that, we do note that `selfdestruct` has been deprecated since EIP-6780 and this would not be possible on most Ethereum-compatible chains.

After the contract is destroyed, when users try to modify liquidity, the call to `_notifyModifyLiquidity` will revert even though there is a `try-catch` block. This is because calling an address without contact code even within a `try-catch` block will revert. Hence, it is impossible to unsubscribe because `try _subscriber.notifyUnsubscribe` reverts. Therefore, the liquidity positions of all the subscribers would be permanently locked as they will not be able to modify liquidity or unsubscribe.

Consider wrapping the call to `notifyUnsubscribe` in another `try-catch` block which would successfully catch reverts caused by calling addresses without contract code.

***Update:*** *Resolved in pull request #318.*

## L-11 Discrepancies Between Interfaces and Implementation Contracts

Throughout the codebase, multiple discrepancies between interfaces and implementation contracts were identified:

- The `INotifier` interface is missing a declaration for the getter function for the `subscriber` mapping, which is declared as `public` within the `Notifier` contract.
- The `INotifier` interface's functions are implemented within `PositionManager`, while `Notifier` only contains the internal functions.
- In `IPositionManager`, the `modifyLiquidities` function uses the parameter name `payload`, while the implementation uses `unlockData`.
- In `IPositionManager`, the `getPositionConfigId` function uses `configId` as the return parameter while the implementation does not.
- In `IQuoter`, the `quoteExactInputSingle` function uses the `calldata` keyword, while the implementation uses `memory`.
- In `IQuoter`, the `quoteExactOutputSingle` function uses `calldata`, while the implementation uses `memory`.

Consider aligning parameter names and storage location keywords between interfaces and their implementation contracts to ensure consistency and reduce potential errors.

***Update:*** *Resolved in pull request #311.*

## L-12 Missing Zero-Address Checks

When assigning an address from a user-provided parameter, it is crucial to ensure that the address is not set to zero. Accidentally setting a variable to the zero address might end up incorrectly configuring the protocol.

Throughout the codebase, multiple instances of assignments being performed without zero-address checks were identified:

- The `poolManager` address in `ImmutableState` contract
- The `permit2` address in `Permit2Forwarder`
- The `params.v3PositionManager` address in `MigratorImmutables`
- The `params.v4PositionManager` address in `MigratorImmutables`

Consider adding a zero address check before assigning a state variable.

***Update:*** *Acknowledged, not resolved.*

## L-13 Missing Docstrings

Throughout the codebase, multiple instances of missing docstrings were identified:

- The `ActionConstants` library in `ActionConstants.sol`
- The entire `BaseHook` abstract contract in `BaseHook.sol`
- The entire `ERC721PermitHashLibrary` library in `ERC721PermitHash.sol`
- The entire `IEIP712_v4` interface in `IEIP712_v4.sol`
- The `IPositionManager` interface in `IPositionManager.sol`
- The entire `ImmutableState` contract in `ImmutableState.sol`
- The entire `Notifier` contract in `Notifier.sol`
- The `PathKeyLib` library and `PathKey struct` in `PathKey.sol`
- The `permit2` state variable in `Permit2Forwarder.sol`
- The entire `PoolInitializer` abstract contract in `PoolInitializer.sol`
- The `PoolTicksCounter` library in `PoolTicksCounter.sol`
- The `PositionManager` contract and `msgSender` function in `PositionManager.sol`
- The `Quoter` contract in `Quoter.sol`
- The `SafeCallback` abstract contract in `SafeCallback.sol`
- The `supportsInterface` function in `Callbacks.sol`
- The `Lock` contract in `Lock.sol`
- The `MigratorImmutables` contract and `MigratorParameters` struct in `MigratorImmutables.sol`
- The `UniversalRouter` contract in `UniversalRouter.sol`
- The entire `V3ToV4Migrator` abstract contract in `V3ToV4Migrator.sol`

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

***Update:*** *Partially resolved in pull request #335.*

# Notes & Additional Information

## N-01 Unimplemented Actions

The following three actions from the Actions.sol library have not been implemented in either `PositionManager.sol` or `V4Router.sol`:

- `DONATE`
- `MINT_6909`
- `BURN_6909`

Consider implementing the above actions with caution. Alternatively, consider removing them from the `Actions.sol` library.

***Update:*** *Acknowledged, not resolved.*

## N-02 Negation of `int256` May Overflow

Negating `type(int256).min` will overflow the `int256` type. The following instances were identified where the execution could revert in such cases.

- The evaluation of a negative `int256` in line 59 of `DeltaResolver.sol`
- The evaluation of a negative `int256` in line 303 of `PositionManager.sol`

It may be unlikely to have the minimum value of `int256` being passed as an argument to the aforementioned functions. Nonetheless, to avoid any unexpected behavior, consider ensuring that the affected elements will not contain the minimum value.

***Update:*** *Acknowledged, not resolved.*

## N-03 Duplicate Imports

Throughout the codebase, multiple instances of duplicate imports were identified:

- `PositionConfig` is imported from both `"../libraries/PositionConfig.sol"` and `"../interfaces/INotifier.sol"` in `Notifier.sol`.
- `Position` is imported twice from `"@uniswap/v4-core/src/libraries/Position.sol"` on lines 10 and 13 of `PositionManager.sol`.

Consider removing duplicate imports to improve the overall clarity and readability of the codebase.

***Update:*** *Resolved in pull request #277, pull request #287.*

## N-04 Unnecessary Cast

Within the `Notifier` contract, the `newSubscriber` parameter is already an `address`, making the cast to `address` unnecessary.

To improve the overall clarity and readability of the codebase, consider removing any unnecessary casts.

***Update:*** *Resolved in pull request #319.*

## N-05 File and Library Names Mismatch

Throughout the codebase, multiple instances of contract and file name mismatch were identified:

- The `ERC721PermitHash.sol` file name does not match the `ERC721PermitHashLibrary` contract name.
- The `PathKey.sol` file name does not match the `PathKeyLib` contract name.
- The `PositionConfig.sol` file name does not match the `PositionConfigLibrary` contract name.
- The `SafeCast.sol` file name does not match the `SafeCastTemp` contract name.
- The `SlippageCheck.sol` file name does not match the `SlippageCheckLibrary` contract name.

To make the codebase easier to understand for developers and reviewers, consider renaming the aforementioned files to match the contract names.

***Update:*** *Resolved in [pull request #335](#).*

## N-06 Unnecessary Unchecked Block

Starting from Solidity version 0.8.22, the compiler automatically optimizes arithmetic increments in `for` loops by removing overflow checks. As such, within `UniversalRouter.sol`, the [`unchecked` block](#) in the `execute` function is unnecessary.

To improve the overall clarity, intent, and readability of the codebase, consider removing any unnecessary `unchecked` blocks.

***Update:*** *Resolved in [pull request #383](#).*

## N-07 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Consider adding a NatSpec comment containing a security contact on each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the [OpenZeppelin Wizard](#) and the [ethereum-lists](#).

***Update:*** *Acknowledged, not resolved.*

## N-08 Unused Imports

Throughout the codebase, multiple instances of unused imports were identified:

- The [`Actions`](#) library is imported within `BaseActionsRouter.sol` but never used.

- The `CustomRevert` library is assigned to `bytes4` but never used within the `Notifier.sol` contract.

Consider removing unused imports to improve the overall clarity and readability of the codebase.

***Update:*** *Resolved in pull request #306.*

## N-09 Incomplete Docstrings

Throughout the codebase, multiple instances of incomplete docstrings were identified:

- The `nonce` parameter is not documented in the `permit` and `permitForAll` functions of `IERC721Permit_v4.sol`.
- The `owner`, `permitSingle`, and `signature` parameters are not documented in the `permit` and `permitBatch` functions of `Permit2Forwarder.sol`.
- The `params` parameter is not documented in the `_quoteExactInput`, `_quoteExactInputSingle`, `_quoteExactOutput`, and `_quoteExactOutputSingle` functions of `Quoter.sol`.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

***Update:*** *Acknowledged, not resolved.*

## N-10 Todo Comments in the Code

During development, having well-described TODO/Fixme comments will make the process of tracking and solving them easier. Without this information, these comments might age and important information for the security of the system might be forgotten by the time it is released to production. These comments should be tracked in the project's issue backlog and resolved before the system is deployed.

Throughout the codebase, multiple instances of TODO/Fixme comments were identified:

- line 5 of `BipsLibrary.sol`.
- line 104 of `ERC721Permit_v4.sol`.
- line 5 of `Locker.sol`.
- line 11 of `Notifier.sol`.

- line 353 of `PositionManager.sol`.
- line 8 of `SafeCast.sol`.
- line 6 of `Locker.sol`.

Consider removing all instances of TODO/Fixme comments and instead tracking them in the issues backlog. Alternatively, consider linking each inline TODO/Fixme to the corresponding issues backlog entry.

***Update:*** *Acknowledged, not resolved.*

## N-11 Inconsistent Function Order Within Contracts

Throughout the codebase, multiple instances of contracts deviating from the Solidity Style Guide due to having an inconsistent ordering of functions were identified:

- The `BaseActionsRouter` contract in `BaseActionsRouter.sol`
- The `BaseHook` contract in `BaseHook.sol`
- The `CalldataDecoder` library in `CalldataDecoder.sol`
- The `EIP712_v4` contract in `EIP712_v4.sol`
- The `ERC721Permit_v4` contract in `ERC721Permit_v4.sol`
- The `IQuoter` interface in `IQuoter.sol`
- The `IV4Router` interface in `IV4Router.sol`
- The `Notifier` contract in `Notifier.sol`
- The `PositionManager` contract in `PositionManager.sol`
- The `Quoter` contract in `Quoter.sol`.
- The `SafeCallback` contract in `SafeCallback.sol`
- The `UnorderedNonce` contract in `UnorderedNonce.sol`
- The `Dispatcher` contract in `Dispatcher.sol`
- The `UniversalRouter` contract in `UniversalRouter.sol`

To improve the project's overall legibility, consider standardizing the ordering throughout the codebase as recommended by the Solidity Style Guide (Order of Functions).

***Update:*** *Acknowledged, not resolved.*

## N-12 Lack of Indexed Event Parameters

Within `Notifier.sol`, multiple instances of events without indexed parameters were identified:

- The `Subscribed` event
- The `Unsubscribed` event

To improve the ability of off-chain services to search and filter for specific events, consider indexing event parameters.

***Update:** Resolved at commit fce887f.*

## N-13 Unused Struct

The `PoolDeltas` struct in `IQuoter.sol` is unused.

To improve the overall clarity, intentionality, and readability of the codebase, consider either using or removing any currently unused structs.

***Update:** Acknowledged, not resolved. The team stated:*

> *For various reasons we are now going to be re-architecting our Quoter contract. As this issue is related to the quoter, we will not be fixing it. However we will keep it in mind when writing our new Quoter contract.*

## N-14 Unnecessary Return Value

The `return` values of the following functions are never used as the revert message contains all the required information.

- The `_quoteExactInput` function in `Quoter.sol`
- The `_quoteExactInputSingle` function in `Quoter.sol`
- The `_quoteExactOutput` function in `Quoter.sol`
- The `_quoteExactOutputSingle` function in `Quoter.sol`

Consider removing unnecessary return values for improved code clarity.

***Update:** Acknowledged, not resolved. The team stated:*

> *For various reasons we are now going to be re-architecting our Quoter contract. As this issue is related to the quoter, we will not be fixing it. However we will keep it in mind when writing our new Quoter contract.*

## N-15 Inconsistent Use of Multiple Solidity Versions

Throughout the codebase, Solidity versions are being used inconsistently. As such, the following issues were identified:

- Multiple contracts use floating pragmas. It is recommended to use fixed pragma directives to ensure consistency.

- Pragma statements that span several Solidity versions can lead to unpredictable behavior due to differences in features, bug fixes, deprecation, and compatibility between versions. For instance:

- The pragma statement in line 2 of `IERC20PermitAllowed.sol`

- The pragma statement in line 2 of `IERC721Permit_v4.sol`
- The pragma statement in line 2 of `PoolTicksCounter.sol`

Consider taking advantage of the latest Solidity version to improve the overall readability and security of the codebase. Regardless of which version of Solidity is used, consider pinning the version consistently throughout the codebase to prevent bugs due to incompatible future releases.

***Update:*** *Resolved in pull request #332.*

## N-16 Typographical Errors

Throughout the codebase, multiple instances of typographical errors were identified

- This TODO comment uses `addresss` whereas it should be `address`.
- The comment in the `decodeActionsRouterParams` function uses `isnt` whereas it should be `isn't`.
- In the comment describing the use of the `msgSender` function, `shouldnt` should be `shouldn't`.
- In the NatSpec comment for the `InvalidEthSender` error, `isnt` should be `isn't`.

Consider fixing the aforementioned typographical errors to improve the readability and clarity of the codebase.

***Update:*** *Resolved in pull request #322 and pull request #388.*

## N-17 Function Visibility Overly Permissive

Throughout the codebase, multiple instances of functions with unnecessarily permissive visibility were identified:

- The `_settlePair`, `_takePair`, `_clearOrTake`, `_sweep`, and `_modifyLiquidity` functions in `PositionManager.sol` with `internal` visibility could be limited to `private`.
- The `multicall` function in `Multicall_v4.sol` with `public` visibility could be limited to `external`.
- The `quoteExactInputSingle`, `quoteExactOutputSingle`, `quoteExactOutput`, `_quoteExactInput`, `_quoteExactInputSingle`, `_quoteExactOutput`, and `_quoteExactOutputSingle` functions in `Quoter.sol` with `public` visibility could be limited to `external`.

To better convey the intended use of functions and to potentially realize some additional gas savings, consider changing a function's visibility to be only as permissive as required.

***Update:*** *Acknowledged, not resolved.*

## N-18 Using Directive Could Be Global

Throughout the codebase, multiple instances of the same library being used locally were identified:

- The use of `PathKeyLib` for `PathKey` in `Quoter.sol`.
- The use of `PathKeyLib` for `PathKey` in `V4Router.sol`.

Consider `using` the `PathKeyLib` library for the `PathKey` struct globally within the `PathKey.sol` file.

***Update:*** *Acknowledged, not resolved.*

## N-19 ERC-721 Permit Could Be Front-Run To Temporarily Prevent Execution

The `PositionManager` implements ERC-721 permit logic that allows the creation of permits, enabling a spender to transfer the owner's positions. The issue lies in the fact that the `permit` and `permitForAll` functions are expected to be used within a `multicall`. An attacker could monitor the mempool and front-run the multicall transaction that includes the permit calls, using the provided signature to approve the spender. In such a case, the legitimate transaction would revert because it attempts to use a signature that has already been used.

Consider adding functionality to the `multicall` contract to specify whether a given call should be allowed to fail. This would allow ERC-721 permit calls within the multicall to be marked accordingly, enabling the execution to proceed.

***Update:*** *Acknowledged, not resolved.*

## N-20 Gas Optimizations

Throughout the codebase, multiple instances of potential gas optimizations were identified:

- Several functions could benefit from using the pre-fix increment and decrement operators instead of the post-fix increment and decrement operators. The pre-fix operators are more gas-efficient as they do not store the original value before performing the operation. The following functions could benefit from this optimization:

- `_executeActionsWithoutUnlock` in `BaseActionsRouter`

- `multicall` in `Multicall_v4`
- `_quoteExactInput` and `_quoteExactOutput` in `Quoter`

- `_swapExactInput` and `_swapExactOutput` in `V4Router`

- In `external` functions, it is more gas-efficient to use `calldata` instead of `memory` for function parameters. The `calldata` is a read-only region of memory that contains the function arguments, making it cheaper and more efficient than memory. For example, in `Quoter.sol`, the `params` parameter should use `calldata` instead of `memory`.

For improved execution efficiency and gas savings, consider applying the aforementioned optimizations throughout the codebase. In addition, ensure that the test suite passes after making these changes.

***Update:*** *Resolved. The team stated:*

> For Quoter-related points, we will not be making these changes as we are rearchitecting the Quoter contract. We made the remaining changes, and they caused gas to increase not decrease, we therefore will not be making these changes.

# Client Reported

## CR-01 Universal Router Does Not Accept Native Tokens

The `receive` function of the universal router does not accept native tokens from the v4 Pool Manager. This limits potential mixed-protocol trades using the native token as an intermediate currency. For example, to trade `USDC` to `UNI`, one could:

1. trade `USDC` for `ETH` on v4,
2. pull `ETH` into universal router,
3. wrap `ETH`, or
4. trade `WETH` for `UNI` on v2 pools.

Consider accepting native tokens from the v4 Pool Manager.

***Update:*** *Resolved in pull request #376.*

## Conclusion

Uniswap V4 has undergone significant design changes, including the adoption of a monolithic architecture, settlement with flash accounting, and the use of hooks. To ensure secure and efficient interaction with the new V4 liquidity pools, the V4 periphery and router contracts have been specifically developed with these advanced features in mind. The Uniswap V4 Position Manager allows users to manage their liquidity as non-fungible tokens, while the Universal Router facilitates V4 swaps with slippage checks and supports the migration of positions from the V3 to the V4 Position Manager.

The audit identified one critical-, one high-, and two medium-severity vulnerabilities, along with several lower-severity issues. In addition, recommendations aimed at enhancing code clarity and optimization were also made. The Uniswap team was highly cooperative and provided clear explanations throughout the audit process.