Uniswap V4 Swap Router Audit Report

Prepared by KupiaSec

Version 1.1

Auditors

KupiaSec

Mar 19th, 2025

Contents

1 About KupiaSec

KupiaSec is a team of Web3 security experts that operates with transparency and a meritocratic spirit.

KupiaSec executes the modified **MPA** model for the Private Audits, a.k.a. **Solo Audit by a Lead + Internal Competition + Mitigation Review**.

- Solo Audit by a Lead

  KupiaSec assigns a senior auditor as a Lead Auditor based on the protocol category. The Lead Auditor is responsible for the first phase and will be the main point of contact for the client. The Lead Auditor shares the analysis and findings with the team.

- Internal Competition

  KupiaSec assigns 5~7 assist auditors to conduct the second phase. The auditors compete to find the most issues and the best solutions. This phase ensures the protocol goes through a rigorous review process by "many eyes" in a competitive environment.

- Mitigation Review

  After the protocol team has fixed the issues, KupiaSec conducts a final review to ensure all the issues are resolved.

2 Disclaimer

The KupiaSec team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

A simple and optimized router for swapping on Uniswap V4.

5 Audit Scope

Solidity source files in `v4-router/src` were in the audit scope for all commit hashes.

Summary of Commits

Project Name	Uniswap V4 Swap Router
Repository	v4-router
Initial Commit	9706c70fbbd4...
Mitigation Commit	7840b40266a8...

6 Executive Summary

KupiaSec executed a modified Multi-Phase Audit model, a.k.a. **Solo Audit by a Lead + Internal Competition + Mitigation Review**.

Auditor conducted the audit as the Lead Auditor and 5 auditors competed in the second phase.

Execution Timeline

Phase-1: Audit by a Lead	Feb 26th, 2025 - Feb 27th, 2025
Phase-2: Internal Competition	Feb 27th, 2025 - Mar 1st, 2025
Initial Report Delivery	Mar 3rd, 2025
Phase-3: Mitigation Review	Mar 18th, 2025 - Mar 19th, 2025
Final Report Delivery	Mar 19th, 2025

Issues Found

Critical Risk	0
High Risk	1
Medium Risk	2
Low Risk	1
Informational	4
Gas Optimizations	0
Total Issues	8

Summary of Findings

[H-1] Incorrect Comparison When Determining `outputAmount` in the `_unlockCallback()` Function	Fixed
[M-1] Too Tight Slippage Check	Fixed
[M-2] Arbitrage Through Path Swap Is Not Possible.	Fixed
[L-1] Lack of Check for `amountIn = msg.value` When Swapping `ETH` for Tokens	Acknowledged

[I-1] Improper Return Value When Path Swap	Fixed
[I-2] The `swapExactTokensForTokens()` Function Should Return an Array of `BalanceDelta` Instead of a Single Value	Acknowledged
[I-3] Inconsistency in Data Location of Function Parameters Between `UniswapV4Router04` and Its Interface	Fixed
[I-4] Missing `sync` Call When Settling Native Currency	Fixed

7 Findings

7.1 High

7.1.1 Incorrect Comparison When Determining `outputAmount` in the `_unlockCallback()` Function

Description: As observed at line 92 of `BaseSwapRouter`, `outputAmount` is determined based on the size relationship between `inputCurrency` and `outputCurrency`. However, this comparison is unreasonable.

`delta` represents the final swap result. The final swap occurs between the last two currencies in the path: `path[path.length - 2].intermediateCurrency` and `path[path.length - 1].intermediateCurrency` (the `outputCurrency`). Consequently, `delta.amount0()` and `delta.amount1()` represent the amounts of these two currencies. We should determine `outputAmount` based on the size relationship between `path[path.length - 2].intermediateCurrency` and `outputCurrency`.

Currently, the implementation determines `outputAmount` based on the size relationship between `inputCurrency` and `outputCurrency`, which could yield incorrect results and lead to the reversal of the transaction.

Consider the Following Scenario:

1. inputCurrency = A
2. `path: B, C (outputCurrency)`
3. Relationship: `A < C < B`

In this case, the swap path is `(A -> B), (B -> C)`, and `delta` represents the final swap result between `B` and `C`.

Since `B > C`, `delta.amount1()` is the amount of `B`, and `delta.amount0()` is the amount of `C`. Therefore, the correct `outputAmount` should be `delta.amount0()`. However, in the current implementation, since `inputCurrency < outputCurrency` (`A < C`), `outputAmount` is incorrectly determined as `delta.amount1()`, which is the amount of `B`, not of `outputCurrency`.

```
    function _unlockCallback(bytes calldata callbackData)
        internal
        virtual
        override(SafeCallback)
        returns (bytes memory)
    {
        unchecked {
            BaseData memory data = abi.decode(callbackData, (BaseData));

            (bool singleSwap, bool exactOutput, bool input6909, bool output6909, bool _permit2) =
                SwapFlags.unpackFlags(data.flags);

            (Currency inputCurrency, Currency outputCurrency, BalanceDelta delta) =
                _parseAndSwap(singleSwap, exactOutput, data.amount, _permit2, callbackData);

            uint256 inputAmount = uint256(-poolManager.currencyDelta(address(this), inputCurrency));
            uint256 outputAmount = exactOutput
                ? data.amount
                : (
92                  inputCurrency < outputCurrency
                        ? uint256(uint128(delta.amount1()))
                        : uint256(uint128(delta.amount0()))
                );

            if (exactOutput ? inputAmount >= data.amountLimit : outputAmount <= data.amountLimit) {
                revert SlippageExceeded();
            }

            ...

            outputCurrency.take(poolManager, data.receiver, outputAmount, output6909);

            ...
        }
    }
```

Impact: `outputAmount` could be incorrect, resulting in the reversal of the transaction.

Recommended Mitigation: The comparison should be made between the two currencies involved in the final swap.

Uniswap: To resolve this issue, we pack a synthetic delta that represents the actual amounts used and owed for single and multihop swaps.

```
    function _unlockCallback(bytes calldata callbackData)
        internal
        virtual
        override(SafeCallback)
        returns (bytes memory)
    {
        . . .
        (Currency inputCurrency, Currency outputCurrency, BalanceDelta delta) =
            _parseAndSwap(singleSwap, exactOutput, data.amount, callbackData);

            uint256 inputAmount = inputCurrency < outputCurrency
                ? uint256(int256(-delta.amount0()))
                : uint256(int256(-delta.amount1()));
            uint256 outputAmount = inputCurrency < outputCurrency
                ? uint256(int256(delta.amount1()))
                : uint256(int256(delta.amount0()));
        . . .
```

```
    }

    function _exactInputMultiSwap(Currency inputCurrency, PathKey[] memory path, uint256 amount)
        internal
        virtual
        returns (BalanceDelta delta)
    {
        . . .
        // create the final delta based on original input and final output
        if (originalInputCurrency < inputCurrency) {
            delta = toBalanceDelta(
                -int128(uint128(amount)), int128(uint128(uint256(-amountSpecified)))
            );
        } else {
            delta = toBalanceDelta(
                int128(uint128(uint256(-amountSpecified))), -int128(uint128(amount))
            );
        }
    }

    function _exactOutputMultiSwap(Currency startCurrency, PathKey[] memory path, uint256 amount)
        internal
        virtual
        returns (BalanceDelta delta)
    {
        . . .
        // create the final delta based on original input and final output
        if (startCurrency < path[pos].intermediateCurrency) {
            delta = toBalanceDelta(
                zeroForOne ? delta.amount0() : delta.amount1(), int128(uint128(amount))
            );
        } else {
            delta = toBalanceDelta(
                int128(uint128(amount)), zeroForOne ? delta.amount0() : delta.amount1()
            );
        }
    }
```

KupiaSec: Verified.

7.2 Medium

7.2.1 Too Tight Slippage Check

Description: The `_unlockCallback()` function performs the slippage check using <= and >=. This means that the `outputAmount` MUST be GREATER than the limit, which is too restrictive. Instead, the slippage check should use < and >.

Since the limit is often set to the exact expected output amount, transactions are very likely to revert due to this strict check.

The UniswapV2Router02 requires that the `outputAmount` be equal to or greater than the limit. As the current V4 router aims to simulate the V2 router, it would be beneficial to adopt a similar slippage check.

```
    function _unlockCallback(bytes calldata callbackData)
        ...

97          if (exactOutput ? inputAmount >= data.amountLimit : outputAmount <= data.amountLimit) {
                revert SlippageExceeded();
            }

            ...
```

Impact: Legitimate swaps could be reverted due to the overly tight slippage check.

Recommended Mitigation: Adjust the check to ensure that the conditions are less restrictive:

```
    function _unlockCallback(bytes calldata callbackData)
        ...

-           if (exactOutput ? inputAmount >= data.amountLimit : outputAmount <= data.amountLimit) {
+           if (exactOutput ? inputAmount > data.amountLimit : outputAmount < data.amountLimit) {
                revert SlippageExceeded();
            }

            ...
    }
```

Uniswap: To resolve this issue, we accept the recommended changes to make limit check inclusive to replicate more familiar behavior.

```
if (exactOutput ? inputAmount > data.amountLimit : outputAmount < data.amountLimit) {
    revert SlippageExceeded();
}
```

KupiaSec: Verified.

7.2.2 Arbitrage Through Path Swap Is Not Possible.

Description: Wherever there is a market, arbitrage opportunities arise. However, executing arbitrage through this router is currently impossible.

When an arbitrage occurs, the function `poolManager.currencyDelta(address(this), inputCurrency)` will yield a value greater than 0. Consequently, in the following code, `inputAmount` becomes an excessively large number, leading to a revert.

```
    function _unlockCallback(bytes calldata callbackData)
        internal
        virtual
        override(SafeCallback)
        returns (bytes memory)
    {
        unchecked {
            BaseData memory data = abi.decode(callbackData, (BaseData));

            (bool singleSwap, bool exactOutput, bool input6909, bool output6909, bool _permit2) =
                SwapFlags.unpackFlags(data.flags);

            (Currency inputCurrency, Currency outputCurrency, BalanceDelta delta) =
                _parseAndSwap(singleSwap, exactOutput, data.amount, _permit2, callbackData);

@>          uint256 inputAmount = uint256(-poolManager.currencyDelta(address(this), inputCurrency));
            uint256 outputAmount = exactOutput
                ? data.amount
                : (
                    inputCurrency < outputCurrency
                        ? uint256(uint128(delta.amount1()))
                        : uint256(uint128(delta.amount0()))
                );

            if (exactOutput ? inputAmount >= data.amountLimit : outputAmount <= data.amountLimit) {
                revert SlippageExceeded();
            }

            ...
        }
    }
```

Impact: Arbitrage through path swaps is not feasible.

Recommended Mitigation: Revise the current logic to facilitate arbitrage opportunities through path swaps, similar to the implementation in `UniswapV2Router02`.

Uniswap: Acknowledged. Logic has been revised slightly to optimize, but otherwise maintains current format for simplicity.

Changes:

```
-uint256 inputAmount = uint256(-poolManager.currencyDelta(address(this), inputCurrency));
+uint256 inputAmount;
+if (exactOutput) {
+    if (singleSwap) {
+        // for single-swap exact output, we can use delta
+        inputAmount = zeroForOne
+            ? uint256(uint128(-delta.amount0()))
+            : uint256(uint128(-delta.amount1()));
+    } else {
+        // for multi-hop exact output, use poolManager.currencyDelta
+        inputAmount = uint256(-poolManager.currencyDelta(address(this), inputCurrency));
+    }
+} else {
+    // for exact input (either single or multi), just use data.amount
+    inputAmount = data.amount;
+}
```

KupiaSec: Verified.

7.3 Low

7.3.1 Lack of Check for `amountIn = msg.value` When Swapping `ETH` for Tokens

Description: The `UniswapV4Router04.swapExactTokensForTokens()` function does not verify whether `msg.value` equals `amountIn` when using `ETH` as the `startCurrency`. If `msg.value` exceeds `amountIn`, the user loses the excess amount `(msg.value - amountIn)` because the function does not refund the remaining `ETH`.

```
    function swapExactTokensForTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        Currency startCurrency,
        PathKey[] calldata path,
        address receiver,
        uint256 deadline
    )
        public
        payable
        virtual
        override(IUniswapV4Router04)
        checkDeadline(deadline)
        returns (BalanceDelta)
    {
        return _unlockAndDecode(
            abi.encode(
                BaseData({
                    amount: amountIn,
                    amountLimit: amountOutMin,
                    payer: msg.sender,
                    receiver: receiver,
                    flags: 0
                }),
                startCurrency,
                path
            )
        );
    }
```

Impact: Excess `ETH` is locked and unrecoverable.

Recommended Mitigation: Implement a check to ensure `msg.value = amountIn`.

```
+   if(startCurrency == CurrencyLibrary.ADDRESS_ZERO && msg.value != amountIn) {
+       revert amountInMismatch();
+   }
```

Uniswap: Acknowledged. The router already includes a refund mechanism at the end of the `_unlockCallback` so any excess ETH is returned.

KupiaSec: Acknowledged.

7.4 Informational

7.4.1 Improper Return Value When Path Swap

Description: The return value of swap functions is of type `BalanceDelta`, which should represent the amounts of `inputCurrency` and `outputCurrency`. However, in the case of a path swap, the final result only represents the amounts of two currencies swapped at the last step.

In a path swap, the final return value is determined by the functions `_exactInputMultiSwap()` or `_exactOutput-MultiSwap()`. These functions set the returning `BalanceDelta` based on the last swap result in the path. However,

since the last swap does not occur between `inputCurrency` and `outputCurrency`, the return value of the path swap fails to accurately represent these amounts.

```
    function _exactInputMultiSwap(Currency inputCurrency, PathKey[] memory path, uint256 amount)
        internal
        virtual
        returns (BalanceDelta finalDelta)
    {
        unchecked {
            PoolKey memory poolKey;
            bool zeroForOne;
            int256 amountSpecified = -(amount.toInt256());
            uint256 len = path.length;

            // cache first path key
            PathKey memory pathKey = path[0];

            for (uint256 i; i < len;) {
                (poolKey, zeroForOne) = pathKey.getPoolAndSwapDirection(inputCurrency);
206             finalDelta = _swap(poolKey, zeroForOne, amountSpecified, pathKey.hookData);

                inputCurrency = pathKey.intermediateCurrency;
                amountSpecified = zeroForOne ? -finalDelta.amount1() : -finalDelta.amount0();

                // load next path key
                if (++i < len) pathKey = path[i];
            }
        }
    }

    function _exactOutputMultiSwap(Currency startCurrency, PathKey[] memory path, uint256 amount)
        internal
        virtual
        returns (BalanceDelta finalDelta)
    {
        unchecked {
            PoolKey memory poolKey;
            bool zeroForOne;
            int256 amountSpecified = amount.toInt256();
            uint256 len = path.length;

            // cache last path key for first iteration
            PathKey memory pathKey = path[len - 1];

            // handle all but the final swap
            for (uint256 i = len - 1; i != 0;) {
                (poolKey, zeroForOne) =
                    pathKey.getPoolAndSwapDirection(path[--i].intermediateCurrency);

                BalanceDelta delta = _swap(poolKey, zeroForOne, amountSpecified, pathKey.hookData);

                // update amount for next iteration
                amountSpecified = zeroForOne ? -delta.amount0() : -delta.amount1();

                // load next pathKey for next iteration
                pathKey = path[i];
            }

            // final swap
            (poolKey, zeroForOne) = path[0].getPoolAndSwapDirection(startCurrency);
247         finalDelta = _swap(poolKey, zeroForOne, amountSpecified, path[0].hookData);
        }
```

```
    }
```

Impact: The return value of the path swap does not accurately represent the amounts of `inputCurrency` and `outputCurrency`, which could lead to significant issues in future integrations.

Recommended Mitigation: The current mechanism should be improved to ensure it returns a proper `BalanceDelta`.

Uniswap: To resolve this issue, we compute the final balance `delta` for multihop swaps within their respective internal swap functions and return this to the `_unlockCallback`. Example:

```
// create the final delta based on original input and final output
if (originalInputCurrency < inputCurrency) {
    delta = toBalanceDelta(
        -int128(uint128(amount)), int128(uint128(uint256(-amountSpecified)))
    );
} else {
    delta = toBalanceDelta(
        int128(uint128(uint256(-amountSpecified))), -int128(uint128(amount))
    );
}
```

KupiaSec: Verified.

7.4.2 The `swapExactTokensForTokens()` Function Should Return an Array of `BalanceDelta` Instead of a Single Value

Description: The `swapExactTokensForTokens()` function currently returns only a single `BalanceDelta`. However, in the case of a path swap, it would be more beneficial to return an array of `BalanceDelta`s to accurately represent the entire swap process, similar to the implementation in UniswapV2Router02.

```
    function swapExactTokensForTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        Currency startCurrency,
        PathKey[] calldata path,
        address receiver,
        uint256 deadline
    )
        public
        payable
        virtual
        override(IUniswapV4Router04)
        checkDeadline(deadline)
@>      returns (BalanceDelta)
    {
        return _unlockAndDecode(
            abi.encode(
                BaseData({
                    amount: amountIn,
                    amountLimit: amountOutMin,
                    payer: msg.sender,
                    receiver: receiver,
                    flags: 0
                }),
                startCurrency,
                path
            )
        );
    }
```

Impact: The current return value does not represent the entire process of the path swap, which could lead to incomplete information regarding the swap execution.

Recommended Mitigation: Modify the function to return all `BalanceDelta`s generated during the path swap process.

Uniswap: Acknowledged. For efficiency and simplicity purposes, we will opt to continue to return just a swap delta of input and output.

KupiaSec: Acknowledged.

7.4.3 Inconsistency in Data Location of Function Parameters Between `UniswapV4Router04` and Its Interface

Description: The 4 th parameter, `poolKey`, of the `swapExactTokensForTokens()` function exhibits differing data locations in the `UniswapV4Router04` contract and its interface.

Specifically, in the interface `IUniswapV4Router04.sol`, `poolKey` is declared with a data location of `memory` at line 82. In contrast, in the `UniswapV4Router04.sol` contract, `poolKey` is declared with a data location of `calldata` at line 120.

```
IUniswapV4Router04.sol

    function swapExactTokensForTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        bool zeroForOne,
82      PoolKey memory poolKey,
        bytes calldata hookData,
        address receiver,
        uint256 deadline
    ) external payable returns (BalanceDelta);

------------------------------------------------------------

UniswapV4Router04.sol

    function swapExactTokensForTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        bool zeroForOne,
120     PoolKey calldata poolKey,
        bytes calldata hookData,
        address receiver,
        uint256 deadline
    )
        ...
```

Recommended Mitigation: Ensure consistency between the contract and its interface.

Uniswap: Resolved. The interface now reflects `calldata` as appropriate.

KupiaSec: Verified.

7.4.4 Missing `sync` Call When Settling Native Currency

Description: In the `_unlockCallback()` function, when settling `inputCurrency`, the `currencySettler.settle()` function is invoked. However, this function does not call `sync` when the currency is native. The v4/PoolManager recommends calling `sync` prior to settling native currency.

```
BaseSwapRouter.sol
```

```
    function _unlockCallback(bytes calldata callbackData)
        ...

            if (_permit2) {
                (, PermitPayload memory permitPayload) =
                    abi.decode(callbackData, (BaseData, PermitPayload));
                poolManager.sync(inputCurrency);
                permit2.permitTransferFrom(
                    permitPayload.permit,
                    ISignatureTransfer.SignatureTransferDetails({
                        to: address(poolManager),
                        requestedAmount: inputAmount
                    }),
                    data.payer,
                    permitPayload.signature
                );
                poolManager.settle();
            } else {
117             inputCurrency.settle(poolManager, data.payer, inputAmount, input6909);
            }

            ...
    }

--------------------

@v4/test/utils/CurrencySettler.sol

    function settle(Currency currency, IPoolManager manager, address payer, uint256 amount, bool burn)
    ↪   internal {
        // for native currencies or burns, calling sync is not required
        // short circuit for ERC-6909 burns to support ERC-6909-wrapped native tokens
        if (burn) {
            manager.burn(payer, currency.toId(), amount);
@>      } else if (currency.isAddressZero()) {
            manager.settle{value: amount}();
        } else {
            manager.sync(currency);
            if (payer != address(this)) {
                IERC20Minimal(Currency.unwrap(currency)).transferFrom(payer, address(manager), amount);
            } else {
                IERC20Minimal(Currency.unwrap(currency)).transfer(address(manager), amount);
            }
            manager.settle();
        }
    }
```

Recommended Mitigation: Ensure that `sync` is called before settling native currency.

```
    function _unlockCallback(bytes calldata callbackData)
        ...

            if (_permit2) {
                (, PermitPayload memory permitPayload) =
                    abi.decode(callbackData, (BaseData, PermitPayload));
                poolManager.sync(inputCurrency);
                permit2.permitTransferFrom(
                    permitPayload.permit,
                    ISignatureTransfer.SignatureTransferDetails({
                        to: address(poolManager),
                        requestedAmount: inputAmount
                    }),
                    data.payer,
                    permitPayload.signature
                );
                poolManager.settle();
            } else {
+               poolManager.sync(inputCurrency);
                inputCurrency.settle(poolManager, data.payer, inputAmount, input6909);
            }

            ...
    }
```

Uniswap: To resolve this issue, we now call `sync` prior to ETH settlement. This improves the use of the `CurrencySettler` library.

KupiaSec: Verified.