hexens x KyberSwap

Security Review Report for KyberSwap

November 2025

Table of Contents

1. About Hexens

Hexens is a pioneering cybersecurity firm dedicated to establishing robust security standards for Web3 infrastructure, driving secure mass adoption through innovative protection technology and frameworks. As an industry elite experts in blockchain security, we deliver comprehensive audit solutions across specialized domains, including infrastructure security, Zero Knowledge Proof, novel cryptography, DeFi protocols, and NFTs.

Our methodology combines industry-standard security practices combined with unique methodology of two teams per audit, continuously advancing the field of Web3 security. This innovative approach has earned us recognition from industry leaders.

Since our founding in 2021, we have built an exceptional portfolio of enterprise clients, including major blockchain ecosystems and Web3 platforms.

2. Executive Summary

This report covers the security review for Kyber Network. The review included the new KSAggregationRouterV3 contract, an upgrade to the existing aggregation router.

Our security assessment was a full review of the code, spanning a total of 1 week.

During our review, we did not identify any major severity vulnerabilities.

We did identify several minor severity vulnerabilities and code optimisations.

All of our reported issues were fixed or acknowledged by the development team and consequently validated by us.

We can confidently say that the overall security and code quality have increased after completion of our audit.

3. Security Review Details

Review Led by

Kasper Zwijsen, Head of Audits

Scope

The analyzed resources are located on:

🔗 https://github.com/KyberNetwork/ks-aggregation-router/tree/88bc5b4ce6734e25bb63d62abe5139b5873654a8

🔗 https://github.com/KyberNetwork/ks-common-sc/tree/c1532e7cba2302f133578de3c9a4392197db66ac

The issues described in this report were fixed in the following commits:

🔗 https://github.com/KyberNetwork/ks-aggregation-router/

📌 Commit: `d7d0e14956695c7204d388148a89b9b8116ca8c0`

📌 Commit: `63a173176940b2c8654c743e23b618c521d26899`

🔗 https://github.com/KyberNetwork/ks-aggregation-router/commit/217f0f7997701e6bd1a6f705d3589f1964fae132
This commit was shared by KyberSwap as a post-audit fix, we reviewed the commit and found no issues with the fix.

Changelog

24 November 2025	Audit start
1 December 2025	Initial report
2 December 2025	Revision received
5 December 2025	Final report

4. Severity Structure

The vulnerability severity is calculated based on two components:

1. Impact of the vulnerability
2. Probability of the vulnerability

Impact	Probability			
	Rare	Unlikely	Likely	Very likely
Low	Low	Low	Medium	Medium
Medium	Low	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

- **Severity Characteristics**

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical

Vulnerabilities that are highly likely to be exploited and can lead to catastrophic outcomes, such as total loss of protocol funds, unauthorized governance control, or permanent disruption of contract functionality.

High

Vulnerabilities that are likely to be exploited and can cause significant financial losses or severe operational disruptions, such as partial fund theft or temporary asset freezing.

Medium	Vulnerabilities that may be exploited under specific conditions and result in moderate harm, such as operational disruptions or limited financial impact without direct profit to the attacker.
Low	Vulnerabilities with low exploitation likelihood or minimal impact, affecting usability or efficiency but posing no significant security risk.
Informational	Issues that do not pose an immediate security risk but are relevant to best practices, code quality, or potential optimizations.

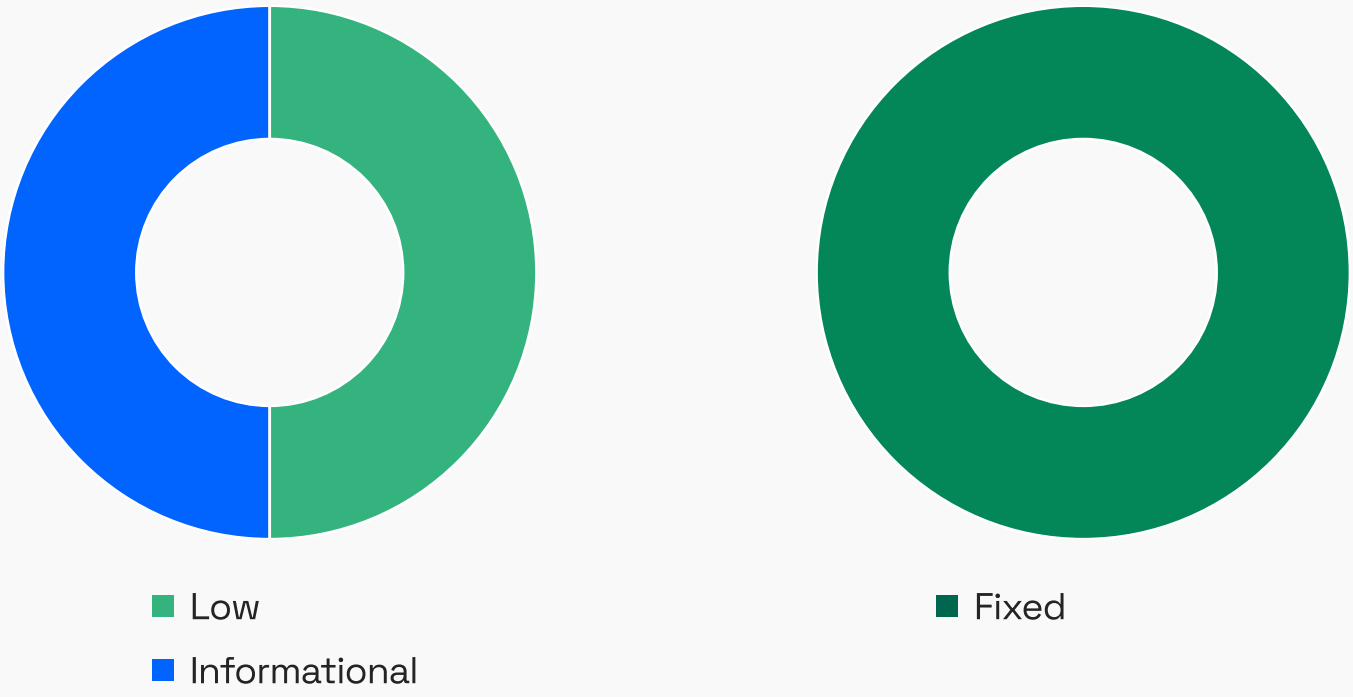
- **Issue Symbolic Codes**

Each identified and validated issue is assigned a unique symbolic code during the security research stage.

Due to the structure of the vulnerability reporting flow, some rejected issues may be missing.

5. Findings Summary

Severity	Number of findings
Critical	0
High	0
Medium	0
Low	1
Informational	1
Total:	**2**



6. Weaknesses

This section contains the list of discovered weaknesses.

KYBER1-1 | totalAmount of input tokens couldn't limit the actual transferred amount

Fixed

Severity:	Probability:	Impact:
Low	Rare	Low

Path:

src/KSAggregationRouterV3.sol#L160-L184

Description:

In the **_collectInputToken()** function, when the input token is the native token, **totalAmount** is required to be equal to **msg.value**, and the actual amount of native tokens used is enforced. However, when the input token is an ERC20 token, there is no limit on the total amount of tokens that can be pulled using Permit2 in the **_collectInputToken()** function.

Although **totalAmount** is used to calculate fees, the function does not guarantee that the actual transferred token amount is equal to or less than **totalAmount**. The actual transferred amount includes both the fee amounts and the target amounts, specified in the input data.

Therefore, the user (**msg.sender**) may be confused about the amount of tokens they need to permit for the router contract. This also creates a risk that the user's permitted funds may be pulled in more than expected for a swap, due to large fees or target amounts contained in platform-provided data. While the output amount is protected by the **minAmountOut** check after charging fees, the permitted amount of input tokens is still unprotected, as the actual transferred amount isn't limited to the **totalAmount** of the input token.

```
if (permitData.length == 0) {
  if (totalAmount >= type(uint160).max) {
    revert TooLargeInputAmount(totalAmount);
  }

  IAllowanceTransfer.AllowanceTransferDetails[] memory details =
    new IAllowanceTransfer.AllowanceTransferDetails[](feeRecipients.length + targets.length);

  for (uint256 i = 0; i < feeRecipients.length; i++) {
    uint256 feeAmount = _computeFeeAmount(totalAmount, fees[i]);
```

```
    details[i] = IAllowanceTransfer.AllowanceTransferDetails({
      from: msg.sender, to: feeRecipients[i], amount: uint160(feeAmount), token: token
    });

    if (feeAmount > 0) {
      emit CollectFee(token, totalAmount, feeAmount, feeRecipients[i]);
    }
  }
  for (uint256 i = 0; i < targets.length; i++) {
    details[i + feeRecipients.length] = IAllowanceTransfer.AllowanceTransferDetails({
      from: msg.sender, to: targets[i], amount: uint160(amounts[i]), token: token
    });
  }

  PERMIT2.transferFrom(details);
}
```

Remediation:

The **_collectInputToken()** function should include a variable to verify that the sum of the actual transferred tokens, including fee amounts and target amounts, is less than or equal to the **totalAmount** of the input.

Here's an example fix:

```
if (permitData.length == 0) {
  if (totalAmount >= type(uint160).max) {
    revert TooLargeInputAmount(totalAmount);
  }

  IAllowanceTransfer.AllowanceTransferDetails[] memory details =
    new IAllowanceTransfer.AllowanceTransferDetails[](feeRecipients.length + targets.length);

+ uint256 totalTransferred = 0;
  for (uint256 i = 0; i < feeRecipients.length; i++) {
    uint256 feeAmount = _computeFeeAmount(totalAmount, fees[i]);
    details[i] = IAllowanceTransfer.AllowanceTransferDetails({
      from: msg.sender, to: feeRecipients[i], amount: uint160(feeAmount), token: token
    });
+   totalTransferred += feeAmount;

    if (feeAmount > 0) {
      emit CollectFee(token, totalAmount, feeAmount, feeRecipients[i]);
```

```
    }
  }
  for (uint256 i = 0; i < targets.length; i++) {
    details[i + feeRecipients.length] = IAllowanceTransfer.AllowanceTransferDetails({
      from: msg.sender, to: targets[i], amount: uint160(amounts[i]), token: token
    });
+   totalTransferred += amounts[i];
  }

+ require(totalTransferred <= totalAmount);
  PERMIT2.transferFrom(details);
}
```

KYBER1-2 | Missing length check for the outputTokens and outputData arrays

Fixed

Severity:	Probability:	Impact:
Informational	Rare	Informational

Path:

src/KSAggregationRouterV3.sol#L210-L223

Description:

In the contract, any group of arrays from the swap parameters that must have the same size undergo length checks using the **checkLengths()** modifiers, which revert with the message **MismatchedArrayLengths** whenever the array lengths don't match.

The **inputTokens–inputAmounts–inputData** arrays are already checked in the **_collectInputTokens()** function. Similarly, the **feeRecipients–fees** and **targets–amounts** arrays are checked in the **_collectInputToken** and **_processOutputToken** functions.

However, **outputTokens - outputData** arrays are never checked for matching lengths like the other arrays. This can cause the process in **_recordOutputBalances()** to revert without the expected error message.

```
function _recordOutputBalances(
  address[] calldata outputTokens,
  OutputTokenData[] calldata outputData,
  address recipient
) internal view returns (uint256[] memory outputBalances) {
  outputBalances = new uint256;
  for (uint256 i = 0; i < outputTokens.length; i++) {
    if (outputData[i].feeRecipients.length == 0) {
      outputBalances[i] = outputTokens[i].balanceOf(recipient);
    } else {
      outputBalances[i] = _selfBalanceMinusMsgValue(outputTokens[i]);
    }
  }
}
```

Remediation:

_recordOutputBalances should also use the **checkLengths()** modifier with the lengths of **outputTokens** and **outputData** to verify that they match.


hexens
x
KyberSwap