hexens x KyberSwap

Security Review Report for KyberSwap

December 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. This review included the Smart Intent protocol, which allows highly customizable automation and delegation of actions.

Our security assessment was a full review of the code, spanning a total of 2 weeks.

During our review, we identified 1 High severity vulnerability, it could have resulted in unauthorized actions being executed on behalf of victim users.

We also identified several minor severity vulnerabilities and code optimisations.

All of our reported issues were fixed 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

Jahyun Koo, Lead Security Researcher

▪ Scope

The analyzed resources are located on:

🔗 https://github.com/KyberNetwork/smart-intent-sc/tree/6c0e85b92b7fb49f721219fbe86160be9b60337b

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

🔗 https://github.com/KyberNetwork/smart-intent-sc/tree/2932bbbf455ca0812150158d3551a9339f1c4028

These PR commits were shared by KyberSwap as a post-audit fix, we reviewed the commits and found no issues with the fixes:

🔗 ▪ PR 55: https://github.com/KyberNetwork/smart-intent-sc/commit/bd65548ae01ca49c6ecbfbe83b6b780c74002eee

🔗 ▪ PR 57: https://github.com/KyberNetwork/smart-intent-sc/commit/c608a0b37846d214ef02666e7a93ead40dae444e

▪ Changelog

22 December 2025	Audit start
7 January 2026	Initial report
15 January 2026	Revision received
16 January 2026	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.

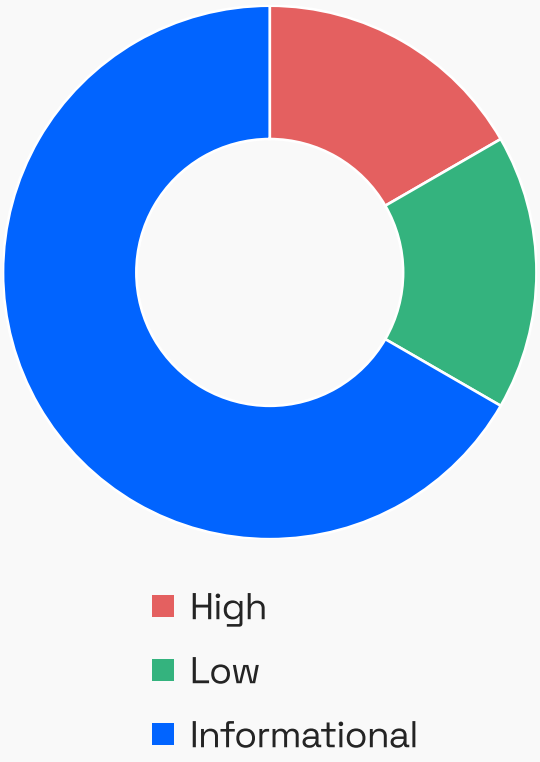
- **Issue Symbolic Codes**

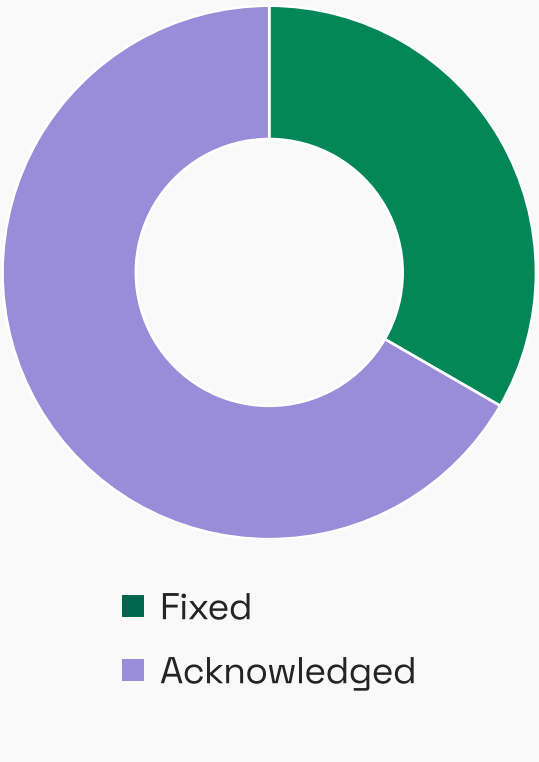
Each identified and validated issue is assigned a unique symbolic code during the security research stage.

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	1
Medium	0
Low	1
Informational	4
Total:	**6**





6. Weaknesses

This section contains the list of discovered weaknesses.

KYBER2-1 | Action signatures are not bound to a specific intent, enabling cross-intent replay

Fixed

Severity:	Probability:	Impact:
High	Likely	High

Path:

src/KSSmartIntentRouter.sol#L147-L173

Description:

The **KSSmartIntentRouter** contract validates execution signatures (**dkSignature** and **gdSignature**) using a hash derived solely from the **ActionData** structure. Crucially, this hash calculation does not incorporate the **intentHash** of the Intent being executed.

```
function _execute(
  bytes32 intentHash,
  IntentData calldata intentData,
  bytes calldata dkSignature,
  address guardian,
  bytes calldata gdSignature,
  ActionData calldata actionData
) internal nonReentrant {
  _checkIntentStatus(intentHash, IntentStatus.DELEGATED);
  if (actionData.actionSelectorId >= intentData.coreData.actionContracts.length) {
    revert InvalidActionSelectorId(actionData.actionSelectorId);
  }
  if (block.timestamp > actionData.deadline) {
    revert ActionExpired();
  }
  _checkRole(KSRoles.GUARDIAN_ROLE, guardian);

  _useUnorderedNonce(intentHash, actionData.nonce);

  _validateActionData(
    intentData.coreData,
```

```
    dkSignature,
    guardian,
    gdSignature,
    _hashTypedDataV4(hasher.hash(actionData))
  );
```

The router verifies DK/Guardian signatures over the **ActionData** hash only, without including the **intentHash**. As a result, a valid signature produced for one intent can be reused to execute a different intent that uses the same delegated key/guardian.

Per-intent status and nonce checks do not prevent this replay because they are scoped to the target intent, not the signature itself.

```
function _useUnorderedNonce(bytes32 intentHash, uint256 nonce) internal {
  uint256 wordPos = nonce >> 8;
  uint256 bitPos = uint8(nonce);

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

  emit UseNonce(intentHash, nonce);
}
```

This breaks intent-specific authorization and can cause execution under different token sets, hooks, or action targets than the signers intended.

Remediation:

Bind action signatures to the intent by including **intentHash** in the signed message (or by signing a typed structure that incorporates it), and validate both DK and Guardian signatures against that combined hash across all supported verification paths.

KYBER2-4 | Unvalidated feesGenerated Allows Yield Condition Bypass

Acknowledged

Severity:	Probability:	Impact:
Low	Rare	Medium

Path:

src/hooks/base/BaseTickBasedRemoveLiquidityHook.sol
src/hooks/base/BaseConditionalHook.sol

Description:

The remove-liquidity hooks for Uniswap V3, V4 and Pancakeswap V4-CL use **feesGenerated** values decoded from **actionData.hookActionData** when evaluating yield-based conditions. These values are supplied by the delegated key and are not verified against on-chain state or any trusted source.

```
function _cacheBaseData(
  RemoveLiquidityHookData calldata validationData,
  bytes calldata hookActionData,
  RemoveLiquidityParams memory removeLiqParams,
  OutputValidationParams memory outputParams
) internal view virtual {
  (
    removeLiqParams.index,
    removeLiqParams.positionInfo.feesGenerated[0],
    removeLiqParams.positionInfo.feesGenerated[1],
    removeLiqParams.liquidityToRemove,
    removeLiqParams.wrapOrUnwrap,
    outputParams.intentFeesPercent
  ) = _decodeHookActionData(hookActionData);
  removeLiqParams.recipient = validationData.recipient;
  removeLiqParams.positionInfo.nftAddress = validationData.nftAddresses[removeLiqParams.index];
  removeLiqParams.positionInfo.nftId = validationData.nftIds[removeLiqParams.index];

  outputParams.router = msg.sender;
  outputParams.maxFees = [
    validationData.maxFees[removeLiqParams.index] >> 128,
    uint128(validationData.maxFees[removeLiqParams.index])
  ];
}
```

As a result, a delegated key can submit inflated fee values to satisfy the yield threshold even when actual fees are lower, allowing liquidity removal that does not match the main address's intended yield condition.

```
function _evaluateYieldCondition(Condition calldata condition, bytes calldata additionalData)
  internal
  pure
  virtual
  returns (bool)
{
  uint256 fee0;
  uint256 fee1;
  uint256 poolPrice;

  assembly ('memory-safe') {
    fee0 := calldataload(additionalData.offset)
    fee1 := calldataload(add(additionalData.offset, 0x20))
    poolPrice := calldataload(add(additionalData.offset, 0x40))
  }

  YieldCondition calldata yieldCondition = _decodeYieldCondition(condition.data);

  uint256 initialAmount0 = yieldCondition.initialAmounts >> 128;
  uint256 initialAmount1 = uint256(uint128(yieldCondition.initialAmounts));

  uint256 numerator = fee0 + _convertToken1ToToken0(poolPrice, fee1);
  uint256 denominator = initialAmount0 + _convertToken1ToToken0(poolPrice, initialAmount1);
  if (denominator == 0) return false;

  uint256 yield = (numerator * PRECISION) / denominator;

  return yield >= yieldCondition.targetYield;
}
```

Remediation:

Derive the fee inputs used for yield checks from on-chain state or other verifiable data instead of relying on **hookActionData**. If off-chain inputs are required, introduce validation that ties the submitted fee values to the position's observable state and reject mismatches.

Commentary from the client:

“We utilize the off-chain input fee to enhance the user experience. This fee accounts for both claimed and unclaimed amounts. Unclaimed fees alone can be reset to zero periodically, which is particularly relevant for Uniswap V4-like implementations.”

KYBER2-2 | Allowance Not Cleared on Intent Revoke

Acknowledged

Severity:	Probability:	Impact:
Informational	Very likely	Informational

Path:

src/KSSmartIntentRouter.sol#L84-L95

Description:

When revoke is executed, the function updates intentStatuses to REVOKED, but it does not clean up the associated erc20Allowances. While this does not have a direct impact, it results in state inconsistency, where the intent is marked as REVOKED but the corresponding allowance data still exists.

```
/// @inheritdoc IKSSmartIntentRouter
function revoke(IntentData calldata intentData) public {
  if (intentData.coreData.mainAddress != msg.sender) {
    revert NotMainAddress();
  }

  bytes32 intentHash = _hashTypedDataV4(hasher.hash(intentData));
  intentStatuses[intentHash] = IntentStatus.REVOKED;

  emit RevokeIntent(intentHash);
}
```

Remediation:

Consider adding to clear allowance.

Commentary from the client:

“This behavior is intentional to reduce gas costs for the revoke action.”

KYBER2-5 | Silent failure of ERC20 permit calls in _approveTokens

Acknowledged

Severity:	Probability:	Impact:
Informational	Rare	Informational

Path:

src/KSSmartIntentRouterAccounting.sol

Description:

In **KSSmartIntentRouterAccounting._approveTokens**, ERC20 permit data is forwarded via **callERC20Permit** when provided, but the return value is not checked. The underlying helper (**PermitHelper.callERC20Permit**) uses **try/catch** and does not revert on failure, so a failed permit is silently ignored. As a result, execution can proceed without the intended allowance being set, and the transaction may revert later during token collection due to insufficient allowance.

```
function _approveTokens(bytes32 intentHash, TokenData calldata tokenData, address mainAddress)
  internal
{
  for (uint256 i = 0; i < tokenData.erc20Data.length; i++) {
    ERC20Data calldata erc20Data = tokenData.erc20Data[i];

    erc20Allowances[intentHash][erc20Data.token] = erc20Data.amount;

    if (erc20Data.permitData.length > 0) {
      erc20Data.token.callERC20Permit(mainAddress, erc20Data.permitData);
    }
  }
  for (uint256 i = 0; i < tokenData.erc721Data.length; i++) {
    ERC721Data calldata erc721Data = tokenData.erc721Data[i];

    if (erc721Data.permitData.length > 0) {
      erc721Data.token.callERC721Permit(erc721Data.tokenId, erc721Data.permitData);
    }
  }
}
```

Remediation:

Treat permit submission as a best-effort step rather than a hard requirement. When permit data is provided, attempt the permit call but gate execution on the post-call approval state (e.g., ensure the router/forwarder has sufficient allowance for the intended spend). If the required allowance is already sufficient, skip the permit call. If it is not sufficient after attempting permit, fail early with a clear error to surface the root cause.

KYBER2-6 | Fee-on-Transfer Tokens Are Not Accounted for During ERC20 Collection

Acknowledged

Severity:	Probability:	Impact:
Informational	Rare	Informational

Path:

src/types/ERC20Data.sol

Description:

ERC20DataLibrary.collect assumes that the receiver gets **amount - fee** after **transferFrom**. For fee-on-transfer (FOT) tokens, the actual received amount can be lower. This mismatch can cause the router or forwarder to have less balance than expected when the action contract runs, leading to failed executions or minor accounting inconsistencies (e.g., recorded volume/fees not matching actual transfers).

```
function collect(
  address token,
  uint256 amount,
  address mainAddress,
  address actionContract,
  uint256 fee,
  bool approvalFlag,
  IKSGenericForwarder forwarder,
  FeeConfig[] calldata partnerFeeConfigs,
  address protocolRecipient
) internal {
  if (address(forwarder) == address(0)) {
    token.safeTransferFrom(mainAddress, address(this), amount - fee);
    if (approvalFlag) {
      token.forceApprove(actionContract, type(uint256).max);
    }
  } else {
    token.safeTransferFrom(mainAddress, address(forwarder), amount - fee);
    if (approvalFlag) {
      _forwardApproveInf(forwarder, token, actionContract);
    }
  }
```

Remediation:

Update the collection flow to base subsequent logic on the actual amount received by the router/forwarder (e.g., using balance deltas), and align any accounting or validations to that effective amount.

KYBER2-7 | Remove Liquidity Hook Does Not Explicitly Compare Intent Fees Against Max Fees

Fixed

Severity:	Probability:	Impact:
Informational	Rare	Informational

Path:

src/hooks/base/BaseTickBasedRemoveLiquidityHook.sol

Description:

The remove-liquidity hook decodes **intentFeesPercent** from **hookActionData** and applies it in fee calculations, while **maxFees** is only used to derive the minimum amount the user must receive. This means an explicit comparison between the two is missing. Although the current **minReceived** check indirectly enforces the limit and causes a revert when fees exceed **maxFees**, the absence of a direct check can make failures less clear.

```
function _cacheBaseData(
  RemoveLiquidityHookData calldata validationData,
  bytes calldata hookActionData,
  RemoveLiquidityParams memory removeLiqParams,
  OutputValidationParams memory outputParams
) internal view virtual {
  (
    removeLiqParams.index,
    removeLiqParams.positionInfo.feesGenerated[0]-
    removeLiqParams.positionInfo.feesGenerated[1],
    removeLiqParams.liquidityToRemove,
    removeLiqParams.wrapOrUnwrap,
    outputParams.intentFeesPercent
  ) = _decodeHookActionData(hookActionData);
  removeLiqParams.recipient = validationData.recipient;
  removeLiqParams.positionInfo.nftAddress = validationData.nftAddresses[removeLiqParams.index];
  removeLiqParams.positionInfo.nftId = validationData.nftIds[removeLiqParams.index];

  outputParams.router = msg.sender;
  outputParams.maxFees = [
    validationData.maxFees[removeLiqParams.index] >> 128,
    uint128(validationData.maxFees[removeLiqParams.index])
  ];
}
```

Remediation:

Add an explicit validation that the decoded **intentFeesPercent** values do not exceed the corresponding **maxFees**, and revert with a clear error when the limit is exceeded.


hexens
x
KyberSwap