OKX DEX Router EVM

Security Audit Report

prepared by

OKX Web3
Audit Team

2025.05.14

Contents

1. Overview

1.1 About OKX DEX Router

OKX DEX Router a decentralized exchange (DEX) aggregator project based on Ethereum, with the primary function of integrating multiple DEX protocols through smart contracts to provide users with the optimal token swap paths and prices.

1.2 Audit Summary

Ecosystem	EVM	**Language**	Solidity
Repository			
Base Commit			
Final Commit			

1.3 Audit Scope

```
contracts/8/
├── DexRouter.sol
├── TokenApprove.sol
├── TokenApproveProxy.sol
├── UnxswapRouter.sol
├── UnxswapV3Router.sol
├── DexRouterExactOut.sol
├── UnxswapExactOutRouter.sol
├── UnxswapV3ExactOutRouter.sol
├── adapter/
├── interfaces/
├── libraries/
│   ├── CommissionLib.sol
│   ├── PMMLib.sol
│   ├── CommonUtils.sol
│   ├── Address.sol
│   ├── SafeERC20.sol
│   ├── SafeMath.sol
│   ├── UniversalERC20.sol
│   └── [... other libraries]
├── storage/
│   ├── DexRouterStorage.sol
│   └── PMMRouterStorage.sol
├── types/
└── utils/
    ├── PmmConstantsTool.sol
    └── WNativeRelayer.sol
```

1.4 Revision History

Version	Date	Commit	Description
V1.0	20240207	229bc2b	Base Audit
V1.1	20240328	4bfb51d	ToToken Commission
V1.2	20241230	46548fc	Refund Feature
V1.3	20250303	63daaa4	TokenApprove V2 Support
V1.4	20250507	4c27678	Dual Commission
V1.5	20250514	3fa2f3b	ExactOut Swap

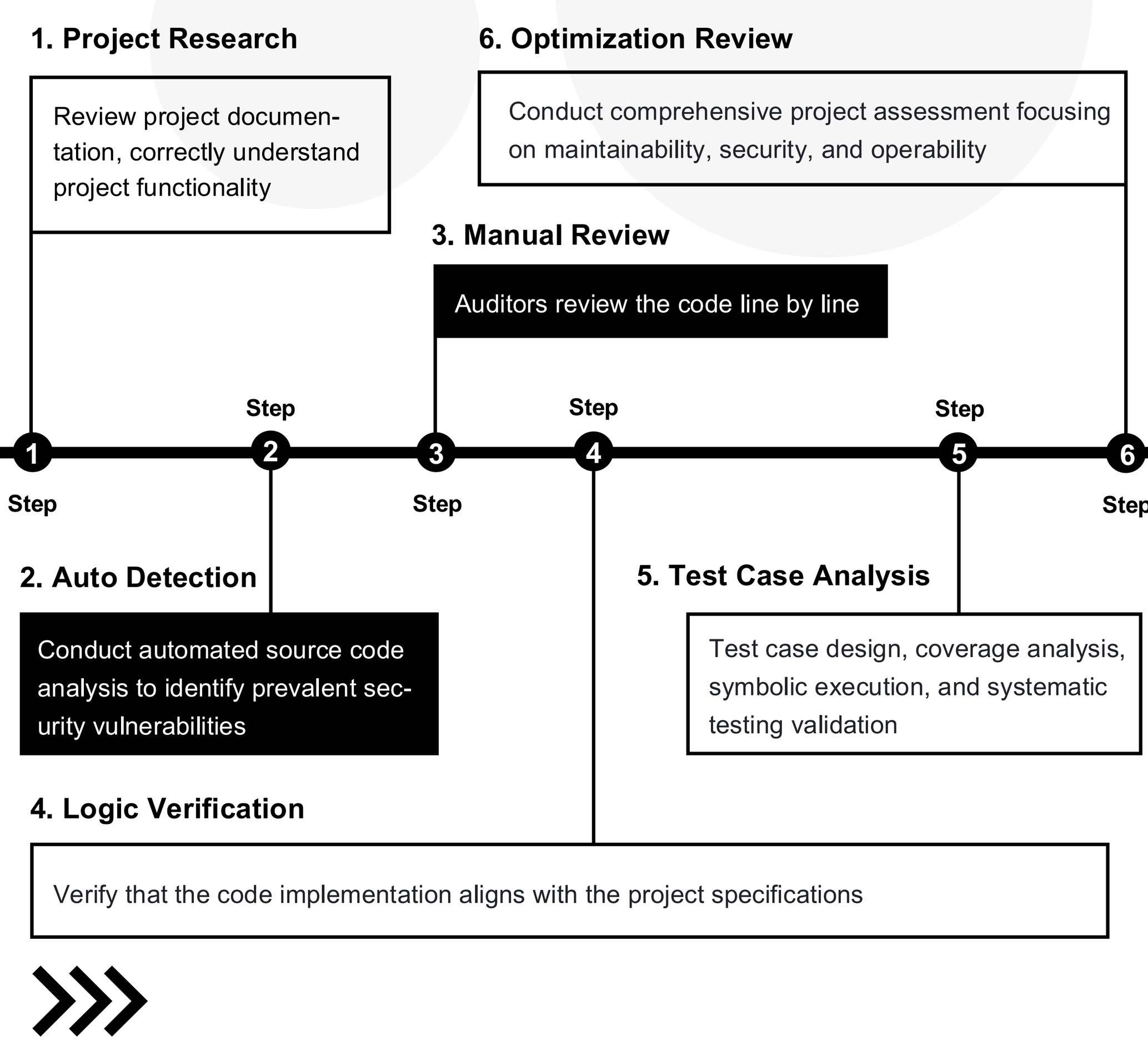
2. Audit Summary

2.1 Audit Methodology

The audit team conducted comprehensive analysis of the contract code through deep understanding of the project's design purpose, operating principles, and implementation methods. By mapping function call relationships, potential security vulnerabilities were systematically identified, with detailed problem descriptions and corresponding remediation recommendations provided.

2.2 Audit Process

The smart contract security audit follows a 6-phase process: Project Research, Automated Detection, Manual Review, Logic Verification, Test Case Analysis, and Optimization Review. During manual auditing, auditors perform comprehensive code review to identify vulnerabilities and provide detailed solutions. After completing all phases, the lead auditor communicates findings with the project team. Following the team's responses, we deliver final audit reports to the project team.



2.3 Risk Classification and Description

Risk items are classified into 5 levels: Critical, High, Medium, Low, and Informational. Critical risks require immediate resolution and re-audit before final report delivery; unresolved critical risks result in audit failure. High risks must be addressed but are less urgent; failure to resolve also results in audit failure. Medium risks indicate potential exposure and require clear documentation of project team notification and response status without affecting report delivery. Low risks and informational items involve compliance or code detail issues that may be deferred without impacting report delivery.

Risk Level	Icon	Risk Description
Critical	🚨	Fatal risks requiring immediate resolution
High	‼️	High-risk vulnerabilities that will cause similar issues, must be resolved
Medium	⚠️	Medium-risk vulnerabilities with potential impact, should be resolved
Low	❗	Low-risk issues with improper handling or warning triggers, can be deferred
Informational	❕	Optimization opportunities, deferrable but recommended for resolution

2.4 Vulnerability Checklist

The vulnerability checklist is divided into two parts: one part is the vulnerability summary of the project audit, and the other part is the detailed vulnerability list.

- **Vulnerability Summary:**

Critical	High	Medium	Low	Informational	Total
0	0	0	8	1	12

- **Vulnerability list:**

No.	Severity	Vulnerability	Category	Status
1	Low	Improper Return of Remaining Assets to tx.origin	Logic Issue	Fixed
2	Low	Inappropriate Leftover Asset Refund to Payer for Earn	Logic Issue	Fixed
3	Low	Incorrect moreInfo Parsing in SmardexAdapter	Logic Issue	Fixed
4	Low	IzumiAdapter Missing Refund Logic for Surplus Assets	Logic Issue	Confirmed
5	Low	Unreasonable Commission Charging from Caller	Logic Issue	Fixed
6	Low	Lack of Token Validation in commissionInfo	Validation	Fixed

No.	Severity	Vulnerability	Category	Status
7	Low	Lack of Non-Zero Check on commissionRate2	Validation	Fixed
8	Low	Potential Centralization Risks	Security	Confirmed
9	Info	Inconsistent ETH Balance Retrieval in _getBalanceOf()	Coding	Fixed

- **Open:** The audit team has notified the project team of the vulnerability, but no reasonable remediation has been implemented.
- **Fixed:** The project team has addressed the vulnerability and the fix has been verified by the audit team.
- **Confirmed:** The project team has confirmed awareness of the vulnerability risk but considers it controllable.

3. Vulnerabilities

This section outlines the risk items identified through manual review and auditing tools. Each item includes the specific file path and code location, along with the assigned risk level. Detailed descriptions of the risks, recommended remediation measures, and relevant code snippets are provided to help clearly illustrate each issue.

3.1 Improper Return of Remaining Assets to tx.origin

Location	File	Status	Severity
Line 52	UniV3Adapter.sol	Fixed	! Low

- **Description**

In the current implementation, during the token swap process, remaining assets are directly returned to the transaction initiator (tx.origin). However, this logic overlooks scenarios where the transaction initiator (tx.origin) may not necessarily be the actual token owner. For example, for users utilizing Account Abstraction (AA) wallets, the transaction initiator is the bundler, not the user who pays the tokens. Therefore, returning remaining assets to tx.origin is inappropriate and may result in assets being returned to the wrong address.

- **Related Code**

```
47      uint256 sellAmount = IERC20(fromToken).balanceOf(address(this));
48      bool zeroForOne = fromToken < toToken;
49      IUniV3(pool).swap(...);
50      uint amount = IERC20(fromToken).balanceOf(address(this));
51      if (amount > 0) {
52          SafeERC20.safeTransfer(IERC20(fromToken), tx.origin, amount);
```

- **Recommendation**

Replace the use of tx.origin with the actual payer's address.

- **Project Team Feedback**

Team Response	Fixed
Re-audit Result	Confirmed

3.2 Inappropriate Leftover Asset Refund to Payer for Earn

Location	File	Status	Severity
Line 554	DexRouter.sol	Fixed	!Low

- **Description**

The smartSwapByInvest() function is specifically designed for Earn project integration. In this scenario, the Earn contract first transfers fromToken directly to the DexRouter contract, then calls this interface to complete the token swap. Consequently, when invoking the _smartSwapInternal() function, the payer parameter is set to address(this) (line 554).

However, in the current implementation, leftover assets are refunded to the payer. Under these circumstances, leftover assets are returned to address(this) (the DexRouter contract itself), causing the refund process to be ineffective as the assets remain stuck in the contract.

- **Related Code**

```
550     returnAmount = _smartSwapInternal(
551         newBaseRequest,
552         newBatchesAmount,
553         batches,
554         address(this),
555         to
```

- **Recommendation**

In Earn scenarios, leftover assets should not be directly refunded to the payer. Instead, implement a proper refund mechanism for the actual user or intended recipient.

- **Project Team Feedback**

Team Response	Fixed
Re-audit Result	Confirmed

3.3 Incorrect moreInfo Parsing in SmardexAdapter

Location	File	Status	Severity
Line 36	SmardexAdapter.sol	Fixed	! Low

- **Description**

The SmardexAdapter::sellBase() function's moreInfo parameter encoding does not include sqrtX96 data, but during decoding, it attempts to decode sqrtX96 as part of the moreInfo. This mismatch causes the decoded data to be incorrect, leading to transaction reverts.

- **Related Code**

```
35      (uint160 sqrtX96, bytes memory data) = abi.decode(
36          moreInfo,
37          (uint160, bytes)
38      );
```

- **Recommendation**

Ensure consistency between encode and decode data structures.

- **Project Team Feedback**

Team Response	Fixed
Re-audit Result	Confirmed

3.4 IzumiAdapter Missing Refund Logic for Surplus Assets

Location	File	Status	Severity
Line 77	IzumiAdapter.sol	Confirmed	! Low

- **Description**

The current protocol supports multiple UniswapV3-like adapters that transfer assets from their respective adapters through callback mechanisms (line 85). If the input asset amount exceeds the actual required payment amount, surplus assets will be generated within the adapter. Therefore, corresponding refund logic must be implemented in these adapters to ensure that surplus assets can be properly returned.

- **Related Code**

```
77      function swapY2XCallback(uint256, uint256 y⬆, bytes memory moreInfo⬆) external override {
78          SwapCallbackData memory dt = abi.decode(moreInfo⬆, (SwapCallbackData));
79          (address token0, address token1, uint24 fee) = dt.path.decodeFirstPool();
80
81          verify(token0, token1, fee);
82          if (token0 > token1) {
83              // token0 is y, amount of token0 is input param
84              // called from swapY2X(...)
85              pay(token0, dt.payer, msg.sender, y⬆);
86          }
87      }
```

- **Recommendation**

Add refund logic to all adapters that may generate surplus assets.

- **Project Team Feedback**

Team Response	Confirmed
Re-audit Result	NA

3.5 Unreasonable Commission Charging from Caller

Location	File	Status	Severity
Line 109	CommissionLib.sol	Fixed	!Low

- **Description**

Currently, the DexRouter's commission fee is deducted from the caller (line 125), but in certain scenarios, this design may be unreasonable because the caller is not necessarily the address (payer) that actually pays the fromToken in the swap operation. This may result in inaccurate fee deduction.

- **Related Code**

```
109 >   function _doCommissionFromToken(…
113     ) internal returns (address, uint256) {
114         ...
115         assembly ("memory-safe") {
116             ...
117             default {
118                 let freePtr := mload(0x40)
119                 mstore(0x40, add(freePtr, 0x84))
120                 mstore(
121                     freePtr,
122                     0x0a5ea46600000000000000000000000000000000000000000000000000000000
123                 ) // claimTokens
124                 mstore(add(freePtr, 0x04), token)
125                 mstore(add(freePtr, 0x24), caller())
126                 mstore(add(freePtr, 0x44), referer)
127                 mstore(add(freePtr, 0x64), amount)
128 >               let success := call(…
136                 )
```

- **Recommendation**

Uniformly adjust the commission payer to the fromToken payer.

- **Project Team Feedback**

Team Response	Fixed
Re-audit Result	Confirmed

3.6 Lack of Token Validation in commissionInfo

Location	File	Status	Severity
-	Multiple Files	Fixed	! Low

- **Description**

Through contract analysis, the commission token type should be consistent with either the swap's input token type or output token type. Specifically:

- If isFromTokenCommission is true, the commission token should match the swap's input token
- If isToTokenCommission is true, the commission token should match the swap's output token

However, current code lacks this validation. Consider this scenario: A user holds BTC with the goal of obtaining USDT, and the contract is configured to collect commission when USDT operations occur. However, during subsequent exploit or malicious contract behavior, the commission token type can be replaced with BTC. This means when users perform swaps using low-value tokens, they could be charged high-value tokens as commission, resulting in user losses.

- **Related Code**

```
38          let token := mload(add(commissionInfo, 0x80))
39          let referer := mload(add(commissionInfo, 0x60))
40          let amount := div(mul(inputAmount, rate), sub(10000, totalRate))
41          switch eq(token, _ETH)
42          case 1 {
43              let success := call(gas(), referer, amount, 0, 0, 0, 0)
44              if eq(success, 0) {
```

- **Recommendation**

Add necessary sanity checks at the contract level to ensure the commission token type remains consistent with the user's swap token types.

- **Project Team Feedback**

Team Response	Fixed
Re-audit Result	Confirmed

3.7 Lack of Non-Zero Check on commissionRate2

Location	File	Status	Severity
Line 149	CommissionLib.sol	Fixed	! Low

- **Description**

The commissionInfo.commissionRate == 0 is checked twice but commissionInfo .commissionRate2 == 0 is not checked, causing the function will not return (receiver, 0, 0) even commissionInfo.commissionRate2 == 0.

- **Related Code**

```
149     if (!commissionInfo.isFromTokenCommission || commissionInfo.commissionRate == 0 ||
150         commissionInfo.commissionRate == 0) {
151             return (receiver, 0, 0);
152         }
153
154         uint256 totalCommissionAmount = 0;
```

- **Recommendation**

The commissionInfo.commissionRate2 == 0 should be checked instead of checking commissionInfo.commissionRate == 0 twice.

- **Project Team Feedback**

Team Response	Fixed
Re-audit Result	Confirmed

3.8 Potential Centralization Risks

Location	File	Status	Severity
Line 413	DexRouter.sol	Confirmed	!Low

- **Description**

The protocol implements several privileged account types (owner, admin, and priorityAddresses) that introduce centralization risks. The owner and admin roles possess administrative privileges to modify critical protocol parameters, including the ability to designate admin accounts and configure priorityAddresses. Additionally, accounts with priorityAddresses status have exclusive authorization to execute swap orders through the XBridge functionality.

- **Related Code**

```
413     function setPriorityAddress(address _priorityAddress, bool valid) external {
414         require(msg.sender == admin || msg.sender == owner(), "na");
415         priorityAddresses[_priorityAddress] = valid;
416         emit PriorityAddressChanged(_priorityAddress, valid);
417     }
418
419     function setProtocolAdmin(address _newAdmin) external {
420         require(msg.sender == admin || msg.sender == owner(), "na");
421         admin = _newAdmin;
422         emit AdminChanged(_newAdmin);
423     }
```

- **Recommendation**

Implement multi-signature governance to manage privileged accounts and separate owner, admin, and priorityAddresses roles. Add timelock mechanisms on privileged functions to enhance security and reduce centralization risks.

- **Project Team Feedback**

Team Response	Confirmed. Owner is passed to asset management team.
Re-audit Result	Confirmed

3.9 Inconsistent ETH Balance Retrieval in _getBalanceOf()

Location	File	Status	Severity
Line 68	CommissionLib.sol	Fixed	!Info

- **Description**

In the _getBalanceOf() function, two parameters (token, user) are used. In the ERC20 token branch, it correctly reads the user's token balance, while in the ETH branch, it uses selfbalance() to directly read the current contract's ETH balance instead of the user's ETH balance.

- **Related Code**

```
68      function _getBalanceOf(
69          address token,
70          address user
71      ) internal returns (uint256 amount) {
72          assembly {
73  >           function _revertWithReason(m, len) {…
84              }
85              switch eq(token, _ETH)
86              case 1 {
87                  amount := selfbalance()
88              }
89  >           default {…
105             }
106         }
107     }
```

- **Recommendation**

Replace selfbalance() with balance(user) in the ETH handling branch to ensure the function consistently returns the specified user's balance regardless of token type.

- **Project Team Feedback**

Team Response	Fixed
Re-audit Result	Confirmed

4. Disclaimer

This audit report only covers the specific audit types stated herein. We assume no responsibility for unknown security vulnerabilities outside this scope.
We rely on audit reports issued before existing attacks or vulnerabilities are published. For future or new vulnerabilities, we cannot guarantee project security impact and assume no responsibility.
Our security audit analysis should be based on documents provided by the project before report release (including contract code). These materials should not contain false information, tampering, deletion, or concealment. If provided materials are false, inaccurate, missing, tampered, deleted, concealed, or modified after report release, we assume no responsibility for resulting losses and adverse effects.
The project team should understand our audit report is based on provided materials and current technical capabilities. Due to institutional technical limitations, our report may not detect all risks. We encourage continued testing and auditing by the development team and stakeholders.
The project team must ensure compliance with applicable laws and regulations.
The audit report is for reference only. Its content, acquisition methods, usage, and related services cannot serve as basis for investment, taxation, legal, regulatory, or construction decisions. Without our prior written consent, the project team may not reference, cite, display, or distribute report content to third parties. Any resulting losses shall be borne by the project team.
This report does not cover contract compiler bugs or scope beyond programming languages. Smart contract risks from underlying vulnerabilities should be borne by the project team.
Force majeure includes unforeseeable, unavoidable events like wars, natural disasters, strikes, epidemics, and legal/regulatory changes preventing contract performance. When occurring, neither party breaches contract obligations. For unaffected economic responsibilities, the project team should pay for completed work.

5. About Us

OKX Web3 Audit Team specializes in blockchain security with expertise in smart contract auditing, token security assessment, and Web3 security tool development. We provide comprehensive security solutions for Web3 projects, conduct pre-listing token audits, and develop security tools to protect OKX Web3 wallet users. Our team combines automated analysis with manual review to deliver thorough security assessments and maintain the highest security standards in the Web3 ecosystem.

PASSED

This audit has been conducted to review the OKX DEX Router project's Solidity-based smart contracts running on EVM chains, examining design, architecture, and implementation to identify potential vulnerabilities

OKX Web3 Audit Team