Sui DexRouter Support Commission + Multihop

Security Audit Report

Prepared by

OKX Web3 Audit Team

September 2025

Contents

1. Overview

1.1 Project Introduction

DEX Router Sui is a sophisticated decentralized exchange aggregation and routing system built on the Sui blockchain that enables optimal token swapping across multiple decentralized exchanges (DEXs) and protocols. The system acts as a unified interface for executing complex multi-path swaps, providing users with the best possible rates by intelligently splitting orders across different liquidity sources within the Sui ecosystem.

1.2 Audit Summary

Ecosystem	Sui Blockchain
Language	Move
Repository	https://github.com/okxlabs/DEX-Router-Sui-V1
Base Commit	
Final Commit	

1.3 Audit Scope

```
DEX-Router-Sui-V1/
├── dexrouter/
│   ├── Move.toml
│   └── sources/
│       └── router.move
└── dexrouter-extended/
    ├── Move.toml
    └── sources/
        └── router.move
```

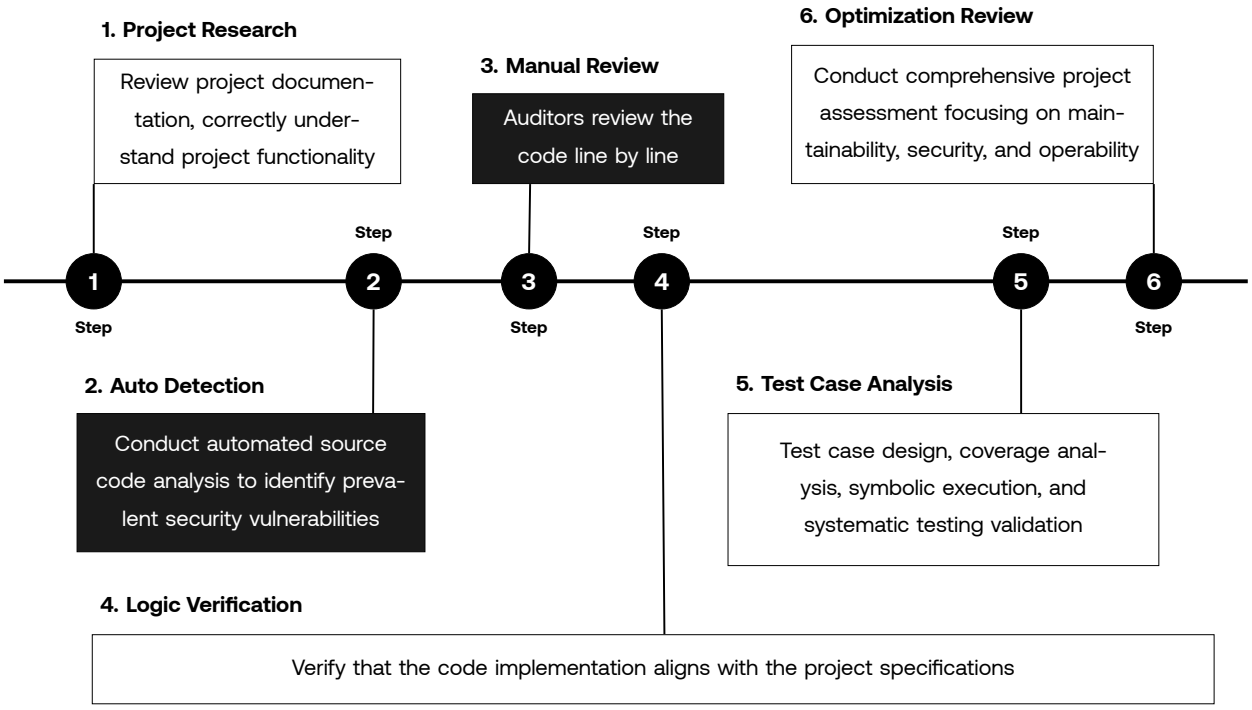
2. Audit Summary

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.

Risk Level	Risk Description
Critical	Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
High	High risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
Medium	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
Low	Low risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
Informational	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

2.4 Results

The audit results are 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	1	0	4	5

Vulnerability list

No.	Severity	Vulnerability	Category	Status
1	Informational	Improper Parameter Naming Convention	Code Quality	Fixed
2	Medium	Missing Slippage Protection in Swap Functions	Security	Fixed
3	Informational	Suboptimal Function Visibility	Code Quality	Acknowledged
4	Informational	Missing Referral Address Validation	Security	Fixed
5	Informational	Code Quality and Style Improvements	Code Quality	Partially Fixed

Status Definitions

- **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.

3.1 Informational - Improper Parameter Naming Convention

Location	File	Category	Status	Severity
Line 49, 167, 184	router.move, pool.move	Code Quality	Fixed	Informational

Description

The TxContext parameter in functions suiswap_y_2_x, do_swap_y_to_x, and do_swap_x_to_y is improperly named as _arg5 instead of following the conventional naming pattern ctx. This violates basic coding principles and reduces code readability and maintainability.

Related Code

```
116 //@OKLink Audit Description: Inappropriate TxContext parameter naming
117 //@OKLink Audit Solution: Rename TxContext parameter to ctx
118 entry fun suiswap_y_2_x<Ty0, Ty1>(
119     pool: &mut pool::Pool<Ty0, Ty1>,
120     coins: vector<Coin<Ty1>>,
121     amount_in: u64,
122     min_out: u64,
123     order_id: u64,
124     decimal: u8,
125     _arg5: &mut TxContext  // Should be renamed to ctx
126 ) {
127     pool::do_swap_y_to_x<Ty0, Ty1>(
128         pool,
129         coins,
130         amount_in,
131         min_out,
132         _arg5
133     );
134 
135     // event
136     event::emit(OrderRecord {
137         order_id: order_id,
138         decimal: decimal
139     })
140 }
```

Recommendation

Rename the TxContext parameter from _arg5 to ctx in all affected functions to follow Move language conventions and improve code readability. This change should be applied consistently across all functions that accept TxContext parameters.

Project Team Feedback

Team Response	Accepted and fixed the parameter naming issue
Re-audit Result	Confirmed as fixed

3.2 Medium - Missing Slippage Protection in Swap Functions

Location	File	Category	Status	Severity
Line 201	router.move	Security	Fixed	Medium

Description

The router::suiswap_y_2_x() and router::suiswap_x_2_y() functions lack essential slippage protection mechanisms. Without proper validation of the min_out parameter, users are vulnerable to sandwich attacks and unfavorable price movements during token swaps. The min_out parameter is accepted but not properly validated or enforced in the swap execution.

Related Code

```
entry fun suiswap_y_2_x<Ty0, Ty1>(
    pool: &mut suiswap_pool::Pool<Ty0, Ty1>,
    coins: vector<Coin<Ty1>>,
    amount_in: u64,
    min_out: u64, //@audit-issue > 0
    clock: &Clock,
    order_id: u64,
    decimal: u8,
    ctx: &mut TxContext
) {
    [...]

    //@audit-info coin_x_value > min_out
    transfer::public_transfer(coin_x_out, 0x2::tx_context::sender(ctx));

    event::emit(OrderRecord {
        order_id: order_id,
        decimal: decimal,
        out_amount: coin_x_value
    });
}

public fun suiswap_y_2_x_with_return<Ty0, Ty1>(
    pool: &mut suiswap_pool::Pool<Ty0, Ty1>,
    coins: vector<Coin<Ty1>>,
    amount_in: u64,
    _min_out: u64,//@audit-issue not used
    clock: &Clock,
    _order_id: u64,
    _decimal: u8,
    ctx: &mut TxContext
):(Coin<Ty0>, u64) {
    [...]
}
```

Recommendation

Implement comprehensive slippage protection by:

- Adding non-zero validation for the min_out parameter in suiswap_y_2_x() and suiswap_x_2_y() functions

- Ensuring that the received tokens are greater than or equal to min_out
- Extending similar validation to other swap functions including movepump_buy(), movepump_sell(), cetus_swap_a2b(), cetus_swap_b2a(), turbos_swap_a_b(), turbos_swap_b_a(), bluemove_swap_exact_input(), bluemove_stable_swap_exact_input(), and finalize()
- Adding proper error handling for cases where slippage tolerance is exceeded

Project Team Feedback

Team Response	Partially accepted. For both single-hop and multi-hop operations, min_return checks are performed in the finalize() function when assembling PTB. The current backend practice sets minout to 0 for each hop and only performs checks in the final finalize() function.
Re-audit Result	Confirmed as fixed

3.3 Informational - Suboptimal Function Visibility

Location	File	Category	Status	Severity
Line 659	router.move	Code Quality	Acknowledged	Informational

Description

Several functions in the router module are defined with public visibility but are only used internally within the module. Functions such as split_with_percentage and split_with_percentage_for_commission could have their visibility restricted to private to limit external access and improve encapsulation.

Related Code

```
173 public fun split_with_percentage<T>(
174     coin: &mut Coin<T>,
175     percentage: u64,
176     ctx: &mut TxContext
177 ): (Coin<T>, u64, Coin<T>, u64) {
178     assert!(percentage <= 10000, E_PERCENTAGE);
179     let total_value = coin::value(coin);
180     let split_amount = (total_value * percentage) / 10000;
181     let split_coin = coin::split(coin, split_amount, ctx);
182     (split_coin, split_amount, coin::split(coin, total_value - split_amount, ctx),
        ↪  total_value - split_amount)
183 }
184
185 public fun split_with_percentage_for_commission<T>(
186     coin: &mut Coin<T>,
187     commissionRate: u64,
188     referalAddress: address,
189     ctx: &mut TxContext
190 ): (Coin<T>, u64) {
191     assert!(commissionRate <= 300, E_COMMISSION_PERCENTAGE);
192     let (coin_split, coin_split_value, coin_left, coin_left_value) =
        ↪  split_with_percentage(coin, commissionRate, ctx);
193     transfer::public_transfer(coin_split, referalAddress);
194     event::emit(CommissionRecord{
195         commission_amount: coin_split_value,
196         referal_address: referalAddress
197     });
198     (coin_left, coin_left_value)
199 }
```

Recommendation

Review the usage patterns of public functions and consider changing the visibility of functions that are only used internally to private. This would:

- Improve module encapsulation
- Reduce the public API surface
- Prevent unintended external usage
- Make the module interface cleaner and more maintainable

Project Team Feedback

Team Response	Rejected. These functions need to be called externally.
Re-audit Result	Confirmed as not requiring resolution

3.4 Informational - Missing Referral Address Validation

Location	File	Category	Status	Severity
Line 621	router.move	Security	Fixed	Informational

Description

The finalize function accepts a referalAddress parameter for receiving commission funds but does not validate that the address is non-zero. If an incorrect or zero address is provided, commission funds could be lost or sent to an unintended recipient, resulting in financial loss.

Related Code

```
223 public fun finalize<CoinType>(
224     coin: Coin<CoinType>,
225     min_amount: u64,
226     toCommissionRate: u64,
227     referalAddress: address,
228     order_id: u64,
229     decimal: u8,
230     ctx: &mut TxContext,
231 ) {
232     let amount_out = coin::value(&coin);
233     if (amount_out < min_amount) {
234         abort E_ROUTER_MIN_RETURN_NOT_REACH
235     };
236 
237     event::emit(OrderRecord{
238         order_id: order_id,
239         decimal: decimal,
240         out_amount: amount_out
241     });
242 
243     if(toCommissionRate > 0){//@audit-info && referalAddress non-zero address
244         let (coin_left, _coin_left_value) = split_with_percentage_for_commission(&mut coin,
        ↪  toCommissionRate, referalAddress, ctx);
245         destroy_or_transfer<CoinType>(coin_left, ctx);
246         destroy_or_transfer<CoinType>(coin, ctx);
247     } else {
248         destroy_or_transfer<CoinType>(coin, ctx);
249     }
250 }
```

Recommendation

Add validation to ensure the referalAddress is not a zero address when commission rate is greater than zero:

- Implement a check for referalAddress != @0x0 before processing commission
- Add appropriate error handling with a descriptive error message
- Consider adding additional validation for address format if required
- Ensure the validation occurs before any commission transfer operations

Project Team Feedback

Team Response	Accepted.
Re-audit Result	Confirmed as fixed

3.5 Informational - Code Quality and Style Improvements

Location	File	Category	Status	Severity
Various	router.move	Code Quality	Partially Fixed	Informational

Description

Several areas in the router contract code can be optimized to improve code style, readability, and efficiency. These improvements include consistent use of transfer functions, proper use of constant variables, conditional transfers, and unnecessary assertions.

Related Code

```
// Multiple code quality issues identified:
// 1. Inconsistent coin transfer methods
// 2. Missing constant variable usage
// 3. Unnecessary transfers when value is zero
// 4. Redundant assertions in utility functions

// Example of improved constant usage:
const E_PERCENTAGE: u64 = 1;
const E_COMMISSION_PERCENTAGE: u64 = 2;
const MAX_PERCENTAGE: u64 = 10000;
const MAX_COMMISSION_RATE: u64 = 300;

// Example of conditional transfer optimization:
if (coin_split_value > 0) {
    transfer::public_transfer(coin_split, referalAddress);
}
```

Recommendation

Implement the following code quality improvements:

- Use consistent destroy_or_transfer() function for coin transfers throughout the contract
- Implement proper constant variables for magic numbers and error codes
- Add conditional checks to only perform transfer operations when coin values are greater than zero
- Review and optimize function visibility where appropriate
- Remove redundant assertions while maintaining necessary validation
- Ensure consistent code formatting and naming conventions

Project Team Feedback

Team Response	Mixed response to different recommendations: • Partially accepted coin transfer optimization • Accepted constant variable usage • Accepted conditional transfer with assertion • Rejected function visibility changes - functions need to remain public for external calls
Re-audit Result	Confirmed as addressed where applicable

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 OKX Web3 Audit Team

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 OKX's internal 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.