MOVEBIT
Securing the Move Ecosystem

Sui-AMM-swap Contracts **Audit Report**

https://twitter.com/movebit_

contact@movebit.xyz

Sui-AMM-swap Contracts Audit Report

1 Executive Summary

1.1 Project Information

Type	DEX
Auditors	MoveBit
Timeline	2022-11-16 to 2022-11-30
Languages	Move
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	Repository: https://github.com/OmniBTC/Sui-AMM-swap Received Commit: 084836dd4c523a85b2d33baa3c4796a1b15fd87 Last Reviewed Commit: fa450398976c15e2e7b9b0e56156274188bfd6dd
Updates	Fixed issue 6.6 on February 21, 2023, Commit: 0de3574e471b8cc13b36b2184c4fa7d0747ff24f

1.2 Issue Statistic

Item	**Count**	**Fixed**	**Pending**
Total	7	7	
Minor			
Medium	6	6	

Major	1	1	
Critical			

1.3 Issue Level

- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

1.4 Issue Status

- **Fixed:** The issue has been resolved.
- **Pending:** The issue has been acknowledged by the code owner, but has not yet been resolved. The code owner may take action to fix it in the future.

2 Summary of Findings

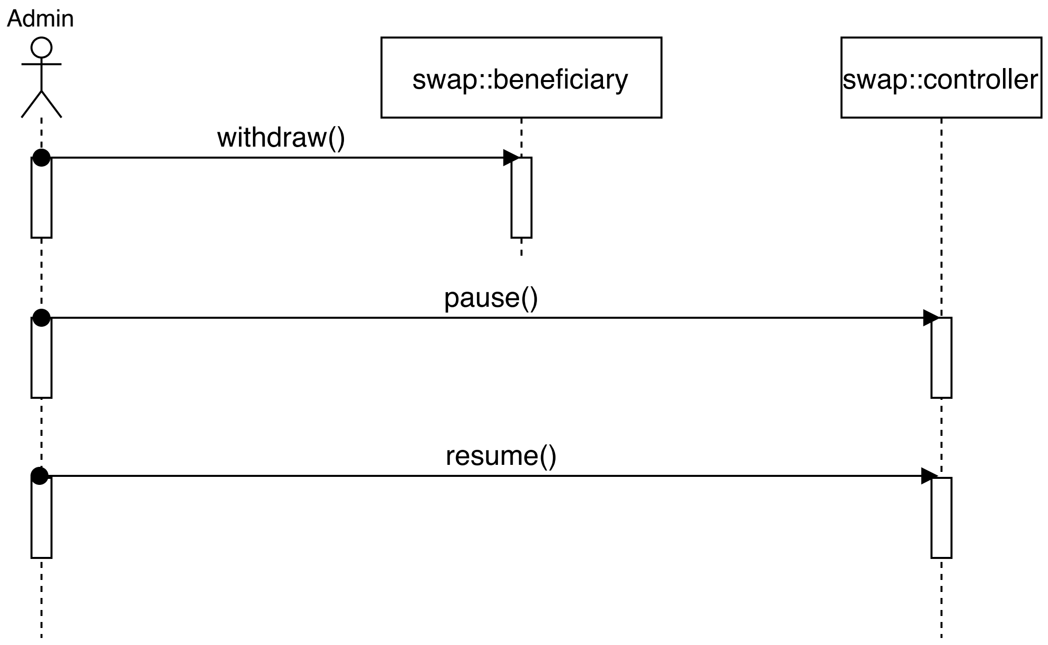
The first open source AMM swap on the Sui. Our team mainly focused on reviewing the Code Security and normative, then conducted code running tests and business logic security tests on the test net, Our team has been in close contact with the developing team for the past few days. As a result, Our team found a total of 7 issues. The team discussed these issues together, and the development team has fixed these 7 issues.

3 Participant Process

Here are the relevant actors with their respective abilities within the Sui-AMM-swap Smart Contract:

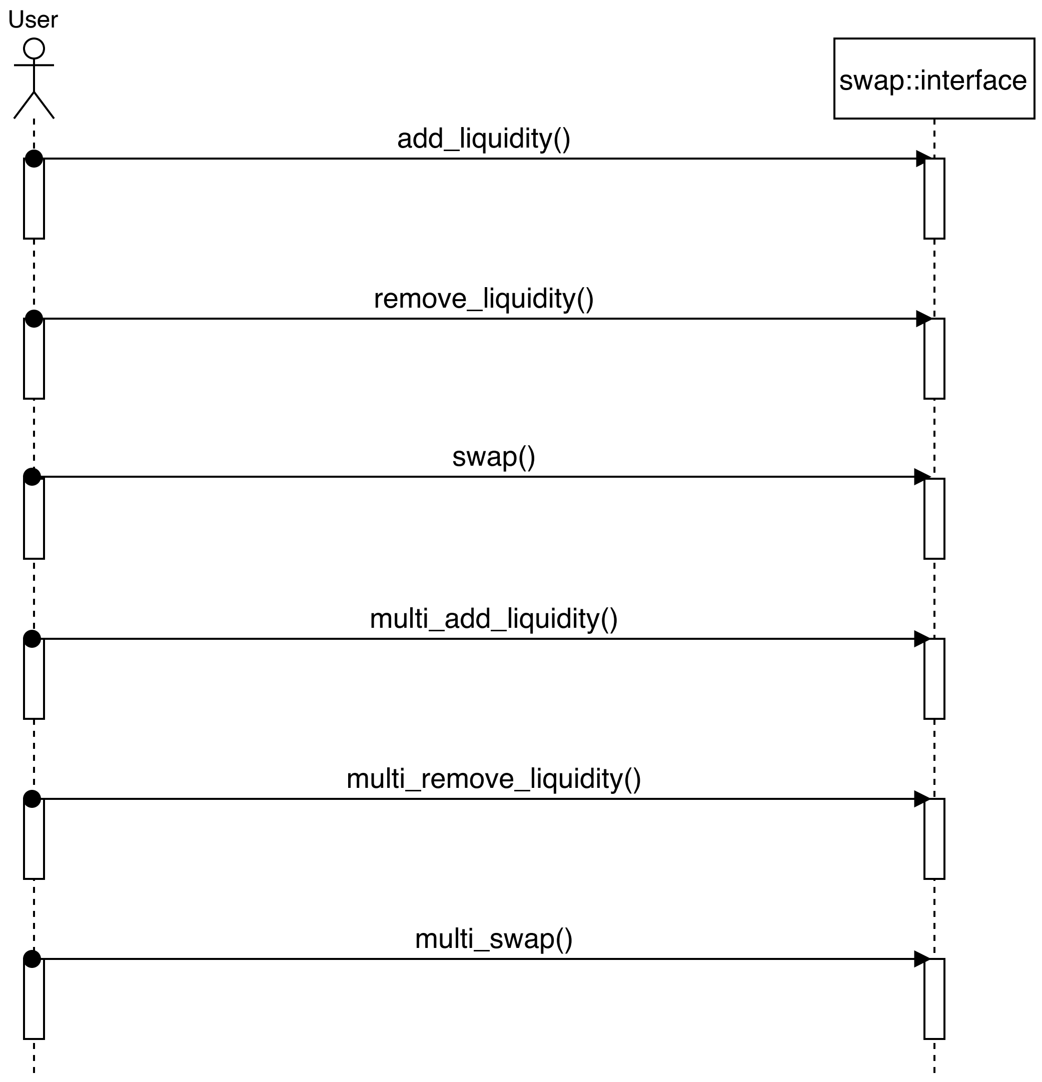
(1) Admin

- Admin can transfer withdraw fee coins to the beneficiary.
- Admin can pause all pools under the global.
- Admin can resume all pools under the global.



(2) User

- User can add liquidity.
- User can remove liquidity.
- User can swap tokens.
- User can multi-add liquidity.
- User can multi-remove liquidity.
- User can multi-swap.



4 MoveBit Audit BreakDown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Denial of service / logical oversights

- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

5 Methodology

The security team adopted the "**Testing and Automated Analysis**", "**Code Review**" and "**Formal Verification**" strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", and that can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

Code scope sees **Appendix 1**.

(3) Formal Verification

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest

stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);

- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

6 Findings

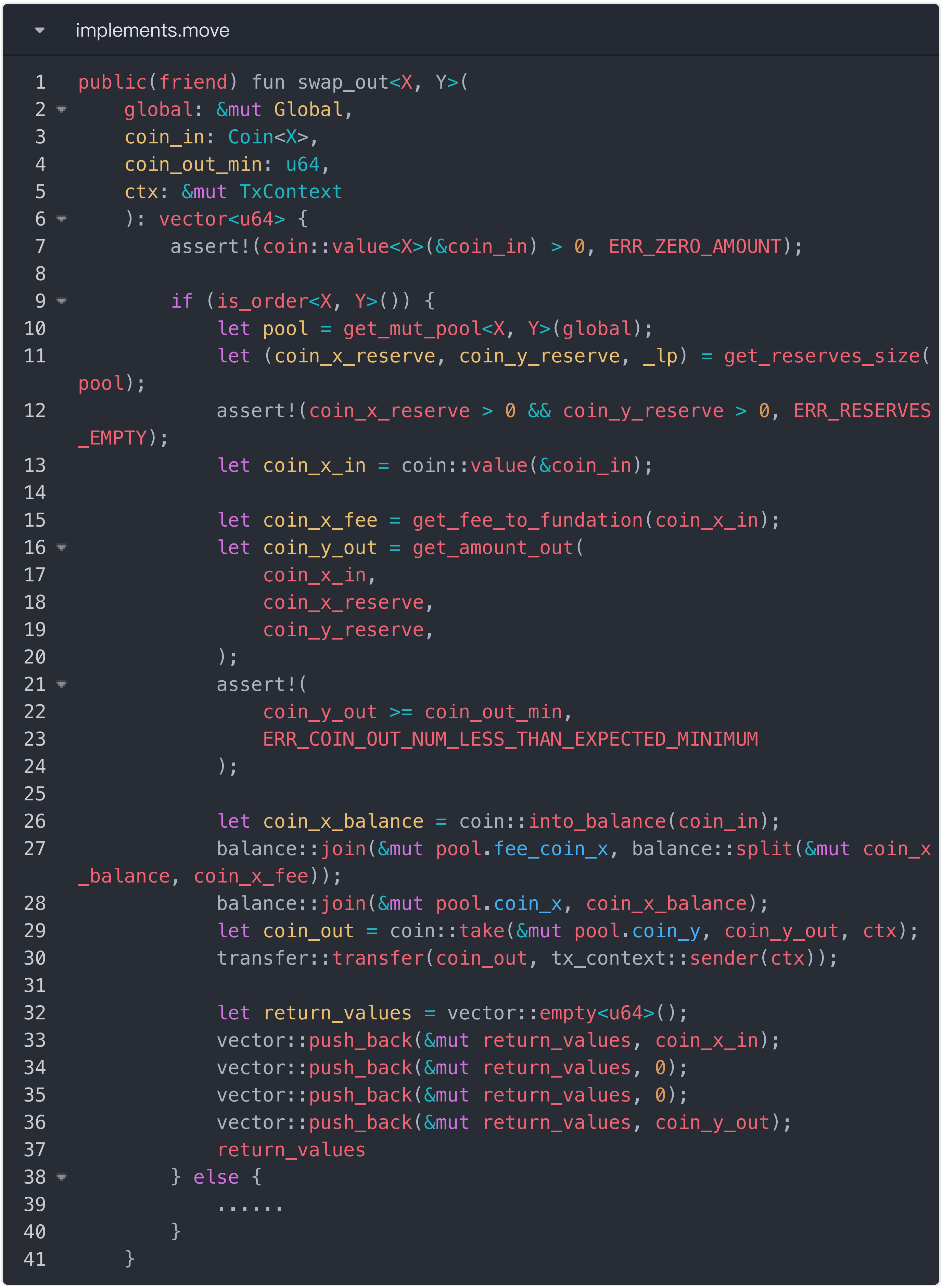
6.1 Missing check K value after swap

Severity: Medium

Status: Fixed

Descriptions: In the function `swap_out()`, it is necessary to check whether the product of the token quantity of the `token` pair after the exchange is strictly greater than or equal to the `K` value. However, since there is a handling fee in the swap process, the product of the number of tokens in the swapped `token` pair should be greater than the previous `K` value.

Code Location: sources/implements.move, line 302.


implements.move

```
public(friend) fun swap_out<X, Y>(
    global: &mut Global,
    coin_in: Coin<X>,
    coin_out_min: u64,
    ctx: &mut TxContext
    ): vector<u64> {
        assert!(coin::value<X>(&coin_in) > 0, ERR_ZERO_AMOUNT);

        if (is_order<X, Y>()) {
            let pool = get_mut_pool<X, Y>(global);
            let (coin_x_reserve, coin_y_reserve, _lp) = get_reserves_size(pool);
            assert!(coin_x_reserve > 0 && coin_y_reserve > 0, ERR_RESERVES_EMPTY);
            let coin_x_in = coin::value(&coin_in);

            let coin_x_fee = get_fee_to_fundation(coin_x_in);
            let coin_y_out = get_amount_out(
                coin_x_in,
                coin_x_reserve,
                coin_y_reserve,
            );
            assert!(
                coin_y_out >= coin_out_min,
                ERR_COIN_OUT_NUM_LESS_THAN_EXPECTED_MINIMUM
            );

            let coin_x_balance = coin::into_balance(coin_in);
            balance::join(&mut pool.fee_coin_x, balance::split(&mut coin_x_balance, coin_x_fee));
            balance::join(&mut pool.coin_x, coin_x_balance);
            let coin_out = coin::take(&mut pool.coin_y, coin_y_out, ctx);
            transfer::transfer(coin_out, tx_context::sender(ctx));

            let return_values = vector::empty<u64>();
            vector::push_back(&mut return_values, coin_x_in);
            vector::push_back(&mut return_values, 0);
            vector::push_back(&mut return_values, 0);
            vector::push_back(&mut return_values, coin_y_out);
            return_values
        } else {
            ......
        }
    }
```

Suggestion: It is recommended to add an `assert!` for `pool.coin_x * pool.coin_y > coin_x_reserve * coin_y_reserve`.

implements.move

```
public(friend) fun swap_out<X, Y>(
    global: &mut Global,
    coin_in: Coin<X>,
    coin_out_min: u64,
    ctx: &mut TxContext
): vector<u64> {
    assert!(coin::value<X>(&coin_in) > 0, ERR_ZERO_AMOUNT);

    if (is_order<X, Y>()) {
        ......
        let (new_reserve_x, new_reserve_y, _lp) = get_reserves_size(pool);
        assert!(
            (coin_x_reserve as u128) * (coin_y_reserve as u128)
                < (new_reserve_x as u128) * (new_reserve_y as u128),
            14
        )

        let return_values = vector::empty<u64>();
        vector::push_back(&mut return_values, coin_x_in);
        vector::push_back(&mut return_values, 0);
        vector::push_back(&mut return_values, 0);
        vector::push_back(&mut return_values, coin_y_out);
        return_values
    } else {
        ......
        let (new_reserve_x, new_reserve_y, _lp) = get_reserves_size(pool);
        assert!(
            (coin_x_reserve as u128) * (coin_y_reserve as u128)
                < (new_reserve_x as u128) * (new_reserve_y as u128),
            14
        )

        let return_values = vector::empty<u64>();
        vector::push_back(&mut return_values, 0);
        vector::push_back(&mut return_values, coin_x_out);
        vector::push_back(&mut return_values, coin_y_in);
        vector::push_back(&mut return_values, 0);
        return_values
    }
}
```

6.2 There is no minting of minimum liquidity, resulting in reduced attack costs

Severity: Medium

Status: Fixed

Descriptions: In the function `add_liquidity()` , if it is the first injection of liquidity, the number of `lp` tokens obtained will be subtracted from the minimum liquidity value (`MINIMAL_LIQUIDITY`). The function of `MINIMAL_LIQUIDITY` is to limit the lower limit of `lp` supply, thereby reducing the unit price of `lp token` and increasing the attack cost of `lp` price manipulation.

This value is directly subtracted in the code, so the value of `lp_supply` does not increase, and this part should be `mint` and stored in an address instead of being directly subtracted.

Code Location: sources/implements.move, line 234.

implements.move

```
let provided_liq = if (0 == lp_supply) {
    let initial_liq = math::sqrt(optimal_coin_x) * math::sqrt(optimal_coin_y);
    assert!(initial_liq > MINIMAL_LIQUIDITY, ERR_LIQUID_NOT_ENOUGH);
    initial_liq - MINIMAL_LIQUIDITY
} else {
    ......
};

```

Suggestion: Call `balance::increase_supply` to increase the total amount of `lp_supply` and transfer it to `@controller` address.

implements.move

```
let provided_liq = if (0 == lp_supply) {
    let initial_liq = math::sqrt(optimal_coin_x) * math::sqrt(optimal_coin_y);
    assert!(initial_liq > MINIMAL_LIQUIDITY, ERR_LIQUID_NOT_ENOUGH);

    let minimal_liquidity_balance = balance::increase_supply(&mut pool.lp_supply, MINIMAL_LIQUIDITY);
    let minimal_liquidity_coin = coin::from_balance(minimal_liquidity_balance, ctx);
    transfer::transfer(minimal_liquidity_coin, @controller);

    initial_liq - MINIMAL_LIQUIDITY

} else {
    ......
};

```

6.3 Multi related functions do not limit the empty Vector

Severity: Medium

Status: Fixed

Descriptions: The functions `multi_add_liquidity`, `multi_remove_liquidity`, and `multi_swap` first use the `pop_back` function for `coins_in` and `lp_coin` in the code to pop up the last element of the `vector`, but this does not judge that the length of the `vector` is 0.

Code Location: sources/interface.move, line 139 and line 190 and line 209.

interface.move

```
public entry fun multi_add_liquidity<X, Y>(
    global: &mut Global,
    coins_x: vector<Coin<X>>,
    coins_x_value: u64,
    coin_x_min: u64,
    coins_y: vector<Coin<Y>>,
    coins_y_value: u64,
    coin_y_min: u64,
    ctx: &mut TxContext
) {
    assert!(!implements::is_emergency(global), ERR_EMERGENCY);

// 1. merge coins
let merged_coin_x = vector::pop_back(&mut coins_x);
......
    let merged_coin_y = vector::pop_back(&mut coins_y);

......
}
```

Suggestion: Add an assert to limit the length of the vector to be greater than 0.

interface.move

```
public entry fun multi_add_liquidity<X, Y>(
    global: &mut Global,
    coins_x: vector<Coin<X>>,
    coins_x_value: u64,
    coin_x_min: u64,
    coins_y: vector<Coin<Y>>,
    coins_y_value: u64,
    coin_y_min: u64,
    ctx: &mut TxContext
) {
    assert!(!implements::is_emergency(global), ERR_EMERGENCY);
assert!(
    !vector::is_empty(&coins_x) && !vector::is_empty(&coins_y),
    105
);

// 1. merge coins
let merged_coin_x = vector::pop_back(&mut coins_x);
......
    let merged_coin_y = vector::pop_back(&mut coins_y);

......
}
```

6.4 Wrong event access permission

Severity: Medium

Status: Fixed

Descriptions: The visibility of `emit` functions in the project is public, so anyone can call these functions to `emit` events. If hacker directly calls the emit function, he can pretend that he has successfully called `add_liquidity/remove_liquidity/swap` , which may cause logic errors in other code.

Code Location: sources/event.move.

event.move

```move
public fun added_event(
    global: ID,
    lp_name: String,
    coin_x_val: u64,
    coin_y_val: u64,
    lp_val: u64
) {
    emit(
        AddedEvent {
            global,
            lp_name,
            coin_x_val,
            coin_y_val,
            lp_val
        }
    )
}
```

Suggestion: Use `friend` to limit the call permission of the function.

event.move

```move
public(friend) fun added_event(
    global: ID,
    lp_name: String,
    coin_x_val: u64,
    coin_y_val: u64,
    lp_val: u64
) {
    emit(
        AddedEvent {
            global,
            lp_name,
            coin_x_val,
            coin_y_val,
            lp_val
        }
    )
}
```

6.5 Sqrt function precision error

Severity: Medium

Status: Fixed

Descriptions: In the function `add_liquidity()` , When injecting liquidity for the first time, the number of `lp` tokens obtained should be the square root of the multiplication of the two injected tokens, but the calculation method in the code is based on the method of first extracting the square and then multiplying, which may cause accuracy problems.

Code Location: sources/implements.move, line 232.

implements.move

```
public(friend) fun add_liquidity<X, Y>(
    pool: &mut Pool<X, Y>,
    coin_x: Coin<X>,
    coin_x_min: u64,
    coin_y: Coin<Y>,
    coin_y_min: u64,
    ctx: &mut TxContext
): (Coin<LP<X, Y>>, vector<u64>) {
    ......

    let provided_liq = if (0 == lp_supply) {
        let initial_liq = math::sqrt(optimal_coin_x) * math::sqrt(optimal_coin_y);
        assert!(initial_liq > MINIMAL_LIQUIDITY, ERR_LIQUID_NOT_ENOUGH);
        initial_liq - MINIMAL_LIQUIDITY
    } else {
        ......
    };

   ......
}
```

Suggestion: Use the `sqrt` function with a higher number of digits and multiply first and then square root.

implements.move

```
public(friend) fun add_liquidity<X, Y>(
    pool: &mut Pool<X, Y>,
    coin_x: Coin<X>,
    coin_x_min: u64,
    coin_y: Coin<Y>,
    coin_y_min: u64,
    ctx: &mut TxContext
): (Coin<LP<X, Y>>, vector<u64>) {
    ......

    let provided_liq = if (0 == lp_supply) {
        let initial_liq = math::sqrt(math::mul_to_u128(optimal_coin_x, optimal_coin_y));
        assert!(initial_liq > MINIMAL_LIQUIDITY, ERR_LIQUID_NOT_ENOUGH);
        initial_liq - MINIMAL_LIQUIDITY
    } else {
        ......
    };

    ......
}
```

6.6 Add an interface to modify the controller as a multi-signature account

Severity: Medium

Status: Fixed

Descriptions: At present, the `@controller` address has great authority and can control the status of the entire contract. In order to ensure asset security, it is recommended to add an interface to support changing the `@controller`. When SUI supports multi-signature accounts in the future, the community can easily change `@controller` to a multi-signature account, and make the contract to be much safer.

Suggestion: Add the following codes in `controller.move` and `implements.move` respectively.

controller.move

```
public entry fun modify_controller(global: &mut Global, new_controller: address,
                                    ctx: &mut TxContext) {
        assert!(implements::controller(global) == tx_context::sender(ctx), ERR_NO_PERMISSIONS);
        implements::modify_controller(global, new_controller)
    }
}
```

implements.move

```
public(friend) fun modify_controller(global: &mut Global, new_controller: address) {
    global.controller = new_controller
}
```

6.7 Missing zero check for added liquidity

Severity: Major

Status: Fixed

Descriptions: In the function `add_liquidity` , a zero check is missing for the `provided_liq` . If a user does not provide enough `coins<X>` and `coins<Y>` to add liquidity, the user will lose `coins<X>` and `coins<Y>` assets, and receive no `Coin<LP<X, Y>>` token.

Code Location: sources/implements.move, line 203.

implements.move

```
public(friend) fun add_liquidity<X, Y>(
    pool: &mut Pool<X, Y>,
    coin_x: Coin<X>,
    coin_x_min: u64,
    coin_y: Coin<Y>,
    coin_y_min: u64,
    ctx: &mut TxContext
): (Coin<LP<X, Y>>, vector<u64>) {
    ......

    let provided_liq = if (0 == lp_supply) {
        let initial_liq = math::sqrt(optimal_coin_x) * math::sqrt(optimal_coin_y);
        assert!(initial_liq > MINIMAL_LIQUIDITY, ERR_LIQUID_NOT_ENOUGH);
        initial_liq - MINIMAL_LIQUIDITY
    } else {
        let x_liq = (lp_supply as u128) * (optimal_coin_x as u128) / (coin_x_reserve as u128);
        let y_liq = (lp_supply as u128) * (optimal_coin_y as u128) / (coin_y_reserve as u128);
        if (x_liq < y_liq) {
            assert!(x_liq < (U64_MAX as u128), ERR_U64_OVERFLOW);
            (x_liq as u64)
        } else {
            assert!(y_liq < (U64_MAX as u128), ERR_U64_OVERFLOW);
            (y_liq as u64)
        }
    };

    ......
}
```

Suggestion: Add an `assert!` on `provided_liq` , if it is equal to zero, aborts the transaction.

implements.move

```
public(friend) fun add_liquidity<X, Y>(
    pool: &mut Pool<X, Y>,
    coin_x: Coin<X>,
    coin_x_min: u64,
    coin_y: Coin<Y>,
    coin_y_min: u64,
    ctx: &mut TxContext
): (Coin<LP<X, Y>>, vector<u64>) {
    ......

    let provided_liq = if (0 == lp_supply) {
        let initial_liq = math::sqrt(optimal_coin_x) * math::sqrt(optimal_coin_y);
        assert!(initial_liq > MINIMAL_LIQUIDITY, ERR_LIQUID_NOT_ENOUGH);
        initial_liq - MINIMAL_LIQUIDITY
    } else {
        let x_liq = (lp_supply as u128) * (optimal_coin_x as u128) / (coin_x_reserve as u128);
        let y_liq = (lp_supply as u128) * (optimal_coin_y as u128) / (coin_y_reserve as u128);
        if (x_liq < y_liq) {
            assert!(x_liq < (U64_MAX as u128), ERR_U64_OVERFLOW);
            (x_liq as u64)
        } else {
            assert!(y_liq < (U64_MAX as u128), ERR_U64_OVERFLOW);
            (y_liq as u64)
        }
    };

    const ERR_INSUFFICIENT_LIQUIDITY_MINTED: u64 = 15;
    assert!(provided_liq > 0, ERR_INSUFFICIENT_LIQUIDITY_MINTED);
    ......
}
```

Appendix 1 – Files in Scope

The following are the SHA1 hashes of the last reviewed files.

Files	SHA-1 Hash

sources/beneficiary.move	c9d1bf9fc31509769d1d588cdb51e0050bcfa8f7
sources/interface.move	05f71edc29f6cb922803950a0bdd10d9b9ae75fa
sources/math.move	0d02552fee51c3c1cc2e832f32171308f956daa2
sources/comparator.move	8caaaec2267d7c05fa6367dcf9444c09e091095e
sources/event.move	b858c7f036a09716b04ee1e35c89afc947b12442
sources/controller.move	a14fd0bcb4c9c5e10a968e4e678c892065da46ae
sources/implements.move	b2bd6b7659901ea8584dea697d1ce34195c57993
Move.toml	32de0edb470e80bba7f9a325e33a573a98af5bc0
test_coins/sources/faucet.move	ba396cd810b6633f6f4cd2d4b06e79bea3ba6a2d
test_coins/sources/coins.move	e2d45a15a50d59c3a4d96ffa15332e2f2e5ec254
test_coins/Move.toml	94ed29619e4798080912809aba9185b5d29ea7b6

Appendix 2 – Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.

MOVEBIT

Securing the Move Ecosystem

https://twitter.com/movebit_

contact@movebit.xyz