Ferra DLMM

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Thu Sep 25 2025

Ferra DLMM Audit Report

1 Executive Summary

1.1 Project Information

Description	This project is the DEX, focusing on liquidity layer infrastructure. Currently, there are two parts: DLMM and CLMM - DLMM is the Dynamic Liquidity Market Maker, the first type on SUI. You can refer to Trader Joe's Liquidity Order Book, or Meteora's DLMM on Solana for reference
Type	DEX
Auditors	Alex,PeiQi
Timeline	Wed Jun 18 2025 - Thu Sep 25 2025
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Ferra-Labs/ferra-dlmm
Commits	0df7c1a9a6630127b0e11dc2a3d499a73a32cf24 2eea25cdbffd3bcbad6b817c31e13fa5370fe536 b66b85b8950d2a8c1537b2efc2bef22a0982c7b0 e5e7aee0fe161e048c09c0a88d51b44e7b1330fb b71c8215831610ecd5d97d82bb43c4f8b00266c1 d8db98838551768d1be6477c7deff5c48dda3ca9

	1c08a51caadede0109f528319e61829703d4e7a7 3566099f5e84c5766e73988866919fd26fe59090 cd692f9926cb963ec53971d9914dabe0e81eeaea de095b82f885cbc7d86a3197d3a6d92da335a8f1 df53af624dea4d7fd87a546c7709d59c5200286e e3cb33436267c1b11c0133fd1b3d6390e4614447

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MOV	Move.toml	85ae2245efa3f2c0047c7e1380c482f5e9c1ebee
BMA	sources/libraries/math/bit_math.move	3a14fb6b2ffe44290e6c21d4469425f215b7fef5
PPH	sources/libraries/pair_parameter_helper.move	5061665413645c6eab5cdc9270155f06bc2c6832
TMA	sources/libraries/math/tree_math.move	68b71378cb190e192ad65d3fb48f47b42f4dad60
LFA	sources/lb_factory.move	d81bb8000a1b09803f19289beaf1cd23d54d2279
FHE	sources/libraries/fee_helper.move	1407cb1497fa5323aeac6c46825f1f85c30d2a7a
PHE	sources/libraries/price_helper.move	0e4d244616e4eb222e0f2d4e59366198cb2e411c
CON	sources/libraries/constants.move	6ef410f0fa49b74cb05c867b65a25a0709d2915d
SMA	sources/libraries/math/safe_math.move	ff3efbb1506f51d7963721b471c565040d6916e9
Q6X6	sources/libraries/math/q64x64.move	4eee515c11ba334e831c72946911d551f0b63807

REW	sources/rewarder.move	ab64695e8ea7f40c34f1832e17f082286c2e5758
CON1	sources/config.move	effdc1e3dde03190d79d3d7bdd44a04e5709de03
LPA	sources/lb_pair.move	bbfdd7709bb36cd248b7cc8e326d7652796a5378
BIN	sources/bin.move	8cb67cf53cc993724ef2c68ad0d95a4038f0aa6d
ACL	sources/acl.move	1adb5e352e053681605526fd30c19e8c57f5f069
LPO	sources/lb_position.move	0c5feec34fd3b0ef0b0fd73364fe41844bb6843d

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	32	32	0
Informational	5	5	0
Minor	6	6	0
Medium	8	8	0
Major	13	13	0
Critical	0	0	0

1.4 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 by bit operations
- 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

1.5 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", which 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

The code scope is illustrated in section 1.2.

(3) Formal Verification(Optional)

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.

2 Summary

This report has been commissioned by Ferra DLMM to identify any potential issues and vulnerabilities in the source code of the Ferra DLMM smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 32 issues of varying severity, listed below.

ID	Title	Severity	Status
ACL-1	`remove_from_all_roles` Permission Removal Is Incomplete	Medium	Fixed
ACL-2	Lack of Vote Threshold Check in `cancel()` Allows Admins to Arbitrarily Cancel Proposals	Medium	Fixed
ACL-3	`propose` Missing Parameter Checking	Minor	Fixed
ACL-4	Repeated error code identification	Informational	Fixed
BIN-1	`sub_fees` Fee Calculation Logical Error	Major	Fixed
BIN-2	`add_reserves_fees` Functions Such As These Should Use Internal Call Modifiers	Informational	Fixed
CON-1	`flashloan_percentage_precission` Spelling Mistake	Informational	Fixed

LFA-1	Missing Validation for `bin_step`	Minor	Fixed
LPA-1	Missing Fee Handling After Fee Collection in `remove_liquidity()` Function	Major	Fixed
LPA-2	Lack of Slippage Protection in `add_liquidity()` and `remove_liquidity()` Functions	Major	Fixed
LPA-3	Missing Reward Collection in `add_liquidity()` and `remove_liquidity()` May Lead to Inaccurate `reward_per_fee_delta` Calculations	Major	Fixed
LPA-4	Incorrect Reward Accrual Due to Delayed Liquidity Addition After Snapshot Update	Major	Fixed
LPA-5	Missing Update to `total_fees_gen` in `remove_liquidity()` Leads to Reduced Reward Calculation	Major	Fixed
LPA-6	Persistent `total_fees` Without Reduction Allows Reward Collection After Position Closure	Major	Fixed
LPA-7	Unsettled Rewards Before reward_factor Update Allow Excess Payouts and Front-Running	Major	Fixed
LPA-8	Bypassing Intended Lock Period by Adding Liquidity After `lock_until_timestamp` Countdown	Major	Fixed

LPA-9	Adding New Rewarder Allows Users to Retroactively Claim Multiple Types of Rewards	Major	Fixed
LPO-1	Potential Out-of-Gas Risk in increase_liquidity() due to Iteration over Excessive Bin IDs	Major	Fixed
LPO-2	Missing Fee Claim Check in `close_position()` May Cause User Fund Loss	Medium	Fixed
LPO-3	`add_bin` Not Used	Minor	Fixed
PPH-1	Missing Validation for `variable_fee_control` and `protocol_share` Upper Limits in `set_static_fee_parameters()`	Informational	Fixed
CON1-1	No Limit To The Traversal Length Of Bins	Medium	Fixed
CON1-2	`add_update_bin_step` And `delete_bin_step` Check For Deficiencies	Minor	Fixed
LPA-10	Rewards Not Settled Before Liquidity Removal	Major	Fixed
LPA-11	Rewards and Fees Not Settled When Adding Liquidity Multiple Times	Major	Fixed
LPA-12	Missing Pause Mechanism in `flash_loan()` Function May Lead to Reentrancy Vulnerability	Medium	Fixed

LPA-13	Inaccurate Repayment Check in `repay_flash_loan()` May Lead to Donation Attack	Medium	Fixed
LPA-14	`remove_liquidity` Logical Error	Medium	Fixed
LPA-15	Incorrect Role Verification in `add_rewarder()` Function	Medium	Fixed
LPA-16	Inconsistent Lock Timestamp Handling Prevents Adding Liquidity	Minor	Fixed
LPA-17	Use `&&` instead of `||`	Minor	Fixed
LPA-18	Redundant Calculation of `lp_comp_fee_x` in `update_bin()` Function	Informational	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Ferra DLMM Smart Contract :

ADMIN

- `set_upgrade_cap` : Store the package's UpgradeCap into the ACL state
- `set_publisher` : Store the package's Publisher object into the ACL state
- `propose` :Create a new governance proposal for a specific action
- `vote` : Cast a vote in favor of an existing
- `execute` : Execute a proposal that has met its conditions
- `cancel` : Cancel an existing proposal

REWARD_ROLE

- `add_rewarder` : Add a reward for the specified trading pair
- `emergent_withdraw` : Emergency withdrawal of funds
- `update_emission` : Update the reward emission rate of the specified trading pair

UPGRADE_ROLE

- `update_package_version` : Store the package's UpgradeCap into the ACL state

POOL_MANAGER_ROLE

- `create_pair` : Create and register a new liquidity pool
- `set_static_fee_parameters` : Set the static cost parameter for the specified LBPair
- `force_decay` : Forcibly triggers the volatility accumulator decay process of the specified LBPair
- `increase_oracle_length` : Increase the oracle data store length for the specified LBPair
- `pause_pair` : Pause or unpause the specified LBPair

- `collect_protocol_fees` : Collect the accumulated agreement fees in the trading pairs

CONFIGROLE

- `add_update_bin_step` : Add or update the cost parameter of bin step
- `delete_bin_step` : Delete the fee parameter of the specified bin step
- `update_flash_loan_max_amount` : Updates the global maximum flash loan amount
- `update_flash_loan_fee_rate` : Updates the global flash loan fee rate
- `add_whitelist_token` : Adds a specified coin type to the whitelist of quote assets
- `delete_whitelist_token` : Removes a specified coin type from the quote asset whitelist
- `set_allow_create_pair` : Sets a boolean flag to globally enable or disable the creation of new liquidity pairs
- `set_pause` : Set the global pause state
- `set_flash_loan_enable` : Set the activation status of the flash loan

User

- `deposit_reward` : Deposit rewards into the global vault
- `swap` : Perform token exchange operations
- `open_position` : Create a new liquidity position
- `lock_position` : Lock the position until the specified time
- `add_liquidity` : Add liquidity to the specified position
- `remove_liquidity` : Remove liquidity from positions and withdraw assets
- `close_position` : Close the position of the specified trading pair
- `collect_position_fees` : Collect the fees for the specified position
- `collect_position_rewards` : Collect rewards for the specified position
- `flash_loan` : Perform the flash loan operation

- `repay_flash_loan` : Repay flash loans

4 Findings

ACL-1 `remove_from_all_roles` Permission Removal Is Incomplete

Severity: Medium

Status: Fixed

Code Location:

sources/acl.move#365

Descriptions:

When a proposal of type 'PROPOSAL_REMOVE_ADMIN' is executed, the system will call the internal function 'remove_from_all_roles' to remove all role permissions of the administrator. However, the 'remove_from_all_roles' function has an implementation oversight. It only removes the permissions whose addresses are in' OPERATOR_ROLE ', 'REWARD_ROLE', and 'PROTOCOL_FEE_ROLE'. However, the removal operation of 'UPGRADE_ROLE' was omitted

```
fun remove_from_all_roles(acl: &mut ACL, target: address) {
    // Remove from all role types
    let operator_set = table::borrow_mut(&mut acl.roles, OPERATOR_ROLE);
    if (vec_set::contains(operator_set, &target)) {
        vec_set::remove(operator_set, &target);
    };

    let reward_set = table::borrow_mut(&mut acl.roles, REWARD_ROLE);
    if (vec_set::contains(reward_set, &target)) {
        vec_set::remove(reward_set, &target);
    };

    let fee_set = table::borrow_mut(&mut acl.roles, PROTOCOL_FEE_ROLE);
    if (vec_set::contains(fee_set, &target)) {
        vec_set::remove(fee_set, &target);
    };
```

```
    }
.....

    const OPERATOR_ROLE: u8 = 0;
    const REWARD_ROLE: u8 = 1;
    const PROTOCOL_FEE_ROLE: u8 = 2;
    const UPGRADE_ROLE: u8 = 3;
```

Suggestion:

Add Code

```
let upgrade_set = table::borrow_mut(&mut acl.roles, UPGRADE_ROLE);
if (vec_set::contains(upgrade_set, &target)) {
    vec_set::remove(upgrade_set, &target);
};
```

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ACL-2 Lack of Vote Threshold Check in `cancel()` Allows Admins to Arbitrarily Cancel Proposals

Severity: Medium

Status: Fixed

Code Location:

sources/acl.move#275-296

Descriptions:

The `cancel()` function is used to cancel a proposal.

```
public fun cancel(
        acl: &mut ACL,
        proposal_id: u64,
        clock: &Clock,
        ctx: &TxContext
    ) {
        let sender = tx_context::sender(ctx);
        assert!(is_admin(acl, sender), E_NOT_ADMIN);
        assert!(table::contains(&acl.proposals, proposal_id), E_PROPOSAL_NOT_FOUND);
        let proposal = table::borrow(&acl.proposals, proposal_id);
        if(proposal.proposal_type == PROPOSAL_REMOVE_ADMIN){
            assert!(proposal.target != sender, E_NOT_CANCEL_PROPOSAL)
        };

        let _proposal = table::remove(&mut acl.proposals, proposal_id);

        event::emit(ProposalCancelled {
            proposal_id,
            cancelled_by: sender,
            cancelled_at: clock::timestamp_ms(clock),
        });
    }
```

However, the protocol does not verify whether the votes are less than 50%. This means that `cancel()` can be used to cancel a valid proposal. If a proposal is unfavorable to a certain admin (e.g., revoking their role), the admin could call `cancel()` to terminate the proposal, preventing it from being executed.

Suggestion:

It is recommended to implement a vote threshold check in the `cancel()` function to ensure that only proposals with less than the required minimum support (e.g., <50%) can be canceled.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ACL-3 `propose` Missing Parameter Checking

Severity: Minor

Status: Fixed

Code Location:

sources/acl.move#179

Descriptions:

In the 'acl::propose' function, the parameter 'proposal_type' is not verified for validity. This function allows administrators to create a proposal type with any u8 value. However, in the 'execute_proposal_action' function, only proposal types from 0 to 7 are processed. If a proposal of type 8 or higher is created, although it can be voted through, it will permanently fail (revert) during the execution stage because no valid actions are matched. This will cause the proposal to remain permanently in storage (unless it is cancelled), resulting in state inflation and potential governance chaos.

Suggestion:

Add validation of the proposal_type parameter at the entrance of the propose function. Only values between 0 and 7 are allowed to pass; otherwise, the transaction should be immediately suspended

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ACL-4 Repeated error code identification

Severity: Informational

Status: Fixed

Code Location:

sources/acl.move#38

Descriptions:

Repeated error codes are defined in the module. The constants `E_PUBLISHER_NOT_AVAILABLE` and `E_FUND_RECEIVER_NOT_SET` both use error code 8. Meanwhile, both `E_NOT_CANCEL_PROPOSAL` and `E_REWARD_RECEIVER_NOT_SET` used error code 9. This will lead to ambiguous error messages returned on the chain when a transaction fails, making it impossible to accurately determine the root cause of the error, thereby seriously affecting the debuggability of the contract and the efficiency of problem-solving.

Suggestion:

Modify the error code identifier

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BIN-1 `sub_fees` Fee Calculation Logical Error

Severity: Major

Status: Fixed

Code Location:

sources/bin.move#229

Descriptions:

In the 'bin_manager' module, there are serious logical errors in the implementation of the 'sub_fees' function. This function is originally applied to subtract the corresponding amount from the total fee reserve of bin after the liquidity provider has claimed the fee. However, the code 'bin.fee_y = safe_math::add_u64(bin.fee_y, sub_fee_y); 'sub_fee_y' was wrongly added to 'bin.fee_y' instead of subtracted. This leads to the situation where whenever a user claims the fees of Y tokens, the total fees recorded in this bin not only do not decrease but increase instead, causing a serious mismatch between the fee status and the actual funds, and may result in the funds being trapped or the protocol status being damaged

```
public fun sub_fees(
    bin: &mut Bin,
    sub_fee_x: u64,
    sub_fee_y: u64
) {
    bin.fee_x = safe_math::sub_u64(bin.fee_x, sub_fee_x);
    bin.fee_y = safe_math::add_u64(bin.fee_y, sub_fee_y);
}
```

Suggestion:

Fix the erroneous logic in the 'sub_fees' function. Set 'bin.fee_y = safe_math::add_u64(bin.fee_y, sub_fee_y); Modify to 'bin.fee_y = safe_math::sub_u64(bin.fee_y, sub_fee_y); To ensure that the corresponding amount can be correctly deducted from the reserve when claiming the fees.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

BIN-2 `add_reserves_fees` Functions Such As These Should Use Internal Call Modifiers

Severity: Informational

Status: Fixed

Code Location:

sources/bin.move#184

Descriptions:

Functions such as add_fee_growth, add_reserves_fees, ub_fees, update_reserves_fees, and subtract_bin should only use the public(friend) modifier when called internally

Suggestion:

Modify the function modifier

Resolution:

This issue has been fixed. The client has adopted our suggestions.

CON-1 `flashloan_percentage_precission` Spelling Mistake

Severity: Informational

Status: Fixed

Code Location:

sources/libraries/constants.move#33

Descriptions:

The correct spelling of the function name flashloan_percentage_precission should be flashloan_percentage_precision

```
    public fun flashloan_percentage_precission(): u64 {
FLASH_LOAN_PERCENTAGE_PRECISION }
```

Suggestion:

Modify the function name

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LFA-1 Missing Validation for `bin_step`

Severity: Minor

Status: Fixed

Code Location:

sources/lb_factory.move#85-98

Descriptions:

In the `create_pair()` function, the protocol retrieves parameter values based on the `bin_step`.

```
let lb_pair = lb_pair::new<X, Y>(
    active_id,
    bin_step,
    config::base_factor(config, bin_step),
    config::filter_period(config, bin_step),
    config::decay_period(config, bin_step),
    config::reduction_factor(config, bin_step),
    config::variable_fee_control(config, bin_step),
    config::protocol_share(config, bin_step),
    config::max_volatility_accumulator(config, bin_step),
    bin_init,
    clock,
    ctx,
);
```

However, it does not verify whether the provided `bin_step` exists. Using a non-existent `bin_step` may lead to unexpected behavior or errors.

Suggestion:

It is recommended to add a validation check to ensure the `bin_step` is valid before proceeding.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-1 Missing Fee Handling After Fee Collection in `remove_liquidity()` Function

Severity: Major

Status: Fixed

Code Location:

sources/lb_pair.move#1504

Descriptions:

In the `remove_liquidity()` function, the protocol collects fees but does not process or distribute them.

```
// Collect fees before modifying bin
        let (_fees_collected_x, _fees_collected_y) = lb_position::collect_fees(
            &mut pair.position_manager,
            position,
            id,
            current_fee_growth_x,
            current_fee_growth_y,
        );
```

Suggestion:

It is recommended to implement proper handling for the collected fees, such as transferring them to a designated fee recipient or distributing them according to the protocol's fee distribution logic.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-2 Lack of Slippage Protection in `add_liquidity()` and `remove_liquidity()` Functions

Severity: Major

Status: Fixed

Code Location:

sources/lb_pair.move#1034-1125

Descriptions:

The `add_liquidity()` and `remove_liquidity()` functions are used to add and remove liquidity, respectively. However, both functions lack slippage protection, which may expose users to unfavorable execution due to price movements.

Suggestion:

It is recommended to add slippage protection mechanisms to ensure users receive expected outcomes and avoid potential losses.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-3 Missing Reward Collection in `add_liquidity()` and `remove_liquidity()` May Lead to Inaccurate `reward_per_fee_delta` Calculations

Severity: Major

Status: Fixed

Code Location:

sources/lb_pair.move#1015-1049

Descriptions:

The `add_liquidity()` function allows users to add liquidity, but the protocol does not collect the pair reward during this process.

```
public fun add_liquidity<X, Y>(
    config: &GlobalConfig,
    pair: &mut LBPair<X, Y>,
    position: &mut LBPosition,
    ids: vector<u32>,
    distribution_x: vector<u64>,
    distribution_y: vector<u64>,
    coin_x: Coin<X>,
    coin_y: Coin<Y>,
    min_amount_x: u64,
    min_amount_y: u64,
    clock: &Clock,
    ctx: &mut TxContext,
) {
    config::checked_package_version(config);
    assert!(!pair.is_pause, E_PAIR_PAUSED);
    assert!(
        object::id<LBPair<X, Y>>(pair) == lb_position::pair_id(position),
        E_POSITION_MISMATCH,
    );
    let sender = tx_context::sender(ctx);
```

Since `reward_per_fee_delta` is calculated as `rewards_generated / total_fees_ever`, and `total_fees_ever` changes over time, not collecting the reward at the correct moment may lead to inaccurate calculations in future updates. The same issue also exists in the `remove_liquidity()` function.

```
// Only distribute if we have both rewards and fees
        if (rewards_generated > 0 && pair.reward_state.total_fees_ever > 0) {
            // reward per fee rate
            let reward_per_fee_delta = safe_math::mul_div_u128(
                rewards_generated,
                q64x64::scale_64x64(),
                pair.reward_state.total_fees_ever
            );
```

Suggestion:

It is recommended to collect the pair reward in both functions to ensure accurate reward accounting.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-4 Incorrect Reward Accrual Due to Delayed Liquidity Addition After Snapshot Update

Severity: Major

Status: Fixed

Code Location:

sources/lb_pair.move#1679

Descriptions:

In the `open_position()` function, the protocol initializes the `reward_per_fee_snapshot`.

```
public fun open_position<X, Y>(
        config: &GlobalConfig,
        pair: &mut LBPair<X, Y>,
        lock_until_timestamp: u64,
        ctx: &mut TxContext,
    ): LBPosition {
        config::checked_package_version(config);
        assert!(!pair.is_pause, E_PAIR_PAUSED);
        let pair_id = object::id<LBPair<X, Y>>(pair);

        ensure_reward_vectors_initialized(pair);

        let position = lb_position::open_position<X, Y>(
            &mut pair.position_manager,
            pair_id,
            lock_until_timestamp,
            ctx,
        );

        event::emit(OpenPositionEvent {
            pair: pair_id,
            position: object::id(&position),
            owner: tx_context::sender(ctx),
        });
```

```
    position
}
```

However, the protocol does not update the position's `reward_per_fee_snapshot` when `add_liquidity()` is called.It only updates it within the `collect_position_rewards()` function.

```
// Update snapshot
    lb_position::update_reward_per_fee_snapshot(
        position_info,
        rewarder_index,
        current_rate
    );
```

This behavior introduces an issue: if a user opens a position and does not immediately add liquidity, but instead first calls `collect_position_rewards()`, the protocol will update the user's position with the current reward rate using `update_reward_per_fee_snapshot()`. If the user then waits for a period of time and later adds liquidity, the next time they call `collect_position_rewards()`, the rewards will be calculated starting from the previously updated snapshot (which may be higher), even though the user did not contribute liquidity during that earlier period. This could result in the user receiving more rewards than they should.

Suggestion:

It is recommended to call `update_reward_per_fee_snapshot()` when adding liquidity to update the user's position reward rate.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-5 Missing Update to `total_fees_gen` in `remove_liquidity()` Leads to Reduced Reward Calculation

Severity: Major

Status: Fixed

Code Location:

sources/lb_pair.move#1482

Descriptions:

A user's reward is calculated based on their `total_fees`, which represents the total fees they have collected.

```
let total_fees = lb_position::get_total_fees_gen(position_info);
    let rewards = safe_math::mul_u128(
        total_fees,
        safe_math::sub_u128(current_rate, last_rate)
    ) >> 64;
```

```
    let fee_normalized = q64x64::liquidity_from_amounts(
        total_fees_x,
        total_fees_y,
        active_price
    );

    // Tracking reward
    // Update position's total normalized fees
    let position_info = lb_position::borrow_mut_position_info(
        &mut pair.position_manager,
        object::id(position)
    );
    lb_position::add_total_fees_gen(position_info, fee_normalized);
```

However, in the `remove_liquidity()` function, although the user collects fees, the protocol does not increase `position_info.total_fees_gen`. As a result, when the user later collects rewards, the reward amount is reduced due to the missing contribution to `total_fees_ever`.

Suggestion:

It is recommended to call `lb_position::add_total_fees_gen()` in the `remove_liquidity()` function to increase the value of `total_fees_gen`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-6 Persistent `total_fees` Without Reduction Allows Reward Collection After Position Closure

Severity: Major

Status: Fixed

Code Location:

sources/lb_pair.move#1669-1673

Descriptions:

The `collect_position_rewards()` function is used to collect rewards from a position. The rewards are calculated using the formula: `rewards = total_fees * delta_rate`.

```
let total_fees = lb_position::get_total_fees_gen(position_info);
  let rewards = safe_math::mul_u128(
    total_fees,
    safe_math::sub_u128(current_rate, last_rate)
  ) >> 64;
```

However, `total_fees` continuously increases and is never reduced, even when the user no longer holds a position. As a result, users can still be able to call `collect_position_rewards()` and receive rewards despite having no active liquidity position, leading to reward over-distribution.

Suggestion:

It is recommended to optimize the reward calculation.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-7 Unsettled Rewards Before reward_factor Update Allow Excess Payouts and Front-Running

Severity: Major

Status: Fixed

Code Location:

sources/lb_pair.move#1462-1489

Descriptions:

The `collect_position_fees()` function is used to collect fees and automatically calculate and distribute rewards.

```
// Collect fees and automatically calculate and collect rewards
  public fun collect_position_fees<X, Y>(
      config: &GlobalConfig,
      pair: &mut LBPair<X, Y>,
      position: &mut LBPosition,
      bin_ids: vector<u32>,
      _clock: &Clock,
      ctx: &mut TxContext,
  ): (Coin<X>, Coin<Y>) {
      config::checked_package_version(config);
      assert!(!pair.is_pause, E_PAIR_PAUSED);
      assert!(
          object::id<LBPair<X, Y>>(pair) == lb_position::pair_id(position),
          E_POSITION_MISMATCH,
      );
      assert!(vector::length(&bin_ids) > 0, E_INVALID_BIN_IDS);
```

The reward is calculated as:

```
reward_amount = total_normalized_fees * reward_factor / REWARD_FACTOR_PRECISION
```

```
if (reward_amount > 0) {
                lb_position::add_reward_debt(
                    &mut pair.position_manager,
                    position,
                    j,
                     reward_amount
                );
            };
```

An operator can call `update_reward_factor()` to change the reward factor.

```
public fun update_reward_factor<X, Y, RewardCoin>(
    config: &GlobalConfig,
    pair: &mut LBPair<X, Y>,
    vault: &RewarderGlobalVault,
    reward_factor: u128,
    _clock: &Clock,
    ctx: &mut TxContext
  ) {
    config::checked_package_version(config);
    assert!(!pair.is_pause, E_PAIR_PAUSED);

    config::check_operator_role(config, tx_context::sender(ctx));

    let pair_id = object::id(pair);

    rewarder::update_reward_factor<RewardCoin>(
        vault,
        &mut pair.reward_manager,
        pair_id,
        reward_factor,
    );
```

However, when the operator updates the factor, the rewards generated under the previous factor are not settled beforehand.

This leads to two issues:

1. **Users may receive excess rewards** because rewards accrued under the old factor are calculated using the new, potentially higher factor.
2. **Front-running risk**: A malicious user who notices the operator is about to call `update_reward_factor()` could quickly perform actions that generate fees, wait for the operator to update the factor, and then immediately call `collect_position_fees()` to receive inflated rewards.

Suggestion:

It is recommended to use a growth-based approach to calculate the reward.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-8 Bypassing Intended Lock Period by Adding Liquidity After `lock_until_timestamp` Countdown

Severity: Major

Status: Fixed

Code Location:

sources/lb_pair.move#920

Descriptions:

When opening a position, the user sets `lock_until_timestamp`.

```
public fun open_position<X, Y>(
        config: &GlobalConfig,
        pair: &mut LBPair<X, Y>,
        lock_until_timestamp: u64,
        ctx: &mut TxContext,
    ): LBPosition {
        config::checked_package_version(config);
        assert!(!pair.is_pause, E_PAIR_PAUSED);
        let pair_id = object::id<LBPair<X, Y>>(pair);

        let position = lb_position::open_position<X, Y>(
            &mut pair.position_manager,
            pair_id,
            lock_until_timestamp,
            ctx,
        );
```

Liquidity can only be removed if the current time is greater than `lock_until_timestamp`.

```
    public(friend) fun decrease_liquidity(
        manager: &mut LBPositionManager,
        position: &LBPosition,
        bin_id: u32,
```

```
    lp_burn: u128,
    clock: &Clock,
){
    let current_time = clock::timestamp_ms(clock);
    assert!(position.lock_until <= current_time, E_POSITION_LOCKED);
```

However, since `open_position()` and `add_liquidity()` are separate functions, there is an issue: a user can set `lock_until_timestamp` when opening the position, wait for a long period of time, then add liquidity later, and subsequently remove liquidity shortly after. This results in the actual lock period being shorter than intended.

Suggestion:

When calling `add_liquidity()`, update or revalidate `lock_until_timestamp` so that the newly added liquidity inherits the same remaining lock period or resets the lock period based on protocol rules.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-9 Adding New Rewarder Allows Users to Retroactively Claim Multiple Types of Rewards

Severity: Major

Status: Fixed

Code Location:

sources/lb_pair.move#1555

Descriptions:

The `add_rewarder()` function allows the operator role to add new rewards. When collecting fees, the protocol settles rewards based on the accumulated fees, regardless of how many different reward types exist, and then distributes them to the user. If there were initially only two reward coins and, after some time, the operator adds a third reward coin, a user who has never collected position fees during both the two-reward and three-reward periods could then collect position fees and receive multiple types of rewards at once.

```
if (total_normalized_fees > 0) {
        let rewarders = rewarder::get_rewarders(&pair.reward_manager);
        let j = 0;
        while (j < vector::length(rewarders)) {
            let rewarder = vector::borrow(rewarders, j);
            let reward_factor = rewarder::get_rewarder_factor(rewarder);

            if (reward_factor > 0) {
                let reward_amount = rewarder::calculate_reward_amount(
                    total_normalized_fees,
                     reward_factor
                );

                if (reward_amount > 0) {
                    lb_position::add_reward_debt(
                        &mut pair.position_manager,
                        position,
                        j,
```

```
                reward_amount
            );
        };
    };
    j = j + 1;
};
};
```

Suggestion:

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPO-1 Potential Out-of-Gas Risk in increase_liquidity() due to Iteration over Excessive Bin IDs

Severity: Major

Status: Fixed

Code Location:

sources/lb_position.move#310-441

Descriptions:

In the `increase_liquidity()` function, the protocol iterates through each bin ID and the corresponding rewards for each bin. If the number of bin IDs is too large, it may lead to an **out-of-gas** issue.

```
while (i < len) {
        let bin_id = *vector::borrow(&bin_ids, i);
        let share = *vector::borrow(&shares, i);
        let fee_growth_x = *vector::borrow(&fee_growths_x, i);
        let fee_growth_y = *vector::borrow(&fee_growths_y, i);
        let current_reward_growth = *vector::borrow(&reward_growths, i);

        let (group_index, position_in_group) = resolve_bin_group_index(bin_id);

        if (group_index != current_group_index) {
            if (!table::contains(&position_info.bins, group_index)) {
                table::add(&mut position_info.bins, group_index, PackedBins {
                    active_bins_bitmap: 0u8,
                    bin_data: vector::empty(),
                });
            };
            current_group = table::borrow_mut(&mut position_info.bins, group_index);
            current_group_index = group_index;
        };
```

```
while (j < min_rewarders) {
                let current_growth = *vector::borrow(&current_reward_growth, j);
                let last_growth = *vector::borrow(&bin.reward_growth_inside_last, j);

                if (current_growth > last_growth) {
                    let pending_reward = safe_math::u128_to_u64(
                        ((bin.amount * (current_growth - last_growth)) >> 64u8)
                    );
```

Suggestion:

It is recommended to introduce safeguards to limit the maximum number of bin IDs that can be processed in a single transaction.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPO-2 Missing Fee Claim Check in `close_position()` May Cause User Fund Loss

Severity: Medium

Status: Fixed

Code Location:

sources/lb_position.move#490

Descriptions:

In the `close_position()` function, the protocol only checks whether the position still holds LP tokens and whether the rewards have been fully claimed.

```
    public(friend) fun close_position(
        manager: &mut LBPositionManager,
        position: LBPosition,
    ) {
        let position_info = table::remove(&mut manager.positions, object::id(&position));
        assert!(is_empty_lp(&position_info) && is_empty_reward(&position),
E_POSITION_NOT_EMPTY);
        destroy_position_info(position_info);
        destroy(position);
    }
```

However, it does not verify whether the position fees have been collected. If the protocol allows a position to be closed without first claiming the fees, this could result in a loss of user funds.

Suggestion:

It is recommended to verify whether all fees have been claimed

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPO-3 `add_bin` Not Used

Severity: Minor

Status: Fixed

Code Location:

sources/lb_position.move#211

Descriptions:

The add_bin function is an internal function and no other function calls it, so it is redundant code. Moreover, after the add_bin function successfully adds a new bin data to the bins table of LBPositionInfo, The total_bins count in the LBPosition structure was not increased synchronously

```
public(friend) fun add_bin(
    manager: &mut LBPositionManager,
    position: &LBPosition,
    bin_id: u32,
    new_bin_data: LBBinPosition,
) {
```

Suggestion:

Delete unused functions

Resolution:

This issue has been fixed. The client has adopted our suggestions.

PPH-1 Missing Validation for `variable_fee_control` and `protocol_share` Upper Limits in `set_static_fee_parameters()`

Severity: Informational

Status: Fixed

Code Location:

sources/libraries/pair_parameter_helper.move#135-141

Descriptions:

In the `public fun set_static_fee_parameters()` function, the protocol verifies `filter_period`, `reduction_factor`, `protocol_share`, and `max_volatility_accumulator`, but it does not check whether `variable_fee_control` and `protocol_share` are below their maximum allowed values.

```
public fun set_static_fee_parameters(
    params: &mut PairParameters,
    base_factor: u32,
    filter_period: u16,
    decay_period: u16,
    reduction_factor: u16,
    variable_fee_control: u32,
    protocol_share: u16,
    max_volatility_accumulator: u32
) {
    assert!(
        filter_period <= decay_period &&
        (reduction_factor as u64) <= constants::basis_point_max() &&
        (protocol_share as u64) <= constants::max_protocol_share() &&
        max_volatility_accumulator <= 0xfffff, // 20 bits max
        E_INVALID_PARAMETER
    );
```

Suggestion:

It is recommended to add validation to ensure that both `variable_fee_control` and `protocol_share` are less than their respective maximum limits.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

CON1-1 No Limit To The Traversal Length Of Bins

Severity: Medium

Status: Fixed

Code Location:

sources/config.move#31

Descriptions:

There is no setting like MAX_BIN_PER_POSITION. Operations such as trading (swap), adding/removing liquidity, etc. all require traversing the bins within the position. The more bins traversed, the higher the Gas consumption will be. Setting the upper limit of the crossed BIN ensures that the computational load of any single operation is within a controllable and predictable range. This can prevent transactions from failing due to exceeding the block Gas limit of the Sui network, thereby ensuring the availability of the protocol.For DLMM contracts of the same type, the configuration value of max_bins_in_position is around 100

Suggestion:

Add the maximum number of bins to be traversed. When there are operations involving bins, limit the maximum number

Resolution:

This issue has been fixed. The client has adopted our suggestions.

CON1-2 `add_update_bin_step` And `delete_bin_step` Check For Deficiencies

Severity: Minor

Status: Fixed

Code Location:

sources/config.move#120

Descriptions:

In the `add_update_bin_step` and `delete_bin_step` functions of the config module, it is only verified that the `bin_step` parameter is not less than `MIN_BIN_STEP`, but it is not verified whether it exceeds `MAX_BIN_STEP`. Although there is a `validate_bin_step` function in the module that contains a full range check, these two key state change functions do not call it. This causes inconsistency in the verification logic and allows setting a `bin_step` beyond the expected range.

Suggestion:

Add checks

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-10 Rewards Not Settled Before Liquidity Removal

Severity: Major

Status: Fixed

Code Location:

sources/lb_pair.move#1366-1372

Descriptions:

In the `remove_liquidity()` function, the protocol decreases the user's liquidity but does not settle the rewards, which could cause the user to receive lower rewards."

```
// Decrease LP from position
    lb_position::decrease_liquidity(
        &mut pair.position_manager,
        position,
        id,
        lp_burn,
        clock,
    );
```

Suggestion:

It is recommended to settle the user's rewards before decreasing the position.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-11 Rewards and Fees Not Settled When Adding Liquidity Multiple Times

Severity: Major

Status: Fixed

Code Location:

sources/lb_pair.move#1114-1123

Descriptions:

The `add_liquidity()` function allows users to add liquidity. Each time a user adds liquidity, the protocol updates the position's reward growth and fee growth.

```
// Update position, fee growth, reward growth
        lb_position::increase_liquidity(
            &mut pair.position_manager,
            &mut pair.bin_manager,
            position,
            id,
            share,
            fee_growth_x,
            fee_growth_y,
            current_reward_growth,
        );
```

However, if a user adds liquidity multiple times to the same position, the repeated updates to reward growth and fee growth will overwrite previous values, causing the user's prior rewards and fees to remain unsettled and resulting in reduced rewards and fees for the user.

Suggestion:

It is recommended to settle the user's accumulated rewards and fees before updating reward growth and fee growth in `add_liquidity()`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-12 Missing Pause Mechanism in `flash_loan()` Function May Lead to Reentrancy Vulnerability

Severity: Medium

Status: Fixed

Code Location:

sources/lb_pair.move#1963-2025

Descriptions:

The `flash_loan()` function allows users to borrow funds. However, the protocol does not set the `pause` flag to `true` during the execution of the function, which may introduce a reentrancy vulnerability.

```
public fun flash_loan<X, Y>(
    config: &GlobalConfig,
    pair: &mut LBPair<X, Y>,
    is_token_x: bool,
    amount: u64,
): (Balance<X>, Balance<Y>, FlashLoanReceipt) {
    config::checked_package_version(config);
    assert!(!pair.is_pause, E_PAIR_PAUSED);

    // Check flash loan amount
    assert!(amount > 0, E_INVALID_FLASH_LOAN);

    let available_balance = if (is_token_x) {
        balance::value<X>(&pair.balance_x)
    } else {
        balance::value<Y>(&pair.balance_y)
    };
```

Suggestion:

It is recommended to set `pause = true` at the beginning of the `flash_loan()` function and reset it to `false` in the `repay_flash_loan()` function to prevent potential reentrant calls

during the flash loan process.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-13 Inaccurate Repayment Check in `repay_flash_loan()` May Lead to Donation Attack

Severity: Medium

Status: Fixed

Code Location:

sources/lb_pair.move#2048

Descriptions:

The `repay_flash_loan()` function is used to repay the flash loan. Within the function, the protocol checks that the repayment balance is **greater than or equal to** the sum of `amount + fee_amount`.

```
        if (loan_x) {
            assert!(balance::value<X>(&balance_x) >= amount + fee_amount,
E_INSUFFICIENT_AMOUNT_IN);
            balance::join<X>(&mut pair.balance_x, balance_x);
            balance::destroy_zero<Y>(balance_y);
        } else {
            assert!(balance::value<Y>(&balance_y) >= amount + fee_amount,
E_INSUFFICIENT_AMOUNT_IN);
            balance::join<Y>(&mut pair.balance_y, balance_y);
            balance::destroy_zero<X>(balance_x);
        };
```

However, this logic may expose the protocol to a donation (donate) attack, where users overpay to manipulate accounting or internal balances.

Suggestion:

It is recommended to strictly check that the repayment balance is **exactly equal** to `amount + fee_amount` to prevent such attacks.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-14 `remove_liquidity` Logical Error

Severity: Medium

Status: Fixed

Code Location:

sources/lb_pair.move#1271

Descriptions:

The contract maintains global 'lp_fee_x' and 'lp_fee_y' counters in the 'LBPair' structure to track the total amount of all fees in the pool that have not been claimed by liquidity providers. These counters increase correctly when users generate fees through 'swap' or 'add_liquidity' (on active bins). When the user explicitly claims the fees by calling the 'collect_position_fees' function, these counters will also decrease correctly. However, the 'remove_liquidity' function also returns the fees that users are due to (including accumulated fees and saved fees) to users in business logic. But after performing this operation, the function does not subtract this part of the paid fees from the global 'lp_fee_x' and 'lp_fee_y' counters.

Suggestion:

Add Code

```
total_fees_x = safe_math::add_u64(total_fees_x, saved_x);
total_fees_y = safe_math::add_u64(total_fees_y, saved_y);


pair.lp_fee_x = safe_math::sub_u64_cape_zero(pair.lp_fee_x, total_fees_x);
pair.lp_fee_y = safe_math::sub_u64_cape_zero(pair.lp_fee_y, total_fees_y);
```

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-15 Incorrect Role Verification in `add_rewarder()` Function

Severity: Medium

Status: Fixed

Code Location:

sources/lb_pair.move#1451

Descriptions:

The contract defines three roles: `OPERATOR_ROLE`, `REWARD_ROLE`, and `PROTOCOL_FEE_ROLE`.

```
// Initialize roles table
    let roles = table::new<u8, VecSet<address>>(ctx);
    table::add(&mut roles, OPERATOR_ROLE, vec_set::empty<address>());
    table::add(&mut roles, REWARD_ROLE, vec_set::empty<address>());
    table::add(&mut roles, PROTOCOL_FEE_ROLE, vec_set::empty<address>());
```

However, in the `add_rewarder()` function, the protocol checks for `OPERATOR_ROLE` instead of `REWARD_ROLE`. The role verification should be updated to check for `REWARD_ROLE`.

```
public fun add_rewarder<X, Y, RewardCoin>(
    config: &GlobalConfig,
    pair: &mut LBPair<X, Y>,
    ctx: &TxContext,
) {
    config::checked_package_version(config);
    assert!(!pair.is_pause, E_PAIR_PAUSED);
    config::check_operator_role(config, tx_context::sender(ctx));

    let pair_id = object::id(pair);
    rewarder::add_rewarder<RewardCoin>(&mut pair.reward_manager);

    event::emit(RewarderAddedEvent {
```

```
        pair: pair_id,
        rewarder_type: type_name::get<RewardCoin>(),
    });
}
```

Suggestion:

It is recommended to update the `add_rewarder()` function to verify `REWARD_ROLE` instead of `OPERATOR_ROLE`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-16 Inconsistent Lock Timestamp Handling Prevents Adding Liquidity

Severity: Minor

Status: Fixed

Code Location:

sources/lb_pair.move#923

Descriptions:

In the `add_liquidity()` function, the protocol verifies that `lb_position::get_lock_until(position) == 0` .

```
config::checked_package_version(config);
    assert!(!pair.is_pause, E_PAIR_PAUSED);
    assert!(
        object::id<LBPair<X, Y>>(pair) == lb_position::pair_id(position),
        E_POSITION_MISMATCH,
    );
    assert!(lb_position::get_lock_until(position) == 0, E_POSITION_LOCKED);
```

However, in the `open_position()` function, the `lock_until_timestamp` is specified by the user. If the user does not input 0, they may not be able to add liquidity later.

```
public fun open_position<X, Y>(
    config: &GlobalConfig,
    pair: &mut LBPair<X, Y>,
    lock_until_timestamp: u64,
    ctx: &mut TxContext,
  ): LBPosition {
    config::checked_package_version(config);
    assert!(!pair.is_pause, E_PAIR_PAUSED);
    let pair_id = object::id<LBPair<X, Y>>(pair);
    let position = lb_position::open_position<X, Y>(
        &mut pair.position_manager,
        pair_id,
```

```
        lock_until_timestamp,
        ctx,
    );
```

Suggestion:

It is recommended to ensure that `open_position()` enforces `lock_until_timestamp = 0` when creating a new position,

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-17 Use `&&` instead of `||`

Severity: Minor

Status: Fixed

Code Location:

sources/lb_pair.move#2343-2348

Descriptions:

In the `set_static_fee_parameters_internal()` function, the validation logic currently uses the `||` operator, meaning the check passes if **any one** of the input parameters meets the condition.

```
/// Helper functions
fun set_static_fee_parameters_internal<X, Y>(
    pair: &mut LBPair<X, Y>,
    base_factor: u32,
    filter_period: u16,
    decay_period: u16,
    reduction_factor: u16,
    variable_fee_control: u32,
    protocol_share: u16,
    max_volatility_accumulator: u32,
    ctx: &TxContext,
) {
    assert!(
        base_factor != 0 || filter_period != 0 || decay_period != 0 ||
            reduction_factor != 0 || variable_fee_control != 0 ||
            protocol_share != 0 || max_volatility_accumulator != 0,
        E_INVALID_STATIC_FEE_PARAMS,
    );
```

However, this is incorrect. The function should use the `&&` operator to ensure that **all** input parameters satisfy the required conditions before proceeding.

Suggestion:

It is recommended to use `&&` instead of `||`

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LPA-18 Redundant Calculation of `lp_comp_fee_x` in `update_bin()` Function

Severity: Informational

Status: Fixed

Code Location:

sources/lb_pair.move#1376-1396

Descriptions:

In the `update_bin()` function, the protocol initializes `composition_fee_x` and `composition_fee_y` to 0.

```
let amounts_in_to_bin_x = amounts_in_x;
    let amounts_in_to_bin_y = amounts_in_y;
    let composition_fee_x = 0u64;
    let composition_fee_y = 0u64;
```

- If `fee_x > 0 || fee_y > 0`, it calculates: `lp_fee_x = fee_x - protocol_fee_x` `lp_fee_y = fee_y - protocol_fee_y`

```
// Update bin fee growth for LP fees
        let lp_fee_x = safe_math::sub_u64(fee_x, protocol_fee_x);
        let lp_fee_y = safe_math::sub_u64(fee_y, protocol_fee_y);
        update_bin_fee_growth(&mut bin, lp_fee_x, lp_fee_y);
```

- If `fee_x == 0 && fee_y == 0`, the protocol only verifies the amounts for non-active bins.

After that, it calculates `lp_comp_fee_x` as follows:

- If `composition_fee_x > 0`, then `lp_comp_fee_x = composition_fee_x - protocol_share`
- Otherwise, `lp_comp_fee_x = 0`

```
// Calculate composition fees for LP
    let lp_comp_fee_x = if (composition_fee_x > 0) {
```

```
            let protocol_share =
pair_parameter_helper::get_protocol_share(&pair.parameters);
            let protocol_fee = fee_helper::get_protocol_fee_amount(
                composition_fee_x,
                (protocol_share as u64),
            );
            safe_math::sub_u64(composition_fee_x,  protocol_fee)
        } else {
            0
        };

        let lp_comp_fee_y = if (composition_fee_y > 0) {
            let protocol_share =
pair_parameter_helper::get_protocol_share(&pair.parameters);
            let protocol_fee = fee_helper::get_protocol_fee_amount(
                composition_fee_y,
                (protocol_share as u64),
            );
            safe_math::sub_u64(composition_fee_y, protocol_fee)
        } else {
            0
        };
```

Issue: When `fee_x > 0 || fee_y > 0`, `lp_comp_fee_x` ends up being equal to `lp_fee_x`. When `fee_x == 0 && fee_y == 0`, `lp_comp_fee_x` is set to 0.

Therefore, the additional calculation of `lp_comp_fee_x` is redundant and unnecessary.

Suggestion:

It is recommended to refactor the `update_bin()` function to remove the redundant calculation of `lp_comp_fee_x/y`

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **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.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

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.