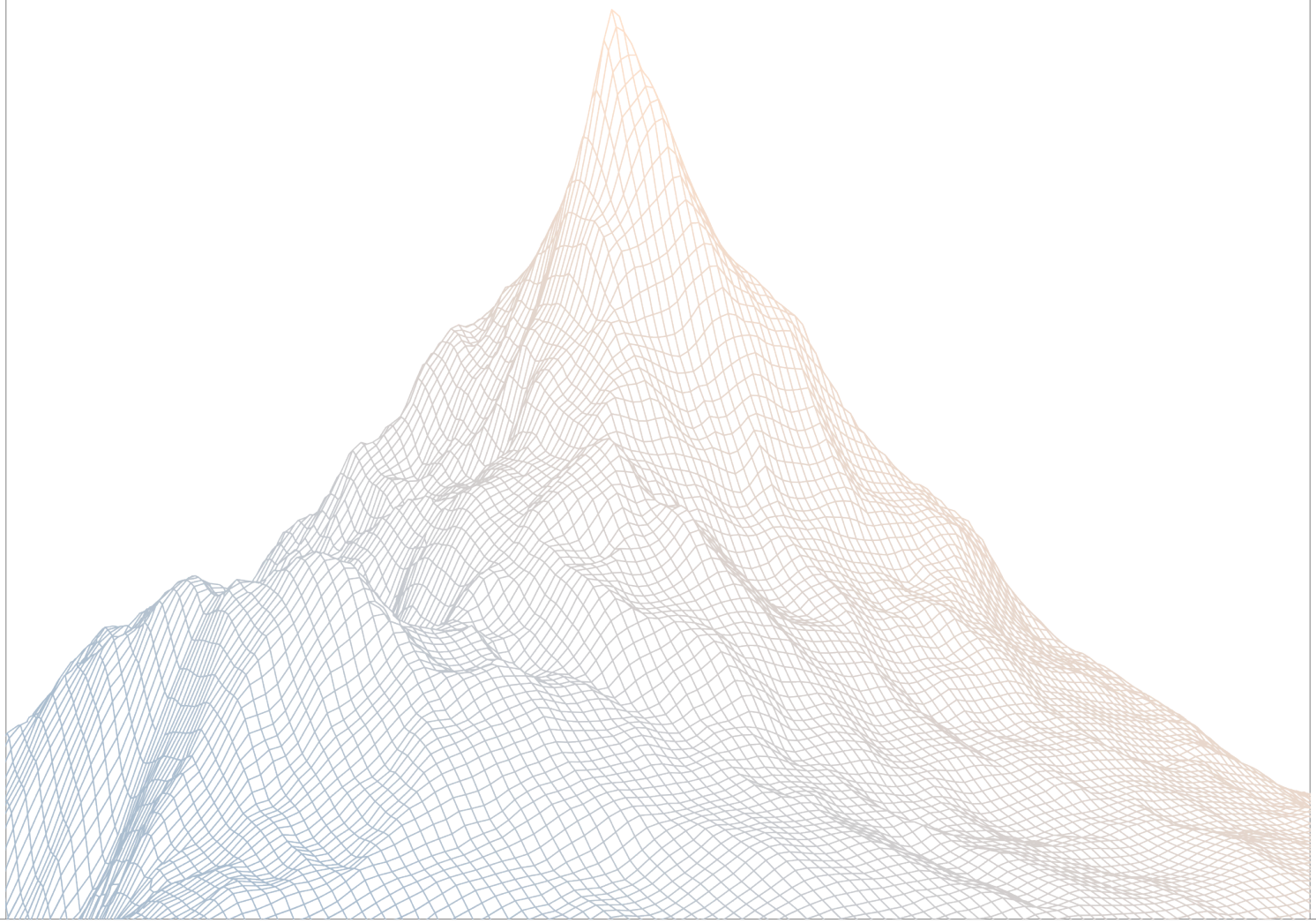
Zenith

Meteora

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: July 7th to August 19th, 2025

AUDITED BY: J4X
Mario Poneder

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Meteora DAMM v2

Our mission is to build the most secure, sustainable and composable liquidity layer for all of Solana and DeFi.

By using Meteora's DLMM and Dynamic AMM Pools, liquidity providers can earn the best fees and yield on their capital.

This would help transform Solana into the ultimate trading hub for mainstream users in crypto by driving sustainable, long-term liquidity to the platform. Join us at Meteora to shape Solana's future as the go-to destination for all crypto participants.

2.2 Scope

The engagement involved a review of the following targets:

Target	DLMM
Repository	https://github.com/MeteoraAg/DLMM
Commit Hash	`b39444d72e458c1239d50929652b0da1187d8914`
Files	`programs/lb_clmm/* (excluding tests)`

2.3 Audit Timeline

July 7, 2025	Audit start
August 19, 2025	Audit end
August 22, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	5
Low Risk	13
Informational	18
Total Issues	**36**

3

Findings Summary

ID	Description	Status
M-1	The rebalance_liquidity instruction fails when position expansion exceeds Solana's MAX_PERMITTED_DATA_INCREASE limit	Resolved
M-2	The rebalance_liquidity instruction allows positions to bypass the POSITION_MAX_LENGTH limit	Resolved
M-3	Price calculation error up to 50% when approaching the boundaries of the supported bin ID ranges	Resolved
M-4	Bin liquidity math severely limits protocol operability due to overflow	Acknowledged
M-5	Multiple v1 functions will become unusable with increased positions	Resolved
L-1	Composition fee is rounded against protocol	Acknowledged
L-2	Inconsistent freeze authority handling among SPL token and Token-2022 mints	Acknowledged
L-3	Owner can take position rent from operator	Acknowledged
L-4	Immutability of TokenMetadata and MetadataPointer is not enforced	Acknowledged
L-5	Price impact will go slightly above the intended slippage	Acknowledged
L-6	validate_mint can be bypassed by using CloseMintAuthority	Acknowledged
L-7	Event at end of swap emits wrong fee BPS	Acknowledged
L-8	Base fee validation can cause unexpected failures due to value constraints	Resolved
L-9	Incorrect memo used for fees transferred in rebalance	Resolved

ID	Description	Status
L-10	Incorrect check of pool limitations in advance_active_bin	Resolved
L-11	Liqudity can't be added for position with only one bin array	Acknowledged
L-12	Bin arrays can be created for bins outside of the supported min/max of pool	Resolved
L-13	realloc_expand_position_account does not consider actual balance of address leading to unnecessary transfer	Resolved
I-1	Unused authorize_migrate_position function in position authorization logic	Resolved
I-2	Only one customizable permissionless LB pair allowed per token X/Y due to seed restriction	Acknowledged
I-3	Token badges (Token-2022 whitelisting) cannot be revoked	Resolved
I-4	Swap and reward distribution failures due to bin array account ordering	Acknowledged
I-5	Incorrect error type used in process_rebalance_transfer function	Resolved
I-6	Artificial limitation on shrink size in decrease_position_length via MAX_RESIZE_LENGTH_USIZE check	Resolved
I-7	Already reset bins are reset again in shift_left	Acknowledged
I-8	Currently unused set_liquidity_share function allows bypassing MINIMUM_LIQUIDITY threshold	Resolved
I-9	Unused is_bin_array_has_liquidity function	Resolved
I-10	Parameter confusion in the is_liquidity_locked function might cause unintended liquidity locks	Resolved

ID	Description	Status
I-11	Unused function get_continuous_bins	Resolved
I-12	Incorrect documentation in resize_position()	Resolved
I-13	Commented out fee update validation logic	Resolved
I-14	Unresolved TODOs in codebase	Resolved
I-15	Unused functions in pda.rs	Resolved
I-16	Unused import in initialize_permission_lb_pair	Resolved
I-17	Multiple permissionless instructions are not tested	Resolved
I-18	max_bin_id_for_unit_bin_step should be defined as constant	Resolved

4

Findings

4.1 Medium Risk

A total of 5 medium risk findings were identified.

[M-1] The `rebalance_liquidity` instruction fails when position expansion exceeds Solana's `MAX_PERMITTED_DATA_INCREASE` limit

`SEVERITY:` Medium	`IMPACT:` Medium
`STATUS:` Resolved	`LIKELIHOOD:` Medium

Target

- programs/lb_clmm/src/instructions/rebalance/process_resize_position.rs#L61-L87

Description:

The `rebalance_liquidity` instruction in the protocol unexpectedly fails if the rebalancing operation requires expanding the underlying position account by more than Solana's `MAX_PERMITTED_DATA_INCREASE` (10,240 bytes) in a single instruction. This is due to a hard limit in the Solana runtime, which restricts the maximum amount by which an account's data can be increased in a single instruction.

If a rebalance operation attempts to add enough bins that the required account size increase exceeds this limit, the instruction will fail with an `InvalidRealloc` error in the runtime, which is currently neither expected nor handled by the protocol.

This limitation in legitimate use cases could result in a denial-of-service (DoS) scenario for integrators (especially programs) if not properly handled in advance, as the desired rebalance operation will be blocked unless the position is rebalanced/resized in smaller increments.

Proof of Concept:

Apply the following change to the `Rebalance: Rebalance in very far range | resize to edge` test case:

```
diff --git a/tests/test_rebalance/test_add_liquidity_and_resize_position.ts
    b/tests/test_rebalance/test_add_liquidity_and_resize_position.ts
```

```
index c440d68..d5cae4a 100644
-- a/tests/test_rebalance/test_add_liquidity_and_resize_position.ts
++ b/tests/test_rebalance/test_add_liquidity_and_resize_position.ts
@@ -346,7 +346,7 @@ describe("Rebalance: Rebalance in very far range", () ⇒
    {

     let positionState = await fetchAndDecodeDynamicPosition(program,
    position);

    const lengthToAdd = 20;
    const lengthToAdd = 46;

     console.log("Rebalance by adding liquidity empty position with
    expansion");
```

Please note that the number of bins in the modified test case is not excessively large and could still represent a legitimate use case. See also: `POSITION_MAX_LENGTH_USIZE` (1400 bins)

Recommendations:

It is recommended to apply the following mitigation measures:

- Consider adding a check and an explicit error message in the instruction handler to detect and explain this specific failure mode, improving developer and user experience.
- Provide helper functions or SDK utilities that automate the process of gradually increasing account size as needed before calling `rebalance_liquidity`.
- Document this limitation clearly for integrators and users, highlighting the need to pre-expand position accounts in increments not exceeding 10,240 bytes per instruction.

Meteora: Resolved with PR-400

Zenith: Resolved by explicitly handling this failure mode.

[M-2] The `rebalance_liquidity` instruction allows positions to bypass the `POSITION_MAX_LENGTH` limit

`SEVERITY`: Medium	`IMPACT`: Medium
`STATUS`: Resolved	`LIKELIHOOD`: Medium

Target

- programs/lb_clmm/src/instructions/rebalance/rebalance_wrapper.rs#L302-L633

Description:

The `rebalance_liquidity` instruction does not enforce the `POSITION_MAX_LENGTH` (1400 bins) limit when expanding a position's bin range. This is in contrast to the `increase_position_length` instruction, which correctly checks and enforces this protocol maximum. As a result, users can repeatedly call `rebalance_liquidity` to gradually expand a position's size well beyond the intended hard cap, breaking a protocol invariant and potentially leading to excessive position sizes, which undermine the protocol's guarantees and may have unexpected downstream effects.

Proof of Concept

Add the following test case, which increases a position's size to 2800 bins:

```
diff --git a/tests/test_rebalance/test_add_liquidity_and_resize_position.ts
    b/tests/test_rebalance/test_add_liquidity_and_resize_position.ts
index c440d68..e9b1054 100644
-- a/tests/test_rebalance/test_add_liquidity_and_resize_position.ts
++ b/tests/test_rebalance/test_add_liquidity_and_resize_position.ts
@@ -335,6 +335,91 @@ describe("Rebalance: Rebalance in very far range", ()
    => {
     }
   });

  it("should allow rebalance_liquidity to gradually exceed POSITION_MAX_
      LENGTH", async () => {
    const program = createLbClmmProgram(wallet);
    const lbPairState = await program.account.lbPair.fetch(lbPair);
```

```
let lowerBinId =
  lbPairState.activeId - DEFAULT_BIN_PER_POSITION.toNumber() / 2;
const position = await createPosition(lbPair, lowerBinId, program);

// Fetch the current position state
let positionState = await fetchAndDecodeDynamicPosition(program,
    position);

// Protocol constants

const MAX_BINS_PER_IX = 90; // slightly below the 10,240 bytes limit per
    instruction
const POSITION_MAX_LENGTH = 1400; // see constants.rs
const TARGET_WIDTH = POSITION_MAX_LENGTH * 2; // go significantly over
    the limit

let currentLower = positionState.globalData.lowerBinId;
let currentUpper = positionState.globalData.upperBinId;
let currentWidth = currentUpper - currentLower + 1;

console.log(`Initial width: ${currentWidth}`);

// Expand only on the right for simplicity
while (currentWidth < TARGET_WIDTH) {

const binsToAdd = Math.min(MAX_BINS_PER_IX, TARGET_WIDTH - currentWidth)
    ;

  // Expand to the right of the current upper bin
  const minDeltaId = currentUpper - lbPairState.activeId + 1;
  const maxDeltaId = minDeltaId + binsToAdd - 1;

  const amountX = ONE_BTC.div(new BN(TARGET_WIDTH));

  await rebalanceLiquidity({
    lbPair,
    position,
    program,
    params: {
```

```
        activeId: lbPairState.activeId,
        maxActiveBinSlippage: 3,
        shouldClaimFee: true,
        shouldClaimReward: true,
        maxDepositXAmount: MAX_AMOUNT,
        minWithdrawXAmount: new BN(0),
        maxDepositYAmount: MAX_AMOUNT,
        minWithdrawYAmount: new BN(0),
        padding: new Array(32).fill(0),
        removes: [],
        adds: [
          {
            minDeltaId,
            maxDeltaId,
            x0: amountX,
            y0: new BN(0),
            deltaX: new BN(0),
            deltaY: new BN(0),
            favorXInActiveId: true,
            padding: new Array(16).fill(0),
          },
        ],
      },
    }).catch((e) => {
      console.dir(e, { maxArrayLength: null });
      throw e;
    });

    // Fetch the new position state

  positionState = await fetchAndDecodeDynamicPosition(program, position);
    currentLower = positionState.globalData.lowerBinId;
    currentUpper = positionState.globalData.upperBinId;
    currentWidth = currentUpper - currentLower + 1;

    // Fetch the account info to get the byte size
    const accountInfo = await program.provider.connection.getAccountInfo(
        position);
    const byteSize = accountInfo?.data.length ?? 0;
```

```
    console.log(`Expanded to width: ${currentWidth}, account byte size: ${
        byteSize}`);
  }

  // This assertion proves the protocol allowed exceeding the max
  expect(currentWidth).to.be.greaterThan(POSITION_MAX_LENGTH);

  console.log(
    `Position width after gradual rebalance: ${currentWidth} exceeding ${
        POSITION_MAX_LENGTH}`
  );
});

it("resize to edge", async () => {
  const program = createLbClmmProgram(wallet);
  const lbPairState = await program.account.lbPair.fetch(lbPair);
```

Recommendations:

It is recommended to add an explicit check in the `rebalance_liquidity` instruction to ensure that the new position width does not exceed `POSITION_MAX_LENGTH` (1400 bins).

Meteora: Resolved with PR-399.

Zenith: Verified.

[M-3] Price calculation error up to 50% when approaching the boundaries of the supported bin ID ranges

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- programs/lb_clmm/src/math/u64x64_math.rs#L22-L183
- programs/lb_clmm/src/math/price_math.rs#L13-L16
- programs/lb_clmm/src/instructions/rebalance/rebalance_params.rs#L579-L640
- programs/lb_clmm/src/math/price_math.rs#L28-L51
- programs/lb_clmm/src/math/weight_to_amounts.rs#L56-L98
- programs/lb_clmm/src/math/weight_to_amounts.rs#L113-L325
- programs/lb_clmm/src/state/bin.rs#L118-L124
- programs/lb_clmm/src/state/preset_parameters2.rs#L187-L194

Description:

The protocol's Q64.64 `pow` function in `u64x64_math.rs` is optimized for bin price calculations with bases between `1.0001` and `1.04` (corresponding to `bin_step` values up to `MAX_BIN_STEP = 400`) and exponents between `-443636` and `443636` (as defined by `MIN_BIN_ID/MAX_BIN_ID`). However, the function's internal precision limitations due to approximation/optimization become apparent in the lower/upper 10% of the supported bin ID range (see `get_min_max_bin_id_from_bin_step` function), where deviations from the reference floating-point implementation can reach up to ~50%.

As a first-order consequence, the `get_price_from_id` function in `price_math.rs` returns identical prices for entire ranges of bin IDs in addition to severe price discontinuity, as can be seen in the following graph that shows the prices in logarithmic scale for the upper 10% of the positive bin IDs using `bin_step = 1`:

As a second-order consequence, the following functions are affected:

- `get_min_price_from_price_impact` (in `price_math.rs`)
  - `get_amount_in_bins_ask_side` (in `rebalance_params.rs`)
  - `to_amount_ask_side` (in `weight_to_amounts.rs`)

- `to_amount_both_side` (in `weight_to_amounts.rs`)
- `get_or_store_bin_price` (in `bin.rs`)

This can lead to miscalculations in swap amounts, liquidity distributions, and price impact calculations.

It is important to note that the protocol maintains internal consistency since it does not depend on external price data. However, the fundamental assumption that the price follows the relationship $\; price = (1 + bin_{step})^{{bin}_{ID}} \;$ is violated.

Proof of Concept:

The following PoC introduces two test cases:

1. *`test_preset_min_max_bin_id_vs_float_pow`*: Demonstrates that, for bin steps between 1 and 100 bps, price calculations can deviate by up to ~50% within the respective bin ID ranges.

2. *`test_output_raw_prices_bin_step_1`*: Produces price data (used for the above graph) for `bin_step = 1`, using the supported bin ID range.

```
diff --git a/programs/lb_clmm/src/constants.rs
    b/programs/lb_clmm/src/constants.rs
index e2f768e..cff872b 100644
-- a/programs/lb_clmm/src/constants.rs
++ b/programs/lb_clmm/src/constants.rs
@@ -164,6 +164,7 @@ pub const MINIMUM_LIQUIDITY: u128 = 1_000_000;
 pub mod tests {
     use super::*;
     use crate::math::price_math;
    use crate::math::u64x64_math::ONE;
     use crate::state::parameters::StaticParameters;
     pub const PRESET_BIN_STEP: [u16; 12] = [1, 2, 4, 5, 8, 10, 15, 20, 25,
    50, 60, 100];

@@ -383,4 +384,57 @@ pub mod tests {
             }
         }
     }

    #[test]
    fn test_preset_min_max_bin_id_vs_float_pow() {
        for bin_step in PRESET_BIN_STEP {
```

```
            let (min_bin_id, max_bin_id) =
                crate::state::preset_parameters2::get_min_max_bin_id_from_
                    bin_step(bin_step).unwrap();

            let base = 1.0 + (bin_step as f64 / BASIS_POINT_MAX as f64);
            let mut max_deviation = 0.0;
            let mut max_deviation_bin_id = 0;

            // Iterate from min_bin_id to max_bin_id
            for bin_id in min_bin_id..=max_bin_id {
                let price = price_math::get_price_from_id(bin_id, bin_step).
                    unwrap() as f64 / ONE as f64;

                // Calculate reference value using floating-point
                let price_ref = base.powi(bin_id);

                // Calculate percentage difference
                let deviation = ((price - price_ref).abs() * 100.0) / price_
                    ref;

                // Track max deviation and its corresponding bin ID
                if deviation > max_deviation {
                    max_deviation = deviation;
                    max_deviation_bin_id = bin_id;
                }
            }

            println!(
                "bin_step: {:>3}, min_id: {:>7}, max_id: {:>6}, max_dev @
                    bin_id: {:>4.2}% @ {:>6}",
                bin_step, min_bin_id, max_bin_id, max_deviation, max_
                    deviation_bin_id
            );
        }
    }

    #[test]
    fn test_output_raw_prices_bin_step_1() {
        use std::fs::File;
```

```
        use std::io::Write;

        let bin_step = 1;
        let (_min_bin_id, max_bin_id) =
                crate::state::preset_parameters2::get_min_max_bin_id_from_
                    bin_step(bin_step).unwrap();

        let mut file = File::create("raw_prices_bin_step_1.csv").expect("
            Failed to create file");

    writeln!(file, "bin_id;raw_price").expect("Failed to write header");


    // output the upper 10% of the bin positive range to demonstrate the
            problem
        for bin_id in (max_bin_id * 9 / 10)..=max_bin_id {
            let price = price_math::get_price_from_id(bin_id, bin_step).
                unwrap();
            writeln!(file, "{};{}", bin_id, price).expect("Failed to write
                data");
        }
    }
}
```

The following output of the *test_preset_min_max_bin_id_vs_float_pow* test case displays the maximum percentage deviation observed, along with the corresponding bin ID where this deviation occurs:

```
bin_step:   1, min_id: -436704, max_id: 436704, max_dev @ bin_id: 50.00% @
    432650
bin_step:   2, min_id: -218363, max_id: 218363, max_dev @ bin_id: 49.99% @
    216336
bin_step:   4, min_id: -109192, max_id: 109192, max_dev @ bin_id: 49.98% @
    108179
bin_step:   5, min_id:  -87358, max_id:  87358, max_dev @ bin_id: 49.94%
    @  86548
bin_step:   8, min_id:  -54190, max_id:  54190, max_dev @ bin_id: 49.90%
    @  54101
bin_step:  10, min_id:  -43690, max_id:  43690, max_dev @ bin_id: 49.92%
    @  43285
```

```
bin_step:  15, min_id:  -29134, max_id:  29134, max_dev @ bin_id: 49.89%
    @  28864
bin_step:  20, min_id:  -21855, max_id:  21855, max_dev @ bin_id: 49.71%
    @  21654
bin_step:  25, min_id:  -17481, max_id:  17481, max_dev @ bin_id: 52.55%
    @  17320
bin_step:  50, min_id:   -8754, max_id:   8754, max_dev @ bin_id: 50.88%
    @   8673
bin_step:  60, min_id:   -7299, max_id:   7299, max_dev @ bin_id: 49.17%
    @   7233
bin_step: 100, min_id:   -4386, max_id:   4386, max_dev @ bin_id: 52.83%
    @   4346
```

Recommendations:

It is recommended to increase the internal precision of the `pow` function from Q64.64 to Q128.128 (if feasible concerning compute units), and/or ensure that the `min_bin_id` and `max_bin_id` returned by the `get_min_max_bin_id_from_bin_step` function (in `preset_parameters2.rs`) are conservatively bounded to avoid price calculations in a numerically unstable range.

For example, apply the following changes to achieve internal Q128.128 precision in the `pow` function:

```
diff --git a/programs/lb_clmm/src/math/u64x64_math.rs
    b/programs/lb_clmm/src/math/u64x64_math.rs
index 2fae387..a4c9c79 100644
-- a/programs/lb_clmm/src/math/u64x64_math.rs
++ b/programs/lb_clmm/src/math/u64x64_math.rs
@@ -7,6 +7,9 @@ pub const PRECISION: u128 = 1_000_000_000_000;
 // Number of bits to scale. This will decide the position of the radix
    point.
 pub const SCALE_OFFSET: u8 = 64;

// Internal scale offset for Q128.128 calculations
const INTERNAL_SCALE_OFFSET: u8 = 128;

 // Where does this value come from ?
 // When smallest bin is used (1 bps), the maximum of bin limit is 887272
    (Check: https://docs.traderjoexyz.com/concepts/bin-math).
 // But in solana, the token amount is represented in 64 bits, therefore, it
    will be (1 + 0.0001)^n < 2 ** 64, solve for n, n ~= 443636
@@ -36,8 +39,9 @@ pub fn pow(base: u128, exp: i32) -> Option<u128> {
         return None;
```

```
     }

    let mut squared_base = base;
    let mut result = ONE;
    // Convert base to Q128.128 format
    let mut squared_base = U256::from(base) << (INTERNAL_SCALE_OFFSET -
        SCALE_OFFSET);
    let mut result = U256::from(1u128) << INTERNAL_SCALE_OFFSET;

     // When multiply the base twice, the number of bits double from 128 →
    256, which overflow.
     // The trick here is to inverse the calculation, which make the upper 64
    bits (number bits) to be 0s.
@@ -48,7 +52,7 @@ pub fn pow(base: u128, exp: i32) -> Option<u128> {
     // By using a calculator, you will find out that 1.001^5 == 1 / (1 /
    1.001^5)
     if squared_base >= result {
         // This inverse the base: 1 / base
        squared_base = u128::MAX.checked_div(squared_base)?;
        squared_base = U256::MAX.checked_div(squared_base)?;
         // If exponent is negative, the above already inverted the result.
    Therefore, at the end of the function, we do not need to invert again.
         invert = !invert;
     }
@@ -71,115 +75,122 @@ pub fn pow(base: u128, exp: i32) -> Option<u128> {
     // squared_base = 1 * base^1
     // 1st bit is 1
     if exp & 0x1 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
     }

     // squared_base = base^2

    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
    squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
     // 2nd bit is 1
     if exp & 0x2 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET
```

```
    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }

    // Example:
    // If the base is 1.001, exponential is 3. Binary form of 3 is ..0011.
The last 2 1's bit fulfill the above 2 bitwise condition.
    // The result will be 1 * base^1 * base^2 == base^3. The process
continues until reach the 20th bit


    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
    if exp & 0x4 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }


    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
    if exp & 0x8 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }


    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
    if exp & 0x10 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }


    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
```

```
    squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
    if exp & 0x20 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }


    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
    squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
    if exp & 0x40 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }


    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
    squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
    if exp & 0x80 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }


    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
    squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
    if exp & 0x100 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }


    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
    squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
```

```
        SCALE_OFFSET;
    if exp & 0x200 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }

    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
    squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
    if exp & 0x400 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }

    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
    squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
    if exp & 0x800 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }

    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
    squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
    if exp & 0x1000 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }

    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
    squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
```

```
    if exp & 0x2000 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }

    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
    squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
    if exp & 0x4000 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }

    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
    squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
    if exp & 0x8000 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }

    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
    squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
    if exp & 0x10000 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }

    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
    squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
    if exp & 0x20000 > 0 {
```

```
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET
    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }

    squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
    squared_base = (squared_base.checked_mul(squared_base)?) >> INTERNAL_
        SCALE_OFFSET;
    if exp & 0x40000 > 0 {
        result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET

    result = (result.checked_mul(squared_base)?) >> INTERNAL_SCALE_OFFSET
    }

    // Stop here as the next is 20th bit, which > MAX_EXPONENTIAL
    if result == 0 {
    if result == U256::ZERO {
        return None;
    }

    if invert {
        result = u128::MAX.checked_div(result)?;
        result = U256::MAX.checked_div(result)?;
    }

    // Convert back to Q64.64 format
    result = result >> (INTERNAL_SCALE_OFFSET - SCALE_OFFSET);

    if result == U256::ZERO {
        return None;
    }

    Some(result)
    Some(result.try_into().ok()?)
}

// Helper function to convert fixed point number to decimal with 10^12
   precision. Decimal form is not being used in program, it's only for UI
   purpose.
```

Meteora: Resolved with PR-417.

Zenith: Resolved by reducing the constant (max bin ID for `bin_step = 1`) in the `get_min_max_bin_id_from_bin_step` function.

[M-4] Bin liquidity math severely limits protocol operability due to overflow

SEVERITY: Medium	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: High

Target

- src/math/bin_math.rs#L23

Description:

The current bin liquidity calculation in `get_liquidity()` function has a critical limitation that severely restricts the protocol's operability. The issue arises from the mathematical constraint that a `u128` price (derived from bin ID) multiplied by a `u64` X amount must fit within `u128`. However if the price gets significantly large, depositing even dust amounts of liquidity will already lead to a overflow when trying to cast the result (`u256`) into a `u128`.

```
pub fn get_liquidity(x: u64, y: u64, price: u128) -> Result<u128> {
    // Q64x0
    let x: U256 = U256::from(x);

    // Multiplication do not require same Q number format. px is in Q64x64
    let price = U256::from(price);
    let px = price.safe_mul(x)?;

    // When perform add, both must be same Q number format. Therefore <<
    SCALE_OFFSET to make y and px Q64x64
    let y = u128::from(y).safe_shl(SCALE_OFFSET.into())?;
    let y = U256::from(y);
    // Liquidity represented with fractional part
    let liquidity = px.safe_add(U256::from(y))?;
    Ok(liquidity.try_into().map_err(|_| LBError::TypeCastFailed)?)
}
```

At the end of the price spectrum, for example, at bin ID 351,639 with `bin_step = 1`, the protocol allows a maximum X amount that equates to less than 0.01 USDC or 0.0001 BTC. Any liquidity attempt above this threshold causes a `TypeCastFailed` error due to the overflow. As a result, the protocol becomes effectively unusable for liquidity provision at these high price levels. This overflow issue becomes increasingly restrictive with higher bin IDs, creating unusable "dead zones" where adding meaningful liquidity is technically

impossible.

Proof of Concept:

Add to `bin_math.rs`:

```
#[test]
fn test_get_max_x_amount_before_overflow() {
    let y_amount = 0;

    let active_id = 351_639; // max bin id @ bin_step=1 (after recent fix,
    previously 436_704)
    let bin_step = 1;

    let price = get_price_from_id(active_id, bin_step).unwrap();
    let max_x_amount:
    u64 = u128::MAX.safe_div(price).unwrap().try_into().unwrap();

    // expect liquidity calculation to succees with max_x_amount
    let result = get_liquidity(max_x_amount, y_amount, price);
    assert!(result.is_ok());

    // expect liquidity calculation to fail with max_x_amount + 1
    let result = get_liquidity(max_x_amount + 1, y_amount, price);
    assert!(result.is_err());
    assert_eq!(result.unwrap_err(), LBError::TypeCastFailed.into());

    // print max_x_amount, also scaled as BTC and USDC
    let max_x_amount_btc = max_x_amount as f64 / 10.0f64.powi(8);
    let max_x_amount_usdc = max_x_amount as f64 / 10.0f64.powi(6);
    println!("max_x_amount (raw)    : {}", max_x_amount);
    println!("max_x_amount (as BTC) : {}", max_x_amount_btc);
    println!("max_x_amount (as USDC): {}", max_x_amount_usdc);
}
```

Output:

```
max_x_amount (raw)    : 9889
max_x_amount (as BTC) : 0.00009889
max_x_amount (as USDC): 0.009889
```

Add to `add_liquidity.ts` (tries to add 0.0001 BTC at bin 351639, fails with `TypeCastFailed`):

```
it("add liquidity to max bin (id=351_639) for bin_step=1", async () => {
  const program = createLbClmmProgram(wallet);

  // Create a new LB pair with bin_step=1 for testing outermost bins
  const binStepOne = new BN(1);
  const maxBinId = 351_639; // max bin id for bin_step=1 (after recent fix,
    previously 436_704)

  // Create preset parameter for bin_step=1
  await createPresetParameter(
    binStepOne,
    createLbClmmProgram(new Wallet(ADMIN))
  );

  // Create a new LB pair with bin_step=1
  const lbPairBinStepOne = await createLbPair(BTC, USDC, new BN(0),
    binStepOne, program);
  const [reserveXBinStepOne] = deriveReserve(BTC, lbPairBinStepOne,
    program.programId);
  const [reserveYBinStepOne] = deriveReserve(USDC, lbPairBinStepOne,
    program.programId);

  const lbPairStateBinStepOne = await
    program.account.lbPair.fetch(lbPairBinStepOne);

  // Update base fee parameters for the new pair
  await createLbClmmProgram(new Wallet(ADMIN))
    .methods.updateBaseFeeParameters({
      protocolShare,
      baseFactor: lbPairStateBinStepOne.parameters.baseFactor,
      baseFeePowerFactor: 0,
    })
    .accountsPartial({
      admin: ADMIN.publicKey,
      lbPair: lbPairBinStepOne,
    })
    .rpc();

  // Create a position at the maximum bin ID
  const positionWidth = 1; // Single bin position
  const positionAtMaxBin = await createPosition(
    lbPairBinStepOne,
    maxBinId,
    program,
    positionWidth
  );
```

```
// Add liquidity to the maximum bin ID
const amountXMaxBin = ONE_BTC.div(new BN(10000)); // 0.0001 BTC
const amountYMaxBin = new BN(0);

// Initialize bitmap extension account for extreme bin IDs
await initializeBinArrayBitmapExtension(lbPairBinStepOne, program);

// Get bin arrays for the position - use getOrCreate to ensure they exist
const { binArrayLower: maxBinArrayLower, binArrayUpper:
  maxBinArrayUpper } =
  await getOrCreateBinArraysForModifyLiquidity(
    lbPairBinStepOne,
    maxBinId,
    program
  );

// Get the bitmap extension account
const [binArrayBitmapExtension] = deriveBinArrayBitmapExtension(
  lbPairBinStepOne,
  program.programId
);

const userTokenX = getAssociatedTokenAddressSync(
  BTC,
  program.provider.publicKey,
  false,
  TOKEN_PROGRAM_ID,
  ASSOCIATED_TOKEN_PROGRAM_ID
);

const userTokenY = getAssociatedTokenAddressSync(
  USDC,
  program.provider.publicKey,
  false,
  TOKEN_PROGRAM_ID,
  ASSOCIATED_TOKEN_PROGRAM_ID
);

await program.methods
  .addLiquidity({
    amountX: amountXMaxBin,
    amountY: amountYMaxBin,
    binLiquidityDist: [
      {
        binId: maxBinId,
        distributionX: 10000, // 100% of X
```

```
          distributionY: 0,
        },
      ],
    })
    .accountsPartial({
      lbPair: lbPairBinStepOne,
      binArrayBitmapExtension,
      position: positionAtMaxBin,
      reserveX: reserveXBinStepOne,
      reserveY: reserveYBinStepOne,
      tokenXMint: BTC,
      tokenYMint: USDC,
      tokenXProgram: TOKEN_PROGRAM_ID,
      tokenYProgram: TOKEN_PROGRAM_ID,
      sender: program.provider.publicKey,
      userTokenX,
      userTokenY,
      binArrayLower: maxBinArrayLower,
      binArrayUpper: maxBinArrayUpper,
    })
    .preInstructions([SET_COMPUTE_UNIT_LIMIT_IX])
    .rpc();
});
```

Recommendations:

Option 1: Restrict Supported Bin/Price Ranges (Immediate Fix)

- Reduce the maximum supported bin ID from 351,639 to a more conservative value
- Implement validation to prevent liquidity addition at extreme bin IDs
- Add clear error messages when users attempt to add liquidity in unsupported ranges

Option 2: Switch to U256 Types Internally (Long-term Solution)

- Modify the `get_liquidity()` function to use `U256` for intermediate calculations
- Update all related math functions to handle larger number ranges
- This would allow the protocol to support realistic amounts across the entire price range

Meteora: Acknowledged.

[M-5] Multiple v1 functions will become unusable with increased positions

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- src/instructions/deposit/add_liquidity.rs#L256-L261
- src/instructions/update_fees_and_rewards.rs#L56

Description:

The `add_liquidity` is intended to be still usable with the new position formats. However, it will no longer be usable once a position spans more than 2 BitArrays. Even if the addition of liquidity will only go into less than 140 bins, the call will always revert as `update_earning_per_token_stored` will try to access the bins for the whole position, but these can't be covered by the 2 BinArrays provided. The call will error in the `update_earning_per_token_stored` with the `InvalidBinArray` function as it will try to

```
/// Update reward + fee earning
pub fn update_earning_per_token_stored(
    &mut self,
    reward_flags: &[bool; NUM_REWARDS_USIZE],
    bin_array_manager: &BinArrayManager,
    min_bin_id: i32,
    max_bin_id: i32,
) -> Result<()> {
    self.assert_valid_bin_range_for_modify_position(min_bin_id,
    max_bin_id)?;
    let (bin_arrays_lower_bin_id, bin_arrays_upper_bin_id) =
        bin_array_manager.get_lower_upper_bin_id()?;

    require!(
        min_bin_id ⩾ bin_arrays_lower_bin_id && max_bin_id ⩽
    bin_arrays_upper_bin_id,
        LBError::InvalidBinArray
    );
```

The same issue is also present in the `UpdateFeesAndRewards` function.

POC:

```
import * as anchor from "@coral-xyz/anchor";
import { Wallet, web3 } from "@coral-xyz/anchor";
import {
  ASSOCIATED_TOKEN_PROGRAM_ID,
  TOKEN_PROGRAM_ID,
  createMint,
  getAssociatedTokenAddressSync,
  getOrCreateAssociatedTokenAccount,
  mintTo,
} from "@solana/spl-token";
import { SystemProgram } from "@solana/web3.js";
import { BN } from "bn.js";
import { expect } from "chai";
import {
  ADMIN,
  Bins,
  CompositionFeeEventCpi,
  Enable,
  MAX_BIN_PER_ARRAY,
  DEFAULT_BIN_PER_POSITION,
  SET_COMPUTE_UNIT_LIMIT_IX,
  addLiquidityRadius,
  assertPositionAndBinLiquidityDelta,
  binIdToBinArrayIndex,
  chunkedGetMultipleAccountInfos,
  createAndFundWallet,
  createBinArrays,
  createLbClmmProgram,
  createLbPair,
  createPosition,
  createPresetParameter,
  decUiAmount,
  deriveBinArray,
  deriveBinArrayBitmapExtension,
  deriveReserve,
  fetchPositionByOwners as fetchPositionByOwner,
  fpToDec,
  getBinArrayLowerUpperBinId,
  getBinArraysForModifyLiquidity,
  getOrCreateBinArraysForModifyLiquidity,
  getPriceFromBinId,
  getSpotPatternDistribution,
  invokeAndAssertError,
  parseEventCpi,
```

```
  sleep,
  fetchAndDecodeDynamicPosition,
} from "../utils";

const provider = anchor.AnchorProvider.env();

describe("Add Liquidity Extended Position", () => {
  const keypair: web3.Keypair = web3.Keypair.generate();
  const wallet: Wallet = new Wallet(keypair);

  const keypair2: web3.Keypair = web3.Keypair.generate();
  const wallet2: Wallet = new Wallet(keypair2);

  const activeId = new BN(5660);
  const binStep = new BN(10);

  const btcDecimal = 8;
  const usdcDecimal = 6;

  const protocolShare = 1000;

  // 5 = Create 5 lower bin arrays, and 5 upper bin arrays surrounding the
    active bin arrays. Total bins = 600 * 11
  const binArrayDelta = 5;
  const binArrayIndex = binIdToBinArrayIndex(activeId);
  const binArrayIndexes = [];

  const ONE_BTC = new BN(1 * 10 ** btcDecimal);
  const ONE_USDC = new BN(1 * 10 ** usdcDecimal);

  let BTC: web3.PublicKey;
  let USDC: web3.PublicKey;
  let lbPair: web3.PublicKey;
  let reserveX: web3.PublicKey;
  let reserveY: web3.PublicKey;

  const balancedDelta = DEFAULT_BIN_PER_POSITION.div(new BN(2));
  const lowerBinId = activeId.sub(balancedDelta);
  const upperBinId
    = lowerBinId.add(DEFAULT_BIN_PER_POSITION.sub(new BN(1)));

  before(async () => {
    await createAndFundWallet(provider.connection, keypair);
    await createAndFundWallet(provider.connection, keypair2);
    // Lower bin arrays
    for (let i = binArrayDelta; i > 0; i--) {
      const idx = binArrayIndex.sub(new BN(i));
```

```
      binArrayIndexes.push(idx);
    }

    binArrayIndexes.push(binArrayIndex);

    // Upper bin arrays
    for (let i = 1; i <= binArrayDelta; i++) {
      const idx = binArrayIndex.add(new BN(i));
      binArrayIndexes.push(idx);
    }
  });

  beforeEach(async () => {
    BTC = await createMint(
      provider.connection,
      keypair,
      wallet.publicKey,
      null,
      btcDecimal,
      web3.Keypair.generate(),
      null,
      TOKEN_PROGRAM_ID
    );

    USDC = await createMint(
      provider.connection,
      keypair,
      wallet.publicKey,
      null,
      usdcDecimal,
      web3.Keypair.generate(),
      null,
      TOKEN_PROGRAM_ID
    );

    const program = createLbClmmProgram(wallet);

    await createPresetParameter(
      binStep,
      createLbClmmProgram(new Wallet(ADMIN))
    );

    lbPair = await createLbPair(BTC, USDC, activeId, binStep, program);
    [reserveX] = deriveReserve(BTC, lbPair, program.programId);
    [reserveY] = deriveReserve(USDC, lbPair, program.programId);

    const lbPairState = await program.account.lbPair.fetch(lbPair);
```

```
await createLbClmmProgram(new Wallet(ADMIN))
  .methods.updateBaseFeeParameters({
    protocolShare,
    baseFactor: lbPairState.parameters.baseFactor,
    baseFeePowerFactor: 0,
  })
  .accountsPartial({
    admin: ADMIN.publicKey,
    lbPair,
  })
  .rpc();

await createBinArrays(lbPair, binArrayIndexes, program);

const userTokenX = await getOrCreateAssociatedTokenAccount(
  provider.connection,
  keypair,
  BTC,
  keypair.publicKey,
  false,
  "confirmed",
  {
    commitment: "confirmed",
  },
  TOKEN_PROGRAM_ID,
  ASSOCIATED_TOKEN_PROGRAM_ID
);

const userTokenY = await getOrCreateAssociatedTokenAccount(
  provider.connection,
  keypair,
  USDC,
  keypair.publicKey,
  false,
  "confirmed",
  {
    commitment: "confirmed",
  },
  TOKEN_PROGRAM_ID,
  ASSOCIATED_TOKEN_PROGRAM_ID
);

await mintTo(
  provider.connection,
  keypair,
  BTC,
```

```
      userTokenX.address,
      keypair.publicKey,
      100 * 10 ** btcDecimal,
      [],
      {
        commitment: "confirmed",
      },
      TOKEN_PROGRAM_ID
    );

    await mintTo(
      provider.connection,
      keypair,
      USDC,
      userTokenY.address,
      keypair.publicKey,
      100_000 * 10 ** usdcDecimal,
      [],
      {
        commitment: "confirmed",
      },
      TOKEN_PROGRAM_ID
    );
  });

it("deposit enlarged position", async () => {
    const program = createLbClmmProgram(wallet);

    const amountX = ONE_BTC; // 1 btc
    const amountY = ONE_USDC.mul(new BN(10000)); // 10k

    const position = await createPosition(
      lbPair,
      10,
      program
    );

    // Increase position size by 70 (from 70 to 140)
    await program.methods
      .increasePositionLength(70, 0) // Add 70 bins, side 0 (lower side)
      .accountsPartial({
        funder: program.provider.publicKey,
        position,
        owner: program.provider.publicKey,
        systemProgram: SystemProgram.programId,
        lbPair,
      })
```

```
    .rpc();

  // Get the position state after enlargement
  const lbPairState = await program.account.lbPair.fetch(lbPair);
  const positionState = await fetchAndDecodeDynamicPosition(program,
  position);

  // Get or create the bin arrays for the enlarged position
  const { binArrayLower, binArrayUpper }
  = await getOrCreateBinArraysForModifyLiquidity(
    lbPair,
    positionState.globalData.lowerBinId,
    program
  );

  // Add liquidity directly using the program method instead of
  addLiquidityRadius
  const mintAccountsInfo = await chunkedGetMultipleAccountInfos(
    program.provider.connection,
    [lbPairState.tokenXMint, lbPairState.tokenYMint]
  );

  const userTokenX = getAssociatedTokenAddressSync(
    lbPairState.tokenXMint,
    program.provider.publicKey,
    false,
    mintAccountsInfo[0].owner,
    ASSOCIATED_TOKEN_PROGRAM_ID
  );

  const userTokenY = getAssociatedTokenAddressSync(
    lbPairState.tokenYMint,
    program.provider.publicKey,
    false,
    mintAccountsInfo[1].owner,
    ASSOCIATED_TOKEN_PROGRAM_ID
  );

  // This should fail because addLiquidity can only handle 2 bin arrays
  // but a 140-bin position spans multiple bin arrays
  try {
    await program.methods
      .addLiquidity({
        amountX,
        amountY,
        binLiquidityDist: [
          {
```

```
            binId: lbPairState.activeId,
            distributionX: 10000,
            distributionY: 10000,
          },
        ],
      })
      .accountsPartial({
        lbPair,
        binArrayBitmapExtension: null,
        position,
        reserveX: lbPairState.reserveX,
        reserveY: lbPairState.reserveY,
        tokenXMint: lbPairState.tokenXMint,
        tokenYMint: lbPairState.tokenYMint,
        tokenXProgram: mintAccountsInfo[0].owner,
        tokenYProgram: mintAccountsInfo[1].owner,
        sender: program.provider.publicKey,
        userTokenX,
        userTokenY,
        binArrayLower,
        binArrayUpper,
      })
      .preInstructions([SET_COMPUTE_UNIT_LIMIT_IX])
      .rpc();
  } catch (error) {
    // Expect the InvalidBinArray error when trying to add liquidity to a
  large position
    console.log("Expected error occurred:", error.toString());
    expect(error.toString()).to.include("InvalidBinArray");
    return; // Test passes if this error occurs
  }
 });
});
```

Recommendations:

We recommend adapting the function to use the same system as `add_liquidity2` or clearly document that it will become unusable once the position spans more than 2 BinArrays.

Meteora: Resolved with PR-423

Zenith: Verified

4.2 Low Risk

A total of 13 low risk findings were identified.

[L-1] Composition fee is rounded against protocol

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- state/lb_pair.rs#L622-L630

Description:

On adding liquidity, the calculated composition fee is split between the protocol and the actual composition fee.

```
let (protocol_fee_x, protocol_fee_y) =
    charge_protocol_fee(lb_pair, composition_fee_x, composition_fee_y)?;

// pay swap fee firstly
bin.deposit_composition_fee(
    composition_fee_x.safe_sub(protocol_fee_x)?,
    composition_fee_y.safe_sub(protocol_fee_y)?,
)?;
```

However, when calculating the protocol part of the fee, the division is rounded down.

```
/// Compute protocol fee
pub fn compute_protocol_fee(&self, fee_amount: u64) -> Result<u64> {
    let protocol_fee = u128::from(fee_amount)
        .safe_mul(self.parameters.protocol_share.into())?
        .safe_div(BASIS_POINT_MAX as u128)?;

    Ok(protocol_fee
        .try_into()
        .map_err(|_| LBError::TypeCastFailed)?)
}
```

This results in unfavorable rounding, although it's only dust amounts.

Recommendations:

We recommend rounding in favor of the protocol.

Meteora: Acknowledged

[L-2] Inconsistent freeze authority handling among SPL token and Token-2022 mints

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- programs/lb_clmm/src/utils/token2022.rs#L64-L66

Description:

In the current protocol implementation, Token-2022 mints are only allowed to have a `freeze_authority` if a token badge has been initialized by the admin (i.e., the mint is explicitly whitelisted). This is enforced in `token2022.rs` via the `validate_mint` function. However, there is no equivalent check for classic SPL token mints, meaning anyone can create permissionless LB pairs with SPL tokens that have an active freeze authority. This creates an inconsistency: freezable Token 2022 mints require admin whitelisting, but freezable SPL tokens do not, potentially exposing users to the risk of their funds being frozen in pools created with such SPL tokens.

Recommendations:

It is recommended to add a check for SPL token mints to ensure that mints with a `freeze_authority` are either disallowed or require explicit whitelisting (e.g., via a token badge or similar mechanism), mirroring the logic used for Token 2022 mints.

Meteora: Acknowledged. We have a UI warning for tokens with freeze authority. Updating it will break integrators, as the behavior of accepting SPL tokens with freeze authority has been there since the beginning.

[L-3] Owner can take position rent from operator

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- instructions/position/initialize_position_by_operator.rs#L28

Description:

In the case of a position creation via `initialize_position_by_operator` the operator covers the rent for the position account. However, the owner can close the position at any time and steal the rent from the operator.

Recommendations:

We recommend tracking the rent payer in the position account and refunding the rent to him on closure.

Meteora: Acknowledged

[L-4] Immutability of `TokenMetadata` and `MetadataPointer` is not enforced

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- programs/lb_clmm/src/utils/token2022.rs#L75-L76

Description:

Currently, the protocol allows tokens with the `TokenMetadata` and `MetadataPointer` extensions to be used in pool creation without checking whether their metadata is mutable. If the `updateAuthority` (for `TokenMetadata`) or the `authority` (for `MetadataPointer`) is not set to `None`, the token's metadata (name, symbol, URI, etc.) can be changed after the pool is created. This opens the door to misleading or malicious metadata changes, such as impersonation, which can harm users and integrators who rely on the displayed metadata.

Recommendations:

It is recommended to require that tokens with mutable metadata are explicitly whitelisted via a token badge.

Meteora: Acknowledged.

[L-5] Price impact will go slightly above the intended slippage

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- src/instructions/swap.rs#L246-L252

Description:

When using the `handle_exact_in_with_price_impact` endpoint, users can set a maximum/minimum price they are willing to pay for their swap. This is done by calculating the `amount_out` they would receive by swapping their `amount_in` at that price and checking if the actual `amount_out` is equal to or higher than it.

```
let min_price =
    get_min_price_from_price_impact(active_id, bin_step,
    max_price_impact_bps, swap_for_y)?;
let min_amount_out = Bin::get_amount_out(amount_in, min_price, swap_for_y)?;
require!(
    amount_out ⩾ min_amount_out,
    LBError::ExceededAmountSlippageTolerance
);
```

This amount is retrieved from the `get_amount_out` function.

```
/// Get out token amount from the bin based in amount in. The result is
    floor-ed.
/// X → Y: inX * bin_price
/// Y → X: inY / bin_price
pub fn get_amount_out(amount_in: u64, price: u128, swap_for_y: bool) →
    Result<u64> {
    if swap_for_y {
        // (Q64x64(price) * Q64x0(amount_in)) >> SCALE_OFFSET
        // price * amount_in = amount_out_token_y (Q64x64)
        // amount_out_in_token_y >> SCALE_OFFSET (convert it back to integer
    form, with some loss of precision [Rounding::Down])
        safe_mul_shr_cast(price, amount_in.into(), SCALE_OFFSET,
    Rounding::Down)
```

```
    } else {
        // (amount_in << SCALE_OFFSET) / price
        // Convert amount_in into Q64x0, if not the result will always in 0
    as price is in Q64x64
        // Division between same Q number format cancel out, result in
    integer
        // amount_in / price = amount_out_token_x (integer [Rounding::Down])
        safe_shl_div_cast(amount_in.into(), price, SCALE_OFFSET,
    Rounding::Down)
    }
}
```

However, this function will round down, resulting in the user's slippage being set slightly below the intended one.

Recommendations:

We recommend adding a rounding direction flag to the `get_amount_out()` function and setting it to round up in the case described above.

Meteora: Acknowledged

[L-6] `validate_mint` can be bypassed by using `CloseMintAuthority`

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- src/utils/token2022.rs#L97-L99

Description:

The `token2022` util restricts which mint extensions are allowed to be enabled on mints for which pools are launched. This is implemented in the `validate_mint()` function. The function currently completely disallows some extensions, while some others are allowed if the mint is whitelisted.

One of the acceptable, if whitelisted, extensions is the `CloseMintAuthority` extension. This extension allows the owner to close the mint if the total supply is 0. A malicious owner could exploit this to bypass the restriction on disallowed extensions, even if they are whitelisted. This can be achieved by creating a pool with a zero supply and no forbidden extensions on the mint. Then the `CloseMintAuthority` can close and reopen the mint with some forbidden extensions, and the pool would still exist and be usable.

The supply of 0 can be achieved by either not minting any tokens and deploying via `initialize_permission_lp_pair` or by using the customizable route, but this can be done either by burning all tokens post-deployment (by owning all of them yourself) or by utilizing the PermanentDelegate feature.

Recommendations:

We recommend either adding `validate_mint` at the start of every ix interacting with the mints or completely disabling `CloseMintAuthority`.

Meteora: Acknowledged

[L-7] Event at end of swap emits wrong fee BPS

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- src/instructions/swap.rs#L422
- src/instructions/v2/swap2.rs#L480

Description:

The `SwapEvent` at the end of each swap emits the `fee_bps` attribute.

```
emit_cpi!(SwapEvent {
    lb_pair: ctx.accounts.lb_pair.key(),
    amount_in: total_amount_in,
    amount_out: total_amount_out,
    fee: total_fee,
    protocol_fee: total_protocol_fee,
    start_bin_id: start_active_id,
    end_bin_id: end_active_id,
    swap_for_y,
    from: ctx.accounts.user.key(),
    fee_bps,
    host_fee: total_host_fee,
});
```

This value is retrieved by calling `get_total_fee()` at the end of the swap.

```
let fee_bps = lb_pair.get_total_fee()?;
```

However, this will return the total fee BPS taken after the swap, as the fee gets updated after every bin due to the dynamic fee changing with every bin. Thus, this value will be higher than the actual fee BPS taken on the `amount_in`.

Recommendations:

We recommend either renaming it to `final_fee_BPS` or calculating the actual fee BPS taken by using the following calculation:

$$(10000 * fee) / amountIn$$

Meteora: Acknowledged

[L-8] Base fee validation can cause unexpected failures due to value constraints

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/lb_clmm/src/state/preset_parameters2.rs#L128-L139
- programs/lb_clmm/src/state/lb_pair.rs#L502-L513

Description:

The validation logic in `PoolPresetParameter::validate_initialize` imposes constraints on the calculated `base_fee` value that can lead to unexpected failures during preset and pool initialization, as well as during fee parameter updates. The issue stems from validating a derived value rather than the input parameters directly.

The `base_fee` is calculated from three user inputs (`base_factor`, `bin_step`, `base_fee_power_factor`) via `LbPair::calculate_base_fee`, but the validation checks the calculated result rather than the input parameters.

The validation requires:

- `base_fee ⩾ MIN_BASE_FEE && base_fee ⩽ MAX_BASE_FEE`
  - `base_fee % MIN_BASE_FEE == 0`

Users may provide seemingly valid input parameters that, when combined, produce a `base_fee` value that fails these constraints, causing the transaction to revert without clear indication of which input parameter is problematic.

Recommendations:

It is recommended to modify the `LbPair::calculate_base_fee` function to ensure the result is always a multiple of `MIN_BASE_FEE`: If modifying the calculation is not desired, improve the validation to provide clearer error messages.

Meteora: Resolved with PR-412.

Zenith: Verified. Resolved by providing a clearer error message.

[L-9] Incorrect memo used for fees transferred in rebalance

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- src/instructions/rebalance/rebalance_wrapper.rs#L248

Description:

The `process_rebalance_transfer()` function is used to bundle the withdrawn fees and funds on a rebalance. It will sum both values and transfer them in one transfer.

```
if removed_amount > added_amount {
    let withdraw_amount = removed_amount.safe_sub(added_amount)?;
    let memo_transfer_ctx = MemoTransferContext {
        memo_program,
        memo: REMOVE_LIQUIDITY_MEMO,
    };
    let excluded_fee_withdraw_amount =
        calculate_transfer_fee_excluded_amount(token_mint,
    withdraw_amount)?.amount;
    require!(
        excluded_fee_withdraw_amount ≥ min_withdraw_amount,
        LBError::ExceededBinSlippageTolerance
    );
    transfer_from_pool_to_user2(
        lb_pair,
        token_mint,
        reserve,
        user_token,
        token_program,
        withdraw_amount,
        Some(memo_transfer_ctx),
        transfer_hook_accounts,
    )?;
```

This will use the `REMOVE_LIQUIDITY_MEMO`, which only mentions that liquidity is removed but does not include that fees were also withdrawn.

Recommendations:

We recommend adding a new memo `pub const REMOVE_LIQUIDITY_AND_FEE_MEMO: &[u8] = b"DLMM RemoveLiquidity && ClaimFees";` and using that in `process_rebalance_transfers`

Meteora: Resolved with PR-414

Zenith: Verified

[L-10] Incorrect check of pool limitations in `advance_active_bin`

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- src/state/lb_pair.rs#L485-L500

Description:

The `advance_active_bin` function is used to advance the bins during a swap. It will advance the bin in the correct direction and checks if we have reached the min/max of the pool.

```
/// Plus / Minus 1 to the active bin based on the swap direction
pub fn advance_active_bin(&mut self, swap_for_y: bool) → Result<()> {
    let next_active_bin_id = if swap_for_y {
        self.active_id.safe_sub(1)?
    } else {
        self.active_id.safe_add(1)?
    };

    require!(
        next_active_bin_id ⩾ MIN_BIN_ID && next_active_bin_id ⩽
    MAX_BIN_ID,
        LBError::PairInsufficientLiquidity
    );

    self.active_id = next_active_bin_id;

    Ok(())
}
```

However, this checks the new id vs the `MIN_BIN_ID/MAX_BIN_ID`, which aren't the actual limitations of the pool. Thus, this would not revert if the swap went outside of the actual limitations of the pool. However, the functions for adding liquidity check the correct min/max, and thus, no liquidity can be added outside of the limitations. As a result, a swap can never go outside of them.

Recommendations:

We recommend enforcing the actual min/max of the pool.

```
/// Plus / Minus 1 to the active bin based on the swap direction
pub fn advance_active_bin(&mut self, swap_for_y: bool) → Result<()> {
    let next_active_bin_id = if swap_for_y {
        self.active_id.safe_sub(1)?
    } else {
        self.active_id.safe_add(1)?
    };

    require!(
        next_active_bin_id ⩾ self.parameters.min_bin_id
    && next_active_bin_id ⩽ self.parameters.max_bin_id,
        LBError::PairInsufficientLiquidity
    );

    self.active_id = next_active_bin_id;

    Ok(())
}
```

Meteora: Resolved with PR-422

Zenith: Verified

[L-11] Liqudity can't be added for position with only one bin array

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- src/instructions/deposit/add_liquidity.rs#L171-L180

Description:

In the `add_liquidity` instruction, the caller has to provide two bin arrays.

```
#[account(
    mut,
    has_one = lb_pair
)]
pub bin_array_lower: AccountLoader<'info, BinArray>,
#[account(
    mut,
    has_one = lb_pair
)]
pub bin_array_upper: AccountLoader<'info, BinArray>,
```

These are then used to create a `BinArrayManager`. However, a position might exactly fit within one bin array as `0 < positionSize < 1400`. Thus, only one bin array would be needed here. If a user provides the same bin array twice, as he is forced to provide two even if just one is required, this will lead to a revert as Rust will try borrowing the same account twice.

POC:

```
import * as anchor from "@coral-xyz/anchor";
import { Wallet, web3 } from "@coral-xyz/anchor";
import {
  ASSOCIATED_TOKEN_PROGRAM_ID,
  TOKEN_PROGRAM_ID,
  createMint,
```

```
  getAssociatedTokenAddressSync,
  getOrCreateAssociatedTokenAccount,
  mintTo,
} from "@solana/spl-token";
import { BN } from "bn.js";
import { expect } from "chai";
import {
  ADMIN,
  Bins,
  CompositionFeeEventCpi,
  Enable,
  MAX_BIN_PER_ARRAY,
  DEFAULT_BIN_PER_POSITION,
  SET_COMPUTE_UNIT_LIMIT_IX,
  addLiquidityRadius,
  assertPositionAndBinLiquidityDelta,
  binIdToBinArrayIndex,
  chunkedGetMultipleAccountInfos,
  createAndFundWallet,
  createBinArrays,
  createLbClmmProgram,
  createLbPair,
  createPosition,
  createPresetParameter,
  decUiAmount,
  deriveBinArray,
  deriveBinArrayBitmapExtension,
  deriveReserve,
  fetchPositionByOwners as fetchPositionByOwner,
  fpToDec,
  getBinArrayLowerUpperBinId,
  getBinArraysForModifyLiquidity,
  getOrCreateBinArraysForModifyLiquidity,
  getPriceFromBinId,
  getSpotPatternDistribution,
  invokeAndAssertError,
  parseEventCpi,
  sleep,
  fetchAndDecodeDynamicPosition,
} from "../utils";

const provider = anchor.AnchorProvider.env();

describe("Add Liquidity", () => {
  const keypair: web3.Keypair = web3.Keypair.generate();
  const wallet: Wallet = new Wallet(keypair);
```

```
const keypair2: web3.Keypair = web3.Keypair.generate();
const wallet2: Wallet = new Wallet(keypair2);

const activeId = new BN(5660);
const binStep = new BN(10);

const btcDecimal = 8;
const usdcDecimal = 6;

const protocolShare = 1000;

// 5 = Create 5 lower bin arrays, and 5 upper bin arrays surrounding the
  active bin arrays. Total bins = 600 * 11
const binArrayDelta = 5;
const binArrayIndex = binIdToBinArrayIndex(activeId);
const binArrayIndexes = [];

const ONE_BTC = new BN(1 * 10 ** btcDecimal);
const ONE_USDC = new BN(1 * 10 ** usdcDecimal);

let BTC: web3.PublicKey;
let USDC: web3.PublicKey;
let lbPair: web3.PublicKey;
let reserveX: web3.PublicKey;
let reserveY: web3.PublicKey;

const balancedDelta = DEFAULT_BIN_PER_POSITION.div(new BN(2));
const lowerBinId = activeId.sub(balancedDelta);
const upperBinId = lowerBinId.add(DEFAULT_BIN_PER_POSITION.sub(new
  BN(1)));

before(async () => {
  await createAndFundWallet(provider.connection, keypair);
  await createAndFundWallet(provider.connection, keypair2);
  // Lower bin arrays
  for (let i = binArrayDelta; i > 0; i--) {
    const idx = binArrayIndex.sub(new BN(i));
    binArrayIndexes.push(idx);
  }

  binArrayIndexes.push(binArrayIndex);

  // Upper bin arrays
  for (let i = 1; i <= binArrayDelta; i++) {
    const idx = binArrayIndex.add(new BN(i));
    binArrayIndexes.push(idx);
  }
```

```
});

beforeEach(async () => {
  BTC = await createMint(
    provider.connection,
    keypair,
    wallet.publicKey,
    null,
    btcDecimal,
    web3.Keypair.generate(),
    null,
    TOKEN_PROGRAM_ID
  );

  USDC = await createMint(
    provider.connection,
    keypair,
    wallet.publicKey,
    null,
    usdcDecimal,
    web3.Keypair.generate(),
    null,
    TOKEN_PROGRAM_ID
  );

  const program = createLbClmmProgram(wallet);

  await createPresetParameter(
    binStep,
    createLbClmmProgram(new Wallet(ADMIN))
  );

  lbPair = await createLbPair(BTC, USDC, activeId, binStep, program);
  [reserveX] = deriveReserve(BTC, lbPair, program.programId);
  [reserveY] = deriveReserve(USDC, lbPair, program.programId);

  const lbPairState = await program.account.lbPair.fetch(lbPair);

  await createLbClmmProgram(new Wallet(ADMIN))
    .methods.updateBaseFeeParameters({
      protocolShare,
      baseFactor: lbPairState.parameters.baseFactor,
      baseFeePowerFactor: 0,
    })
    .accountsPartial({
      admin: ADMIN.publicKey,
      lbPair,
```

```
    })
    .rpc();

  await createBinArrays(lbPair, binArrayIndexes, program);

  const userTokenX = await getOrCreateAssociatedTokenAccount(
    provider.connection,
    keypair,
    BTC,
    keypair.publicKey,
    false,
    "confirmed",
    {
      commitment: "confirmed",
    },
    TOKEN_PROGRAM_ID,
    ASSOCIATED_TOKEN_PROGRAM_ID
  );

  const userTokenY = await getOrCreateAssociatedTokenAccount(
    provider.connection,
    keypair,
    USDC,
    keypair.publicKey,
    false,
    "confirmed",
    {
      commitment: "confirmed",
    },
    TOKEN_PROGRAM_ID,
    ASSOCIATED_TOKEN_PROGRAM_ID
  );

  await mintTo(
    provider.connection,
    keypair,
    BTC,
    userTokenX.address,
    keypair.publicKey,
    100 * 10 ** btcDecimal,
    [],
    {
      commitment: "confirmed",
    },
    TOKEN_PROGRAM_ID
  );
```

```
    await mintTo(
      provider.connection,
      keypair,
      USDC,
      userTokenY.address,
      keypair.publicKey,
      100_000 * 10 ** usdcDecimal,
      [],
      {
        commitment: "confirmed",
      },
      TOKEN_PROGRAM_ID
    );
  });

it("deposit one bin array", async () => {
    const program = createLbClmmProgram(wallet);

    const amountX = ONE_BTC; // 1 btc
    const amountY = ONE_USDC.mul(new BN(10000)); // 10k

    const position = await createPosition(
      lbPair,
      0,
      program
    );

    let [binArrayBitmapExtension] = deriveBinArrayBitmapExtension(
        lbPair,
        program.programId
      );
      let binArrayBitmapExtensionState =
        await program.account.binArrayBitmapExtension.fetchNullable(
          binArrayBitmapExtension
        );
      if (!binArrayBitmapExtensionState) {
        binArrayBitmapExtension = null;
      }

      const lbPairState = await program.account.lbPair.fetch(lbPair);
      const positionStateInner = await
    fetchAndDecodeDynamicPosition(program, position);
      const { binArrayLower, binArrayUpper }
    = await getOrCreateBinArraysForModifyLiquidity(
        lbPair,
        positionStateInner.globalData.lowerBinId,
        program
```

```
    );
    const mintAccountsInfo = await chunkedGetMultipleAccountInfos(
      program.provider.connection,
      [lbPairState.tokenXMint, lbPairState.tokenYMint]
    );
    const userTokenX = getAssociatedTokenAddressSync(
      lbPairState.tokenXMint,
      program.provider.publicKey,
      false,
      mintAccountsInfo[0].owner,
      ASSOCIATED_TOKEN_PROGRAM_ID
    );

    const userTokenY = getAssociatedTokenAddressSync(
      lbPairState.tokenYMint,
      program.provider.publicKey,
      false,
      mintAccountsInfo[1].owner,
      ASSOCIATED_TOKEN_PROGRAM_ID
    );

    const binLiquidityDist = getSpotPatternDistribution(
      1,
      lbPairState.activeId
    );

    try {
      const txHash = await program.methods
        .addLiquidity({
          amountX,
          amountY,
          binLiquidityDist,
        })
        .accountsPartial({
          lbPair,
          binArrayBitmapExtension,
          position,
          reserveX: lbPairState.reserveX,
          reserveY: lbPairState.reserveY,
          tokenXMint: lbPairState.tokenXMint,
          tokenYMint: lbPairState.tokenYMint,
          tokenXProgram: mintAccountsInfo[0].owner,
          tokenYProgram: mintAccountsInfo[1].owner,
          sender: program.provider.publicKey,
          userTokenX,
          userTokenY,
          binArrayLower,
```

```
            binArrayUpper: binArrayLower, // Pass the same account as both
    lower and upper
          })
          .preInstructions([SET_COMPUTE_UNIT_LIMIT_IX])
          .rpc({
            commitment: "confirmed",
          });
      } catch (error) {
        // Expect any error to occur (the test should fail with an error)
        console.log("Expected error occurred:", error.toString());
        expect(error).to.exist;
        return; // Test passes if any error occurs
      }


    let positionState = await fetchAndDecodeDynamicPosition(
      program,
      position
    );

    const activeBinIdxInPosition = activeId
      .sub(new BN(positionState.globalData.lowerBinId))
      .toNumber();

    // Deposit to active bin + surrounding bin with 1 radius
    await addLiquidityRadius(
      lbPair,
      activeId,
      position,
      amountX,
      amountY,
      1,
      program
    );
  });
});
```

Recommendations:

We recommend making the `bin_array_upper` optional and only adding it to the `BinArrayManager` if needed.

Meteora: Acknowledged

[L-12] Bin arrays can be created for bins outside of the supported min/max of pool

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- src/instructions/permissionless_operations/initialize_bin_array.rs#L31

Description:

The `initialize_bin_array` function allows anyone to initialize new bin arrays. However, it currently only restricts these to be between `MIN_BIN_ID` and `MAX_BIN_ID`. This is enforced by the `check_valid_index()` which is called during the initialization of the bin array.

```
/// Check that the index within MAX and MIN bin id
pub fn check_valid_index(index: i32) -> Result<()> {
    let (lower_bin_id, upper_bin_id)
    = BinArray::get_bin_array_lower_upper_bin_id(index)?;

    require!(
        lower_bin_id ≥ MIN_BIN_ID && upper_bin_id ≤ MAX_BIN_ID,
        LBError::InvalidStartBinIndex
    );

    Ok(())
}
```

The actual min and max of the bins could be significantly lower depending on the `bin_step` chosen for the pool. This is calculated in the `get_min_max_bin_id_from_bin_step`.

```
// get min max bin id by using approximation method
// (1 + b) ^ (k *x) ~= (1 + kb) ^ x if b is small
// check the test test_get_min_max_bin_id_from_bin_step
pub fn get_min_max_bin_id_from_bin_step(bin_step: u16) -> Result<(i32, i32)>
    {
    let max_bin_id_for_unit_bin_step = 436_704; // hardcoded number
    let max_bin_id =
    max_bin_id_for_unit_bin_step.safe_div(bin_step.into())?;
```

```
    Ok((-max_bin_id, max_bin_id))
}
```

For example, for a bin step of 3, the actual min/max would be `[-145_568, 145_568]`, which is significantly lower than the `[-443636, 443636]` that is checked in `check_valid_index()`. As a result, this will allow users to create bin arrays outside of the actual min/max of the pool.

A second safeguard is the `get_price_from_id()` function, which will be called during the loop in `fill_prices`. This should overflow if a value significantly out of the actual price range is added. However, for ones close to the intended min/max, it will not overflow, thus not being an adequate safeguard. The following POC that can be added to `price_math.rs` showcases that `get_price_from_id()` overflowing is not enough of a safeguard, as the used bin here is 100 bins outside of the max and will still not revert.

```
#[test]
fn test_get_price_from_positive_id2() {
    let active_id = 145668;
    let bin_step = 3;

    let price = get_price_from_id(active_id, bin_step).unwrap();
    let decimal_price = to_decimal(price).unwrap();
    println!("Decimal price (10^12 precision) {}", decimal_price);

    let base = 1.0f64 + bin_step as f64 / BASIS_POINT_MAX as f64;
    let price = base.powi(active_id);
    println!("Float price {}", price);
}
```

Recommendations:

We recommend adapting `check_valid_index()` in the following way:

```
/// Check that the index within MAX and MIN bin id
pub fn check_valid_index(index: i32, lb_pair: Pubkey) -> Result<()> {
    let (lower_bin_id, upper_bin_id)
    = BinArray::get_bin_array_lower_upper_bin_id(index)?;

    require!(
        lower_bin_id ⩾ lb_pair.min_id && upper_bin_id ⩽ lb_pair.max_id,
        LBError::InvalidStartBinIndex
    );

    Ok(())
```

```
}
```

Meteora: Resolved with PR-426

Zenith: Verified

[L-13] `realloc_expand_position_account` does not consider actual balance of address leading to unnecessary transfer

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- src/instructions/rebalance/process_resize_position.rs#L69-L100

Description:

The `realloc_expand_position_account` function is used to calculate the difference between the new and old size of the accounts and transferring the required rent.

```
pub fn realloc_expand_position_account<'info>(
position: AccountInfo<'info>,
system_program: AccountInfo<'info>,
rent_payer: AccountInfo<'info>,
old_space: usize,
new_space: usize,
) → Result<()> {
let rent = Rent::get()?;

let old_rental = rent.minimum_balance(old_space);
let new_rental = rent.minimum_balance(new_space);
let extra_lamports = new_rental.safe_sub(old_rental)?;

if new_space.saturating_sub(old_space) > MAX_PERMITTED_DATA_INCREASE {
    return Err(LBError::ReallocExceedMaxLengthPerInstruction.into());
}

transfer(
    CpiContext::new(
        system_program,
        Transfer {
            from: rent_payer,
            to: position.clone(),
        },
    ),
    extra_lamports,
```

```
)?;

position.realloc(new_space, true)?;

Ok(())
}
```

However, the function forgets to verify the balance held by the account. The account could already have enough funds for the new size, requiring no transfer at all.

Recommendations:

We recommend checking if the current balance of the account is lower than the newly needed minimum balance and only transferring the difference if required.

Meteora: Resolved with PR-428

Zenith: Verified

4.3 Informational

A total of 18 informational findings were identified.

[I-1] Unused `authorize_migrate_position` function in position authorization logic

`SEVERITY:` Informational	`IMPACT:` Informational
`STATUS:` Resolved	`LIKELIHOOD:` Low

Target

- programs/lb_clmm/src/instructions/position_authorize.rs#L28-L34

Description:

The function `authorize_migrate_position` is currently unused in the codebase. Its purpose appears to be to check whether a given sender is authorized to migrate a `PositionV2` account (i.e., if the sender is either the owner or the operator of the position). However, there are no references to this function in the current implementation, and it is not invoked during any migration or authorization flows.

It is unclear whether this function is reserved for future use, such as supporting migration of V2 positions to a potential V3 version, or if it is a leftover from a previous refactor.

Recommendations:

It is recommended to document whether this function is intended for future use (e.g., V2 → V3 migration) or remove it if it is obsolete.

Meteora: Resolved with PR-395.

Zenith: Verified. Resolved by removing the function.

[I-2] Only one customizable permissionless LB pair allowed per token X/Y due to seed restriction

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- src/instructions/initialize_pool/initialize_customizable_permissionless_lb_pair.rs#L149-L160
- src/instructions/v2/initialize_customizable_permissionless_lb_pair2.rs#L23-L34

Description:

In the current implementation of `initialize_customizable_permissionless_lb_pair` and `initialize_customizable_permissionless_lb_pair2`, the PDA seeds for the `lb_pair` account are constructed as follows:

```
seeds = [
    ILM_BASE_KEY.as_ref(),
    min(token_mint_x.key().as_ref(), token_mint_y.key().as_ref()),
    max(token_mint_x.key().as_ref(), token_mint_y.key().as_ref()),
]
```

This means that for any given pair of token mints X and Y, there can only be one customizable permissionless LB pair, regardless of any other parameters or customizations. This is in contrast to permissioned and permissionless LB pairs, which may allow for multiple pairs with the same tokens but different configurations.

This restriction limits the protocol's flexibility, as it prevents the creation of multiple customizable pools for the same token pair, even if they are intended to have different parameters or use cases.

Recommendations:

It is recommended to introduce an extra seed value to the PDA derivation for `lb_pair`.

Meteora: Acknowledged. It's intended design. We wish to have only a single token pair for each token launch.

[I-3] Token badges (Token-2022 whitelisting) cannot be revoked

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/lb_clmm/src/instructions/v2/initialize_token_badge.rs

Description:

In the current implementation of `initialize_token_badge`, once a token badge is created for a Token-2022 mint (i.e., the mint is whitelisted for use in the protocol), there is no mechanism to revoke or remove the token badge. This means that whitelisting is permanent and cannot be undone by the admin or any other authority.

If a token becomes compromised, or if the protocol's risk assessment changes, there is no way to remove its whitelisted status. While revoking a token badge would not affect existing pools that have already been created with the whitelisted token, it would be valuable to prevent the creation of new pools with that token in the future.

Recommendations:

It is recommended to add an instruction (e.g., `revoke_token_badge`) that allows an admin to close or invalidate a token badge, thereby removing the token from the whitelist for future pool creation.

Meteora: Resolved with PR-396.

Zenith: Verified.

[I-4] Swap and reward distribution failures due to bin array account ordering

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- programs/lb_clmm/src/instructions/v2/swap2.rs#L361-L501
- programs/lb_clmm/src/instructions/v2/swap2.rs#L601-L642

Description:

In the function `internal_handle_swap2`, the swap logic assumes that the order of bin array accounts in `remaining_accounts` matches the order in which bins are traversed during the swap (i.e., the swap direction). The code uses `BinArrayAccount::try_accounts` to sequentially consume accounts from `remaining_accounts`. If the swap spans multiple bin arrays and the accounts are not ordered according to the swap direction, the loop cannot access bin arrays with a lower index in the account list after advancing, causing the swap to fail.

Although the test suite (e.g., via `getBinArraysForSwap`) appears to always provide bin arrays in the correct order, this requirement is not enforced or documented, and may be a pitfall for integrators who do not strictly order their accounts.

A similar issue exists in `distribute_rewards_to_swapped_bin`. The function uses an iterator (`loaded_bin_arrays_iter`) over the loaded bin arrays, and only advances the iterator when a bin is not found in the current bin array. If a bin is not found and the iterator reaches the end, it does not reset, and any subsequent bins that reside in earlier bin arrays will never be found. This can result in an endless loop if the bins are not distributed in strictly increasing order of bin array accounts.

Recommendations:

It is recommended to consider the following mitigation measures:

- For `internal_handle_swap2`:

  When advancing the active bin (i.e., after calling `lb_pair.advance_active_bin(swap_for_y)?;`), reset the `remaining_accounts` slice and the associated index to the first bin array account. This ensures that all bin arrays can be

checked for the new active bin, regardless of their order in the account list.

- For `distribute_rewards_to_swapped_bin`:

  When the `loaded_bin_arrays_iter` reaches the end (returns `None`), reset the iterator to the beginning of the loaded bin arrays. This allows the function to continue searching for bins in all bin arrays, regardless of their order, and prevents an endless loop if bins are not distributed in strictly increasing order of bin array accounts.
- Documentation:

  Clearly document the requirement for bin array account ordering in both the code and any developer/integrator documentation.

Meteora: Acknowledged. Will be documented that bin arrays need to be in strict order.

[I-5] Incorrect error type used in `process_rebalance_transfer` function

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/lb_clmm/src/instructions/rebalance/rebalance_wrapper.rs#L252-L255
- programs/lb_clmm/src/instructions/rebalance/rebalance_wrapper.rs#L271-L274
- programs/lb_clmm/src/errors.rs#L15-L19

Description:

In the `process_rebalance_transfer` function, the code incorrectly uses `LBError::ExceededBinSlippageTolerance` in two locations where it should be using `LBError::ExceededAmountSlippageTolerance`, because both of these checks are related to amount slippage tolerance (ensuring the actual amounts fall within acceptable ranges).

Recommendations:

It is recommended to replace the incorrect error types with `LBError::ExceededAmountSlippageTolerance` in both locations to accurately reflect the nature of the validation being performed.

Meteora: Resolved with PR-388.

Zenith: Verified.

[I-6] Artificial limitation on shrink size in `decrease_position_length` via `MAX_RESIZE_LENGTH_USIZE` check

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/lb_clmm/src/instructions/position/decrease_position_length.rs#L30-L33

Description:

In the `decrease_position_length` instruction handler, the following check is enforced:

```
require!(
    length_to_remove > 0 && (length_to_remove as usize)
    ⩽ MAX_RESIZE_LENGTH_USIZE,
    LBError::InvalidResizeLength
);
```

This restricts the maximum number of bins that can be removed in a single call to `MAX_RESIZE_LENGTH_USIZE`. However, this is an artificial limitation: the underlying Solana `realloc` function only enforces the `MAX_PERMITTED_DATA_INCREASE` (10,240 bytes) limit when increasing account size, not when decreasing it. There is no technical reason to restrict the shrink size, and this limitation is inconsistent with the behavior of the `realloc_shrink_position_account` function used during rebalancing, which does not enforce such a restriction.

Recommendations:

It is recommended to allow users to decrease the position length by any amount, as long as it is valid and does not violate other protocol invariants (e.g., at least one bin remains).

Meteora: Resolved with PR-410.

Zenith: Verified.

[I-7] Already reset bins are reset again in `shift_left`

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- src/state/dynamic_position.rs#L146-L150)

Description:

The `shift_left` function is used when a position is shrunk on the left side. It first moves the bins from the right to the left.

```
for i in 0..length {
    // set data from i+count to i
    let from_idx = BinDataIndex::new(i.safe_add(count)?)?;
    let to_idx = BinDataIndex::new(i)?;
    self.set_bin_data(&from_idx, &to_idx);
}
```

After this, it will reset the bins outside the position (`length ... length+count`, which will be unset anyway.

```
for i in length..(length + count) {
    // reset data in i
    let idx = BinDataIndex::new(i)?;
    self.reset_bin_data(&idx);
}
```

Recommendations:

We recommend removing the reset as it is not needed.

Meteora: Acknowledged.

[I-8] Currently unused `set_liquidity_share` function allows bypassing `MINIMUM_LIQUIDITY` threshold

`SEVERITY:` Informational	`IMPACT:` Informational
`STATUS:` Resolved	`LIKELIHOOD:` Low

Target

- programs/lb_clmm/src/state/dynamic_position.rs#L539-L548

Description:

The `set_liquidity_share` function in `dynamic_position.rs` directly sets the liquidity share for a given bin without enforcing the `MINIMUM_LIQUIDITY` threshold. Unlike the `withdraw` function, which contains explicit checks to ensure that the liquidity share is either zero or greater than or equal to `MINIMUM_LIQUIDITY`, this function.

Currently, `set_liquidity_share` is not used anywhere in the codebase. However, if it is used in the future, it would allow setting a nonzero liquidity share below the required minimum, violating protocol invariants and potentially leading to unexpected states.

Recommendations:

It is recommended to remove the `set_liquidity_share` function from the codebase to prevent misuse in the future.

Meteora: Resolved with PR-405.

Zenith: Verified.

[I-9] Unused `is_bin_array_has_liquidity` function

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/lb_clmm/src/state/lb_pair.rs#L677-L692

Description:

The function `is_bin_array_has_liquidity` in `LbPair` is currently unused in the codebase. Its name and implementation suggest it is intended to provide a unified and reliable way to check whether a given bin array (by index) has any liquidity.

Recommendations:

It is recommended to review and clarify the intended use case of the `is_bin_array_has_liquidity` function.

Meteora: Resolved with PR-407.

Zenith: Verified.

[I-10] Parameter confusion in the `is_liquidity_locked` function might cause unintended liquidity locks

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- lb_clmm/src/state/dynamic_position.rs#L550-L552
- lb_clmm/src/instructions/rebalance/process_remove_liquidity.rs#L36-L41
- lb_clmm/src/instructions/v2/remove_liquidity2.rs#L213-L218
- lb_clmm/src/instructions/withdraw/remove_all_liquidity.rs#L83-L88
- lb_clmm/src/instructions/withdraw/remove_liquidity.rs#L139-L144
- lb_clmm/src/pair_action_access/customizable_permissionless_pair.rs#L20-L25

Description:

The `is_liquidity_locked` function of the `dynamic_position` is called in four instances with the return value of `get_current_point`, which can be either a slot number or a timestamp depending on the `activation_type` of the pool. However, the function signature of `is_liquidity_locked` expects a parameter named `current_slot`, suggesting that the checked `lock_release_point` field is also denominated in slots.

This mismatch can lead to situations where liquidity remains locked longer than intended (or is unlocked prematurely) if `lock_release_point` and `activation_type` are initialized using different units.

Recommendations:

It is recommended to rename the parameter in `is_liquidity_locked` from `current_slot` to `current_point` to accurately reflect its dual usage. Furthermore, ensure that `lock_release_point` is always initialized and compared using the same unit (slot or timestamp) as determined by the pool's `activation_type`.

Meteora: Resolved with PR-406.

Zenith: Verified.

[I-11] Unused function `get_continuous_bins`

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: High

Target

- src/manager/bin_array_manager.rs#L101-L106

Description:

The `get_continuous_bins()` function in the `bin_array_manager` is never used and can thus be removed to reduce complexity.

Recommendations:

We recommend removing the unused function.

Meteora: Resolved with PR-403.

Zenith: Verified.

[I-12] Incorrect documentation in `resize_position()`

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- src/instructions/rebalance/process_resize_position.rs#L130

Description:

The description of the `resize` function describes the following case.

```
// 1. Expanded both side
// [new lower bin id ........... new upper bin id]
//    [old lower bin id .... new upper bin id]
```

However the "new upper bin id" here should actually be the "old upper bin id".

Recommendations:

We recommend adapting the comment to the following:

```
// 1. Expanded both side
// [new lower bin id ........... new upper bin id]
//    [old lower bin id .... old upper bin id]
```

Meteora: Resolved with PR-402

Zenith: Verified

[I-13] Commented out fee update validation logic

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/lb_clmm/src/state/parameters.rs#L36-L52

Description:

The `StaticParameters::update` function contains commented out validation logic for fee parameter updates that appears to be accidentally left in the codebase. This commented code represents safety checks that were intended to prevent excessive fee changes during pool parameter updates.

The commented code shows two validations that are currently disabled:

- Ensures fee changes don't exceed 100% of the current fee rate.
  - Ensures fee changes don't exceed 100 basis points (1%).

The presence of commented validation logic suggests this was either:

- Accidentally left commented during development.
  - Intentionally disabled but not properly documented.

Recommendations:

It is recommended to review whether these validations are still needed or if they can be removed entirely to improve code clarity.

Meteora: Resolved with PR-416

Zenith: Verified. Resolved by removing the commented out logic.

[I-14] Unresolved TODOs in codebase

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/lb_clmm/src/constants.rs#L4
- programs/lb_clmm/src/constants.rs#L175
- programs/lb_clmm/src/instructions/claim_reward.rs#L88
- programs/lb_clmm/src/instructions/swap.rs#L315
- programs/lb_clmm/src/instructions/position/migrate_position.rs#L23
- programs/lb_clmm/src/instructions/rebalance/process_resize_position.rs#L112
- programs/lb_clmm/src/instructions/v2/claim_reward2.rs#L91
- programs/lb_clmm/src/instructions/v2/swap2.rs#L219
- programs/lb_clmm/src/instructions/v2/swap2.rs#L373
- programs/lb_clmm/src/math/weight_to_amounts.rs#L417
- programs/lb_clmm/src/math/weight_to_amounts.rs#L440
- programs/lb_clmm/src/state/bin_array_bitmap_extension.rs#L146
- programs/lb_clmm/src/state/bin.rs#L221
- programs/lb_clmm/src/state/lb_pair.rs#L115
- programs/lb_clmm/src/state/lb_pair.rs#L197
- programs/lb_clmm/src/state/lb_pair.rs#L199
- programs/lb_clmm/src/state/lb_pair.rs#L201
- programs/lb_clmm/src/state/lb_pair.rs#L1412
- programs/lb_clmm/src/state/lb_pair.rs#L1413
- programs/lb_clmm/src/tests/reward_integration_tests.rs#L74

Description:

The DLMM codebase contains 20 unresolved TODO comments across multiple Rust files:

`programs/lb_clmm/src/constants.rs`

> Line 4: Macro to compute the constants which changes based on the bit system used ? Line 175: enable protocol share back later

programs/lb_clmm/src/instructions/claim_reward.rs

Line 88: Should we pass in range of bin we are going to collect reward ? It could help us in heap / compute unit issue by chunking into multiple tx.

programs/lb_clmm/src/instructions/position/migrate_position.rs

Line 23: do we need to check whether it is pda?

programs/lb_clmm/src/instructions/rebalance/process_resize_position.rs

Line 112: verify this

programs/lb_clmm/src/instructions/swap.rs

Line 315: when out of liquidity, this function will use up all the CU

programs/lb_clmm/src/instructions/v2/claim_reward2.rs

Line 91: Should we pass in range of bin we are going to collect reward ? It could help us in heap / compute unit issue by chunking into multiple tx.

programs/lb_clmm/src/instructions/v2/swap2.rs

Line 219: do we need this check? Line 373: when out of liquidity, this function will use up all the CU

programs/lb_clmm/src/math/weight_to_amounts.rs

Line 417: fix this assertion Line 440: fix this assertion

programs/lb_clmm/src/state/bin.rs

Line 221: User possible to bypass fee by swapping small amount ? User do a "normal" swap by just bundling all small swap that bypass fee ?

programs/lb_clmm/src/state/bin_array_bitmap_extension.rs

Line 146: do we need validate bin_array_index again?

`programs/lb_clmm/src/state/lb_pair.rs`

Line 115: Bug in anchor IDL parser when using InitSpace macro. Temp hard-code it. https://github.com/coral-xyz/anchor/issues/2556 Line 197: check whether we need to store it in pool Line 199: check whether we need to store it in pool Line 201: check whether we need to store it in pool Line 1412: add test overflow for BIN_ARRAY_BITMAP_SIZE Line 1413: add test overflow for -BIN_ARRAY_BITMAP_SIZE-1

`programs/lb_clmm/src/tests/reward_integration_tests.rs`

Line 74: add function clawback in case passed_duration, but no one actually deposit liquidity

Recommendations:

It is recommended to either resolve, remove, or convert the TODOs to normal comments.

Meteora: Resolved with PR-415.

Zenith: Verified.

[I-15] Unused functions in `pda.rs`

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- src/utils/pda.rs#L5
- src/utils/pda.rs#L23
- src/utils/pda.rs#L38
- src/utils/pda.rs#L55
- src/utils/pda.rs#L73
- src/utils/pda.rs#L84
- src/utils/pda.rs#L88
- src/utils/pda.rs#L92
- src/utils/pda.rs#L103
- src/utils/pda.rs#L107

Description:

The `pda.rs` file includes multiple unused functions.

Recommendations:

We recommend removing the unused functions to reduce complexity.

Meteora: Resolved with PR-413

Zenith: Verified

[I-16] Unused import in `initialize_permission_lb_pair`

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- src/instructions/initialize_pool/initialize_permission_lb_pair.rs#L4

Description:

The `initialize_permission_lb_pair` includes an import for `QUOTE_MINTS` which is never used.

```
use crate::constants::QUOTE_MINTS;
```

Recommendations:

We recommend removing the unused import.

Meteora: Resolved with PR-425

Zenith: Verified

[I-17] Multiple permissionless instructions are not tested

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- src/instructions/permissionless_operations/go_to_a_bin.rs#L11
- src/instructions/permissionless_operations/increase_oracle_length.rs#L12
- src/instructions/permissionless_operations/initialize_bin_array_bitmap_extension.rs#L7
- src/instructions/permissionless_operations/initialize_bin_array.rs

Description:

Many of the permissionless operations lack test cases that cover their functionality. This can lead to undetected bugs and is also not in line with the thorough test suite that is implemented for the rest of the repo.

Recommendations:

We recommend adding testcases to the affected files.

Meteora: Resolved with PR-424

Zenith: Verified

[I-18] `max_bin_id_for_unit_bin_step` should be defined as constant

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- src/state/preset_parameters2.rs#L191

Description:

The `max_bin_id_for_unit_bin_step` is hardcoded which is not recommended.

```
// get min max bin id by using approximation method
// (1 + b) ^ (k *x) ~= (1 + kb) ^ x if b is small
// check the test test_get_min_max_bin_id_from_bin_step
pub fn get_min_max_bin_id_from_bin_step(bin_step: u16) → Result<(i32, i32)>
    {
    let max_bin_id_for_unit_bin_step = 436_704; // hardcoded number
    let max_bin_id =
    max_bin_id_for_unit_bin_step.safe_div(bin_step.into())?;
    Ok((-max_bin_id, max_bin_id))
}
```

Recommendations:

We recommend defining a new constant for the value.

Meteora: Resolved with PR-427

Zenith: Verified