Meteora DLMM

Smart Contract Security Assessment

January 2024

Prepared for:

Meteora

Prepared by:

Offside Labs

Ronny Xing

Siji Feng

Contents

1 About Offside Labs

Offside Labs is a leading security research team, composed of top talented hackers from both academia and industry.

We possess a wide range of expertise in modern software systems, including, but not limited to, *browsers*, *operating systems*, *IoT devices*, and *hypervisors*. We are also at the forefront of innovative areas like *cryptocurrencies* and *blockchain technologies*. Among our notable accomplishments are remote jailbreaks of devices such as the **iPhone** and **PlayStation 4**, and addressing critical vulnerabilities in the **Tron Network**.

Our team actively engages with and contributes to the security community. Having won and also co-organized *DEFCON CTF*, the most famous CTF competition in the Web2 era, we also triumphed in the **Paradigm CTF 2023** within the Web3 space. In addition, our efforts in responsibly disclosing numerous vulnerabilities to leading tech companies, such as *Apple*, *Google*, and *Microsoft*, have protected digital assets valued at over **$300 million**.

In the transition towards Web3, Offside Labs has achieved remarkable success. We have earned over **$9 million** in bug bounties, and **three** of our innovative techniques were recognized among the **top 10 blockchain hacking techniques of 2022** by the Web3 security community.

https://offside.io/

https://github.com/offsidelabs

https://twitter.com/offside_labs

2 Executive Summary

Introduction

Offside Labs completed a security audit of *Meteora*'s *Dynamic Liquidity Market Maker (DLMM)* smart contracts, starting on December 11, 2023, and concluding on December 22, 2023.

DLMM Project Overview

Meteora's *DLMM* protocol revolutionizes liquidity provision by offering real-time dynamic fees and precise liquidity concentration. This slippage-resistant system is structured around discrete price bins, allowing for efficient asset pair exchanges.

Audit Scope

The assessment scope contains mainly the smart contracts of the *LB-clmm* program for the *DLMM* project by the Dec 6, 2023.

The audit is based on the following specific branches and commit hashes of the codebase repositories:

- DLMM
  - Branch: master
  - Commit Hash: 69a7f2057539fbd754270379b02ed77f5121d9b5
  - Codebase Link

We listed the files we have audited below:

- DLMM
  - programs/lb_clmm/src/**/*.rs
  - Exclude
    - programs/lb_clmm/src/tests/*
    - inline tests

Findings

The security audit revealed:

- 1 critical issue
- 2 high issues
- 2 medium issues
- 2 low issues
- 4 informational issues

Further details, including the nature of these issues and recommendations for their remediation, are detailed in the subsequent sections of this report.

3 Summary of Findings

ID	Title	Severity	Status
01	Inflation Attack in the Bin	Critical	Fixed
02	Actual Composition Fee is Less Than Swap Fee	High	Fixed
03	Rewards Can be Stolen if the Active Bin is Empty	High	Fixed
04	Rewards are Skipped if the Swap is Crossing Multiple Bins	Medium	Fixed
05	Attacker Can Keep Fees Max at Low Cost	Medium	Fixed
06	total_amount_out Can be Equal to the min_amount_out	Low	Fixed
07	add_liquidity Fails to Verify that Distribution Sum Does Not Exceed 100%	Low	Fixed
08	Redundant Checks	Informational	Fixed
09	No Validations of bin_arrays When Claiming Fees or Rewards	Informational	Fixed
10	Mathematical Simplification	Informational	Acknowledged
11	Pair Can be Initialized With an Active Bin ID Outside the Preset Range	Informational	Fixed

4 Key Findings and Recommendations

4.1 Inflation Attack in the Bin

Severity: Critical	Status: Fixed
Target: Smart Contract	Category: Precision

Description

When adding liquidity to bins, the `get_liquidity_shr_cast` function is used to calculate how much liquidity, in terms of the current `price / bin`, is represented by the input of `tokenx + tokeny`. Furthermore, if the current `liquidity_supply` is 0, the liquidity is directly considered as the number of shares.

The main issue is that, although the liquidity calculation here utilizes Q64x64, it is converted to u64 type as shares, which truncates all decimals during the conversion.

```rust
    let liquidity_q64 = get_liquidity(x, y, price)?;
    let one = U256::from(1);
    let denominator = one.safe_shl(scale_offset as usize)?;
    let (liquidity, rem) = liquidity_q64.div_rem(denominator);
    let liquidity = match rounding {
        ...
    };
    Ok(liquidity.try_into().map_err(|_| LBError::TypeCastFailed)?)
```

programs/lb_clmm/src/math/bin_math.rs#L33-L48

This allows attackers to exploit an inflation attack to exponentially enlarge the share ratio of an empty bin.

Proof of Concept

Attack Flow:

1. Assume that the current active bin is empty. And the price is 1000.
2. An attacker adds liquidity to the active bin with 1 unit of `tokenY` in `position1`.
3. The attacker swaps 2 units of `tokenX` for 1 unit of `tokenY`. Due to rounding, after the swap, the current bin has 1 `tokenX` and 0 `tokenY`.
4. The attacker adds liquidity to `position2` with 1999 unit of `tokenY` and receives 1 share.
5. Remove all liquidity of the `position2`. After removing, the current bin has 1 `tokenX` and 500 `tokenY` for 1 share.
6. The attacker repeats steps 4 and 5, with each repetition making the share ratio 1.5 times larger than the previous iteration.
7. After repeating the process 40 times, the shares will inflate approximately 22 * 1e9 times.

Impact

Due to *DLMM* being based on a liquidity book, empty bins frequently occur as prices move, or attackers can prearrange malicious bins outside the current active price range.

The protocol and users are primarily compromised in the following two scenarios:

1. Any attempt to add liquidity below the liquidity of 1 share will fail. This will partially DOS the protocol and may result in the attacker becoming the sole liquidity provider.
2. If a user adds liquidity greater than the liquidity of 1 share, the remainder liquidity will be donated to shares of the current bin.

Recommendation

Use Q64x64 as shares, instead of u64. Considering the CU cost, it is also advisable to calc liquidity and shares as fixed-point numbers scaled in WAD.

Mitigation Review Log

Meteora Team: We follow your recommendation, however the fix requires user to migrate position state to store q-number (u128), BinArray account state that store total liquidity also need to migrate to q-number: PR-163

Offside Labs: **Fixed**. The current bin's liquidity has been changed from a u64 type to a Q64x64 type represented by u128, and this issue has been fixed.

During the review process, attention was paid to auditing the upgrade migration process for the bin array and position. With the fixed-point representation of liquidity shifted 64 bits to the left and extended to 128 bits, it is necessary to convert `liquidity_supply` and `liquidity_shares` in the bin array and position accounts into fixed-point numbers before any liquidity changes/swaps occur. To ensure that there is no misuse of different versions of liquidity, the review checked and confirmed the following items:

- The calculation and conversion process for liquidity supply/shares has been checked. Note: Redundant type conversion for `U256::from(y)` at programs/lb_clmm/src/math/bin_math.rs #L22
- In any instruction, the BinArray account has already had the `BinArray::migrate_to_v2` method called before use.
- No instruction uses the outdated `state::position::Position` account type; it is necessary to call the `migrate_position` instruction to upgrade to `PositionV2` . Note: CLI and some unit tests still use the Position type.
- BinArray accounts are all loaded using AccountLoader with demand zero copy deserialization, ensuring that updates to BinArray are synchronized within the current instruction context.

Note: In the calculation of swap fees and reward distribution, there are multiple instances of `safe_shr` operations on `liquidity_shares / liquidity_supply` , which as explained in the comments, "to make it simple we truncate decimals of `liquidity_supply` for the calculation". Code lines at commit d3bc99713288d8a682d7141b834fbf944b338505:

- Global rate calculation with liquidity_supply:
  - programs/lb_clmm/src/state/bin.rs#L125
  - programs/lb_clmm/src/state/bin.rs#L468
  - programs/lb_clmm/src/instructions/swap.rs#L438
- Fee/Reward amount calculation with liquidity_shares:
  - programs/lb_clmm/src/state/position.rs#L239
  - programs/lb_clmm/src/state/position.rs#L254
  - programs/lb_clmm/src/state/position.rs#L277

Since users first discard the fractional part of `liquidity_shares` when calculating the token amount, they receive less fee/reward than the accumulated amount. This ultimately leaves a small portion of fee/reward stuck in the contract that cannot be withdrawn, although this loss is likely negligible.

Since both the ratio calculation and the amount calculation discard the fractional part, this issue will not lead to more serious security implications.

4.2 Actual Composition Fee is Less Than Swap Fee

Severity: High	Status: Fixed
Target: Smart Contract	Category: Logic

Description

We can find the formula of the Composition Fee in the “4.4 Active Bin Composition Fee” of Joe v2 Liquidity Book Whitepaper . In fact, the code simplifies the formula to `totalFee * (1 + totalFee)` .

However, the composition fee is also distributed as liquidity to the shares deposited in the `add_liquidity` ix. The fee rate reduction becomes more pronounced during large amount liquidity operations.

The code details are in the `charge_fee_and_deposit` function:

```
    let in_liquidity = get_liquidity_shr_cast(
        amount_x_into_bin_after_fee,
        amount_y_into_bin_after_fee,
        bin_price,
        SCALE_OFFSET.into(),
        Rounding::Down,
    )?;

    // Calculate liquidity share to mint after charge swap fee
    let liquidity_share = get_liquidity_share(in_liquidity, bin_liquidity,
        bin.liquidity_supply)?;
```

programs/lb_clmm/src/instructions/add_liquidity.rs#L315-L331

If the current depositing bin id is `activeId` and the composition fee is not zero, it will sub the fees from the input amount, and use the new input amount to recalculate the actual shares minted for user.

But, after that shares calculation, the composition fees, without protocol fee, are added to the bin liquidity.

```
    // Protocol fee is not accumulated in the bin liquidity.
    bin.deposit(
        amount_x - protocol_fee_x,
        amount_y - protocol_fee_y,
        liquidity_share,
    )?;
```

programs/lb_clmm/src/instructions/add_liquidity.rs#L327-L331

Since the fee accounting is done after the calculation of shares, the shares being minted will also share these composition fees.

Impact

The actual composition fee is much less than the swap fee rate. The fee rate reduction becomes more pronounced during large amount liquidity operations. If an attacker uses a significant amount of tokens or feeless flash loans, the composition fees of this implicit swap can be reduced to nearly zero, as long as the proportion of minted shares to the total supply is sufficiently large.

Proof of Concept

We calculated the actual composition fees and the theoretical fees according to the above method, respectively, and compared them:

```
const real_fee_x = input_token_x.sub(new
    BN(Math.floor(userTokenYWithdraw.sub(ONE_USDC).toNumber() /
    binPrice)));
const real_fee_rate = real_fee_x.mul(FEE_PRECISION).div(input_token_x);
// lp.get_total_fee() is too hard to simulate, we just cmp the fee
    amount from event here
expect(real_fee_x.lt(
    compositionFeeEvent.data.tokenXFeeAmount)
).to.be.true;
```

Recommendation

Calc the minted shares after adding the deposited liquidity to the bin.

Mitigation Review Log

Meteora Team: We follow the recommendation: PR-174

Offside Labs: **Fixed**. After thorough mathematical verification, we have confirmed that the calculation of the composition fee (includes protocol fee) here is completely equivalent to the mathematical formula given in the white paper, with the exception of the unavoidable precision rounding.

4.3 Rewards Can be Stolen if the Active Bin is Empty

Severity: High	Status: Fixed
Target: Smart Contract	Category: Timing

Description

Every time there is a change in liquidity, it calls the `BinArray:: update_all_rewards` function to update the accumulated rewards for the current active bin.

However, due to the current rewards emission algorithm, if the current active bin is empty, the update will be skipped.

```
421 if reward_info.initialized() && bin.liquidity_supply > 0 {
```

programs/lb_clmm/src/state/bin.rs#L421

Impact

It will result in theft of unclaimed rewards. An attacker can add and remove liquidity in the same transaction to claim all the accumulated rewards of the current active bin.

Proof of Concept

Attack flow in the PoC test:

1. Wait until reward duration ends and the current active bin is empty.
2. The attacker creates a position and adds liquidity to the active bin.
3. The attacker calls the `claimReward` ix and gets all rewards in this period.
4. After claiming, the attacker can remove liquidity in the same tx.

Recommendation

When the liquidity is 0, the rewards timestamp should also be updated.

However, due to the current rewards emission algorithm, rewards must be fully emitted within a specified period. Implementing the above change would result in a portion of rewards being locked in the contract. To keep the emission algorithm unchanged, one option is to add an additional interface to claim rewards after the period ends. This change may also involve modifications to the `fund_reward` instruction.

Mitigation Review Log

Meteora Team: When the bin is empty liquidity, we will not distribute rewards to the next LP, instead we accumulate the time liquidity is zero. For the next funding, funder can choose to carry forward the remaining rewards to the next funding, or he/she can withdraw it: PR-182

Offside Labs: Fixed. To solve this issue, `cumulative_seconds_with_empty_liquidity_reward` field has been added to RewardInfo, `withdraw_ineligible_reward` instruction has been introduced to withdraw unclaimed rewards, and the `fund_reward` instruction has been modified accordingly.

Note: If the funder wants to extract both the precision residuals and external transferred reward tokens in the `withdraw_ineligible_reward` instruction, consider using a formula to calculate the transfer amount like

```
token_account.amount - reward_rate * (reward_duration - cumulative_seconds_with_empty_liquidity_reward)
```

When implementing this feature, be careful to handle floor/ceiling correctly to avoid off-by-one DOS.

4.4 Rewards are Skipped if the Swap is Crossing Multiple Bins

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Profit Distribution

Description

The rewards only accumulate on the current active bin. When iterating through the bins in a swap, only the first bin, which is the current active bin, accumulates rewards.

```
230        if start_active_id == lb_pair.active_id {
231            active_bin_array.update_all_rewards(&mut lb_pair, current_timestamp as u64)?;
232        }
```

programs/lb_clmm/src/instructions/swap.rs#L230-L232

So, when a swap is crossing multiple bins, only the first bin will get the accumulated rewards. The last bin of the swap can get the next accumulated rewards in the next swap. But all the intermediate bins are skipped.

Impact

This leads to uneven distribution of rewards. The following PoC demonstrates an edge case to illustrate why the distribution is unfair.

Proof of Concept

Scenario:

1. Assume the current active bin is empty.
2. `User1` adds liquidity to the right next bin of the active bin.
3. `User2` adds liquidity to the `right + 2` bin of the active bin, and only deposits a smaller amount.
4. Wait for half of rewards.
5. Swap from the active bin id to the active bin `id + 2`.
6. Users claim rewards and the following values have passed the check:

```
const user1_rewards = await claimReward(position1);
const user2_rewards = await claimReward(position2);
expect(user1_rewards.eq(new BN(0))).to.be.true;
expect(user2_rewards.sub(new BN(fundingAmount)).lte(new
    BN(1))).to.be.true;
```

Recommendation

If the swap crosses multiple bins, distribute the rewards among these bins.

Mitigation Review Log

Meteora Team: PR-171

We follow the recommendation, but instead of distributing evenly for all crossed bins, we only distribute rewards for the next 15 bins. That is due to limit of compute unit for a swap transaction.

Offside Labs: Mitigation Pending.

In the `distribute_rewards_to_swapped_bin` function, when iterating `bin_id`, if the current `bin_id` is not in the current `bin_array`, the next `bin_array` will be loaded. However, since the loop continues, this `bin_id` will be skipped and its rewards will not be accumulated.

```
        let mut bin_array = bin_array.load_mut()?;

        let bin_in_bin_array =
            bin_array.is_bin_id_within_range(*bin_id).is_ok();
        if bin_in_bin_array {
            let bin = bin_array.get_bin_mut(*bin_id)?;
```

d3bc99713288d8a682d7141b834fbf944b338505/programs/lb_-
clmm/src/instructions/swap.rs#L428-433

Note: Currently, the rewards are distributed evenly across multiple bins, which may lead to an unfair distribution of rewards due to differences in liquidity among the bins.

Meteora Team: We've created the fix on the PR-211.

Offside Labs: Fixed.

4.5 Attacker Can Keep Fees Max at Low Cost

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic

Description

The variable fee rate is calculated in the `lp_pair:: compute_variable_fee` function, with its variations primarily determined by the `volatility_accumulator`. The update of `volatility_accumulator` is in the `VariableParameters::update_volatility_parameter` function.

```
volatility_accumulator = min(volatility_reference + num_of_bin_crossed,
    max_volatility_accumulator)
```

And the above `volatility_reference` is calculated depending on time passed since last swap:

```
    let elapsed = current_timestamp.safe_sub(self.last_update_timestamp)?;

    // Not high frequency trade
    if elapsed >= static_params.get_filter_period() as i64 {
        // Update active id of last transaction
        self.index_reference = active_id;
        // filter period < t < decay_period. Decay time window.
        if elapsed < static_params.get_decay_period() as i64 {
            // UPDATE self.volatility_reference
            ...
        }
```

programs/lb_clmm/src/state/parameters.rs#L135-L149

Impact

The issue is that when the time since last swap is below `filter_period`, `volatility_reference` does not change, yet the `last_update_timestamp` is updated. Therefore, an attacker can keep fees extremely high at virtually 0 cost, by swapping just under every `filter_period` window with a zero-ish amount. Since Vr will forever stay the same, the calculated Va will stay `max_volatility_accumulator` , making the protocol completely uncompetitive around the clock.

It's an acknowledged issue in Trader Joe. The mitigation included adding a `forceDecay` function to reset the `index_reference` and `volatility_reference` by admin:

```
804 function forceDecay() external override onlyFactory {
```

traderjoe-xyz/joe-v2/src/LBPair.sol#L804

But the *DLMM* doesn't have such an interface.

Recommendation

Add an instruction that allows the admin to reset the `index_reference` and `volatility_reference` .

Mitigation Review Log

Meteora Team: We fix it by only update the `last_update_timestamp` if a bin is crossed in swap function: PR-178

Offside Labs: **Fixed**.

The mitigation has modified the update logic for `volatility_reference` . It will only update the last timestamp if bins are crossed. This may result in reduced dynamic fees affecting the protocol's revenue; please ensure that such a change is acceptable and intentional by design.

4.6 total_amount_out Can be Equal to the min_amount_out

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Data Validation

Description

If the user wants to precompute the expected `total_amount_out` and set it to the minimum value, the `total_amount_out` can also be equal to the `min_amount_out` .

```
300     require!(
301         total_amount_out > min_amount_out,
302         LBError::ExceededAmountSlippageTolerance
303     );
```

programs/lb_clmm/src/instructions/swap.rs#L300-L303

This is also known as a zero-slippage scenario, and in fact, most DEXs support this condition, such as Trader Joe.

```
if (amountOutMin > amountOut) revert
↪  LBRouter__InsufficientAmountOut(amountOutMin, amountOut);
```

https://github.com/traderjoe-xyz/joe-v2/blob/c6870ed6615ac9d96663c40216e5ed7c420b06e6/src/LBRouter.sol#L400

Recommendation

Change the condition to `total_amount_out >= min_amount_out`

Mitigation Review Log

Meteora Team: We pushed the fix here PR-166

Offside Labs: Fixed.

4.7 add_liquidity Fails to Verify that Distribution Sum Does Not Exceed 100%

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Data Validation

Description

For the `bin_liquidity_dist` array passed into the `add_liquidity` instruction, only individual distributions are checked in `get_amount_into_bin` to ensure they do not exceed 100%, without verifying that the sum of all distributions does not exceed 100%.

As a result, the final amounts, `amount_x` and `amount_y` , deposited may exceed the parameters provided by the user.

Recommendation

Add a check in the `LiquidityParameter:: validate` function to make sure the sum of distributions does not exceed 100%.

Mitigation Review Log

Meteora Team: We pushed the fix here PR-165

Offside Labs: Fixed.

4.8 Informational and Undetermined Issues

Redundant Checks

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Optimization

There are some redundant checks in the `validate` function of `LiquidityParameter` / `LiquidityParameterByWeight` / `LiquidityOneSideParameter` :

```
require!(bin_count > 0, LBError::InvalidInput);
require!(!self.bin_liquidity_dist.is_empty(), LBError::InvalidInput);
```

The `is_empty()` function also checks if the `bin_count==0` .

No Validations of bin_arrays When Claiming Fees or Rewards

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Data Validation

When the `bin_array_manager` is initialized with two `bin_arrays` (`bin_array_lower` and `bin_array_upper`), the manager could call `validate_bin_arrays` to check these two `bin_arrays` . This check is performed in the instruction handlers for `add_liquidity` , `add_liquidity_one_side` , `add_liquidity_by_weight` , `close_position` , and `remove_liquidity` , but is missing in the handler for `claim_fee` and `claim_reward` .

This check is not mandatory because the assert in `update_earning_per_token_stored` ensures that all the bins of this position are covered by the `bin_arrays` . However, if the requirement is to be removed in the future, it could potentially cause troubles due to the missing validation.

```
let mut bin_array_manager = BinArrayManager::new(&mut bin_arrays)?;
```

programs/lb_clmm/src/instructions/claim_reward.rs#L96

programs/lb_clmm/src/instructions/claim_fee.rs#L104

Mathematical Simplification

Severity: Informational	Status: Acknowledged
Target: Smart Contract & Math	Category: Math

The calculation of the ratio between the weights of x(wx0) and y(wy0) in the active bin(w0) can be simplified. The simplification process is as follows:

$$\omega_x = \frac{\omega_0}{p_0 + \frac{y}{x}}$$

$$\omega_y = \frac{\omega_0}{1 + \frac{p_0 * x}{y}}$$

$$\frac{\omega_x}{\omega_y} = \frac{1 + \frac{p_0 * x}{y}}{p_0 + \frac{y}{x}} = \frac{x * y + p_0 * x * x}{x * y * p_0 + y * y} = \frac{x(y + p_0 * x)}{y(y + p_0 * x)} = \frac{x}{y}$$

Where p_0 is the current price of the active bin, and x and y represent the amounts of tokens x and y in the current active bin.

Pair Can be Initialized With an Active Bin ID Outside the Preset Range

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Data Validation

The `initialize_lb_pair` instruction does not have a check, like `initialize_position` , to verify whether the input active bin ID is within the range (`min_bin_id` , `max_bin_id`) specified by `constants::get_preset` .

In practice, this does not result in a logical error in the program because bins outside the range are skipped since liquidity cannot be added to them.

However, it may lead to unexpected behavior in the frontend/CLI during the initial liquidity addition or swapping, so attention should be paid to this edge case.

Meteora Team: PR-185

5 Disclaimer

This audit report is provided for informational purposes only and is not intended to be used as investment advice. While we strive to thoroughly review and analyze the smart contracts in question, we must clarify that our services do not encompass an exhaustive security examination. Our audit aims to identify potential security vulnerabilities to the best of our ability, but it does not serve as a guarantee that the smart contracts are completely free from security risks.

We expressly disclaim any liability for any losses or damages arising from the use of this report or from any security breaches that may occur in the future. We also recommend that our clients engage in multiple independent audits and establish a public bug bounty program as additional measures to bolster the security of their smart contracts.

It is important to note that the scope of our audit is limited to the areas outlined within our engagement and does not include every possible risk or vulnerability. Continuous security practices, including regular audits and monitoring, are essential for maintaining the security of smart contracts over time.

Please note: we are not liable for any security issues stemming from developer errors or misconfigurations at the time of contract deployment; we do not assume responsibility for any centralized governance risks within the project; we are not accountable for any impact on the project's security or availability due to significant damage to the underlying blockchain infrastructure.

By using this report, the client acknowledges the inherent limitations of the audit process and agrees that our firm shall not be held liable for any incidents that may occur subsequent to our engagement.

This report is considered null and void if the report (or any portion thereof) is altered in any manner.