Dynamic Bonding Curve

Security Assessment

May 5th, 2025 — Prepared by OtterSec

Xiang Yin	soreatu@osec.io
Gabriel Ottoboni	ottoboni@osec.io
Kevin Chow	kchow@osec.io
Robert Chen	r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

MeteoraAg engaged OtterSec to assess the `dynamic-bonding-curve` program. This assessment was conducted between April 7th and April 18th, 2025. For more information on our auditing methodology, refer to Appendix C.

Key Findings

We produced 7 findings throughout this audit engagement.

In particular, we identified a vulnerability while applying the swap result, where the logic subtracts the full output (including fees) from the quote reserve during a base-to-quote swap with fees on output, resulting in the quote reserve to understate actual liquidity and potentially break migrations or fee claims (OS-MVC-ADV-00).

Additionally, the base vault amount may be stale when checking liquidity post-swap, risking incorrect migration readiness and leaving the pool stuck in an unusable completed state (OS-MVC-ADV-01). Furthermore, allowing `migration_quote_threshold` to be set to zero results in the pool skipping its launch phase and immediately transitioning to the migrated state, restricting any swaps (OS-MVC-ADV-02).

We made recommendations for modifying the codebase to rectify informational and typographical errors, and for improved functionality (OS-MVC-SUG-02), and suggested removing redundant or unutilized code instances (OS-MVC-SUG-03). We further advised utilizing the correct assumption while performing the proof derivation (OS-MVC-SUG-00).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/MeteoraAg/dynamic-bonding-curve. This audit was performed against commit 48cdb05. Follow-up reviews were performed for PR#75 and PR#78.

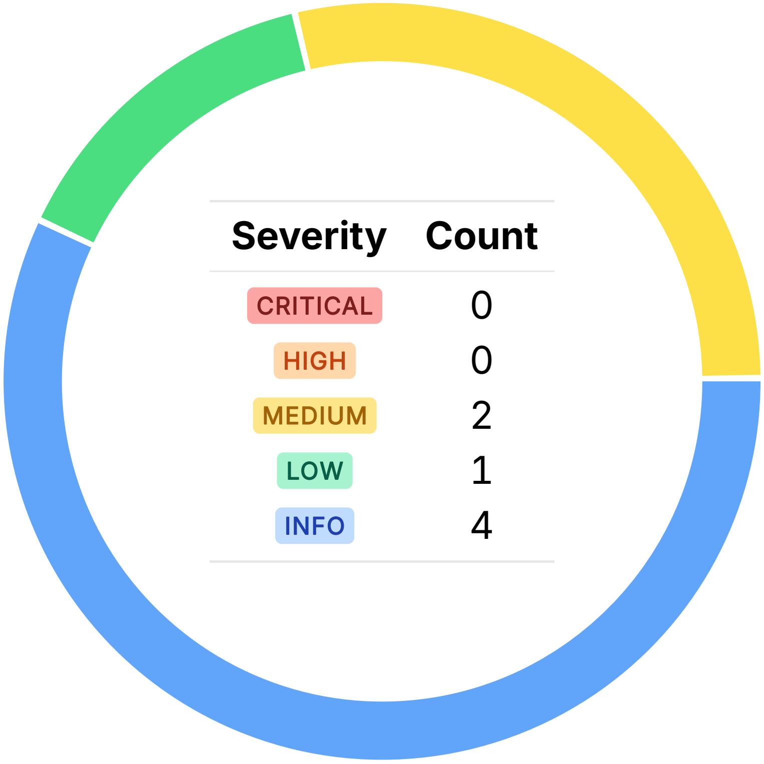
A brief description of the program is as follows:

Name	Description
dynamic-bonding-curve	It allows projects to launch tokens with customizable virtual liquidity curves, enabling immediate trading through integrations like Jupiter. The system supports flexible configurations, including quote tokens, fee structures, and liquidity distribution.

03 — Findings

Overall, we reported 7 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	0
HIGH	0
MEDIUM	2
LOW	1
INFO	4

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix B.

ID	Severity	Status	Description
OS-MVC-ADV-00	MEDIUM	RESOLVED ✓	`apply_swap_result` subtracts the full output (including fees) from `quote_reserve` during a base-to-quote swap with fees on output, resulting in `quote_reserve` to understate actual liquidity and potentially break migrations or fee claims.
OS-MVC-ADV-01	MEDIUM	RESOLVED ✓	The `base_vault.amount` may be stale when checking liquidity post-swap, risking incorrect migration readiness and leaving the pool stuck in an unusable completed state.
OS-MVC-ADV-02	LOW	RESOLVED ✓	Allowing `migration_quote_threshold` to be set to zero results in the pool skipping its launch phase and immediately transitioning to the migrated state, restricting any swaps.

Incorrect Quote Reserve Accounting MEDIUM

OS-MVC-ADV-00

Description

When performing a base-to-quote swap with fees charged on the output token, `get_swap_result` correctly returns the net amount (after fees) to the user. However, in `apply_swap_result`, the pre-fee `output_amount` is subtracted from `quote_reserve` (as shown below). This results in the on-chain `quote_reserve` to become higher than the actual token balance allocated to it. Consequently, since the migration logic requires at least `migration_quote_threshold`, either `migrate` or fee claims will fail.

>_ *virtual-curve/src/state/virtual_pool.rs* RUST

```rust
pub fn apply_swap_result(
    &mut self,
    swap_result: &SwapResult,
    fee_mode: &FeeMode,
    trade_direction: TradeDirection,
    current_timestamp: u64,
) -> Result<()> {
    [...]
    if trade_direction == TradeDirection::BaseToQuote {
        self.base_reserve = self.base_reserve.safe_add(actual_input_amount)?;
        self.quote_reserve = self.quote_reserve.safe_sub(output_amount)?;
    } else {
        self.quote_reserve = self.quote_reserve.safe_add(actual_input_amount)?;
        self.base_reserve = self.base_reserve.safe_sub(output_amount)?;
    }
    self.update_post_swap(old_sqrt_price, current_timestamp)?;
    Ok(())
}
```

Remediation

Ensure the amount subtracted includes fees as at least `migration_quote_threshold` available to migrate, otherwise migrate or fee claims will fail.

Patch

Resolved in 6fba296.

Stale Vault Balance Check MEDIUM

OS-MVC-ADV-01

Description

The vulnerability arises from utilizing a potentially stale `base_vault.amount` when checking if the pool has enough base-side liquidity to complete migration. Since the vault account is loaded before the swap executes, its `.amount` field may not reflect real-time changes. If this outdated balance is utilized, the check `base_vault_balance >= required_base_balance` may incorrectly succeed. This results in the pool emitting the `EvtCurveComplete` event and marking itself as completed, even though it lacks sufficient liquidity. As a result, the pool becomes stuck, further swaps are blocked, and migration fails due to insufficient funds.

>_ virtual-curve/src/instructions/ix_swap.rs RUST

```rust
pub fn handle_swap(ctx: Context<SwapCtx>, params: SwapParameters) -> Result<()> {
    [...]
    if pool.is_curve_complete(config.migration_quote_threshold) {
        // validate if base reserve is enough token for migration
        let base_vault_balance = ctx.accounts.base_vault.amount;
        let required_base_balance = config
            .migration_base_threshold
            .safe_add(pool.get_protocol_and_partner_base_fee()?)?;

        require!(
            base_vault_balance >= required_base_balance,
            PoolError::InsufficentLiquidityForMigration
        );

        emit_cpi!(EvtCurveComplete {[...]})
    }
    Ok(())
}
```

Remediation

Explicitly reload the vault after the swap logic and before the liquidity check.

Patch

Resolved in d3674b1.

Risk of Zero Threshold Migration LOW OS-MVC-ADV-02

Description

The `migration_quote_threshold` defines how much quote token must be swapped into the pool before it migrates into an AMM. Once this threshold is reached, the launch phase ends, and the token is assumed to have completed its initial distribution. However, currently, `migration_quote_threshold` may be set to zero when creating a configuration. Consequently, this will render it impossible to swap, because the pool starts already in a completed phase.

Remediation

Ensure the validation logic in `ConfigParameters::validate` enforces a minimum threshold for swaps before migration may be triggered.

Patch

Resolved in 02a1562.

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-MVC-SUG-00	The proof incorrectly assumes that y is constant during a swap, whereas liquidity (L) is actually constant. Additionally, the derivation of L' as $L + x * \sqrt{P}$ is flawed.
OS-MVC-SUG-01	Suggestions regarding inconsistencies in the codebase and ensuring adherence to coding best practices.
OS-MVC-SUG-02	Modifications to the codebase to rectify informational and typographical errors and improve functionality.
OS-MVC-SUG-03	The codebase contains multiple cases of redundant or unutilized code that should be removed.

Improper Proof Derivation

OS-MVC-SUG-00

Description

In `curve::get_next_sqrt_price_from_amount_a_rounding_up`, the function's inline proof incorrectly assumes constant y during a swap, which is not correct. Instead, liquidity (L) remains constant while token balances (`x` and `y`) change. Thus, it is not possible to derive L' as $L + \Delta x * \sqrt{P}$ correctly, because: $L + \Delta x * \sqrt{P} \Rightarrow x * \sqrt{P} + \Delta x * \sqrt{P} \Rightarrow x' * \sqrt{P} \neq x' * \sqrt{P'} = L'$.

>_ *dynamic-bonding-curve/src/curve.rs* RUST

```rust
/// # Formula
///
/// * `√P' = √P * L / (L + Δx * √P)`
/// * If Δx * √P overflows, use alternate form `√P' = L / (L/√P + Δx)`
///
/// # Proof
///
/// For constant y,
/// √P * L = y
/// √P' * L' = √P * L
/// √P' = √P * L / L'
/// √P' = √P * L / L'
/// √P' = √P * L / (L + Δx*√P)
///
pub fn get_next_sqrt_price_from_amount_a_rounding_up([...])
```

Remediation

Ensure to utilize the correct assumption while performing the derivation. A derivation that results in $\sqrt{P'} = \sqrt{P} * L / (L + \Delta x * \sqrt{P})$ is attached for reference in Appendix A.

Code Maturity

OS-MVC-SUG-01

Description

1. `validate_config_key` requires that a configuration's `pool_creator_authority` matches the caller's key, but only admins may create configurations in `dammv2`. This implies users may create virtual pools without having a valid configuration, rendering those pools unmigratable later. The virtual pool creation flow should check if a valid configuration exists and is owned by the user.

   >_ *instructions/migration/dynamic_amm_v2/migrate_damm_v2_initialize_pool.rs* RUST

   ```rust
   impl<'info> MigrateDammV2Ctx<'info> {
       fn validate_config_key(&self, damm_config: &damm_v2::accounts::Config) -> Result<()> {
           require!(
               damm_config.pool_creator_authority == self.pool_authority.key(),
               PoolError::InvalidConfigAccount
           );
           [...]
       }
       [...]
   }
   ```

2. `handle_migrate_meteora_damm_lock_lp_token` allows either the partner or pool creator to lock their portion of LP tokens. However, if the signer is not one of the expected parties, the function currently fails silently. An explicit `else` check should be added (similar to what `handle_migrate_meteora_damm_claim_lp_token` does) to ensure unrecognized signers trigger an immediate error, instead of failing silently.

3. Consider returning the remaining lamports sent to `pool_authority` back to the payer after the `initialize_pool` CPI call in `create_pool`, or alternatively, introduce a new endpoint to allow claiming them. Otherwise, these lamports will remain idle and un-utilized.

   >_ *instructions/migration/dynamic_amm_v2/migrate_damm_v2_initialize_pool.rs* RUST

   ```rust
   fn create_pool([...]) -> Result<()> {
       [...]
       msg!("transfer lamport to pool_authority");
       invoke(
           &system_instruction::transfer(
               &self.payer.key(),
               &self.pool_authority.key(),
               50_000_000, // TODO calculate correct lamport here
           ),
           &[[...]],
       )?;[...]
   }
   ```

4. Add a sanity check on `max_limiter_duration` to prevent misconfiguration or abuse by capping how long rate limits apply, ensuring predictable and safe throttling behavior.
5. The current logic may block all swaps in a transaction if they utilize the same pool. Consider verifying whether the swap instructions share a pool to avoid unnecessarily blocking valid swaps.

Remediation

Incorporate the above refactors.

Code Clarity

OS-MVC-SUG-02

Description

1. The comment in `get_current_base_fee_numerator` states that *"If trading occurs before the activation point, it's an alpha vault"*. This is misleading because it references "alpha-vault" behavior, which is not supported in the Virtual Curve protocol. The logic actually forces the fee to the minimum before the activation point by setting the period to the maximum. The intent is to apply the lowest fee when the decay schedule has not started, not to simulate alpha vault functionality. The comment should be updated to accurately reflect this purpose.
2. There is a typographical error in the `PoolFeeParamters` structure name in `fee_parameters`. Ensure to rectify the spelling.
3. The current implementation of `handle_swap` utilizes `clone().unwrap()` to access the referral account, which is unnecessary. This may be improved by replacing the `if has_referral` check with `if let Some(referral_account)`, avoiding cloning and unwrapping.

>_ *virtual-curve/src/curve.rs* RUST

```rust
pub fn handle_swap(ctx: Context<SwapCtx>, params: SwapParameters) -> Result<()> {
    [...]
    // send to referral
    if has_referral {
        if fee_mode.fees_on_base_token {
            transfer_from_pool(
                [...]
                &ctx.accounts.referral_token_account.clone().unwrap(),
                &ctx.accounts.token_base_program,
                swap_result.referral_fee,
                ctx.bumps.pool_authority,
            )?;
        } else {
            transfer_from_pool(
                ctx.accounts.pool_authority.to_account_info(),
                &ctx.accounts.quote_mint,
                &ctx.accounts.quote_vault,
                &ctx.accounts.referral_token_account.clone().unwrap(),
                [...]
            )?;
        }
    }
    [...]
}
```

4. The comments describing `token_a_account` and `token_b_account` as treasury accounts are misleading because there are no constraints enforcing that these accounts belong to a treasury and thus should be removed.

Remediation

Implement the above-mentioned suggestions.

Unutilized/Redundant Code

OS-MVC-SUG-03

Description

1. The `pool` account in `MigrateMeteoraDammClaimLpTokenCtx` is currently unutilized. Also, the owner field in `PoolConfig` does not seem to be utilized for anything.
2. The `match` block in `MigrateDammV2Ctx::validate_config_key` is redundant, as it contains only a single arm that is always executed. Replacing it with a direct `require` statement would achieve the same effect more clearly.

   >_ *instructions/migration/dynamic_amm_v2/migrate_damm_v2_initialize_pool.rs* RUST

   ```rust
   impl<'info> MigrateDammV2Ctx<'info> {
       fn validate_config_key(
           &self,
           damm_config: &damm_v2::accounts::Config,
           migration_fee_option: u8,
       ) -> Result<()> {
           [...]
           // validate non fee scheduler
           match migration_fee_option {
               MigrationFeeOption::FixedBps25
               | MigrationFeeOption::FixedBps30
               | MigrationFeeOption::FixedBps100
               | MigrationFeeOption::FixedBps200 => {
                 require!(
                     damm_config.pool_fees.base_fee.period_frequency == 0,
                     PoolError::InvalidConfigAccount
                 );
               }
           }
           [...]
       }
       [...]
   }
   ```

3. The logic in `creator_claim_lp_from_meteora_dynamic_amm` and `migrate_meteora_damm_creator_claim_lp_token` is redundant, as they both call `handle_migrate_meteora_damm_creator_claim_lp_token(ctx)` internally. Similarly, `partner_claim_lp_from_meteora_dynamic_amm` and `migrate_meteora_damm_partner_claim_lp_token` call `handle_migrate_meteora_damm_creator_claim_lp_token`.

4. In `handle_initialize_virtual_pool_with_token2022`, the `config` account is loaded twice, once at the beginning of the function, and then again following the call to `update_account_lamports_to_minimum_balance`. Remove the duplicate line.

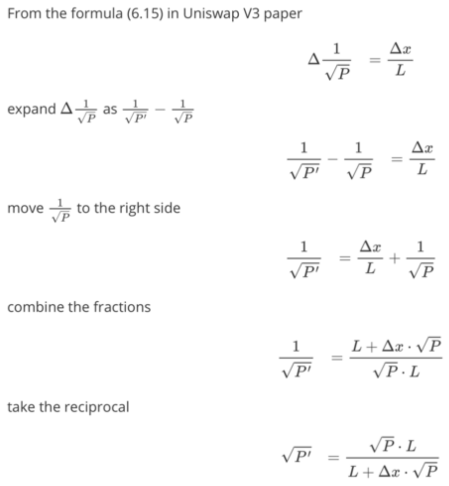
   ```rust
   >_ dynamic-bonding-curve/src/instructions/initialize_pool /ix_initialize_virtual_pool_with_token2022.rs   RUST
   pub fn handle_initialize_virtual_pool_with_token2022<'c: 'info, 'info>(
       ctx: Context<'_, '_, 'c, 'info, InitializeVirtualPoolWithToken2022Ctx<'info>>,
       params: InitializePoolParameters,
   ) -> Result<()> {
       let config = ctx.accounts.config.load()?;
       [...]
       // transfer minimum rent to mint account
       update_account_lamports_to_minimum_balance(
           ctx.accounts.base_mint.to_account_info(),
           ctx.accounts.payer.to_account_info(),
           ctx.accounts.system_program.to_account_info(),
       )?;
       let config = ctx.accounts.config.load()?;
       [...]
   }
   ```

5. Standardize the `signer` constraint during account initialization if it is redundant by adding it or removing it.
6. Reduce redundancy in `get_base_fee_numerator` by re-utilizing the `is_rate_limiter_applied` check. If the rate limiter is not applied, the function should directly return `cliff_fee_numerator`, simplifying the logic.

Remediation

Remove the redundant and unutilized code instances.

A — Uniswap V3 Reference



From the formula (6.15) in Uniswap V3 paper

$$\Delta\frac{1}{\sqrt{P}} = \frac{\Delta x}{L}$$

expand $\Delta\frac{1}{\sqrt{P}}$ as $\frac{1}{\sqrt{P'}} - \frac{1}{\sqrt{P}}$

$$\frac{1}{\sqrt{P'}} - \frac{1}{\sqrt{P}} = \frac{\Delta x}{L}$$

move $\frac{1}{\sqrt{P}}$ to the right side

$$\frac{1}{\sqrt{P'}} = \frac{\Delta x}{L} + \frac{1}{\sqrt{P}}$$

combine the fractions

$$\frac{1}{\sqrt{P'}} = \frac{L + \Delta x \cdot \sqrt{P}}{\sqrt{P} \cdot L}$$

take the reciprocal

$$\sqrt{P'} = \frac{\sqrt{P} \cdot L}{L + \Delta x \cdot \sqrt{P}}$$

Figure A.1: A derivation that leads to the result $\sqrt{}P' = \sqrt{}P * L/(L + x * \sqrt{}P)$

B — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

C — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.