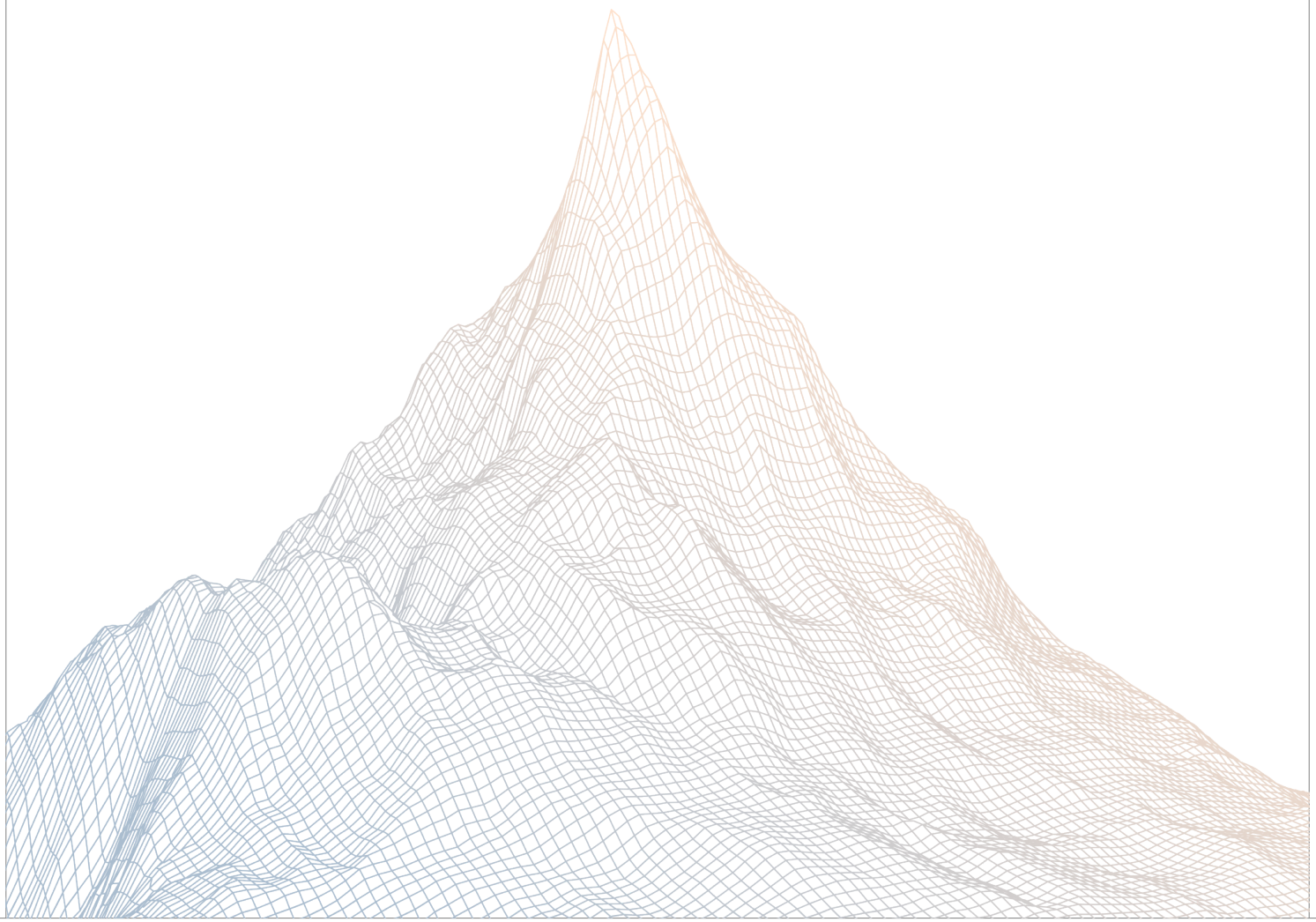
Zenith

Meteora

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: November 28th to December 8th, 2025

AUDITED BY: peakbolt

Contents

1 Introduction **2**

1.1 About Zenith 3

1.2 Disclaimer 3

1.3 Risk Classification 3

2 Executive Summary **3**

2.1 About Meteora 4

2.2 Scope 4

2.3 Audit Timeline 5

2.4 Issues Found 5

3 Findings Summary **5**

4 Findings **6**

4.1 Medium Risk 7

4.2 Low Risk 14

4.3 Informational 17

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

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	MeteoraAg/damm-v2#124
Repository	https://github.com/MeteoraAg/damm-v2
Commit Hash	`425cdd128ab139f97c4f2b855423b6290d5ac28b`
Files	`Changes in release 0.1.6`

Target	MeteoraAg/damm-v2#124 Mitigation Review
Repository	https://github.com/MeteoraAg/damm-v2
Commit Hash	`6b3dced7f6392190a4400e5ce56821b1b741a7f3`
Files	`Changes in release 0.1.6`

2.3 Audit Timeline

November 28, 2025	Audit start
December 8, 2025	Audit end
December 10, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	4
Low Risk	2
Informational	2
Total Issues	**8**

3

Findings Summary

ID	Description	Status
M-1	Incorrect max fee validation for update_pool_fees	Resolved
M-2	Infinite loop in validate_single_swap_instruction will cause swap TX to fail	Resolved
M-3	MarketCapScheduler's fee decay will be slower than expected due to wrong unit for price_step_bps	Resolved
M-4	Missing bound checks for accounts.split_at_unchecked() in p_entrypoint()	Resolved
L-1	validate() for Market Cap Scheduler should enforce number_of_period > 0	Resolved
L-2	Missing account data size and alignment checks inp_load_mut_checked()	Resolved
I-1	Incorrect error messages used for update_pool_fees	Resolved
I-2	close_operator_account() should emit event	Acknowledged

4

Findings

4.1 Medium Risk

A total of 4 medium risk findings were identified.

[M-1] Incorrect max fee validation for `update_pool_fees`

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

Target

- fee_rate_limiter.rs#L419
- fee_market_cap_scheduler.rs#L199
- fee_time_scheduler.rs#L163

Description:

During swaps, it will get the max fee numerator using `get_max_fee_numerator(self.verison)`, which will use the value based on the pool's version.

However, `update_pool_fees` will get the max fee numerator based on `get_max_fee_numerator(CURRENT_POOL_VERSION)`, intead of the pool's version.

That means for existing pool on version 0 (max fee 50%), it will incorrectly validate based on version 1 (max 99%) and allow pool fee to exceeds 50%.

Though during swaps it still clamps down to the max fee based on pool's version.

Recommendations:

This issue can be resolved by ensuring that `update_pool_fees` will validate max fee based on the pool's version and not the program's current version.

Meteora: Resolved with `@3f45ffd6ac ...`

Zenith: Verified. Resolved by validating max fee with a lower max fee numerator (10%), which will be lower than both pool's and program's.

[M-2] Infinite loop in `validate_single_swap_instruction` will cause swap TX to fail

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

Target

- ix_p_swap.rs#L315-L345

Description:

In `validate_single_swap_instruction`, the `for` loop will look at earlier IXs in the TX before the current swap IX and check that there are no other IXs for the same pool.

However, the inner `loop` uses the same index `i` as the outer `for`, and this will cause an infinite loop as the inner `loop` fails to increment the index `i`. Furthermore, using the same index `i` causes the inner `loop` to start from the wrong index.

The finite loop will cause TX with swap/swap2 IX to fail if there are other non-DAMM-v2 IX before the swap/swap2 IX.

```
pub fn validate_single_swap_instruction<'c, 'info>(
....
for i in 0..current_index {
        let instruction = instruction_sysvar_instructions
            .load_instruction_at(i.into())
            .map_err(|err| ProgramError::from(u64::from(err)))?;

        if instruction.get_program_id() ≠ crate::ID.as_array() {
            // we treat any instruction including that pool address is other
    swap ix
            loop {
                match instruction.get_account_meta_at(i.into()) {
                    Ok(account_metadata) ⇒ {
                        if &account_metadata.key == pool.as_array() {
                            msg!("Multiple swaps not allowed");
                            return
    Err(PoolError::FailToValidateSingleSwapInstruction.into());
                        }
                    }
```

```
                Err(err) => {
                    if err ==
pinocchio::program_error::ProgramError::InvalidArgument {
                        break;
                    } else {
                        return Err(PoolError::UndeterminedError.into());
                    }
                }
            }
        }
    } else {
        require!(
            !is_p_instruction_include_pool_swap(&instruction, pool)?,
            PoolError::FailToValidateSingleSwapInstruction
        );
    }
}
```

Recommendations:

Fix the infinite loop issue by using a different loop index and increment it on each iteration.

Meteora: Resolved with `@60749bf3b6 ...`

Zenith: Verified. Resolved by using a different loop index with a for loop.

[M-3] MarketCapScheduler's fee decay will be slower than expected due to wrong unit for `price_step_bps`

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

Target

- fee_market_cap_scheduler.rs#L148-L153

Description:

MarketCapScheduler measures the relative change in `sqrt_price`, expressed in bps and then divided by `price_step_bps`.

However, the config name `price_step_bps` suggests “price move in bps”, instead of "sqrt price move in bps". They are not the same unit of measurements and will differ significantly for large price move.

For example,

- Suppose initial price = 100.
- And the current price increased to 200 (price doubled, +100% = 10,000 bps)
- That means `initial_sqrt_price` = sqrt(100) = 10.
- And `current_sqrt_price` = sqrt(200) = 14.1421356.
- The relative sqrt price change = (14.1421456 - 10) / 10 = 41.42%
- So for a +100% price move, the sqrt price move is only 41.42%.

That means the fee decay will be slower than expected, and the mismatch is larger for bigger price move.

```
pub fn get_base_fee_numerator(
    ...
                    let price_step_bps = U256::from(self.price_step_bps);
                    let passed_period = current_sqrt_price
                        .safe_sub(init_sqrt_price)?
                        .safe_mul(max_bps)?
                        .safe_div(init_sqrt_price)?
                        .safe_div(price_step_bps)?;
```

Recommendations:

Rename `price_step_bps` to `sqrt_price_step_bps` and implement the SDK/frontend accordingly.

Meteora: Resolved with `@3de8101e2e ...`

Zenith: Verified.

[M-4] Missing bound checks for `accounts.split_at_unchecked()` in `p_entrypoint()`

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

Target

- entrypoint.rs#L34

Description:

`p_entrypoint()` does `accounts.split_at_unchecked(SWAP_IX_ACCOUNTS)` to split the input `accounts` into `accounts` and `remaining_accounts`.

However, it does not check perform a bound check to ensure that number of input `accounts` is >= `SWAP_IX_ACCOUNTS`. If the number of `accounts` < `SWAP_IX_ACCOUNTS`, it will lead to undefined behavior.

```
unsafe fn p_entrypoint(input: *mut u8) -> Option<u64> {
    ....
    let result = match instruction_bits {
        [true, false, false] | [false, true, false] => {
            let (left, right)
    = accounts.split_at_unchecked(SWAP_IX_ACCOUNTS);
            let accounts = core::slice::from_raw_parts(left.as_ptr() as _,
    SWAP_IX_ACCOUNTS);
            let remaining_accounts = core::slice::from_raw_parts(
                right.as_ptr() as _,
                count.checked_sub(SWAP_IX_ACCOUNTS)?,
            );
```

The rust docs for split_at_unchecked recommends bound checking as described below,

> Calling this method with an out-of-bounds index is undefined behavior even if the resulting reference is not used. The caller has to ensure that 0 <= mid <= self.len().

Recommendations:

Consider moving up the `count.checked_sub(SWAP_IX_ACCOUNTS)?` to throw an error when `count < SWAP_IX_ACCOUNTS`, before it hits `accounts.split_at_unchecked(SWAP_IX_ACCOUNTS)`.

Meteora: Resolved with `@4664e9025d ...`

Zenith: Verified. Resolved by throwing an error when `accounts.len() < SWAP_IX_ACCOUNTS`.

4.2 Low Risk

A total of 2 low risk findings were identified.

[L-1] `validate()` for Market Cap Scheduler should enforce `number_of_period > 0`

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

Target

- fee_market_cap_scheduler.rs#L171-L190

Description:

In `validate()` for Market Cap Scheduler, it validates that `reduction_factor > 0`, `price_step_bps > 0`, and `scheduler_expiration_duration > 0`.

However, it did not enforce `number_of_period > 0`.

That means its possible to set `number_of_period = 0` and configure a 'zero' Market Cap Scheduler, that does not reduces the base fee.

Recommendations:

Consider adding a check to ensure `number_of_period > 0`.

Meteora: Resolved with `@ab6b056ff1 ...`

Zenith: Verified.

[L-2] Missing account data size and alignment checks in `p_load_mut_checked()`

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

Target

- p_helper.rs#L61-L87

Description:

`p_load_mut_checked()` is supposed to mirror Anchor's AccountLoader `load_mut()`.

However, it is missing account data size and alignment checks, which are performed by bytemuck deserialization.

WIthout these checks, it could result in undefined behavior (UB) if there are differences in the account data layout due to future changes, leading to panic or data corruption.

```
// same as AccountLoader load_mut() but check for discriminator and owner
pub fn p_load_mut_checked<T: Discriminator + Owner>(acc_info: &AccountInfo)
    -> Result<&mut T> {
    // validate owner
    require!(
        acc_info.owner().eq(&T::owner().to_bytes()),
        ErrorCode::AccountOwnedByWrongProgram
    );

    if !acc_info.is_writable() {
        return Err(ErrorCode::AccountNotMutable.into());
    }

    let disc = T::DISCRIMINATOR;
    let mut data = acc_info
        .try_borrow_mut_data()
        .map_err(|_| ProgramError::AccountBorrowFailed)?;

    if data.len() < disc.len() {
        return Err(ErrorCode::AccountDiscriminatorNotFound.into());
    }
```

```
    let given_disc = &data[..disc.len()];
    if given_disc ≠ disc {
        return Err(ErrorCode::AccountDiscriminatorMismatch.into());
    }

    Ok(unsafe { &mut *(data[disc.len()..].as_mut_ptr() as *mut T) })
}
```

If we see bytemuck::try_from_bytes_mut(), it will actually check the data size and data alignment when deserializing the on-chain data (see below).

```
/// Re-interprets `&mut [u8]` as `&mut T`.
///
/// ## Failure
///
/// * If the slice isn't aligned for the new type
/// * If the slice's length isn't exactly the size of the new type
#[inline]
pub(crate) unsafe fn try_from_bytes_mut<T: Copy>(
  s: &mut [u8],
) -> Result<&mut T, PodCastError> {
  if s.len() ≠ size_of::<T>() {
    Err(PodCastError::SizeMismatch)
  } else if !is_aligned_to(s.as_ptr() as *const (), align_of::<T>()) {
    Err(PodCastError::TargetAlignmentGreaterAndInputNotAligned)
  } else {
    Ok(unsafe { &mut *(s.as_mut_ptr() as *mut T) })
  }
}
```

Recommendations:

Use `bytemuck::try_from_bytes_mut()` or manually check account data size and alignment checks in `p_load_mut_checked()`.

Meteora: Resolved with @e3968325cf ...

Zenith: Verified. Resolved by using `bytemuck::try_from_bytes_mut()` and panic when it fails to deserialize.

4.3 Informational

A total of 2 informational findings were identified.

[I-1] Incorrect error messages used for `update_pool_fees`

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

Target

- ix_update_pool_fees.rs#L59-L62
- pool.rs#L1359-L1362

Description:

In `UpdatePoolFeesParameters.validate()`, it will throw `InvalidDynamicFeeParameters` when both `cliff_fee_numerator` and `dynamic_fee` are `None`.

It should instead throw the error `InvalidUpdatePoolFeesParameters` as it could be an issue with either base fee or dynamic fee parameters.

```
fn validate(&self) -> Result<()> {
    // We don't need to validate `cliff_fee_numerator` in case we update it.
    // Because after update pool fee we will validate pool fee with new
    updated parameters
    require!(
        self.cliff_fee_numerator.is_some() || self.dynamic_fee.is_some(),
        PoolError::InvalidDynamicFeeParameters
    );
```

In `validate_and_update_pool_fees()`, it will throw the error `InvalidUpdatePoolFeesParameters` even though it is an error due to dynamic fee parameters.

It should instead throw `InvalidDynamicFeeParameters`.

```
// update dynamic fee
match params.get_dynamic_fee_update_mode() {
```

```
    DynamicFeeUpdateMode::Disable => {
        require!(
            self.pool_fees.dynamic_fee.is_dynamic_fee_enable(),
            PoolError::InvalidUpdatePoolFeesParameters
        );
        self.pool_fees.dynamic_fee = DynamicFeeStruct::default();
    }
```

is only used in the dynamic-fee disable path.

Recommendations:

Throw the appropriate error messages in `UpdatePoolFeesParameters.validate()` and `validate_and_update_pool_fees()`.

Meteora: Resolved with `@af5eace62f...`

Zenith: Verified.

[I-2] `close_operator_account()` should emit event

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

Target

- lib.rs#L96

Description:

`close_operator_account()` does not emit any event and this is inconsistent with the other close IX (close_config, close_token_badge, close_claim_fee_operator).

This makes it slightly harder to track operator account closure on chain.

Recommendations:

Consider emitting an event for `close_operator_account()`.

Meteora: Acknowledged, caused it is permissioned action.