audita

MageLabs DEX

Smart Contract Security Audit

May 17th, 2025

Prepared for:
Magelabs - Decentralized Exchange on Solana (*magelabs.com*)

Presented By:
Audita Security (*audita.io*)

Document

Network: Solana
Programming language: Rust
Method: Manual Audit
Client Website: https://magelabs.com
Timeline: 07/05/2025 - 17/05/2025

Table of Contents

Executive Summary

Manual Audit

During the manual audit conducted by our experts, we did not identify any Critical or High severity vulnerabilities.

We identified 2 Medium and 4 Low severity vulnerabilities, as well as 4 Informational issues.

Overall Assessment

After a detailed and thorough security review, our researchers did not identify any vulnerabilities of critical or high severity.
MageLabs DEX is, to the best of our knowledge, safe to use.

Severity	Count	Acknowledged
Critical	0	-
High	0	-
Medium	2	**Acknowledged**
Low	4	**Acknowledged**
Informational	4	**Acknowledged**

Documentation

We recommend this report, as well as specific information from this report to be included in MageLabs DEX's official protocol Documentation.

Audita Vulnerability Classifications

Audita follows the most recent standards for vulnerability severities, taking into consideration both the possible impact and the likelihood of an attack occurring due to a certain vulnerability.

Severity	Description
Critical	Critical vulnerability is one where the attack is more straightforward to execute and can lead to exposure of users' data, with catastrophic financial consequences for clients and users of the smart contracts.
High	The vulnerability is of high importance and impact, as it has the potential to reveal the majority of users' sensitive information and can lead to significant financial consequences for clients and users of the smart contracts.
Medium	The issue at hand poses a potential risk to the sensitive information of a select group of individual users. If exploited, it has the potential to cause harm to the client's reputation and could result in unpleasant financial consequences.
Low	The vulnerability is relatively minor and not likely to be exploited repeatedly, or is a risk that the client has indicated is not impactful or significant, given their unique business situation.
Informational	The issue may not pose an immediate threat to ongoing operation or utilization, but it's essential to consider implementing security and software engineering best practices, or employing backup measures as a safety net.

Scope

The security assessment was scoped to the following files in MageLabs DEX's code repository:

Files
-src
-curve
calculator.rs
constant_product.rs
fees.rs
mod.rs
snapper.rs
-instructions
-admin
collect_fund_fee.rs
collect_protocol_fee.rs
create_config.rs
mod.rs
update_config.rs
update_pool_status.rs
close.rs

deposit.rs
initialize_metadata.rs
initialize.rs
mod.rs
swap_base_input.rs
swap_base_output.rs
withdraw.rs
-states
config.rs
events.rs
mod.rs
oracle.rs
pool.rs
user.rs
-utils
account_load.rs
math.rs
mod.rs
token.rs
error.rs

```
lib.rs
```

The codebase has been audited up to and including commit:

2773cee9c0a86d3eb70e5e91f932820613e7db93

Findings

Summary

Code	Description	Severity	Fixes
[MAGE-01]	Observations can be Insufficient for TWAP Calculation	Medium	Acknowledged
[MAGE-02]	TWAP Oracle may not correctly reflect the price in a volatile market	Medium	Acknowledged
[MAGE-03]	Inconsistent or Incomplete Access Control Between Admin and Owner in Fee Collection Instructions	Low	Acknowledged
[MAGE-04]	Missing Validation for Fee Parameters in *update_amm_config*	Low	Acknowledged
[MAGE-05]	Dangerous NatSpec Documentation in *update_amm_config*	Low	Acknowledged
[MAGE-06]	The *close* function does not have any restrictions in place	Low	Acknowledged
[MAGE-07]	Unused function level variables	Informational	Acknowledged

[MAGE-08]	The *Fees::trading_fee* function is not being used anywhere	Informational	Acknowledged
[MAGE-09]	Typos	Informational	Acknowledged
[MAGE-10]	Redundant functions	Informational	Acknowledged

Detailed Findings

[MAGE-01]	Observations can be Insufficient for TWAP Calculation	Medium

Details:

The *ObservationState* in oracle.rs uses a fixed-size circular buffer to store observations, with the size set to *OBSERVATION_NUM = 100.*

```
pub const OBSERVATION_NUM: usize = 100;
```

Each observation is updated every *OBSERVATION_UPDATE_DURATION_DEFAULT = 15* seconds.

```
OBSERVATION_UPDATE_DURATION_DEFAULT = 15
```

This means the buffer retains a maximum of 25 minutes of historical price data.

```
 if delta_time < OBSERVATION_UPDATE_DURATION_DEFAULT
      return;
 }
```

However, this design may not be sufficient. As a result, TWAP calculations may fail to include the intended time frame, leading to inaccuracies in price-sensitive operations.

Impact:

A fixed 25-minute window could lead to inaccuracies.

A larger observation capacity would provide more accurate TWAP calculations, particularly in highly active pools or during volatile market conditions where frequent updates shorten the effective lookback window.

Recommendation:

Increase *OBSERVATION_NUM* to 150–200, allowing the system to retain a broader historical window without overwriting data too quickly.

[MAGE-02]	TWAP Oracle may not correctly reflect the price in a volatile market	Medium

Details:

The current implementation of the TWAP oracle updates the observation array every 15 seconds, as defined by *OBSERVATION_UPDATE_DURATION_DEFAULT*:

```
pub const OBSERVATION_UPDATE_DURATION_DEFAULT: u64 = 15;
```

```
    pub fn update(
        &mut self,
        block_timestamp: u64,
        token_0_price_x32: u128,
        token_1_price_x32: u128,
    ) {
        let observation_index = self.observation_index;
        if !self.initialized {
            self.initialized = true;
            self.observations[observation_index as usize].block_timestamp =
block_timestamp;
            self.observations[observation_index as
usize].cumulative_token_0_price_x32 = 0;
            self.observations[observation_index as
usize].cumulative_token_1_price_x32 = 0;
        } else {
            let last_observation = self.observations[observation_index as
usize];
            let delta_time =
block_timestamp.saturating_sub(last_observation.block_timestamp);
--->            if delta_time < OBSERVATION_UPDATE_DURATION_DEFAULT {
                return;
            }
.....
.....
```

While this interval might be reasonable for networks like Ethereum, which has block times of approximately 12 seconds, it introduces essential limitations on Solana, which has block times of approximately 400 milliseconds.

Due to the longer update interval:

- Rapid price changes within a 15-second period may go completely unnoticed by the oracle.
- On Solana, where 15 seconds can span more than 30 blocks, sharp price movements can occur and be entirely excluded from the TWAP calculations.

For example, in a volatile market scenario, if a token experiences a sharp price drop or spike within a few seconds, this event will not be reflected in the TWAP. Instead, the TWAP will inaccurately portray the market as stable, skewing averages and leading to incorrect pricing.

Impact:

Significant price changes within the 15-second interval are ignored, leading to delayed or inaccurate TWAP calculations. The oracle may not reflect market conditions during critical moments, creating discrepancies in price-sensitive operations.

Recommendation:

Reduce the *OBSERVATION_UPDATE_DURATION_DEFAULT* to around 5 seconds to align better with Solana's block time and capture more granular price data.

[MAGE-03]	Inconsistent or Incomplete Access Control Between Admin and Owner in Fee Collection Instructions	Low

Details:

The fee collection instructions *collect_fund_fee* and *collect_protocol_fee* both contain outdated or unimplemented access control logic, as indicated by commented-out code and TODO-style comments in NatSpec:

In *collect_fund_fee*:

```
/// Only admin or fund_owner can collect fee now
// #[account(constraint = (owner.key() == amm_config.fund_owner ||
owner.key() == crate::admin::id()) @ ErrorCode::InvalidOwner)]
#[account(constraint = (owner.key() == crate::admin::id()) @
ErrorCode::InvalidOwner)]
```

In *collect_protocol_fee*:

```
/// Only admin or owner can collect fee now
// #[account(constraint = (owner.key() == amm_config.protocol_owner ||
owner.key() == crate::admin::id()) @ ErrorCode::InvalidOwner)]
#[account(constraint = (owner.key() == crate::admin::id()) @
ErrorCode::InvalidOwner)]
```

Currently, both instructions only allow the hardcoded admin to collect fees. However, the presence of comments and unused access paths referencing *fund_owner* and *protocol_owner* suggests that the protocol intends to support decentralized or delegated control, but it is not yet implemented.

Impact:

The severity of this issue is Low/Info.

- The presence of unused or outdated comments/code may confuse integrators or future contributors.
- Current setup allows only the admin to collect fees, which centralizes control and may violate decentralization or DAO expectations.

Recommendation:

You have two options:

- Implement the Intended Access Control
- Remove Inaccurate Comments and Dead Code

[MAGE-04]	Missing Validation for Fee Parameters in *update_amm_config*	Low

Details:

The *update_amm_config* instruction allows updating fee-related fields in the *AmmConfig* account using a *param*-based switch. Specifically, the instruction permits direct assignment of values to the following fields without validating that they are within acceptable bounds:

```
Some(3) => amm_config.protocol_fee = value,
Some(4) => amm_config.creator_fee = value,
```

All of these parameters are expected to be basis-point-style values where the denominator is *FEE_RATE_DENOMINATOR_VALUE = 1_000_000* (i.e., representing 100%). However, no *require!* checks are performed to ensure that the provided *value* is less than or equal to this denominator.

Additionally, no checks are in place to ensure that the sum of all fee rates does not exceed the denominator, which could result in invalid fee splits or arithmetic underflows elsewhere in the protocol.

Impact:

Setting a fee rate above 1_000_000 (e.g., 1_500_000) can cause:

- Swap and deposit operations to revert or panic due to underflows or invalid math (especially in fee split calculations).
- Protocol or fund revenue loss due to incorrect fee allocation.
- Denial of service on the pool if the fee math fails consistently due to bad config.

Recommendation:

Add *require*! checks to enforce per-parameter upper bounds:

```
require!(value <= FEE_RATE_DENOMINATOR_VALUE, ErrorCode::FeeTooHigh);
```

[MAGE-05]	Dangerous NatSpec Documentation in *update_amm_config*	Low

Details:

The *update_amm_config* instruction is documented with NatSpec-style comments that **do not reflect the actual logic implemented** in the underlying instruction (*update_config.rs*). Specifically, the documented meanings of the param argument are incorrect or outdated:

Current NatSpec claims:

```
/// * `param` - The value can be 0 | 1 | 2 | 3 | 4, otherwise will report a
error
```

```
/// * `trade_fee_rate` - The new trade fee rate of amm config, be set when
`param` is 0
/// * `protocol_fee_rate` - ..., when `param` is 1
/// * `fund_fee_rate` - ..., when `param` is 2
/// * `new_owner` - when `param` is 3
/// * `new_fund_owner` - when `param` is 4
```

But this is the actual implementation:

```
pub fn update_amm_config(ctx: Context<UpdateAmmConfig>, param: u8, value:
u64) -> Result<()> {
    let amm_config = &mut ctx.accounts.amm_config;
    let match_param = Some(param);
    match match_param {
        Some(0) => {
            let new_fund_owner =
*ctx.remaining_accounts.iter().next().unwrap().key;
            set_new_fund_owner(amm_config, new_fund_owner)?;
        }
        Some(1) => amm_config.token_1_lp_rate = value,
        Some(2) => amm_config.token_0_lp_rate = value,
        Some(3) => amm_config.protocol_fee = value,
        Some(4) => amm_config.creator_fee = value,
        Some(5) => amm_config.disable_create_pool = if value == 0 { false }
else { true },
        _ => return err!(ErrorCode::InvalidInput),
    }
    Ok(())
}
```

There is **no logic** for *setting trade_fee_rate*, *new_owner*, or *protocol_owner*, which are referenced in the comments.

In addition, for example, it states that 0 sets *trade_fee_rate* but actually sets *new_fund_owner*.

Impact:

The protocol may pass wrong *param* values and unintentionally mutate incorrect configuration fields.

Recommendation:

Update the NatSpec documentation to match the actual logic in *update_amm_config*.

[MAGE-06]	The *close* function does not have any restrictions in place	Low

Details:

With the current implementation of the close function, no type is being enforced on the *account_to_close* parameter:

```
/// CHECK: The account to close
   #[account(mut)]
   pub account_to_close: UncheckedAccount<'info>,
```

However, this can prove to be problematic in some scenarios. For example, if we close a PDA for a given pool without first emptying its token vaults, the token balances of that pool will be lost forever.

Impact:

In certain scenarios funds might become stuck in the protocol.

Recommendation:

Consider adding additional validation logic to the *close* function or limiting its functionality to only certain types of PDAs.

[MAGE-07]	Unused function level variables	Informational

Details:

The following function level variables are not being used and can safely be removed:

withdraw.rs#L125

```
    let user = ctx.accounts.user.key();
```

withdraw.rs#L146-L151

```
  // if user is just inititializing
    let user_last_slot = if user_state.last_snap_slot == 0 {
        current_snap_slot
    } else {
        user_state.last_snap_slot
    };
```

deposit.rs#L117

```
 let user = ctx.accounts.user.key();
```

deposit.rs#L128-L133

```
  // if user is just inititializing
    let user_last_slot = if user_state.last_snap_slot == 0 {
        current_snap_slot
    } else {
        user_state.last_snap_slot
    };
```

Recommendation:

Remove the unused variables.

[MAGE-08]	The *Fees::trading_fee* function is not being used anywhere	Informational

Details:

The *Fees::trading_fee* function is not being anywhere within the codebase as of its current state. This means that it can safely be removed.

Recommendation:

Consider removing the *Fees::trading_fee* function.

[MAGE-09]	Typos	Informational

Details:

During our audit of the codebase, we came across some typos in the code comments, that are worth mentioning:

deposit.rs#L45

tokan → token

```
/// user lp tokan account     #[account(mut,  token::authority = user)]
    pub user_lp_token: Box<InterfaceAccount<'info, TokenAccount>>,
```

initialize.rs#L75

must smaller than → must be smaller than

```
    /// Token_0 mint, the key must smaller then token_1 mint.
    #[account(
        constraint = token_0_mint.key() < token_1_mint.key(),
        mint::token_program = token_0_program,
    )]
    pub token_0_mint: Box<InterfaceAccount<'info, Mint>>,
```

Recommendation:

Fix the above mentioned typos.

[MAGE-10]	Redundant functions	Informational

Details:

The current implementations of both the *Fees::fund_fee* and *Fees::protocol_fee* functions hold the exact same functionality as one another. This is redundant and can be avoided by creating a function with a more generalized name that is going to be used in the place of those two.

```
  /// Calculate the owner trading fee in trading tokens
    pub fn protocol_fee(amount: u128, protocol_fee_rate: u64) ->
Option<u128> {
        floor_div(
            amount,
            u128::from(protocol_fee_rate),
            u128::from(FEE_RATE_DENOMINATOR_VALUE),
        )
    }

    /// Calculate the owner trading fee in trading tokens
    pub fn fund_fee(amount: u128, fund_fee_rate: u64) -> Option<u128> {
        floor_div(
            amount,
            u128::from(fund_fee_rate),
            u128::from(FEE_RATE_DENOMINATOR_VALUE),
        )
    }
```

Recommendation:

Consider creating a function with a more generalized name with the same functionality as *Fees::fund_fee* and *Fees::protocol_fee*, and use it in the places where those two are currently being used.

Overall Assessment

After a detailed and thorough security review, our researchers did not identify any vulnerabilities of critical or high severity.

MageLabs DEX is, to the best of our knowledge, safe to use.

Severity	Count	Acknowledged
Critical	0	-
High	0	-
Medium	2	Acknowledged
Low	4	Acknowledged
Informational	4	Acknowledged

Recommendations

Audita Security has put forward the following recommendations for MageLabs DEX:

- Increase *OBSERVATION_NUM* to 150–200, allowing the system to retain a broader historical window without overwriting data too quickly.
- Reduce the *OBSERVATION_UPDATE_DURATION_DEFAULT* to around 5 seconds to align better with Solana's block time and capture more granular price data.
- Fix incomplete access control between admin and owner by either:
    - Implementing the intended access control, or
    - Removing inaccurate comments and dead code
- Add *require*! checks to enforce per-parameter upper bounds:

```
require!(value <= FEE_RATE_DENOMINATOR_VALUE, ErrorCode::FeeTooHigh);
```

- Update the NatSpec documentation to match the actual logic in *update_amm_config*.
- Remove unused variables.
- Consider removing the *Fees::trading_fee* function.
- Fix the mentioned typos.
- Consider creating a function with a more generalized name with the same functionality as *Fees::fund_fee* and *Fees::protocol_fee*, and use it in the places where those two are currently being used.

Disclaimer

This audit makes no statements or warranties on the security of the code. While we have conducted the analysis to our best abilities and produced this report in line with latest industry developments, it is important to not rely on this report only. In order for contracts to be considered as safe as possible, the industry standard requires them to be checked by several independent auditing bodies. Those can be other audit firms or public bounty programs.

Smart contract platforms, their programming languages, and other software components are not immune to vulnerabilities that can be exploited by hackers. As a result, although a smart contract audit can help identify potential security issues, it cannot provide an absolute guarantee of the audited smart contract's security.