Raydium Constant Product AMM Program Audit Report

25.03.2024

– Mad Shield

Table of Contents

1. Introduction

Raydium engaged Mad Shield to audit the Constant Product AMM program. The audit is a full analysis of the core functionality of the program. The AMM is an improvement over the original Raydium AMM implementation, however it eliminates the functionality related to OpenBook while adding support for market making and trading of SPL-Token-22 pairs.

In summary, our audit yielded 3 vulnerabilities within the Constant Product AMM program: 2 categorized as Critical and 1 High-Severity. Additionally, we reported 1 Critical finding in the Token-22 program that may impact any program interacting with the Token-22 program, specifically ones using tokens with the `TransferFeeConfig` extension. A detailed walk-through of these findings is provided in Section [4].

We communicated all our findings with proper mitigations through private channels to the Raydium developer team. **We are glad to confirm that all the vulnerabilities disclosed here have since been addressed and resolved accordingly.** This document outlines the audit procedure, the assessment methodology, key findings and suggestions to further improve the program.

1.1 Overview

The Raydium AMM program, is an implementation of a constant product formulae market that is the latest release of the Raydium AMM decentralized exchange (DEX). The primary features that differentiate this release from the previous versions are as follows:

- **Support for SPL-Token-22** – As Token-22 tokens are gaining traction on the Solana blockchain, the Raydium AMM now allows users to create markets to trade these pairs alongside the canonical Token program.

- **Removing the OpenBook orderbook integration** – The program eliminates the OpenBook market making component, hence, clarifying the code to be more concise while preserving the core functionality of the program.

Raydium proactively engaged Mad Shield for periodic security audits after each major development milestone. This report presents a comprehensive analysis of the implemented changes and their security implications, focusing on identifying and mitigating program vulnerabilities.

1.2 System Specification

A complete review of the constant product AMM logic used by Raydium is provided in our previous report [1]. Here, we only enumerate the most recent changes that are relevant to the context of this audit.

- The set of Token-22 extensions currently supported by the program are `TransferFeeConfig`, `MetadataPointer`, and `TokenMetadata`. A full set of the Token-22 extensions and their description are provided in [2].

- The program is rewritten in Anchor, the standard programming framework for developing programs on Solana, migrating from the vanilla Solana used by the original implementation.

2. Scope and Objectives

The primary objectives of the audit are defined as:

- Minimizing the possible presence of any critical vulnerabilities in the program. This would include detailed examination of the code and edge case scrutinization to find as many vulnerabilities.

- 2-way communication during the audit process. This included for Mad Shield to reach a perfect understanding of the design of Token Metadata and the goals of the Raydium team.

- Provide clear and thorough explanations of all vulnerabilities discovered during the process with potential suggestions and recommendations for fixes and code improvements.

- Clear attention to the documentation of the vulnerabilities with an eventual publication of a comprehensive audit report to the public audience for all stakeholders to understand the security status of the programs.

Raydium has delivered these programs to Mad Shield at the following Github repositories.

Repository URL	https://github.com/raydium-io/raydium-cp-swap
Commit (start of audit)	0d56d2cbde2e6215fdfefdbfb13fb92b32479a55
Commit (end of audit)	8f10f251da354f14dc8e317f1fea1bbc78d9c19b

3. Methodology

After the initial request from Raydium to review we did a quick read-through of the code to evaluate the scope of the work and recognize early potential footguns.

As our team was familiar with the legacy Raydium AMM, we were quick to verify the business logic of the program. The code is a pleasant read, much simpler, less redundant and cleaner to follow compared to the legacy implementation due to the use of the Anchor framework. In addition, the removal of the market making strategy on Serum's orderbook has significantly reduced code complexity.

During this audit, for the first time, we made use of our integrated testing suite which is currently in early stages of development to check the account constraints, account substitution and other preliminary implementation bugs. This technique turned out to be fruitful as we unveiled many bugs such as **[MAD_RAY_02]** and **[MAD_RAY_03]**. Interestingly, these were not discovered during our manual overview showcasing the potential of our testing suite.

As the support for Token-22 was the core feature added to the AMM logic, we did an overview of that program's implementation to assess the composability of the AMM with the new Token program. CP Swap introduces some changes to adapt the constant product formulae with the `TransferFeeConfig` extension (The only supported extension with effects on the swap logic) where transfers impose an additional fee on token amounts. During this evaluation we discovered an issue within Token-22 itself where the calculation of the fee and its inverse was incorrect as explained by **[MAD_T22_01].**

A final manual review revealed residual inconsistencies in the program both impacting the execution and data keeping of the program. The first issue was regarding the rounding of numerator and denominator result of a division arithmetic described in **[MAD_RAY_04]** and an improvement to avoid discrepancy in the data monitored internally by the Raydium team **[MAD_RAY_IMPR_01]**.

4. Findings & Recommendations

Our severity classification system adheres to the criteria outlined here.

Severity Level	Exploitability	Potential Impact	Examples
Critical	Low to moderate difficulty, 3rd-party attacker	Irreparable financial harm	Direct theft of funds, permanent freezing of tokens/NFTs
High	High difficulty, external attacker or specific user interactions	Recoverable financial harm	Temporary freezing of assets
Medium	Unexpected behavior, potential for misuse	Limited to no financial harm, non-critical disruption	Escalation of non-sensitive privilege, program malfunctions, inefficient execution
Low	Implementation variance, uncommon scenarios	Zero financial implications, minor inconvenience	Program crashes in rare situations, parameter adjustments by authorized entities
Informational	N/A	Recommendations for improvement	Design enhancements, best practices, usability suggestions

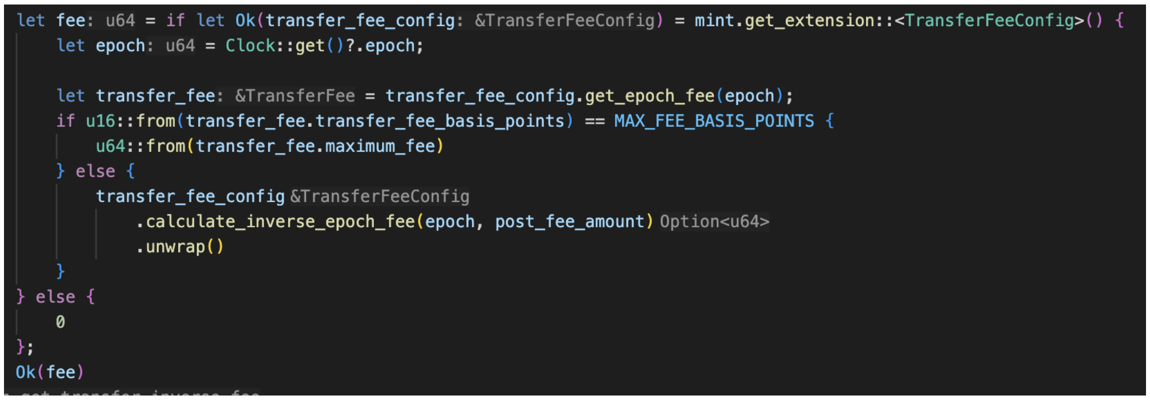
In the following, we enumerate some of the findings and issues we discovered and explain their implications and corresponding resolutions.

Finding	Description	Severity Level
MAD_T22_01 [RESOLVED]	Calculating inverse fee doesn't properly account for mints with `ONE_IN_BASIS_POINTS` fee rate and `max_fee` for SPL-Token-22	*Critical*
MAD_RAY_02 [RESOLVED]	The pool account could not be initialized due to the stack limits of the Solana runtime.	*Critical*
MAD_RAY_03 [RESOLVED]	Wrong evaluation of pool vaults dyadic relationship with input token accounts in swap instructions	*Critical*
MAD_RAY_04 [RESOLVED]	Misplaced rounding for division computation in the swap calculation	*High*
MAD_RAY_IMPR_01 [ACKNOWLEDGED]	Emitting the wrong `token_amount` in the swap event results in logging incorrect swap data.	*Improvement*

1. **MAD_T22_01 - Calculating inverse fee doesn't account for mints with ONE_IN_BASIS_POINT fee-rate and max_fee for SPL-Token-22[Critical]**

   To support swaps of Token-22 mints with `TransferFeeConfig` the program calculates and deducts the fee from the amount passed into the swap function to give users a fair price and make sure the constant is always increasing. It then recalculates the required fees of the `TransferFeeConfig` to update these amounts accordingly.

   Token-22 itself has a utility function to calculate_inverse_fee_amount that upon inspection was revealed to incorrectly return ZERO for tokens with fee_rate of ONE_IN_BASIS_POINTS. However, if the token has a max_fee set, this is not always true and results in no tokens to be transferred instead of re-upping it with the max_fee.



```
let fee: u64 = if let Ok(transfer_fee_config: &TransferFeeConfig) = mint.get_extension::<TransferFeeConfig>() {
    let epoch: u64 = Clock::get()?.epoch;

    let transfer_fee: &TransferFee = transfer_fee_config.get_epoch_fee(epoch);
    if u16::from(transfer_fee.transfer_fee_basis_points) == MAX_FEE_BASIS_POINTS {
        u64::from(transfer_fee.maximum_fee)
    } else {
        transfer_fee_config &TransferFeeConfig
            .calculate_inverse_epoch_fee(epoch, post_fee_amount) Option<u64>
            .unwrap()
    }
} else {
    0
};
Ok(fee)
```

Fig 2. The Raydium implementation for calculate_inverse_fee..

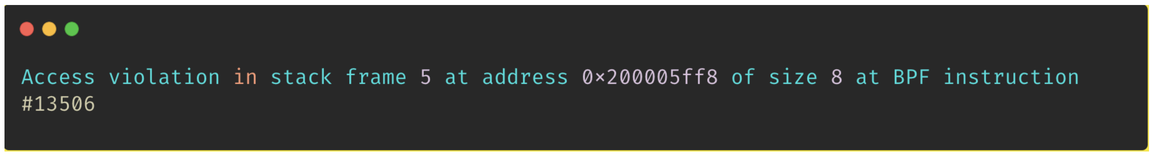
   Recommended Fix

   There is currently an open discussion on how to address this, however, this issue does not affect Raydium as they appropriately do a custom implementation to calculate the inverse_fee as in Fig. 2.

   SPL-Token-22 Issue: [#6451](#6451)

2. **MAD_RAY_02 - The pool account could not be initialized due to the stack limits of the Solana runtime *[Critical]***

   During our intensive testing of the `cp_swap` program, we noticed the initialize instruction uses 5 init constraints in the account context of the instruction. This violates the Solana runtime stack limit as it only supports 4 KB stack size as our tests show below.



```
Access violation in stack frame 5 at address 0x200005ff8 of size 8 at BPF instruction
#13506
```

Fig 1. The error code returned by the initialize instruction.

Recommended Fix

To address the problem, we recommended modifying the `initialize` instruction to remove the init constraint for creating `token_0_vault` and `token_1_vault` and instead creating them inside the instruction itself.

patch commit: [b05d1dd1]()

3. MAD_RAY_03 - Wrong evaluation of pool vaults dyadic relationship with input token accounts in swap instructions [Critical]

The program would take in as input accounts an `input_vault` and an `output_vault` and then compare them to the pool's vault accounts to determine the trade direction. However, it would incorrectly replace the order of the token accounts returned in each scenario. It is always the case that `input_vault` gives the input_amount and the `output_vault` gives the `output_amount`. The condition should only determine the `trade_direction`.

This was revealed during our fuzzy testing, where the swap would fail as the constant product increasing invariant was violated.

Recommended Fix

The issue is fixed by interchanging the token_amounts returned in the code block that determines the trade_direction. Ideally this will be refactored to be a match statement where both readability and reliability of the code increases. This is currently under work by the Raydium team but our testing shows that the code operates correctly as it stands.

The patch commit: [0f4d5270]()

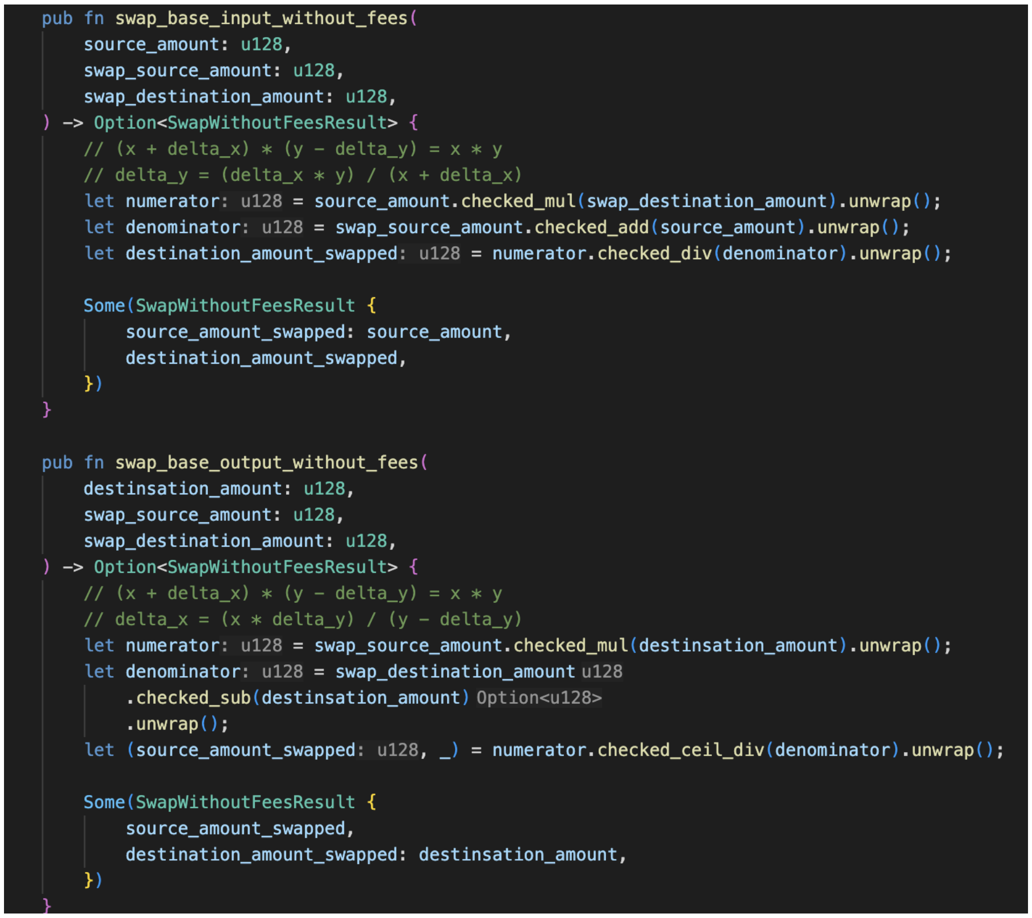
4. MAD_RAY_04 - Misplaced rounding for division computation in the swap calculation [High]

The instruction for the `swap_amount_input` used a misplaced rounding calculation. For swaps of `base_input_amount`, the division function would use a `checked_ceil_div` function that rounded down the `swap_src` and `swap_dest` amounts resulting in less than optimized trades as the user would receive fewer output tokens while transferring fewer input tokens. This rounding works for `swap_amount_output` though as the new `swap_dest` is rounded down and the new `swap_src` is rounded up, thus, achieving the desiderata.

Recommended Fix

We recommended breaking down the calculations for each swap type into separate corresponding procedures as shown in Fig. 4.

The patch commits: 707cd6ec ,



```
pub fn swap_base_input_without_fees(
    source_amount: u128,
    swap_source_amount: u128,
    swap_destination_amount: u128,
) -> Option<SwapWithoutFeesResult> {
    // (x + delta_x) * (y - delta_y) = x * y
    // delta_y = (delta_x * y) / (x + delta_x)
    let numerator: u128 = source_amount.checked_mul(swap_destination_amount).unwrap();
    let denominator: u128 = swap_source_amount.checked_add(source_amount).unwrap();
    let destination_amount_swapped: u128 = numerator.checked_div(denominator).unwrap();

    Some(SwapWithoutFeesResult {
        source_amount_swapped: source_amount,
        destination_amount_swapped,
    })
}

pub fn swap_base_output_without_fees(
    destinsation_amount: u128,
    swap_source_amount: u128,
    swap_destination_amount: u128,
) -> Option<SwapWithoutFeesResult> {
    // (x + delta_x) * (y - delta_y) = x * y
    // delta_x = (x * delta_y) / (y - delta_y)
    let numerator: u128 = swap_source_amount.checked_mul(destinsation_amount).unwrap();
    let denominator: u128 = swap_destination_amount u128
        .checked_sub(destinsation_amount) Option<u128>
        .unwrap();
    let (source_amount_swapped: u128, _) = numerator.checked_ceil_div(denominator).unwrap();

    Some(SwapWithoutFeesResult {
        source_amount_swapped,
        destination_amount_swapped: destinsation_amount,
    })
}
```

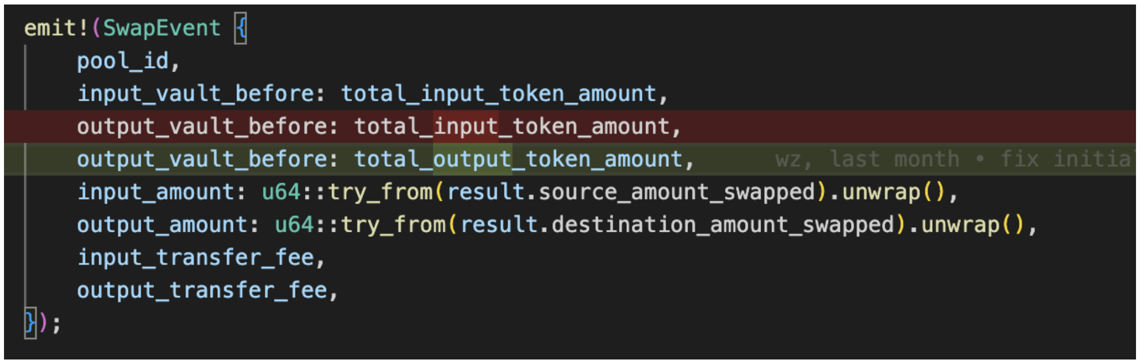
Fig 4. Separation of swap calculation for input vs output base

4. General Recommendations

In this section, we provide general recommendations and informational issues that we found during the process of the audit.

1. **MAD_RAY_IMPR_01 - Emitting the wrong token amount in the swap event results in logging incorrect swap data.**

   The swap instruction emits an event that is used to internally monitor trades by Raydium. We noticed that the event is returning the same amount for both `input_vault_before` and `output_vault_before` as seen in the snippet below.



```
emit!(SwapEvent {
    pool_id,
    input_vault_before: total_input_token_amount,
    output_vault_before: total_input_token_amount,
    output_vault_before: total_output_token_amount,        wz, last month • fix initia
    input_amount: u64::try_from(result.source_amount_swapped).unwrap(),
    output_amount: u64::try_from(result.destination_amount_swapped).unwrap(),
    input_transfer_fee,
    output_transfer_fee,
});
```

Fig 3. The wrong Swap event emitted

Recommended Fix

The fix involved an easy update as shown in Fig .3 that will prevent inaccurate record keeping.

The patch commit: b05d1dd1

Conclusion

In conclusion, Mad Shield's audit of the `cp-swap` program has been a thorough effort to enhance security. The audit underscores our focus on security within the expanding Solana DeFi landscape, emphasizing the ongoing need for security, penetration testing and transparent documentation. We would like to acknowledge the Raydium team for their cooperative approach throughout the process, which has contributed to the overall effectiveness of our collaboration.

References

[1] Raydium AMM Program Audit – https://github.com/madshieldio/Publications/blob/main/Raydium/raydium-amm-v-1.0.0.pdf
[2] Token-22 Extension Guide – https://spl.solana.com/token-2022#extensions