Raydium AMM

Audit

Presented by:

OtterSec contact@osec.io

Maher Azzouzi maher@osec.io

William Wang defund@osec.io

Robert Chen r@osec.io

Contents

Appendices

© 2022 Otter Audits LLC. All Rights Reserved.

01 | Executive Summary

Overview

Raydium engaged OtterSec to perform an assessment of the `raydium-amm` program. This assessment was conducted between October 24th and November 11th, 2022.

After our initial engagement concluded, we performed a number of incremental reviews until Feb 16th, 2022. This included confirming patches, various refactors, and other miscellaneous changes throughout this period prior to the finalization of the program code.

Key Findings

Over the course of this audit engagement, we produced 6 findings total.

In particular, we noted issues with improper liquidity pool initialization (OS-RAY-ADV-00) and stale account information which could lead to the improper settlement of funds (OS-RAY-ADV-01).

We also made suggestions around code hardening (OS-RAY-SUG-01, OS-RAY-SUG-02) and potentially incorrect state transitions (OS-RAY-SUG-00). We also made recommendations around improving PNL precision (OS-RAY-SUG-03).

Overall, we commend the Raydium team for their patience and attention to detail throughout our engagement.

© 2022 Otter Audits LLC. All Rights Reserved.

02 | Scope

The source code was delivered to us in a git repository at github.com/raydium-io/raydium-amm. This audit was performed against commits d70a8fb. We performed additional incremental reviews up to 8da289a.

A brief description of the programs is as follows.

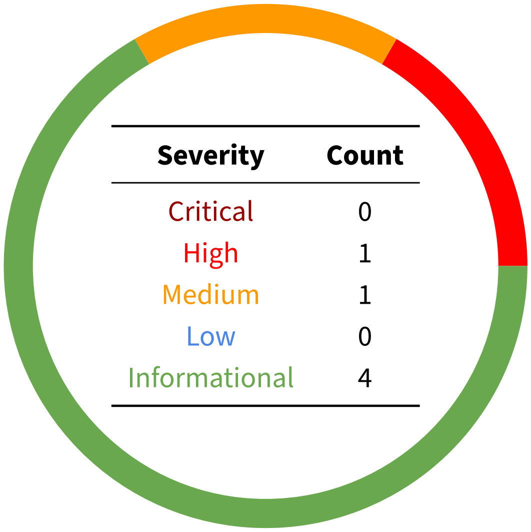
Name	Description
raydium-amm	Constant product automated market maker built on Solana. Raydium also automatically plans, places, and settles orders against an underlying orderbook, allowing for the sharing of liquidity between Raydium and other ecosystem projects. The Raydium AMM supports a variety of modes, including orderbook only, liquidity only, and a mix of both. Orderbook interactions happen through a series of state transitions between planning, placing, and settling orders. This orderbook implementation is currently OpenBook.

© 2022 Otter Audits LLC. All Rights Reserved.

03 | Findings

Overall, we report 6 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 don’t have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	0
High	1
Medium	1
Low	0
Informational	4

© 2022 Otter Audits LLC. All Rights Reserved.

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-RAY-ADV-00	High	Resolved	A malicious liquidity provider has the ability to monopolize the liquidity pool by manipulating the share-to-token ratio.
OS-RAY-ADV-01	Medium	Resolved	The open orders account is not properly refreshed during swap operations, potentially leading to the improper settlement of funds.

© 2022 Otter Audits LLC. All Rights Reserved.

OS-RAY-ADV-00 [high] | Liquidity Pool Monopolization

Description

A malicious early liquidity provider can maliciously raise the LP token ratio, stealing funds from other users who deposit insufficient funds into the pool due to rounding errors.

When a pool is started, an attacker can initialize the pool with a small amount of LP tokens, and then send tokens to the pool to increase the exchange rate. Subsequent deposit operations will then round down the output LP token amount, stealing money due to the rounding behavior of LP token minting.

Alternatively, this could also represent a denial of service scenario due to conditions that prevent minting zero LP tokens.

src/processor.rs RUST

```rust
pub fn process_deposit(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    deposit: DepositInstruction,
) -> ProgramResult {
    ...
    if mint_lp_amount == 0 || deduct_coin_amount == 0 || deduct_pc_amount
    ↪   == 0 {
            return Err(AmmError::InvalidInput.into());
    }
    ...
}
```

Remediation

Similar to what Uniswap does, when providing liquidity to the pool for the very first time, the contract can lock away or burn a fixed number of LP tokens to ensure that the initial ratio will be preserved, even if an attacker directly sends funds to the vaults.

Patch

Resolved in 3dc6a26.

© 2022 Otter Audits LLC. All Rights Reserved.

OS-RAY-ADV-01 [med] | Stale Open Orders Calculations

Description

During swap operations, the Raydium AMM will cancel orders on the orderbook in the direction of the swap in order to free up liquidity for potentially large swaps.

src/processor.rs RUST

```rust
    for ids in amm_order_ids_vec.iter() {
        Invokers::invoke_dex_cancel_orders_by_client_order_ids(
            serum_dex_info.clone(),
            market_info.clone(),
            bids_info.clone(),
            asks_info.clone(),
            open_orders_info.clone(),
            authority_info.clone(),
            event_queue_info.clone(),
            AUTHORITY_AMM,
            amm.nonce as u8,
            *ids,
        )?;
    }
```

However, the subsequent calculation for settlement uses a cached `OpenOrders` account. This account is loaded previously in the instruction, but crucially represents a copy of the actual underlying account data.

src/processor.rs RUST

```rust
    let (market_state, open_orders) = Processor::load_serum_market_order(
        market_info,
        open_orders_info,
        authority_info,
        &amm,
        false,
    )?;
```

This means that the cancellation of orders will not update the calculated free token counts, causing the AMM to incorrectly skip the settlement of funds on large swaps.

© 2022 Otter Audits LLC. All Rights Reserved.

src/processor.rs RUST

```rust
    if swap_amount_out > token_pc.amount
        && swap_amount_out
            <= token_pc
                .amount
                .checked_add(open_orders.native_pc_free)
                .unwrap()
    {
```

As a result, large swaps will fail. More precisely, if the amount of funds in the AMM's reserves are insufficient to satisfy the swap, the swap will fail.

Remediation

Consider either refreshing the open orders account to properly load the native pc and coin counts.

Alternatively, remove this check to attempt settlement of funds regardless of the balance in the open orders account. Note that assuming the AMM invariant holds, this check should always be true.

Patch

Resolved in 0b25381.

© 2022 Otter Audits LLC. All Rights Reserved.

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 antipatterns and could lead to security issues in the future.

ID	Description
OS-RAY-SUG-00	Consider stricter admin-based state transitions to mitigate against centralization risk.
OS-RAY-SUG-01	When performing typecasting, prefer the use of `try_from` over `as`.
OS-RAY-SUG-02	Arithmetic overflow checks are not enabled by default in release builds.
OS-RAY-SUG-03	Add auxiliary fields to improve the precision of fee calculations.

© 2022 Otter Audits LLC. All Rights Reserved.

OS-RAY-SUG-00 | Stricter State Transitions

Description

Raydium implements a complex state machine to facilitate placing liquidity on the orderbook. The AMM status can also be set to various different values, which determines what sorts of operations are allowed.

processor.rs RUST

```rust
pub enum AmmStatus {
    Uninitialized = 0u64,
    Initialized = 1u64,
    Disabled = 2u64,
    WithdrawOnly = 3u64,
    // pool only can add or remove liquidity, can't swap and plan orders
    LiquidityOnly = 4u64,
    // pool only can add or remove liquidity and plan orders, can't swap
    OrderBookOnly = 5u64,
    // pool only can add or remove liquidity and swap, can't plan orders
    SwapOnly = 6u64,
    // transfer user swap_in token to the pool token vault as punishment
     ↪  before the ido pool open period through swap instruction
    SwapPunish = 7u64,
}
```

For example, AMMs placed in the `LiquidityOnly` state can no longer interact with the orderbook.

This state can be set via the admin controlled `process_set_params`.

processor.rs RUST

```rust
    match AmmParams::from_u64(param as u64) {
        AmmParams::Status => {
            let value = match setparams.value {
                Some(a) => a,
                None => return Err(AmmError::InvalidInput.into()),
            };
            if AmmStatus::valid_status(value) {
                amm.status = value as u64;
                set_valid = true;
            }
        }
```

© 2022 Otter Audits LLC. All Rights Reserved.

However, this setting does not place any restrictions on where the AMM is in the state machine. More specifically, if the AMM still has orders on the orderbook while this state transition is performed, these orders will no longer be factored into the liquidity calculations.

processor.rs RUST

```rust
    if AmmStatus::from_u64(amm.status).orderbook_permission() {
        enable_orderbook = true;
    } else {
        enable_orderbook = false;
    }
```

This means that stale orders could violate the critical constant product invariant, leading to a loss of funds. As an additional complexity, the reset flag will be set, meaning that such stale orders need to be filled prior to the cranking.

In practice, it is likely that this is only exploitable if the admin key is compromised. Hence, we rate the severity as a suggestion.

Remediation

Perform proper state transition checks before allowing for a transition from orderbook enabled to disabled.

Patch

Resolved in 33bc7a4.

OS-RAY-SUG-01 | Prevent Unsafe Type Casting

Description

Throughout the codebase, there are certain instances of unchecked typecasting. However, this operation does not check for truncation, which could lead to unexpected behavior if the source numerical type doesn't fit in the destination.

As an example, the below code is responsible for updating the calculated PNL in the deposit instruction and performs an unchecked cast.

src/processor.rs.rs RUST

```rust
pub fn process_deposit(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    deposit: DepositInstruction,
) -> ProgramResult {
    ...
    target_orders.calc_pnl_y = (after_k2
        .checked_mul(after_y.as_u128().into())
        .unwrap()
        .checked_div(after_x.as_u128().into())
        .unwrap()
        .as_u128())
    .integer_sqrt() as u64;
    target_orders.calc_pnl_x = after_k2
        .as_u128()
        .checked_div(target_orders.calc_pnl_y as u128)
        .unwrap() as u64;
    ...
}
```

This particular operation is unlikely to overflow because that would imply a PNL which is larger than the total token supply. However, as a best practice, it could make sense to avoid such potentially unsafe operations to harden the codebase and improve readability.

Note that this is a general pattern throughout the codebase, and we recommend remediating all instances.

Remediation

As one potential solution, consider using `try_from` over `as` and explicitly unwrap the result to generate a panic on truncation.

© 2022 Otter Audits LLC. All Rights Reserved.

Alternatively, consider changing the type of the calculated PNL to u128 to avoid any truncation operations. This would involve more refactoring throughout the codebase, but would also be more resilient to changes.

Patch

Resolved in 6d64388.

© 2022 Otter Audits LLC. All Rights Reserved.

OS-RAY-SUG-02 | Arithmetic Overflow Checks

Description

Currently, there exists usage of unchecked numerical operations throughout the codebase. While there does not appear to be any immediate impact, it would be safer to mitigate against these overflows at the compiler level by enabling overflow checks.

Remediation

Consider appending the following in `Cargo.toml`:

```TOML
[profile.release]
overflow-checks = true
```

This will enable integer overflow checks by default in release builds.

Patch

Resolved in f323096.

© 2022 Otter Audits LLC. All Rights Reserved.

OS-RAY-SUG-03 | Improve Fee Precision

Description

Fee calculations are tracked via the `need_take_pnl_pc` and `need_take_pnl_coin` fields. These fields are accrued via `calc_take_pnl`, which is invoked during deposits and withdrawals.

This function looks at the accrual of the constant product invariant and takes a percentage of that as the fee. Unfortunately, due to the limited precision of the numerical type, these calculations may truncate more often than expected, leading to lost profits.

processor.rs RUST

```rust
        let coin_pnl_amount = diff_coin_pnl_amount
            .checked_mul(amm.fees.pnl_numerator.into())
            .unwrap()
            .checked_div(amm.fees.pnl_denominator.into())
            .unwrap()
```

Remediation

One possible solution would be to round up the accrued PNL upon each calculation. This would bias the calculations in favor of the pool admin, harming the users. However, this could also lead to greatly more fees collected than expected.

Alternatively, consider accruing the unused differences in new fields on `AmmInfo` to ensure no loss of precision.

© 2022 Otter Audits LLC. All Rights Reserved.

A | 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 implementation of the program 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 sum, 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 get a comprehensive understanding of 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 the other 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.

© 2022 Otter Audits LLC. All Rights Reserved.

B | Vulnerability Rating Scale

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

Critical

Vulnerabilities that immediately lead to 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 could lead to 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 could lead to denial of service scenarios or degraded usability.

Examples:

- Malicious input that causes computational limit exhaustion
- Forced exceptions in normal user flow

Low

Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions

Informational

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

Examples:

- Explicit assertion of critical internal invariants
- Improved input validation