Raydium AMM v3

Audit

Presented by:

OtterSec contact@osec.io

Maher Azzouzi maher@osec.io

William Wang defund@osec.io

Michal Bochnak embe221ed@osec.io

Robert Chen notdeghost@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Raydium engaged OtterSec to perform an assessment of the `raydium-amm-v3` program. This assessment was conducted between September 25th and December 9th, 2022. For more information on our auditing methodology, see Appendix B.

Vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team over to streamline patches and confirm remediation.

We delivered final confirmation of the patches December 30th, 2022.

Key Findings

Over the course of this audit engagement, we produced 12 findings total.

In particular, we have found two issues that could lead to loss of funds related to unchecked type castings (OS-RAY-ADV-00) and closing personal positions (OS-RAY-ADV-01). We also found a number of other issues such as missing account checks on the AMM Config (OS-RAY-ADV-02), incorrectly closed accounts (OS-RAY-ADV-03), and a denial of service related to reward state calculations (OS-RAY-ADV-05).

We also made a number of recommendations around improving code quality and resilience, including arithmetic overflow checks (OS-RAY-SUG-01, better initialization sanity checks (OS-RAY-SUG-02, OS-RAY-SUG-03, OS-RAY-SUG-00), and more.

Overall, we commend the Raydium team for being responsive and knowledgeable throughout the audit.

02 | Scope

The source code was delivered to us in a git repository at github.com/raydium-io/raydium-amm-v3. This audit was performed against commit `4fe73c7`.

There was a total of 1 program included in this audit. A brief description of the program is as follows.

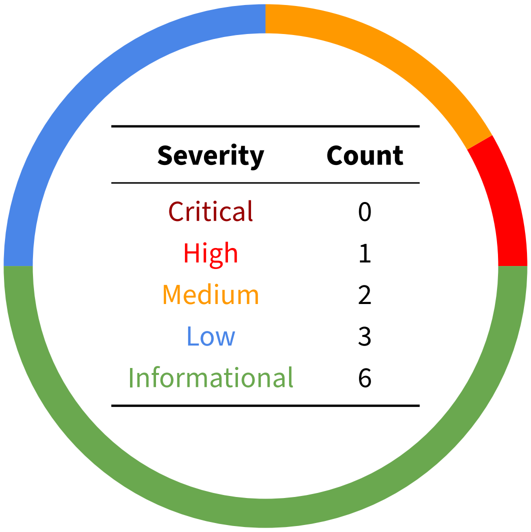
Name	Description
raydium-amm-v3	Concentrated liquidity market maker program.

03 | Findings

Overall, we report 12 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.

The below chart displays the findings by severity.



Severity	Count
Critical	0
High	1
Medium	2
Low	3
Informational	6

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 A.

ID	Severity	Status	Description
OS-RAY-ADV-00	High	Resolved	Liquidity math performed an unchecked cast from u64 to i64, which is unsafe.
OS-RAY-ADV-01	Medium	Resolved	Any user can close a personal position and steal the lamports used for rent, since NFT ownership is not checked.
OS-RAY-ADV-02	Medium	Resolved	It is possible to pass arbitrary AMM Config to SwapRouterBaseIn instruction
OS-RAY-ADV-03	Low	Resolved	The custom implementation for closing accounts is not safe as it does not overwrite the Anchor discriminator.
OS-RAY-ADV-04	Low	Resolved	Invoking admin-gated instructions which close active pools, protocol positions, or tick arrays might lead to an invalid state.
OS-RAY-ADV-05	Low	Resolved	Incorrect rounding direction leads to potential denial of service in reward calculations

OS-RAY-ADV-00 [high] [resolved] | Unchecked Type Casting

Description

In the `get_delta_amount_0_signed` function, there are unchecked conversions from u64 (which `get_delta_amount_0_unsigned` returns) to i64. The issue is that the value of the u64 returned by `get_delta_amount_0_unsigned` might be larger than what i64 can represent, which would result in a faulty conversion.

src/libraries/liquidity_math.rs RUST

```rust
pub fn get_delta_amount_0_signed(
    sqrt_ratio_a_x64: u128,
    sqrt_ratio_b_x64: u128,
    liquidity: i128,
) -> i64 {
    if liquidity < 0 {
        -(get_delta_amount_0_unsigned(
            sqrt_ratio_a_x64,
            sqrt_ratio_b_x64,
            -liquidity as u128,
            false,
        ) as i64)
    } else {
        // TODO check overflow, since i64::MAX < u64::MAX
        get_delta_amount_0_unsigned(sqrt_ratio_a_x64, sqrt_ratio_b_x64,
    ↪   liquidity as u128, true)
            as i64
    }
}
```

The same applies for `get_delta_amount_1_signed`.

We prepared a simple proof of concept test case demonstrating that this could be exploited against the burn liquidity instruction.

BASH

```bash
amount_0: 11760396439755885211, amount_1: 335938481032035582
real profit = amount_0 - (amount_0_add * 40) = 2532157683947193710
test
    ↪   instructions::decrease_liquidity::burn_liquidity_test::burn_liquidity_test_overf
    ↪   ... ok
```

Remediation

A possible method of remediation is using `i64::try_from` instead of an unchecked cast.

Patch

Fixed in #27.

OS-RAY-ADV-01 [med] [resolved] | Closing Personal Positions Is Not Gated

Description

When a personal position is created, the program mints an NFT to the user's wallet. Subsequent instructions which require authorization of the position, such as increasing and decreasing liquidity, use the `is_authorized_for_token` function to check that the signer holds the NFT.

The `ClosePosition` instruction should also be privileged, but it does not check NFT ownership. An attacker can create their own empty NFT account, thus spoofing NFT ownership. This allows them to harvest the lamports used for rent.

Remediation

In order for the issue to be remediated, the `ClosePosition` instruction should verify that the position NFT account, which should hold the NFT, is non-empty. This can be done with `is_authorized_for_token` or, as shown below, with an Anchor constraint.

src/instructions/close_position.rs RUST

```rust
#[account(
       mut,
       associated_token::mint = position_nft_mint,
       associated_token::authority = nft_owner,
       constraint = position_nft_account.amount == 1
   )]
   pub position_nft_account: Box<Account<'info, TokenAccount>>,
```

Patch

Fixed in #26.

OS-RAY-ADV-02 [med] [resolved] | Arbitrary AMM Config Possible Usage

Description

Every `PoolState` is created using one of existing AMM configs. The `AmmConfig` structure is used in `swap_internal` function to determine `trade_fee_rate`, `protocol_fee_rate` and `fund_fee_rate` that is used in the current pool. In Swap instruction, there are implemented anchor checks that validate the given `AmmConfig` to be the one that `PoolState` was initialised with. However, the instruction `SwapRouterBaseIn` doesn't implement those checks. The lack of checks makes it possible to pass any AMM Config to the `UwapRouterBaseIn` and as a result to manipulate the fee value.

Remediation

In order for the issue to be remediated, the `SwapRouterBaseIn` instruction should verify that the `amm_config`, which was passed to the instruction, is the one assigned to the `PoolState`. This can be done by adding the same check that is implemented for Swap instruction.

src/instructions/swap_router_base_in.rs RUST

```rust
require!(pool_state.amm_config == amm_config.key());
```

Patch

Fixed in #35.

OS-RAY-ADV-03 [low] [resolved] | Accounts Are Incorrectly Closed

Description

The `close_account` function is used by the program to close Solana accounts. It does so by transferring any remaining lamports to another address. The issue is that Solana does not guarantee that the account will be immediately purged. Theoretically, closed accounts may remain accessible for a short period of time, which poses a security risk.

Remediation

To remediate this issue, use Anchor's `close` constraint, which, in addition to transferring lamports, disables the account by overwriting the account's discriminator.

src/instructions/close_position.rs RUST

```rust
#[account(
    mut,
    seeds = [POSITION_SEED.as_bytes(), position_nft_mint.key().as_ref()],
    bump,
    close = nft_owner
)]
pub personal_position: Box<Account<'info, PersonalPositionState>>,
```

Patch

Fixed in #26.

OS-RAY-ADV-04 [low] [resolved] | Closing Active Accounts Is Dangerous

Description

Listed below are admin-gated instructions which will close accounts, even if they are active:

- `ClosePersonalPosition`
- `ClosePool`
- `CloseProtocolPosition`
- `CloseTickArray`

This is problematic, as there may still be outstanding accounts which point to the now-empty address. For example, suppose the admin closes a pool, even though there is still liquidity stored in personal position accounts. If someone recreates the pool at the same address, the personal positions will appear valid, even though there is zero liquidity.

Remediation

To remediate this issue, it should be ensured that the admin is not able to close accounts arbitrarily.

Patch

Fixed in #26.

OS-RAY-ADV-05 [low] [resolved] | Rewards State DOS

Description

Raydium tracks the total amount of rewards which are supposed to be emitted via `reward_total_emissioned`. This field is then subtracted from when claiming rewards.

Unfortunately, because of small differences in rounding, it is possible to create a discrepancy between the expected rewards emissions and the actual total amount.

More specifically, because `reward_total_emissioned` is updated every time rewards information is updated, but individual user reward emission calculations are only tracked in aggregate, more rewards could be lost due to rounding in the former scenario.

Proof of Concept

1. Let's assume following setup:
    - `open_time = 1665982800` (epoch in seconds),
    - `end_time = open_time + (90 * 24 * 60 * 60)` (epoch in seconds, 90 days),
    - `emissions_per_second = 80_000 / (90 * 24 * 60 * 60) = 0.102`,
    - `pool_state.liquidity = 100`
2. If in `curr_timestamp = open_time + 60` the user runs `PoolState::update_reward_infos()` the `reward_infos` will be updated with `reward_0_total_emissioned = 0`
3. It is possible to update reward infos to the state in which the `reward_growth_global_x64 > 0 && reward_x_total_emissioned == 0`
4. By running `update_reward_infos` instruction every 1 minute (60 seconds), with following setup, it is possible to keep `PoolState::reward_x_total_emissioned` at value of 0
5. If `reward_growth_global_x64` will become big enough it is possible for `reward_amount_owed` of an user to be greater than 0
6. If `personal_position.reward_amount_owed > 0 && reward_x_total_emissioned = 0` the check in `decrease_liquidity`: `check_unclaimed_reward` will panic and make it impossible to decrease the liquidity
7. That can result in higher probability of more users gaining Impermanent Loss
8. Even if `PoolState::reward_x_total_emissioned` will be greater than 0 not all users will be able to decrease liquidity and gain a reward

Remediation

Consider changing the code of `check_unclaimed_reward` to `get_unclaimed_reward` or replacing `mul_div_floor()` with `mul_div_ceil` when calculating `reward_total_emissioned` *src/states/pool.rs* RUST

```rust
pub fn get_unclaimed_reward(&self, index: usize, reward_amount_owed: u64)
    ↪  -> u64 {
    assert!(index < REWARD_NUM);
    let unclaimed_reward = self.reward_infos[index]
        .reward_total_emissioned
        .saturating_sub(self.reward_infos[index].reward_claimed)
        .unwrap();
    unclaimed_reward
}
```

Patch

Fixed in #35.

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 do represent antipatterns and could introduce a vulnerability in the future.

ID	Description
OS-RAY-SUG-00	Missing sanity checks on critical admin functionality, such as updating configuration and price
OS-RAY-SUG-01	Arithmetic overflow checks are not enabled by default in release builds.
OS-RAY-SUG-02	When creating a pool, consider explicitly initializing all fields of the struct, even if they will be set to zero.
OS-RAY-SUG-03	The program detects initialization by checking whether an account's bump is zero, which is not strictly correct.
OS-RAY-SUG-04	The token authority should be allowed to increase and decrease liquidity.
OS-RAY-SUG-05	The list of possible reward funders is inconsistent

OS-RAY-SUG-00 [resolved] | Admin Sanity Checks

Description

The `CreateAMMConfig` configuration accepts arbitrary values for parameters which should be bounded in practice, e.g. fee rates and tick spacing. Strict checks would make the protocol more robust to accidental misuse.

src/instructions/admin/create_amm_config.rs RUST

```rust
pub fn create_amm_config(
    ctx: Context<CreateAmmConfig>,
    index: u16,
    tick_spacing: u16,
    protocol_fee_rate: u32,
    trade_fee_rate: u32,
) -> Result<()> {
    let amm_config = ctx.accounts.amm_config.deref_mut();
    amm_config.owner = ctx.accounts.owner.key();
    amm_config.bump = *ctx.bumps.get("amm_config").unwrap();
    amm_config.index = index;
    amm_config.protocol_fee_rate = protocol_fee_rate;
    amm_config.trade_fee_rate = trade_fee_rate;
    amm_config.tick_spacing = tick_spacing;

    emit!(CreateConfigEvent {
        index: amm_config.index,
        owner: ctx.accounts.owner.key(),
        protocol_fee_rate: amm_config.protocol_fee_rate,
        trade_fee_rate: amm_config.trade_fee_rate,
        tick_spacing: amm_config.tick_spacing,
    });

    Ok(())
}
```

A similar issue affects the `admin_reset_sqrt_price_instr` instruction.

Remediation

During critical admin operations, consider enforcing restrictions to ensure sane parameters.

Patch

Fixed in bbf8f40 and aec59eb.

OS-RAY-SUG-01 [resolved] | Arithmetic Overflow Checks

Description

Present in the code are some unchecked additions and subtractions. These may be secure due to the nature of the arguments, but it would be safer to mitigate against overflows by checking these operations regardless.

Remediation

Consider appending the following in `Cargo.toml`:

TOML

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

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

Patch

Fixed in #26.

OS-RAY-SUG-02 [resolved] | Explicitly Initialize All Pool State Fields

Description

The CreatePool instruction only initializes certain fields of the PoolState struct, implicitly setting the rest to zero. It would be clearer if these assignments were made explicit.

Remediation

Explicitly initialize all fields of the PoolState struct.

Patch

Fixed in #26.

OS-RAY-SUG-03 [resolved] | Ineffective Initialization Check

Description

In the OpenPosition instruction, the program will initialize the protocol position if it does not already exist. If the bump field is zero, it assumes the protocol position is uninitialized. The issue is that an initialized account's bump can legitimately be zero — hence this check is ineffective.

src/instructions/open_position.rs RUST

```rust
// check if protocol position is initilized
if ctx.accounts.protocol_position.bump == 0 {
    let protocol_position = &mut ctx.accounts.protocol_position;
    protocol_position.bump = *ctx.bumps.get("protocol_position").unwrap();
    protocol_position.pool_id = ctx.accounts.pool_state.key();
}
```

Remediation

A better method for detecting initialization is checking whether the pool_id field is all-zeros, i.e. the default public key.

src/instructions/open_position.rs RUST

```rust
// check if protocol position is initilized
if ctx.accounts.protocol_position.pool_id == Pubkey::default() {
    ...
}
```

Patch

Fixed in a768d0f.

OS-RAY-SUG-04 | Allow Token Authority To Modify The Position

Description

The function `is_authorized_for_token` treats only `token_account.owner` as an user authorized for the token. However, according to the comment above that function and our understanding, it should also include `token_account.delegate`.

Remediation

Consider adding second possible case to the list of conditions

src/util/access_control.rs RUST

```rust
pub fn is_authorized_for_token<'info>(
    signer: &Signer<'info>,
    token_account: &Box<Account<'info, TokenAccount>>,
) -> Result<()> {
  let delegate: COption<Pubkey> = token_account.delegate;
  if delegate.contains(&signer.key()) {
      require!(
          token_account.amount == 1 && token_account.delegated_amount == 1,
          ErrorCode::NotApproved
      )
    }
  require!(
      token_account.amount == 1 && token_account.owner == signer.key(),
      ErrorCode::NotApproved
  );
  Ok(())
}
```

This will include `token_account.delegate` in the list of users authorized for the token.

Patch

Fixed in #XXX.

OS-RAY-SUG-05 | Inconsistent List Of Possible Reward Funders

Description

The InitializeReward instruction requires the reward_funder to be one of those users:

RUST

```
require!(
    ctx.accounts.reward_funder.key() == ctx.accounts.amm_config.owner
        || ctx.accounts.reward_funder.key() == crate::admin::id()
        || ctx.accounts.reward_funder.key() ==
    ↪   ctx.accounts.pool_state.load()?.owner
        ||
    ↪   operation_state.validate_operation_owner(ctx.accounts.reward_funder.key()),
    ErrorCode::NotApproved
);
```

After the initial check, another one is performed if the last reward is being initialized:

RUST

```
else if lowest_index == REWARD_NUM - 1 {
    // the last reward token must be controled by the admin
    require!(
        *authority == *amm_config_owner
            || *authority == crate::admin::id()
            || operation_state.validate_operation_owner(*authority),
        ErrorCode::NotApproved
    );
}
```

However, the CollectRemainingRewards instruction allows only the pool_state.owner to collect the remaining rewards

RUST

```
require_keys_eq!(reward_funder.key(), pool_state.owner);
```

Remediation

Users that are able to provide a reward should also be able to collect the remaining reward.

Patch

Fixed in #XXX.

A | 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

B | 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.