Security Audit – Orca Whirlpools

Neodyme AG

May 05, 2022

Contents

Introduction

Orca engaged Neodyme to do a detailed security analysis of their new Orca Whirlpools smart contract on Solana. The full audit took place in the month leading up to the Whirlpools launch, beginning on 7th March 2022 and ending on 1st April 2022.

The audit revealed one severe vulnerability as well as some medium and low-priority findings, all of which were resolved by the Orca team subsequently.

In this report, we outline the most relevant findings.

Project Overview

Orca Whirlpools brings Concentrated Liquidity, known from Uniswap v3, to Solana. In traditional AMM liquidity pools, a user provides liquidity across the entire continuous price curve. With Whirlpools, each user can specify the price range over which they provide liquidity.

This enables higher efficiency of deposited liquidity but also increases the risk for Impermanent Loss.

From a technical perspective, the entire price curve gets segmented through ticks and a user can choose a lower tick and a higher tick as boundaries in which he wants to provide his liquidity. Each tick tracks the net-liquidity that gets added/removed when the price crosses the corresponding tick. Between two initialized ticks, Whirlpools act as a traditional Constant Product AMM. The authority over a liquidity-position is represented through an NFT and thereby leverages the power of SPL-Tokens.

If the token is traded in the provided liquidity range, the user earns the corresponding trading fees. Furthermore, Orca Whirlpools has integrated support for yield farming.

The contract is written idiomatically with the up-to-date Anchor Framework, currently the leading framework for writing Solana contracts.

Scope

The scope for this audit is the Orca Whirpools Smart Contract. At the time of the audit, the contract was developed in a private GitHub repository and corresponding commithashes still refer to that. After finishing the audit, Orca open sourced the code at https://github.com/orca-so/whirlpools with commit hash c55588a6eae1144be58bd348992f693e8b5ddc01 matching the state Neodyme audited and all reported bugs being fixed.

The audit also included the custom U256 math implementation used by Orca for swap calculations.

Methodology

Neodyme's audit team, which consists of security engineers with extensive experience in Solana smart contract security, reviewed the code of the on-chain contract, paying particular attention to the following:

- Ruling out common classes of Solana contract vulnerabilities, such as:
  - Missing ownership checks,
  - Missing signer checks,
  - Signed invocation of unverified programs,
  - Solana account confusions,
  - Re-initiation with cross-instance confusion,
  - Missing freeze authority checks,
  - Insufficient SPL token account verification,
  - Missing rent exemption assertion,
  - Casting truncation,
  - Arithmetic over- or underflows,
  - Numerical precision errors.
- Checking for unsafe design that might lead to common vulnerabilities being introduced in the future,
- Checking for any other, as-of-yet unknown classes of vulnerabilities arising from the structure of the Solana blockchain,
- Ensuring that the contract logic correctly implements the project specifications,
- Examining the code in detail for contract-specific low-level vulnerabilities,
- Ruling out denial-of-service attacks,
- Ruling out economic attacks,
- Checking for instructions that allow front-running or sandwiching attacks,
- Checking for rug-pull mechanisms or hidden backdoors,
- Checking for replay protection.

Findings

This section discusses the overall design of Orcas Whirlpool contract, followed by a detailed description of all our findings and their resolutions.

Orca takes great care not to run into one of Solana's most common sources of vulnerabilities: account confusions and missing signer and owner checks. They use the Anchor framework, which enforces account ownership and type, and additionally make good use of Program Derived Addresses (PDAs).

The codebase is of very good quality and documentation. Especially the math part and corresponding design choices are explained in-depth and provide clear guidance through the codebase.

Orca makes extensive use of unit- and fuzz-testing, ensuring the correctness of the program.

All findings are classified in one of four severity levels:

- **Critical**: Bugs that will likely cause a loss of funds. This means that an attacker can trigger them with little or no preparation or even accidentally. Effects are difficult to undo after they are detected.
- **High**: Bugs which can be used to set up a loss of funds in a more limited capacity, or to render the contract unusable.
- **Medium**: Bugs that do not cause a direct loss of funds but lead to other exploitable mechanisms.
- **Low**: Bugs that do not have a significant immediate impact and could be fixed easily after detection.

Name	Severity	Status
Lower Tick could be larger than upper Tick	High	Resolved
Integer Overflow when swapping	Medium	Resolved
Overflow in checked_mul_shift_right_round_up_if	Informational	Resolved

Lower Tick could be larger than upper Tick (High; Resolved)

Severity	Impact	Affected Component	Status
High	Corruption of curve invariant	Position logic	Resolved

When opening a position, the user specifies the interval, in which the position should provide liquidity through a lower tick boundary and a higher tick boundary. Orca was missing a check for `lower_tick < upper_tick` on position creation. The creation of those invalid positions would have led to uncovered liquidity. Assuming the current whirlpool price tick is 100 and an attacker creates a position with lower boundary 101 and upper boundary 99, then the liquidity of the whirlpool would not change, as this test does not pass:

```
if whirlpool.tick_current_index < tick_upper_index
        && whirlpool.tick_current_index >= tick_lower_index
```

Furthermore, the attacker would only have to pay `get_amount_delta_a()` of token A, as if the whole position would be above the current tick.

```
    if current_tick_index < position.tick_lower_index {
        // current tick below position
        delta_a = get_amount_delta_a(lower_price, upper_price,
            liquidity, round_up)?;
    } else if current_tick_index < position.tick_upper_index {
        // current tick inside position
        delta_a = get_amount_delta_a(sqrt_price, upper_price, liquidity
            , round_up)?;
        delta_b = get_amount_delta_b(lower_price, sqrt_price, liquidity
            , round_up)?;
    } else {
        // current tick above position
        delta_b = get_amount_delta_b(lower_price, upper_price,
            liquidity, round_up)?;
    }
```

Considering a swap from A to B that moves the price left, crossing our “upper” boundary 99, the liquidity will be increased by `tick_upper. liquidity` . In theory, this liquidity should consist solely of token B, which will get swapped to A in the following A to B trades. But our attacker has only supplied tokens A, so the newly added liquidity is uncovered and will corrupt all the following calculations.

Fix Orca quickly fixed this bug in commit `78ecd7e2f81c2036710ba9859192757a353b7e21` by failing if `tick_lower_index >= tick_upper_index` is true on creating a position. Neodyme verified this fix.

Integer Overflow when swapping (Medium; Resolved)

Severity	Impact	Affected Component	Status
Medium	Partial loss of funds	Swap Manager	Resolved

In swap_manager.rs, Orca used the following code snippet to calculate the input and output amount of tokens when swapping:

```
    if amount_specified_is_input {
        amount_remaining =
            amount_remaining - (swap_computation.amount_in +
               swap_computation.fee_amount);
        amount_calculated = amount_calculated + swap_computation.
           amount_out
    } else {
        amount_remaining = amount_remaining - swap_computation.
           amount_out;
        amount_calculated =
            amount_calculated + (swap_computation.amount_in +
               swap_computation.fee_amount);
    }
```

If the price of an asset is very high/low and we use a quite high prepared input/output amount, the swap_computation.amount_in + swap_computation.fee_amount or swap_computation.amount_out + swap_computation.fee_amount (all u64) calculations can overflow and the assets get priced wrongly. This can lead to sales of an expensive token for a fraction of its real worth.

Fix Orca subsequently fixed the overflow in commit 8199f59ed385c4ad9fccd6bc7743c77c448f4111 by using checked math. Neodyme verified the fix.

Overflow in checked_mul_shift_right_round_up_if (Informational; Resolved)

Severity	Impact	Affected Component	Status
Informational	None	Bit Math	Resolved

In bit_math.rs checked_mul_shift_right_round_up_if can overflow, if round_up is set to true and p equals u128::MAX:

```
    let result = (p >> Q64_RESOLUTION) as u64;
    Ok(if round_up && (p & Q64_MASK > 0) {
        result + 1
    } else {
        result
    })
```

With the current use of the function by Orca it is impossible to trigger this overflow, so we do not see any impact with the current implementation. Nevertheless, checked implies that also this overflow must not happen.

Fix Orca quickly fixed the overflow in commit 5e035a1db513b848dd8729e1c992ff4d6199ae78. Neodyme verified the fix.

Neodyme AG

Dirnismaning 55
Halle 13
85748 Garching
E-Mail: contact@neodyme.io

https://neodyme.io