Marco Polo DEX

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen notdeghost@osec.io

Akash Gurugunti Sud0u53r.ak@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Marco Polo engaged OtterSec to perform an assessment of the `marcopolo-dex` program.

This assessment was conducted between August 2nd and August 12th, 2022.

Critical 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 September 30th, 2022.

Key Findings

The following is a summary of the major findings in this audit.

- 8 findings total
- 3 vulnerabilities which could lead to loss of funds
  - OS-MCP-ADV-00: Improper Validation in add_tokens Instruction
  - OS-MCP-ADV-01: Improper Farm Reward Calculations
  - OS-MCP-SUG-04: Improper Calculation of delta_out in Swap Instruction

02 | Scope

The source code was delivered to us in a git repository at github.com/marcopolo-org/Dex. This audit was performed against commit `6f9cfd2`.

There was a total of 1 program included in this audit. A brief description of the programs is as follows. A full list of program files and hashes can be found in Appendix A.

Name	Description
marcopolo-dex	AMM bridging the gap between traditional Defi and NFTs

03 | 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 onchain program. In other words, there is no way to steal tokens or deny service, ignoring any Solana specific quirks such as account ownership issues. An example of a design vulnerability would be an onchain oracle which could be manipulated by flash loans or large deposits.

On the other hand, auditing the implementation of the program requires a deep understanding of Solana's execution model. Some common implementation vulnerabilities include account ownership issues, arithmetic overflows, and rounding bugs. For a non-exhaustive list of security issues we check for, see Appendix B.

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 any target in a team of two. This allows us to share thoughts and collaborate, picking up on details that the other missed.

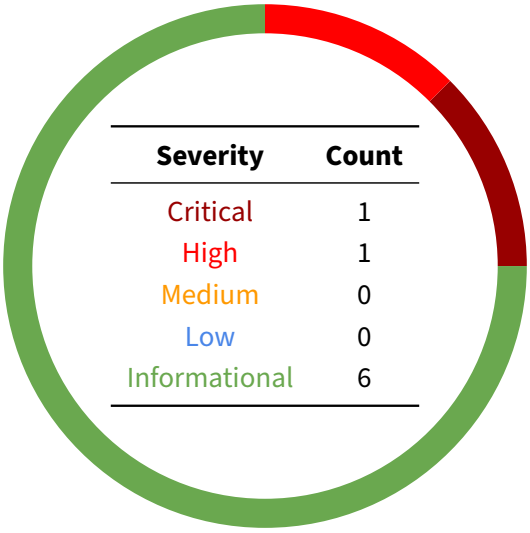
While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.

04 | Findings

Overall, we report 8 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	1
High	1
Medium	0
Low	0
Informational	6

05 | 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 C.

ID	Severity	Status	Description
OS-MCP-ADV-00	Critical	Resolved	Incorrect constraints on pool_x_account and pool_y_account in add_token instruction.
OS-MCP-ADV-01	High	Resolved	Farm rewards are incorrectly distributed

OS-MCP-ADV-00 [crit] [resolved] | Improper Validation in add_tokens

Description

The constraints on `pool_x_account` and `pool_y_account` in the `add_token` instruction don't properly validate that the accounts are actual pool reserve accounts. This allows a malicious provider to give their own token accounts as a `pool_x_account` and `pool_y_account` pair, essentially receiving liquidity provider shares for those tokens for free.

The below code snippet shows the improper checks:

src/instructions/add_tokens.rs RUST

```
35      #[account(mut,
36          constraint = owner_y_account.owner == owner.key()
37      )]
38      pub pool_x_account: Box<Account<'info, TokenAccount>>,
39      #[account(mut,
40          constraint = owner_y_account.owner == owner.key()
41      )]
42      pub pool_y_account: Box<Account<'info, TokenAccount>>,
```

Proof of Concept

Consider the following attack scenario:

1. A provider calls the `add_token` instruction with `pool_x_account = owner_x_account` and `pool_y_account = owner_y_account`.
2. The shares are actually transferred back to their accounts and they get shares for those tokens into their provider account.

Remediation

Enforcing proper checks on `pool_x_account` and `pool_y_account` to validate if they match with the pool reserve accounts or not.

Patch

Resolved in eff8e40.

OS-MCP-ADV-01 [high] [resolved] | Incorrect Farm Reward Distribution

Description

The rewards for a provider are accumulated using `farm.update_accumulator` function into the `farm.accumulated_seconds_per_share`. Whenever there is a change in the share amount, the reward amount that is collected up to that point should be either stored somewhere or should be transferred to the provider.

Proof of Concept

Consider the following scenario:

1. A pool and a farm are initiated by the admin
2. A provider is created by a user with some initial tokens
3. After some time, the user withdraws 90% of his shares
4. Now if the user tries to withdraw the remaining 10% of his shares, he will only get rewards for 10% of his shares

Remediation

The rewards should be calculated and distributed whenever there is a change in the provider shares, i.e., in `add_tokens` and `withdraw_shares`.

Patch

Resolved in 39431e3, a7e042d, and c249da4.

06 | 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-MCP-SUG-00	The create_state instruction should enforce that the program_authority account is a PDA.
OS-MCP-SUG-01	Unnecessary re-implementation of account closing instruction.
OS-MCP-SUG-02	Code refactoring is recommended in the close_pool instruction.
OS-MCP-SUG-03	It is recommended to define constant strings used in seeds at one place.
OS-MCP-SUG-04	Improper calculation of delta_out in swap instruction.
OS-MCP-SUG-05	There are a variety of improper constraint checks.

OS-MCP-SUG-00 [resolved] | Enforcing program_authority to be a PDA

Description

The `program_authority` account in state account is used as the owner for the pool reserve accounts and farm token accounts. Whenever funds are transferred out of those accounts, the `get_program_signature` macro is used to generate signer seeds for the `program_authority` PDA.

This should be enforced in the `create_state` instruction instead of taking an unchecked account from the input accounts.

Remediation

The following change to the `create_state` instruction is recommended:

src/instructions/create_state.rs DIFF

```
use anchor_lang::prelude::*;

use crate::structs::State;
+ use crate::PROGRAM_SEED;

#[derive(Accounts)]
#[instruction(nonce: u8)]
pub struct CreateState<'info> {
    #[account(init, seeds = [b"marcostatev1".as_ref()], payer = admin,
        ↪  bump)]
    pub state: AccountLoader<'info, State>,
    #[account(mut)]
    pub admin: Signer<'info>,
+   #[account(seeds = [PROGRAM_SEED.as_bytes()], bump = nonce)]
    pub program_authority: AccountInfo<'info>,
    pub system_program: AccountInfo<'info>,
}
```

Patch

Resolved in eff8e40.

OS-MCP-SUG-01 [resolved] | Unnecessary Re-implementation of Predefined instruction

Description

A function named `close` is implemented in `utils.rs` which has the same functionality as a function implemented by `anchor-lang`–namely, closing a Solana account. (here). This re-implemenation of the close function is unnecessary and the `anchor-lang` close function should be used instead.

Below is a code snippet showing the unnecessary function in `utils.rs`:

src/utils/utils.rs RUST

```rust
pub fn close<'info>(
    info: AccountInfo<'info>,
    sol_destination: AccountInfo<'info>,
) -> ProgramResult {
    // Transfer tokens from the account to the sol_destination.
    let dest_starting_lamports = sol_destination.lamports();
    **sol_destination.lamports.borrow_mut() =
        dest_starting_lamports.checked_add(info.lamports()).unwrap();
    **info.lamports.borrow_mut() = 0;

    // Mark the account discriminator as closed.
    let mut data = info.try_borrow_mut_data()?;
    let dst: &mut [u8] = &mut data;
    let mut cursor = std::io::Cursor::new(dst);
    cursor
        .write_all(&CLOSED_ACCOUNT_DISCRIMINATOR)
        .map_err(|_| ErrorCode::AccountDidNotSerialize)?;
    Ok(())
}
```

Remediation

Delete the `close` function implementation in the `utils.rs` and use the `anchor-lang` close function instead, as shown in the below code snippets:

src/instructions/close_pool.rs DIFF

```
use anchor_lang::prelude::*;
use anchor_spl::token::{self, CloseAccount, Mint, TokenAccount,
    ↪ Transfer};
use decimal::*;
+ use anchor_lang::AccountsClose;
```

src/instructions/close_pool.rs DIFF

```
-   close(
-       ctx.accounts.pool.to_account_info(),
-       ctx.accounts.admin.to_account_info(),
-   )?;
+   ctx.accounts
+       .pool
+       .close(ctx.accounts.admin.to_account_info())?;
```

src/instructions/close_pool.rs DIFF

```
-   close(
-       ctx.accounts.farm.to_account_info(),
-       ctx.accounts.admin.to_account_info(),
-   )?;
+   ctx.accounts
+       .farm
+       .close(ctx.accounts.admin.to_account_info())?;
```

Patch

Resolved in eff8e40.

OS-MCP-SUG-02 [resolved] | Miscellaneous Code Refactoring

Description

There is a lot of unnecessary code in the `close_pool` instruction that tracks the lamports transferred to admin. The `diff` variable in the `close_pool` instruction keeps track of the lamports added to the admin account due to the closing of other PDA accounts by removing the added lamports from the admin account and keeping track of them.

All the lamports collected by the `diff` variable are added back to the admin account and since the `diff` is not stored anywhere, it is unnecessary to keep track of the lamports.

Remediation

The code in the below code snippet can be removed:

src/instructions/close_pool.rs RUST

```
226     let mut diff = 0;
227     let initial_lamports = **ctx
228         .accounts
229         .admin
230         .to_account_info()
231         .try_borrow_mut_lamports()?;
```

The code in the below code snippet is repeated in the file several times and all of it can be removed:

src/instructions/close_pool.rs RUST

```
243         diff += **ctx
244             .accounts
245             .admin
246             .to_account_info()
247             .try_borrow_mut_lamports()?
248             - initial_lamports;
249         **ctx
250             .accounts
251             .admin
252             .to_account_info()
253             .try_borrow_mut_lamports()? = initial_lamports;
```

The code in the below code snippet can be removed:

src/instructions/close_pool.rs RUST

```
386     **ctx
387         .accounts
388         .admin
389         .to_account_info()
390         .try_borrow_mut_lamports()? += diff;
```

Patch

Resolved in eff8e40.

OS-MCP-SUG-03 [resolved] | PDA Seed Constants

Description

The strings that are provided as seeds for the PDA accounts like ”marcostatev1”, ”marcopoolv1”, ”marcofarmv1”, ”marcoproviderv1”, etc., should be defined at one place as constants and used in the required places, just like the constant PROGRAM_SEED in `lib.rs`. This way, in the event of any future changes to seeds of the PDAs in the protocol, it will be easier for the developer to edit all the strings, as they will be in one place.

Remediation

Define all the constant strings that are used in PDAs in one place and use them by importing them wherever required as shown in the below code snippets.

The code snippet that shows defining the constants:

src/lib.rs DIFF

```
const PROGRAM_SEED: &str = "MarcoPolo";
+ const MARCO_STATE_SEED: &[u8] = b"marcostatev1";
+ const MARCO_POOL_SEED: &[u8] = b"marcopoolv1";
+ const MARCO_FARM_SEED: &[u8] = b"marcofarmv1";
+ const MARCO_PROVIDER_SEED: &[u8] = b"marcoproviderv1";
```

The code snippet that shows the use of the constants:

src/instructions/create_state.rs DIFF

```
use crate::structs::State;
+ use crate::{MARCO_STATE_SEED};

#[derive(Accounts)]
#[instruction(nonce: u8)]
pub struct CreateState<'info> {
-    #[account(init, seeds = [b"marcostatev1".as_ref()], payer = admin,
      ↪  bump)]
+    #[account(init, seeds = [MARCO_STATE_SEED], payer = admin, bump)]
     pub state: AccountLoader<'info, State>,
     #[account(mut)]
     pub admin: Signer<'info>,
```

OS-MCP-SUG-04 [resolved] | Improper Calculation of delta_out in Swap Instruction

Description

While calculating the `delta_out` amount inside the swap instruction using the `calculate_delta_out` function, the fraction is floored by default, resulting in a higher value than the original `delta_out`.

Consider the following example:

1. Let `token_x_reserve = 10000`, `token_y_reserve = 1000`, `delta_in = 500`, `x_to_y = true` and `const_k = 10**7`
2. denominator = 10000 + 500 = 10500
3. fraction = 10**7 / 10500 = 952.3809523809524 = 952 (floored)
4. `delta_out` = 1000 - 952 = 48 (but the original value is 47.61904761904759)
5. The value of `const_k` after the swap will be 10500*952 = 9996000, which is less than original `const_k`

The function that does this calculation is in the below code snippet:

src/utils/math.rs RUST

```
31      let denominator = if x_to_y {
32          delta_in + token_x_reserve
33      } else {
34          delta_in + token_y_reserve
35      };
36
37      let fraction = Token::from_decimal(const_k / denominator);
38
39      let delta_out = match x_to_y {
40          true => token_y_reserve - fraction,
41          false => token_x_reserve - fraction,
42      };
43
44      Ok(delta_out)
45  }
```

Patch

Resolved in eff8e40.

Remediation

While calculating the fraction, `spl_math::checked_ceil_div` should be implemented on `Token` and used instead of default division to avoid flooring.

OS-MCP-SUG-05 [resolved] | Improper Constraint Checks

Description

The constraints on `owner_project_first_account` and `owner_project_second_account` in the `withdraw_rewards` instruction don't validate that the accounts have `owner.key()` as the owner.

The constraints on `admin_project_first_account` and `admin_project_second_account` in the `close_pool` instruction also don't validate that the accounts have `admin.key()` as the owner.

Remediation

The below code snippets shows the suggested changes to the constraints:

src/instructions/withdraw_rewards.rs DIFF

```
-       constraint = owner_marco_account.owner == owner.key(),
+       constraint = owner_project_first_account.owner == owner.key(),
    )]
    pub owner_project_first_account: Box<Account<'info,
        ↪ TokenAccount>>,
    #[account(mut,
-       constraint = owner_marco_account.owner == owner.key(),
+       constraint = owner_project_second_account.owner ==
    ↪ owner.key(),
    )]
    pub owner_project_second_account: Box<Account<'info,
        ↪ TokenAccount>>,
```

src/instructions/close_pool.rs DIFF

```
-       constraint = admin_marco_account.owner == admin.key()
+       constraint = admin_project_first_account.owner == admin.key()
    )]
    pub admin_project_first_account: Box<Account<'info,
        ↪ TokenAccount>>,
    #[account(mut,
-       constraint = admin_marco_account.owner == admin.key()
+       constraint = admin_project_second_account.owner == admin.key()
    )]
```

```
73      pub admin_project_second_account: Box<Account<'info,
            ↪  TokenAccount>>,
```

Patch

Resolved in eff8e40.

A | Program Files

Below are the files in scope for this audit and their corresponding SHA256 hashes.

```
src/
  lib.rs                                  c12447a5ca6cc4d963d27d560e6721d3ee140e028ce7fbd658
  instructions/
    add_supply.rs                         8bc68ccc7733bc9751245994a36889c52f627c451fc19c5176
    add_tokens.rs                         1abae0fad615a0e99eeafe7e2f830ee990e5c3788094caf9b3
    close_pool.rs                         f48db67dc4e783ce329399958adff7ddd4ecccb2de7d02414e
    create_dual_farm.rs                   62598f01844906e20190523871a816cdfd4c408383026352ad
    create_farm.rs                        29c34cbd0e6cce68cd10a8378c9d66aff0b7eb6d78928d07d7
    create_pool.rs                        8cec00b283c607ab5944e7dfed3dfe110e95e9c831c371b573
    create_provider.rs                    66df4e7841866aef4718ba327236946943f79180b66591bdc4
    create_state.rs                       3d394c3e7d433ea1ee3dbe15ecc9c036bc3eb4e4d595566db1
    create_triple_farm.rs                 a71d0bc7a20e3d1baa2f30dadbfed7d45fd33c98d485c3acc8
    mod.rs                                6c6a36196d71ab006e11a3a49f5d73a9bb5bd760d737877822
    swap.rs                               891154cb0dfb5a31f3c0eac035d8940fc327570ab068f4f8f5
    withdraw_buyback.rs                   3a7a32acb0f1670e20639586d94754ead5f7d18c0ccfa4f081
    withdraw_lp_fee.rs                    fac5ccccf399b30d3d2680a8597a1f03b1aebaf11f16d18604
    withdraw_mercanti_fee.rs              648d3c425dfe71e7c5dbfd32068185421a9a25dda37ff66cc0
    withdraw_project_fee.rs               cb976f2126434cbc10bebb391ccab956da54a232a797238be6
    withdraw_rewards.rs                   4d918eee79fdc35b9fe6fcc345cf379404dff70e732c1e9bf6
    withdraw_shares.rs                    980fd0a37bfa6eff72335f0e15b54a42ec406d88aa32b4eadc
  interfaces/
    close_token_account.rs                0c003a51d1cca966c6e9798987d20e588dcffd09534cc54642
    mod.rs                                2d2682daad8ae4307e99aeea30b7e97764b0edf55585d1c015
    send_farm.rs                          6029f22020fd6e67258e439a96610d382f905c3b0d58f81b23
    send_tokens.rs                        8694af7245aaceed6fe2a0e8f170d85693ffd34f7c655793f8
    take_farm.rs                          4d088aa6cfe5e1beefe8a9cd94aff118b7db65d8904123e2e7
    take_tokens.rs                        11389226ca14a9a4a79a62c5bb120155d58adfd05ad7c1c48f
  structs/
    farm.rs                               f38db8e6855cfcb29601bdaddf284b55ff4f2744771c3f08e0
    mod.rs                                df9b16342d7740b668fc21616a74ff9adfea76acf9438034df
    pool.rs                               f0f79dd3d87dbf7e6658cb596b9b89f9a32310b1f41c75caff
    provider.rs                           4399f4c4d5a95894d1e8805d2773b9a83e50fd66f0284986d2
    state.rs                              804426c0fd96bf77221364bd619c621508bb6f1cf020a0d746
  utils/
    decimal.rs                            b40063b711e73bf8a04c13ba57a30d900a8fa28a817082bc5a
    errors.rs                             bcdd7ed9e95da65541715e6031ceab1593afc8a7db8d4bf180
    math.rs                               83eb11ddd4129eeeb331ed9015e33440a2d1a5e0ffbb5ffdb8
    mod.rs                                804753de3719c7207e9047b870dfc7855e4b0bb3a4167bc11e
    uint.rs                               6120e5f6efb85989300e324e4e0ec13aa7588c86f8465cfb19
    utils.rs                              deb377cc4dff0f13d470cdb2f3959635d9c08338d930f9516b
```

B | Implementation Security Checklist

Unsafe arithmetic

Integer underflows or overflows	Unconstrained input sizes could lead to integer over or underflows, causing potentially unexpected behavior. Ensure that for unchecked arithmetic, all integers are properly bounded.
Rounding	Rounding should always be done against the user to avoid potentially exploitable off-by-one vulnerabilities.
Conversions	Rust as conversions can cause truncation if the source value does not fit into the destination type. While this is not undefined behavior, such truncation could still lead to unexpected behavior by the program.

Account security

Account Ownership	Account ownership should be properly checked to avoid type confusion attacks. For Anchor, the safety of unchecked accounts should be clearly justified and immediately obvious.
Accounts	For non-Anchor programs, the type of the account should be explicitly validated to avoid type confusion attacks.
Signer Checks	Privileged operations should ensure that the operation is signed by the correct accounts.
PDA Seeds	PDA seeds are uniquely chosen to differentiate between different object classes, avoiding collision.

Input validation

Timestamps	Timestamp inputs should be properly validated against the current clock time. Timestamps which are meant to be in the future should be explicitly validated so.
Numbers	Sane limits should be put on numerical input data to mitigate the risk of unexpected over and underflows. Input data should be constrained to the smallest size type possible, and upcasted for unchecked arithmetic.
Strings	Strings should have sane size restrictions to prevent denial of service conditions
Internal State	If there is internal state, ensure that there is explicit validation on the input account's state before engaging in any state transitions. For example, only open accounts should be eligible for closing.

Miscellaneous

Libraries	Out of date libraries should not include any publicly disclosed vulnerabilities
Clippy	cargo clippy is an effective linter to detect potential anti-patterns.

C | 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 which immediately lead to loss of user funds with minimal preconditions Examples: • Misconfigured authority/token account validation • Rounding errors on token transfers
High	Vulnerabilities which 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 which could lead to denial of service scenarios or degraded usability. Examples: • Malicious input cause computation limit exhaustion • Forced exceptions preventing normal use
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 • Uncaught Rust errors (vector out of bounds indexing)