# RAYDIUM

# CONSTANT PRODUCT AMM

# SWAP PROGRAM

# FUZZ REPORT

## INTRODUCTION

Raydium cp-swap program is a constant product swap program on the solana blockchain that is been used to swap a token into another token provided a market for it has already been created/configured for the two dissimilar tokens.

It’s major functionalities include

- Admin creates a pool that stores two dissimilar tokens that users can use to swap.
- Users can swap between two dissimilar tokens either from token a to token b or vice-versa.
- Users can supply to the pool that stores the tokens in order to earn yield.

## Scope and Objectives

- Setup Trident for the Anchor based program and configure it to ensure it is working perfectly.
- Identify key crictical functions.
- Run fuzz tests and anlayze the program’s behaviour to unexpected, invalid and even valid input.

## Methodology

In order to setup the raydium cp-swap program to accommodate for trident which has a limited functionality on most on-chain programs, the cp-swap program setup configurations were modified. The following modifications were made

- Add

```
[toolchain]
anchor_version = "0.29.0"
solana_version = "1.18.23"
```

to the Anchor.toml file.

- Add

```
[dependencies]
ahash = "0.8.9"
```

to the Cargo.toml file in programs/cp-swap/* directory.

- A local clone of the trident repo was used and added to the same directory as the raydium cp-swap program, this was to ensure that more code could be added to the trident binary to enable more custom functionalities without needing to modify the root trident binary or await a pull request from the trident team for missing functionality.
- Add

  ```
  [dependencies]
  solana-sdk = ">=1.18"
  solana-client = ">=1.18"
  solana-account-decoder = ">=1.18"
  solana-transaction-status = ">=1.18"
  ```

  to the Cargo.toml file in client/* directory.(This is optional as it enables testing on the client side)

**Instructions in Scope**

Fuzzing was carried out on three instructions and analyzed through trident.

These instructions are as follows

- create_amm_config – This is an admin gated instruction that is responsible for creating the configuration details unique to a protocol.
- update_config – This is also an admin gated instruction but it is responsible for updating or modifying a protocol's configuration details.
- Initialize – This instruction performs the creation of the pool callable by any address.

  It handles the creation of the token vaults and gives the first deposit to these vaults.

  Also, it creates the lp mints and mints to the address with which it was called from.

  Lastly, it stores the current pool configuration details.

## Fuzzing Results

### Approach

For the `UpdateAmmConfigData` and `CreateConfigData` Structs in `fuzz_instructions.rs` file, the arbitrary trait was implemented to ensure passed in values are randomized but do not ever go above the `FEE_RATE_DENOMINATOR_VALUE`, this is because the corresponding instructions have constraints everywhere that already check for these and we want to limit crashes or failing instructions as much as possible.

Also, the token balances before and after the `initialize` instruction were evaluated in the `check` invariant function, this to monitor the expected behaviour of of the account data before and after the instruction call.

Furthermore, a custom function was added to our local trident binary named `set_token_account_custom`, it mimics the functionality of `set_token_account` but creates a token account to a custom address rather than to a new address as the latter.

**Results**

The instructions were set to be called in the following sequence `create_amm_config`, `update_config` and `initialize` respectively.
From the fuzzing stats, it is easy to say `create_amm_config` instruction is usually successful except in cases where the address is already in use or there is no longer lamports for creation of new accounts.

`update_config` instruction has a very high success output but fails sometimes due to the poor seeds generation.

`initialize` instruction has a very high instruction failure output. From my observations, this is usually caused by poor seeds generation or the `SyncNative` CPI failing to succeed with the error output `Invalid account state for operation`.

## References

Trident - https://github.com/Ackee-Blockchain/trident

Raydium cp-swap - https://github.com/raydium-io/raydium-cp-swap

Fuzz report - https://github.com/Solana-Auditors-Bootcamp/fuzzing-with-trident-chinepun