audita

# SnapperDEX

## Smart Contract Security Audit

January 29th, 2025

Prepared for:
Solana.fun - DEX and Launchpad
(*solana.fun*)

Presented By:
Audita Security (*audita.io*)

# Document

The contents of this document may include confidential information pertaining to the IT systems, intellectual property, and possible vulnerabilities along with methods of exploitation that the Client may possess. The report that contains this confidential information can be utilized internally by the Client, and can be made available to the public after all vulnerabilities are addressed, depending on the decision of the Client.

**Network**: Solana
**Programming language**: Rust
**Method**: Manual Audit
**Auditors:** @arabadzhiew @cvetanovv
**Client Website**: [https://www.solana.fun/](https://www.solana.fun/)
**Timeline:** 20/01/2025 - 28/01/2025

# Table of Contents

**Document** 1

**Executive Summary** 3

**Audita Vulnerability Classifications** 4

**Scope** 5

**Findings** 7

Summary 7

Detailed Findings 9

**Recommendations** 27

**Fixes** 28

# Executive Summary

### Manual Audit

During the manual audit conducted by our experts, we did not identify any Critical severity vulnerabilities.

We identified 2 High and 2 Medium severity vulnerabilities.
2 Low severity vulnerabilities were marked, as well as 7 Informational issues.

### Overall Assessment

Once the High and Medium severity vulnerabilities are addressed, Snapper's smart contracts are safe and pose no risks to the protocol and its users.

| Severity | Count | Fixes |
|---|---|---|
| Critical | 0 | - |
| High | 2 | Fixed |
| Medium | 2 | Acknowledged |
| Low | 2 | Acknowledged |
| Informational | 7 | Acknowledged |

### Documentation

We recommend this report, as well as specific information from this report to be included in protocol's official Documentation.

# Audita Vulnerability Classifications

Audita follows the most recent standards for vulnerability severities, taking into consideration both the possible impact and the likelihood of an attack occurring due to a certain vulnerability.

| Severity | Description |
|---|---|
| Critical | Critical vulnerability is one where the attack is more straightforward to execute and can lead to exposure of users' data, with catastrophic financial consequences for clients and users of the smart contracts. |
| High | The vulnerability is of high importance and impact, as it has the potential to reveal the majority of users' sensitive information and can lead to significant financial consequences for clients and users of the smart contracts. |
| Medium | The issue at hand poses a potential risk to the sensitive information of a select group of individual users. If exploited, it has the potential to cause harm to the client's reputation and could result in unpleasant financial consequences. |
| Low | The vulnerability is relatively minor and not likely to be exploited repeatedly, or is a risk that the client has indicated is not impactful or significant, given their unique business situation. |
| Informational | The issue may not pose an immediate threat  to ongoing operation or utilization, but it's essential to consider implementing security and software engineering best practices, or employing backup measures as a safety net. |

# Scope

The security assessment was scoped to the following files in Snapper DEX's code repository:

| Files |
|---|
| **-curve** |
| *calculator.rs* |
| *constant_product.rs* |
| *fees.rs* |
| *mod.rs* |
| *snapper.rs* |
| ***-instructions/admin*** |
| *collect_fund_fee.rs* |
| *collect_protocol_fee.rs* |
| *create_config.rs* |
| *mod.rs* |
| *update_config.rs* |
| *update_pool_status.rs* |
| ***-instructions*** |
| *deposit.rs* |
| *initialize_metadata.rs* |

| |
|---|
| *initialize.rs* |
| *mod.rs* |
| *swap_base_input.rs* |
| *swap_base_output.rs* |
| *withdraw.rs* |
| ***-states*** |
| *config.rs* |
| *events.rs* |
| *mod.rs* |
| *oracle.rs* |
| *pool.rs* |
| *user.rs* |
| ***-utils*** |
| *account_load.rs* |
| *math.rs* |
| *mod.rs* |
| *token.rs* |
| *error.rs* |
| *lib.rs* |

The codebase has been audited up to and including commit:

***ae421fc13a1c83ad5853d702d00efaf0000a6abd***

# Findings

## Summary

| Code | Description | Severity | Fixes |
|---|---|---|---|
| **[SNAP-01]** | The calculations for the input tokens in the deposit function are being rounded in the wrong direction | High | Fixed |
| **[SNAP-02]** | The constant product calculations in *swap_base_input* and *swap_base_output* are performed with token amounts that include the accumulated swap fees | High | Fixed |
| **[SNAP-03]** | Observations can be Insufficient for TWAP Calculation | Medium | Acknowledged |
| **[SNAP-04]** | TWAP Oracle may not correctly reflect the price in a volatile market | Medium | Acknowledged |
| **[SNAP-05]** | Missing Deadline Check in Swap Functions [irrelevant] | - | Voided |
| **[SNAP-06]** | Timing variability in curve25519-dalek's Scalar29::sub/Scalar52::sub | Low | Acknowledged |

| | | | |
|---|---|---|---|
| **[SNAP-07]** | Double Public Key Signing Function Oracle Attack on ed25519-dalek | Low | Acknowledged |
| **[SNAP-08]** | Lack of integration tests | Informational | Acknowledged |
| **[SNAP-09]** | Unused function level variables | Informational | Acknowledged |
| **[SNAP-10]** | Unused imports | Informational | Acknowledged |
| **[SNAP-11]** | The *Fees::trading_fee* function is not being used anywhere | Informational | Acknowledged |
| **[SNAP-12]** | Typos | Informational | Acknowledged |
| **[SNAP-13]** | Redundant functions | Informational | Acknowledged |
| **[SNAP-14]** | Commented out code | Informational | Acknowledged |

## Detailed Findings

| | | |
|---|---|---|
| **[SNAP-01]** | The calculations for the input tokens in the deposit function are being rounded in the wrong direction | High |

**Function:** deposit

**Details:**

Within the current implementation of the deposit function, the input amounts of token0 and token1 are being calculated in the following way:

```
let results = CurveCalculator::lp_tokens_to_trading_tokens(
       u128::from(lp_token_amount),
       u128::from(pool_state.lp_supply),
       u128::from(total_token_0_amount),
       u128::from(total_token_1_amount),
       RoundDirection::Floor,
   )
   .ok_or(ErrorCode::ZeroTradingTokens)?;
```

As it can be seen, the calculations are being performed by the CurveCalculator::lp_tokens_to_trading_tokens helper function, which accepts a round_direction value as its last parameter. And as it can also be seen, in the deposit function, the value that is being passed in for that parameter is RoundDirection::Floor. What this essentially means is that the values for the token0 and token1 input amounts will always be rounded down, which favours the depositors, not the protocol owners. Furthermore, this also means that it can be abused to mint small amounts of LP shares for free, as can be observed in the calculation logic itself:

```
pub fn lp_tokens_to_trading_tokens(
       lp_token_amount: u128,
       lp_token_supply: u128,
       swap_token_0_amount: u128,
       swap_token_1_amount: u128,
       round_direction: RoundDirection,
   ) -> Option<TradingTokenResult> {
       let mut token_0_amount = lp_token_amount
           .checked_mul(swap_token_0_amount)?
           .checked_div(lp_token_supply)?;
       let mut token_1_amount = lp_token_amount
           .checked_mul(swap_token_1_amount)?
           .checked_div(lp_token_supply)?;
```

This holds true even more so for pool pairs with low-decimal tokens.

**Impact:**

Malicious users will be able to mint LP shares for free, effectively stealing funds from other liquidity providers.

**Recommendation:**

When performing the calculations for the input amounts in the deposit function, round up instead of down:

```
    let results = CurveCalculator::lp_tokens_to_trading_tokens(
        u128::from(lp_token_amount),
        u128::from(pool_state.lp_supply),
        u128::from(total_token_0_amount),
        u128::from(total_token_1_amount),
-       RoundDirection::Floor,
+       RoundDirection::Ceiling,
    )
    .ok_or(ErrorCode::ZeroTradingTokens)?;
```

| | | |
|---|---|---|
| **[SNAP-02]** | The constant product calculations in *swap_base_input* and *swap_base_output* are performed with token amounts that include the accumulated swap fees | High |

**Details:**

The current implementation of both the swap_base_input and swap_base_output functions calculate the total pool token amounts in the following way:

```
            let (total_input_token_amount, total_output_token_amount) =
pool_state
                .vault_amount_without_fee(
                    ctx.accounts.input_vault.amount,
                    ctx.accounts.output_vault.amount,
                );

            (
                TradeDirection::ZeroForOne,
                total_input_token_amount,
                total_output_token_amount,
            )
```

As it can be seen, the vault_amount_without_fee function of the PoolState account struct is being used in order to subtract the fees from the total token amounts.

```
    pub fn vault_amount_without_fee(&self, vault_0: u64, vault_1: u64) ->
(u64, u64) {
        (
            vault_0
                .checked_sub(self.creator_fees_token_0 +
self.protocol_fees_token_0)
                .unwrap(),
            vault_1
                .checked_sub(self.creator_fees_token_1 +
self.protocol_fees_token_1)
                .unwrap(),
        )
    }
```

However, as it can be seen within the implementation of that function, it does not subtract the swap fees from the total amounts. What this essentially means is that since the constant product calculations won't be made with the "pure" `token0` and `token1` amounts, they will be incorrect, due to the fact that the swap fees can't and won't follow the same proportions as the trading token amounts in their respective pools all of the time.

**Impact:**

The constant product calculations will be incorrect.

**Recommendation:**

In both swap functions, use the total token amounts without the swap fees for the constant product calculations:

```
// Calculate the trade amounts
    let (trade_direction, total_input_token_amount,
total_output_token_amount) =
        if ctx.accounts.input_vault.key() == pool_state.token_0_vault
            && ctx.accounts.output_vault.key() == pool_state.token_1_vault
        {
            let (total_input_token_amount, total_output_token_amount) =
pool_state
-               .vault_amount_without_fee(
+               .vault_amount_without_swap_fee(
                    ctx.accounts.input_vault.amount,
                    ctx.accounts.output_vault.amount,
                );

            (
                TradeDirection::ZeroForOne,
                total_input_token_amount,
                total_output_token_amount,
            )
        } else if ctx.accounts.input_vault.key() ==
pool_state.token_1_vault
            && ctx.accounts.output_vault.key() == pool_state.token_0_vault
        {
            let (total_output_token_amount, total_input_token_amount) =
pool_state
-               .vault_amount_without_fee(
+               .vault_amount_without_swap_fee(
                    ctx.accounts.output_vault.amount,
                    ctx.accounts.input_vault.amount,
                );

            (
                TradeDirection::OneForZero,
                total_input_token_amount,
                total_output_token_amount,
            )
        } else {
            return err!(ErrorCode::InvalidVault);
        };
```

| | | |
|---|---|---|
| **[SNAP-03]** | Observations can be Insufficient for TWAP Calculation | Medium |

**Details:**

The ObservationState in *oracle.rs* uses a fixed-size circular buffer to store observations, with the size set to OBSERVATION_NUM = 100.

```
pub const OBSERVATION_NUM: usize = 100;
```

Each observation is updated every `OBSERVATION_UPDATE_DURATION_DEFAULT = 15` seconds.

```
OBSERVATION_UPDATE_DURATION_DEFAULT = 15
```

This means the buffer retains a maximum of 25 minutes of historical price data.

```
if delta_time < OBSERVATION_UPDATE_DURATION_DEFAULT
    return;
}
```

However, this design may not be sufficient. As a result, TWAP calculations may fail to include the intended time frame, leading to inaccuracies in price-sensitive operations.

**Impact:**

A fixed 25-minute window could lead to inaccuracies.

A larger observation capacity would provide more accurate TWAP calculations, particularly in highly active pools or during volatile market conditions where frequent updates shorten the effective lookback window.

**Recommendation:**

Increase `OBSERVATION_NUM` to 150–200, allowing the system to retain a broader historical window without overwriting data too quickly.

| [SNAP-04] | TWAP Oracle may not correctly reflect the price in a volatile market | Medium |
|---|---|---|

**Details:**

The current implementation of the TWAP oracle updates the observation array every 15 seconds, as defined by `OBSERVATION_UPDATE_DURATION_DEFAULT`:

```
pub const OBSERVATION_UPDATE_DURATION_DEFAULT: u64 = 15;
```

```
    pub fn update(
        &mut self,
        block_timestamp: u64,
        token_0_price_x32: u128,
        token_1_price_x32: u128,
    ) {
        let observation_index = self.observation_index;
        if !self.initialized {
            self.initialized = true;
            self.observations[observation_index as usize].block_timestamp =
block_timestamp;
            self.observations[observation_index as
usize].cumulative_token_0_price_x32 = 0;
            self.observations[observation_index as
usize].cumulative_token_1_price_x32 = 0;
        } else {
            let last_observation = self.observations[observation_index as
usize];
            let delta_time =
block_timestamp.saturating_sub(last_observation.block_timestamp);
--->            if delta_time < OBSERVATION_UPDATE_DURATION_DEFAULT {
                return;
            }
.....
.....
```

While this interval might be reasonable for networks like Ethereum, which has block times of approximately 12 seconds, it introduces essential limitations on Solana, which has block times of approximately 400 milliseconds.

Due to the longer update interval:

- Rapid price changes within a 15-second period may go completely unnoticed by the oracle.
- On Solana, where 15 seconds can span more than 30 blocks, sharp price movements can occur and be entirely excluded from the TWAP calculations.

For example, in a volatile market scenario, if a token experiences a sharp price drop or spike within a few seconds, this event will not be reflected in the TWAP. Instead, the TWAP will inaccurately portray the market as stable, skewing averages and leading to incorrect pricing.

**Impact:**

Significant price changes within the 15-second interval are ignored, leading to delayed or inaccurate TWAP calculations. The oracle may not reflect market conditions during critical moments, creating discrepancies in price-sensitive operations.

**Recommendation:**

Reduce the `OBSERVATION_UPDATE_DURATION_DEFAULT` to around 5 seconds to align better with Solana's block time and capture more granular price data.

As we have written in another issue, increase `OBSERVATION_NUM` to 150–200 to accommodate the additional observations generated by a shorter update interval while maintaining a sufficient historical window for TWAP calculations.

| [SNAP-06] | Timing variability in curve25519-dalek's Scalar29::sub/Scalar52::sub | Low |
|---|---|---|

**Details:**

Package vulnerability.

While it has no direct effect on the protocol, it is important to be mentioned.

Reference: https://rustsec.org/advisories/RUSTSEC-2024-0344

**Recommendation:**

Update the package to the appropriate version as specified in the reference.

| [SNAP-07] | Double Public Key Signing Function Oracle Attack on ed25519-dalek | Low |
|---|---|---|

**Details:**

Package vulnerability.

While it has no direct effect on the protocol, it is important to be mentioned.

Reference: https://rustsec.org/advisories/RUSTSEC-2022-0093

**Recommendation:**

Update the package to the appropriate version as specified in the reference.

| | | |
|---|---|---|
| **[SNAP-08]** | Lack of integration tests | Informational |

**Details:**

The current state of the codebase lacks one important part - an integration testing suite. This goes against the general best coding practices. Furthermore, without those tests, the intended functionality of the protocol will be much harder to nail down, making it easier to apply breaking changes to its current working state, should the need to change some part of the codebase arise.

**Recommendation:**

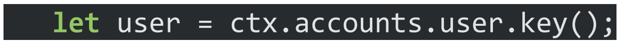
Consider writing an integration testing suite that verifies that all of the functionality of the protocol works as it should.

| | | |
|---|---|---|
| **[SNAP-09]** | Unused function level variables | Informational |

**Details:**

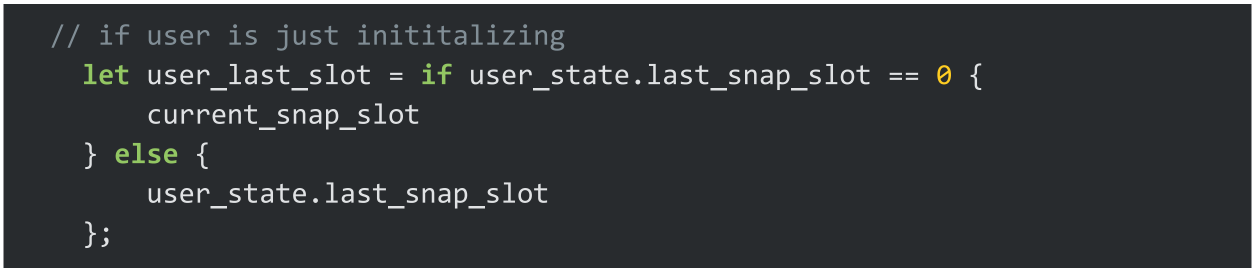
The following function level variables are not being used and can safely be removed:

withdraw.rs#L125



```
    let user = ctx.accounts.user.key();
```

withdraw.rs#L145-L150



```
  // if user is just inititalizing
    let user_last_slot = if user_state.last_snap_slot == 0 {
        current_snap_slot
    } else {
        user_state.last_snap_slot
    };
```

deposit.rs#L117

```
let user = ctx.accounts.user.key();
```

deposit.rs#L129-L134

```
 // if user is just inititalizing
   let user_last_slot = if user_state.last_snap_slot == 0 {
       current_snap_slot
   } else {
       user_state.last_snap_slot
   };
```

**Recommendation:**

Remove the unused variables.

| | | |
|---|---|---|
| **[SNAP-10]** | Unused imports | Informational |

**Details:**

At its current state, the codebase has unused imports within some of its source files. The following list showcases some of them:

deposit.rs/#L5

```
use crate::curve::CONSTANT_SNAP_RATE;
```

deposit.rs/#L10

```
use std::ops::Deref;
```

initialize_metadata.rs/#L1-L2

```
use std::ops::Deref;
use crate::curve::CurveCalculator;
```

```rust
use crate::error::ErrorCode;
```

initialize_metadata.rs/#L21-L22

```rust
use solana_program::program::invoke_signed;
use spl_associated_token_account::create_associated_token_account;
```

initialize.rs#L16-L19

```rust
use anchor_spl::metadata::create_metadata_accounts_v3;
use anchor_spl::metadata::mpl_token_metadata::types::DataV2;
use anchor_spl::metadata::CreateMetadataAccountsV3;
use anchor_spl::metadata::Metadata;
```

initialize.rs#L27

```rust
use spl_associated_token_account::create_associated_token_account;
```

initialize.rs#L30

```rust
use std::ops::DerefMut;
```

withdraw.rs#L5

```rust
use crate::curve::CONSTANT_SNAP_RATE;
```

withdraw.rs#L7

```rust
use crate::utils::*;
```

withdraw.rs#L11

```rust
use std::ops::Deref;
```

**Recommendation:**

Remove the unused imports.

| | | |
|---|---|---|
| **[SNAP-11]** | The *Fees::trading_fee* function is not being used anywhere | Informational |

**Details:**

The Fees::trading_fee function is not being anywhere within the codebase as of its current state. This means that it can safely be removed.

**Recommendation:**

Consider removing the Fees::trading_fee function.

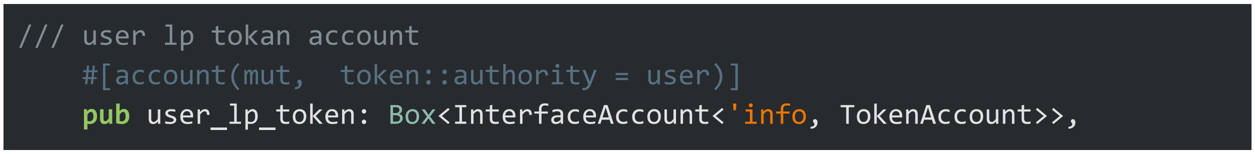
| | | |
|---|---|---|
| **[SNAP-12]** | Typos | Informational |

**Details:**

During our audit of the codebase, we came across some typos within its code comments, that are worth mentioning:

deposit.rs#L45

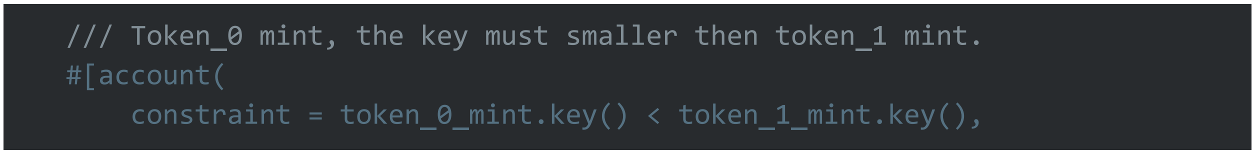
tokan → token



```
/// user lp tokan account
    #[account(mut,  token::authority = user)]
    pub user_lp_token: Box<InterfaceAccount<'info, TokenAccount>>,
```

initialize.rs#L75

must → must be



```
    /// Token_0 mint, the key must smaller then token_1 mint.
    #[account(
        constraint = token_0_mint.key() < token_1_mint.key(),
        mint::token_program = token_0_program,
    )]
    pub token_0_mint: Box<InterfaceAccount<'info, Mint>>,
```

**Recommendation:**

Fix the above mentioned typos.

| | | |
|---|---|---|
| **[SNAP-13]** | Redundant functions | Informational |

**Details:**

The current implementations of both the Fees::fund_fee and Fees::protocol_fee functions hold the exact same functionality as one another. This is redundant and can be avoided by creating a function with a more generalized name that is going to be used in the place of those two.

```
  /// Calculate the owner trading fee in trading tokens
    pub fn protocol_fee(amount: u128, protocol_fee_rate: u64) ->
Option<u128> {
        floor_div(
            amount,
            u128::from(protocol_fee_rate),
            u128::from(FEE_RATE_DENOMINATOR_VALUE),
        )
    }

    /// Calculate the owner trading fee in trading tokens
    pub fn fund_fee(amount: u128, fund_fee_rate: u64) -> Option<u128> {
        floor_div(
            amount,
            u128::from(fund_fee_rate),
            u128::from(FEE_RATE_DENOMINATOR_VALUE),
        )
    }
```

**Recommendation:**

Consider creating a function with a more generalized name with the same functionality as Fees::fund_fee and Fees::protocol_fee, and use it in the places where those two are currently being used.

| | | |
|---|---|---|
| **[SNAP-14]** | Commented out code | Informational |

**Details:**

Commented out code is generally always thought of as a bad coding practice, due to the fact that it makes all source code around it harder to comprehend. There are a few instances of such code that were identified during our security review of Snapper:

collect_fund_fee.rs#L13

```
 // #[account(constraint = (owner.key() == amm_config.fund_owner ||
owner.key() == crate::admin::id()) @ ErrorCode::InvalidOwner)]
    #[account(constraint = (owner.key() == crate::admin::id()) @
ErrorCode::InvalidOwner)]
```

snapper.rs#L61-L106

```
// pub fn get_buy_price(&self, tokens: u128) -> Option<u128> {
    //     if tokens == 0 || tokens > self.virtual_token_reserves {
    //         return None;
    //     }
    //     let product_of_reserves =
self.virtual_liquidity.checked_mul(self.virtual_token_reserves)?;
    //     let new_virtual_token_reserves =
self.virtual_token_reserves.checked_sub(tokens)?;
    //     let new_virtual_liquidity =
product_of_reserves.checked_div(new_virtual_token_reserves)?.checked_add(1)
?;
    //     let amount_needed =
new_virtual_liquidity.checked_sub(self.virtual_liquidity)?;
    //     Some(amount_needed)
    // }

    // pub fn buy_snap(&mut self, delta_r: u128) -> Option<u128> {
    //     let  = self.r;
    //     let  = r.checked_add(delta_r).unwrap();
    //     let  = self.rv.checked_div(new_r).unwrap();
    //     let  = self.v.checked_sub(new_v).unwrap();
    //     self.r = ;
    //     self.v = ;
    //     self.rv = self.r.checked_mul(self.v).unwrap();
    //     Some(delta_v)
    // }

    // pub fn apply_sell(&mut self, delta_r: u128) -> Option<u128> {
    //     let r = self.r;
    //     let new_r = r.checked_sub(delta_r).unwrap();
    //     let new_v = self.rv.checked_div(new_r).unwrap();
    //     let delta_v = new_v.checked_sub(new_r).unwrap();
    //     self.r = new_r;
    //     self.v = new_v;
    //     self.rv = self.r.checked_mul(self.v).unwrap();
    //     Some(delta_v)
    // }

    // pub fn get_sell_price(&self, tokens: u128) -> Option<u128> {
    //     if tokens == 0 || tokens > self.virtual_token_reserves {
    //         return None;
    //     }

    //     let scaling_factor = self.initial_virtual_token_reserves;

    //     let scaled_tokens = tokens.checked_mul(scaling_factor)?;
    //     let token_sell_proportion =
scaled_tokens.checked_div(self.virtual_token_reserves)?;
    //     let sol_received =
(self.virtual_liquidity.checked_mul(token_sell_proportion)?).checked_div(sc
aling_factor)?;

    //     Some(sol_received.min(self.real_sol_reserves))
    // }
```

**Recommendation:**

Remove the commented code.

# Overall Assessment

Once the High and Medium severity vulnerabilities are addressed, Snapper DEX's smart contracts are safe and pose no risks to the protocol and its users.

| Severity | Count | Fixes |
|---|---|---|
| Critical | 0 | - |
| High | 2 | Fixed |
| Medium | 2 | Acknowledged |
| Low | 2 | Acknowledged |
| Informational | 7 | Acknowledged |

# Recommendations

Audita Security has put forward the following recommendations for Snapper DEX:

- When performing the calculations for the input amounts in the deposit function, round up instead of down.
- In both swap functions, use the total token amounts without the swap fees for the constant product calculations.
- Increase OBSERVATION_NUM to 150–200, allowing the system to retain a broader historical window without overwriting data too quickly.
- Reduce the OBSERVATION_UPDATE_DURATION_DEFAULT to around 5 seconds to align better with Solana's block time and capture more granular price data. Increase OBSERVATION_NUM to 150–200 to accommodate the additional observations generated by a shorter update interval while maintaining a sufficient historical window for TWAP calculations.
- Update the packages to the appropriate versions as specified in the references:
    - https://rustsec.org/advisories/RUSTSEC-2024-0344
    - https://rustsec.org/advisories/RUSTSEC-2022-0093
- Consider writing an integration testing suite that verifies that all of the functionality of the protocol works as it should.
- Remove the unused variables.
- Remove the unused imports.
- Consider removing the Fees::trading_fee function.
- Fix the above mentioned typos.
- Consider creating a function with a more generalized name with the same functionality as Fees::fund_fee and Fees::protocol_fee, and use it in the places where those two are currently being used.
- Remove the commented code.

# Fixes

Snapper DEX' team is dedicated and responsive, cooperating to acknowledge and implement the above recommendations.

**Fixed issues:**

[SNAP-01] - High

[SNAP-02] - High

# Disclaimer

This audit makes no statements or warranties on the security of the code. While we have conducted the analysis to our best abilities and produced this report in line with latest industry developments, it is important to not rely on this report only. In order for contracts to be considered as safe as possible, the industry standard requires them to be checked by several independent auditing bodies. Those can be other audit firms or public bounty programs.

Smart contract platforms, their programming languages, and other software components are not immune to vulnerabilities that can be exploited by hackers. As a result, although a smart contract audit can help identify potential security issues, it cannot provide an absolute guarantee of the audited smart contract's security.