# TSwap Protocol Audit Report

Version 1.0

*QV.io*

May 4, 2024

# TSWAP Protocol Audit Report

QV

May 4, 2024

Prepared by: [QV]

## Table of Contents

- Medium
  * [M-1] `TSwapPool::deposit` is missing deadline check causing extended transaction duration
- Low
  * [L-1] `TSwapPool::LiquidityAdded` has parameter in a wrong order, swapping the third parameter to the second one
- Informational
  * [I-1]: **`public`** functions not used internally could be marked `external`
  * [I-2]: Define and use `constant` variables instead of using literals
  * [I-3]: Event is missing `indexed` fields
  * [I-4]: PUSH0 is not supported by all chains
  * [I-5]: Large literal values multiples of 10000 can be replaced with scientific notation

## Protocol Summary

T-Swap as a decentralized asset/token exchange (DEX) and is known as an Automated Market Maker (AMM)because it doesn't use a normal "order book" style exchange, instead it uses "Pools" of an asset.

## Disclaimer

The team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

## Risk Classification

| | | Impact | | |
|---|---|---|---|---|
| | | High | Medium | Low |
| | High | H | H/M | M |
| Likelihood | Medium | H/M | M | M/L |

| Impact | | | |
|---|---|---|---|
| Low | M | M/L | L |

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

## Audit Details

### Scope

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda
- In Scope:

```
./src/
#-- PoolFactory.sol
#-- TSwapPool.sol
```

- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum
- Tokens:
  - Any ERC20 token

### Roles

Liquidity Providers: Users who have liquidity deposited into the pools. Their shares are represented by the LP ERC20 tokens. They gain a 0.3% fee every time a swap is made. Users: Users who want to swap tokens.

### Executive Summary

This project is meant to be a permissionless way for users to swap assets between each other at a fair price. You can think of T-Swap as a decentralized asset/token exchange (DEX). T-Swap is known as an Automated Market Maker (AMM)

## Issues found

| Severtity | # of issues found |
|---|---|
| High | 4 |
| Medium | 1 |
| Low | 1 |
| Info | 4 |
| Total | 10 |

## High

### [H-1] `TSwapTool::_swap` the extra token given to users after every `swapCount` break to protocol invariant of `x*y  =k`

**Description** The natspec mentioned that every 10 swaps the user receive extra token as an incentive, meaning that the invariant k will be break if the attacker conduct 10+ swaps in a row. Protocal token fund will be drained over time.

```
        swap_count++;
        if (swap_count >= SWAP_COUNT_MAX) {
            swap_count = 0;
            outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000)
                ;
        }
```

**Impact** Protocal core invariant broken and all fund would be gone after a numbers of swaps

**Proof of Concept** Place the following test into `TSwapPool.t.sol` 1. An user swaps 10 times in order to receive 1`_000_000_000_000_000_000` incentive token 2. That user continues doing so untill all the protocol funds are drained.

```
function testInvariantBroken() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        uint256 outputWeth = 1e17;
        int256 startingY = int256(weth.balanceOf(address(pool)));
        int256 expectedDeltaY = int256(-1) * int256(outputWeth);


        vm.startPrank(user);
        poolToken.approve(address(pool),type(uint256).max);
        poolToken.mint(user,100e18);
        pool.swapExactOutput(poolToken,weth,outputWeth,uint64(block.
           timestamp));
        pool.swapExactOutput(poolToken,weth,outputWeth,uint64(block.
           timestamp));
        pool.swapExactOutput(poolToken,weth,outputWeth,uint64(block.
           timestamp));
        pool.swapExactOutput(poolToken,weth,outputWeth,uint64(block.
           timestamp));
        pool.swapExactOutput(poolToken,weth,outputWeth,uint64(block.
           timestamp));
        pool.swapExactOutput(poolToken,weth,outputWeth,uint64(block.
           timestamp));
        pool.swapExactOutput(poolToken,weth,outputWeth,uint64(block.
           timestamp));
        pool.swapExactOutput(poolToken,weth,outputWeth,uint64(block.
           timestamp));
        pool.swapExactOutput(poolToken,weth,outputWeth,uint64(block.
           timestamp));
        pool.swapExactOutput(poolToken,weth,outputWeth,uint64(block.
           timestamp));


        vm.stopPrank();

        uint256 endingY = weth.balanceOf(address(pool));

        int256 actualDeltaY = int256(endingY) - int256(startingY);

        assertEq(actualDeltaY,expectedDeltaY);
    }
```

**Recommended Mitigation** Reove the incentive to make sure protocal invariant is not broken

```
-     swap_count++;
-     if (swap_count >= SWAP_COUNT_MAX) {
-         swap_count = 0;
-           outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000
    );
-     }
```

### [H-2] Fee calculation in the `TSwapPool::deposit` is wrong, causing user need to put more token input for a given amount of token output.

**Description** The `getInputAmountBasedOnOutput` scales the amount by 10_000 instead of 1_000. **Impact** User pay much more fees (10 times more) **Recommended Mitigation**

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
        return
-        ((inputReserves * outputAmount) * 10000) /
+        ((inputReserves * outputAmount) * 1000) /
        ((outputReserves - outputAmount) * 997);
    }
```

### [H-3] Lack of slippage proctection in `TSwapPool::swapExactOutput` causes users to potentially receive way fewer tokens

**Description** The `SwapExactOutput` function does not include any sort of slipage protection. In oder works, no protection for users when they swap token when market condition is unflavorable. **Impact** Users postion highly affected if the market changes during the course of the transaction. **Proof of Concept** 1. The current price of 1 WETH is 1,000 USDC 2. User input a `swapExactOutput` looking for 1 WETH 1. inputtoken = USDC 2. outputtoken = WETH 3. outputAmount = 1 4. dealine = whatever 3. The fuction does not offer slippage protection: maxInputAmount 4. As the transaction is pending in the mempool, the market price of 1 WETH is now 5,000 USDC. 5. Transaction completed. User send the protocol 5,000 USDC instead of 1,000 USDC.

**Recommended Mitigation** Consider adding slippage protection `maxInputAmout` to the `swapExactOutput` to protect users so they can predict max amout they will spend on the swap.

```
    function swapExactOutput(
        IERC20 inputToken,
        IERC20 outputToken,
        uint256 outputAmount,
        uint64 deadline
+       uint256 maxInputAmout,

    )
        public
        revertIfZero(outputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 inputAmount)
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));
+        if (inputAmount > maxInputAmount){
            revert();
        }

        inputAmount = getInputAmountBasedOnOutput(
            outputAmount,
            inputReserves,
            outputReserves
        );

        _swap(inputToken, inputAmount, outputToken, outputAmount);
    }
```

**[H-4] The `sellPoolTokens` function is intended to allow users to easily sell pool tokens and receive WETH in exchange. Users indicate how many pool tokens they're willing to sell in the `poolTokenAmount` parameter. However, the function currently miscalculaes the swapped amount.**

This is due to the fact that the swapExactOutput function is called, whereas the swapExactInput function is the one that should be called. Because users specify the exact amount of input tokens, not output. Impact: Users will swap the wrong amount of tokens, which is a severe disruption of protcol functionality.

**Proof of Concept**

**Recommended Mitigation**

Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require changing the `sellPoolTokens` function to accept a new parameter (ie `minWethToReceive` to be passed to `swapExactInput`)

```
    function sellPoolTokens(
        uint256 poolTokenAmount,
+        uint256 minWethToReceive,
        ) external returns (uint256 wethAmount) {
-         return swapExactOutput(i_poolToken, i_wethToken,
    poolTokenAmount, uint64(block.timestamp));
+         return swapExactInput(i_poolToken, poolTokenAmount,
    i_wethToken, minWethToReceive, uint64(block.timestamp));
    }
```

## Medium

### [M-1] `TSwapPool::deposit` is missing deadline check causing extended transaction duration

**Description:** The `deadline` parameter can be manipulated to extend transection duration. If someone set the dealine is, let's say, next block, they could still have time to deposit.

**Impact:** Failed depost will go through because deadline is extended.

**Proof of Concept:** The `deadline` parameter is unused.

**Recommended Mitigation:** Consider making the following change to the function

```

      function deposit(
          uint256 wethToDeposit,
          uint256 minimumLiquidityTokensToMint,
          uint256 maximumPoolTokensToDeposit,
          uint64 deadline
      )
          external
+         revertIfDeadlinePasssed(deadline)
          revertIfZero(wethToDeposit)
          returns (uint256 liquidityTokensToMint)
```

## Low

### [L-1] `TSwapPool::LiquidityAdded` has parameter in a wrong order, swapping the third parameter to the second one

**Description** When `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTransfe` fuction, it logs values in the wrong order.

**Impact** Event emission is incorrect, leading to off-chain functions potentially malfuctioning.

**Recommended Mitigation**

```
-  emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
+  emit LiquidityAdded(msg.sender, wethToDeposit,poolTokensToDeposit);
```

## Informational

### [I-1]: `public` functions not used internally could be marked `external`

Instead of marking a function as `public`, consider marking it as `external` if it is not used internally.

- Found in src/TSwapPool.sol Line: 305

```
    function swapExactInput(
```

### [I-2]: Define and use `constant` variables instead of using literals

If the same constant literal value is used multiple times, create a constant state variable and reference it throughout the contract.

- Found in src/TSwapPool.sol Line: 283

```
        uint256 inputAmountMinusFee = inputAmount * 997;
```

- Found in src/TSwapPool.sol Line: 302

```
            ((outputReserves - outputAmount) * 997);
```

- Found in src/TSwapPool.sol Line: 464

```
                1e18,
```

- Found in src/TSwapPool.sol Line: 473

```
                1e18,
```

### [I-3]: Event is missing `indexed` fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

- Found in src/PoolFactory.sol Line: 35

```
    event PoolCreated(address tokenAddress, address poolAddress);
```

- Found in src/TSwapPool.sol Line: 52

```
    event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
    event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
    event Swap(
```

#### [I-4]: PUSH0 is not supported by all chains

Solc compiler version 0.8.20 switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain other than mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.

- Found in src/PoolFactory.sol Line: 15

```
pragma solidity 0.8.20;
```

- Found in src/TSwapPool.sol Line: 15

```
pragma solidity 0.8.20;
```

#### [I-5]: Large literal values multiples of 10000 can be replaced with scientific notation

Use `e` notation, for example: `1e18`, instead of its full numeric value.

- Found in src/TSwapPool.sol Line: 301

```
            ((inputReserves * outputAmount) * 10000) /
```