MK
Smart Contract
Audits

# TSwap Protocol Audit Report

Version 1.0

*Kotte Mohan krishna*

May 28, 2025

# Protocol Audit Report

Kotte Mohan Krishna

May, 2025

Prepared by: Mohan Kotte Lead Security Researcher: Mohan Kotte

## Table of Contents

    * [H-5] In `TSWpPool::_swap` the extra tokens given to users after every `swapCount` breaks the protocol invariant of `x * y = k`
  - Low
    * [L-1] `TSWapPool::LiquidityAdded` event has paramaters out of order, cause event to emit incorrecr information
    * [L-2] Default value returned by `TSwap::swapExactInput` results in incorrect return value given
  - Informationals
    * [I-1] `PoolFatcory::PoolFactory__PoolDoesNotExist` is not used and should be removed
    * [I-2] `PoolFactory::Constructor` in Lacking Zero address checks
    * [I-3] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`
    * [I-4] Event is missing `indexed` fields
    * [I-5] `TSwapPool::Constructor` in Lacking Zero address checks

## Protocol 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) because it doesn't use a normal "order book" style exchange, instead it uses "Pools" of an asset. It is similar to Uniswap. To understand Uniswap, please watch this video: Uniswap Explained

The protocol starts as simply a PoolFactory contract. This contract is used to create new "pools" of tokens. It helps make sure every pool token uses the correct logic. But all the magic is in each TSwapPool contract.

You can think of each TSwapPool contract as it's own exchange between exactly 2 assets. Any ERC20 and the WETH token. These pools allow users to permissionlessly swap between an ERC20 that has a pool and WETH. Once enough pools are created, users can easily "hop" between supported ERC20s.

## Disclaimer

The Mohan Kotte 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 |
| Likelihood | High | H | H/M | M |
| | Medium | H/M | M | M/L |
| | Low | M | M/L | L |

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

## Audit Details

**The findings described in this document correspond the following commit hash:**

```
1ec3c30253423eb4199827f59cf564cc575b46db
```

### Scope

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

### 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

### Issues found

| severity | Number of issues found |
|---|---|
| High | 5 |
| Medium | 0 |
| Low | 2 |
| Info | 5 |
| Total | 12 |

## High

### [H-1] `TSwapPool::Deposit` is missing deadline check causing transactions to complete even after the deadline

**Description:** The `deposit` function accepts the dealine parameter which according to the documenting is the "The dealine for the transaction to completed by" However, this parameter is never used As a consequence, operations that add liquidity to the pool might be executed at unxexpected times, in the market conditions where the deposit rate is unfavourable

**Impact:** Transactions could be sent when maket conditions are unfavourable to deposit, even when adding a deadline parameter

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

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

### [H-2] Incorrect Fee Calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users as a fee

**Description:** The`getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens a user should deposit given amount of tokens of output tokens. However, the function currently miscalculates the resulting amount. when the calculating the fee, it scales the amount by 10_000 instaed of 1_000. **Impact:** Protocol takes more fees thean expected from users

**Recommended Mitigation:**

```
  function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {

        return
-            ((inputReserves * outputAmount) * 10000) /
-            ((outputReserves - outputAmount) * 997);
+ ((inputReserves * outputAmount) * 1000) /
+          ((outputReserves - outputAmount) * 997);
    }
```

**Proof of Concept:**

Code

```

       function testFeeCalculationMisMatchedSwapExactoutput() public {
        uint256 initialLiquidity = 100e18;
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), initialLiquidity);
        poolToken.approve(address(pool),initialLiquidity);

        pool.deposit({
            wethToDeposit: initialLiquidity,
            minimumLiquidityTokensToMint: 0,
            maximumPoolTokensToDeposit: initialLiquidity,
            deadline: uint64(block.timestamp)
        });
        vm.stopPrank();

        address someUser = makeAddr("SomeUser");
        uint256 UserInitialPoolTokenBalance = 11e18;
        poolToken.mint(someUser, UserInitialPoolTokenBalance);

        vm.startPrank(someUser);
        poolToken.approve(address(pool),type(uint256).max);
        pool.swapExactOutput(
            poolToken,
            weth,
            1 ether,
            uint64(block.timestamp)
        );
        assertLt(poolToken.balanceOf(someUser), 1 ether);
        vm.stopPrank();

        vm.startPrank(liquidityProvider);
        pool.withdraw(
            pool.balanceOf(address(liquidityProvider)),
            1,
            1,
            uint64(block.timestamp)
        );

        assertEq(weth.balanceOf(address(pool)),0);
        assertEq(poolToken.balanceOf(address(pool)),0);
    }
```

#### [H-3] There is No slippage protection in `TSwapPool.swapExactOutput` causes users to potentially receive way fewer tokens

**Description:** The `swapExactOutput` function doesnot include any sort of slippage protection. This function is similar to what is done in `TSwapPool::swapExactInput`, where the function specifies a `minOutPutAmount`, The `swapExactOutput` function should specify a `maxInputAmount`.

**Impact:** If Market conditions changes before the transaction processes, the user could get a much worse swap.

**Proof of Concept:** 1. The price of 1 weth right now is 1,000 USDC 2. User inputs a `swapExactOutput` looking for 1 WETH 1. inputToken = USDC 2. outputToken = WETH 3. outputAmount = 1 4. deadline = some dealine 3. The function doesnt offer a maxInput amount 4. As the transaction is pending in the mempool, the market changes! And he price moves Huge -> 1 WETH is now 10,000 USDC. 10x more than the user expected 5. The Transaction complete, but the user sent the protocol 10,000 USDC instead of expected 1,000 USDC

**Recommended Mitigation:** We should include a `maxInputAmount` so the user only has to spend up to specific amount, and can predict how much they will spend on the protocol

```

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

```
    )
    inputAmount = getInputAmountBasedOnOutput(
            outputAmount,
            inputReserves,
            outputReserves
        );
+   if(inputAmount > maxInputAmount){
+    revert()
+    }
```

### [H-4] `TSwapPool::sellPoolTokens` mismacthes input and output tokens causing users to receive the incorrect amount of tokens

**Description:** 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 miscalculates the swapped amount.

This is due to the fact that the `swapExactOutput` function is called where as the `swapExactInput` function is the one that should be called. Because users specify the exact amount of input tokens, not output tokens

**Impact:** Users will swap the wrong a amount of tokens, which is a severe disruption of protocol functionality

**Proof of Concept:**

Code

```
function testIncorrectSellTokenFunctions() public {
      uint256 initialLiquidity = 100e18;
      vm.startPrank(liquidityProvider);
      weth.approve(address(pool), initialLiquidity);
      poolToken.approve(address(pool),initialLiquidity);

      pool.deposit({
          wethToDeposit: initialLiquidity,
          minimumLiquidityTokensToMint: 0,
          maximumPoolTokensToDeposit: 50e18,
          deadline: uint64(block.timestamp)
      });
      vm.stopPrank();

      address someUser = makeAddr("SomeUser");
      uint256 UserInitialPoolTokenBalance = 1e18;
      poolToken.mint(someUser, UserInitialPoolTokenBalance);

```

```
        vm.startPrank(someUser);
        poolToken.approve(address(pool),type(uint256).max);
        pool.sellPoolTokens(UserInitialPoolTokenBalance);
        assertGt(weth.balanceOf(someUser), 1.5 ether);
        vm.stopPrank();
    }
```

**Recommended Mitigation:**

Consider changing the implmentation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require to change the parameters of `TSwapPool::sellPoolTokens` function (i.e `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));
-              swapExactInput(i_poolToken,poolTokenAmount,i_wethToken,
   minWethToReceive, uint64(block.timestamp));
    }
```

### [H-5] In `TSWpPool::_swap` the extra tokens given to users after every `swapCount` breaks the protocol invariant of `x * y = k`

**Description:** The protocol follows of strict invariant of `x * y = k`. Where:

- `x`: The balance of the pool token
- `y`: The balance of WETH
- `k`: The constant product of the two balances

  This means, that whenever the balances change in the protocol, the ratio between the two amounts should remain constant, hence the `k`. However, this is broken due to the extra incentive in the `_swap` function. Meaning that over time the protocoll funds will be drained.

  The FOllowing code is reponisble for the issue

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

> **Impact:** A user could maliciously drain the protocol of funds by doing a lot of swaps and collecting the extra incentive given out by the protocol

Most simply put, the protocol's core invariant is broken

**Proof of Concept:** 1. A user swaps 10 times, and collects the extra incentives of 1`_000_000_000_000_000_000` 2. That user continues to swap until all the protocol funds are drained

Code

```
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;

       vm.startPrank(user);
        poolToken.approve(address(pool),type(uint64).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));

       int256 startingY = int256(weth.balanceOf(address(pool)));
       int256  expectedDeltaY = int256(-1) * int256(outputWeth);
        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:** Remove the extra incentive Mechanism. If you want to keep this in, we should account for the change in the `x * y = k` protocol invariant. Or, we should set aside tokens in the same way we do with the fees.

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

### Low

#### [L-1] `TSwapPool::LiquidityAdded` event has paramaters out of order, cause event to emit incorrecr information

**Description:** When `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTransfe` funtion, it logs values in incorrect positions. The `PoolTokensToDeposit` value should go in the third parameter position, whereas the `WethToDeposit` value should go second.

**Impact:** Event emission is incorrect, leading to off-chain funcions potentially malfunctioning

**Recommended Mitigation:**

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

#### [L-2] Default value returned by `TSwap::swapExactInput` results in incorrect return value given

**Description:** The `TSwap::swapExactInput` function is expected to return the actual amount of tokens bought by the caller. However, while it declares the named return value `output` it is never assigned a value nor uses an explict return statement

**Impact:** The return value always be 0, giving incorrect information to User

**Proof of Concept:**

Code

```

        function testFeeCalculationMisMatchedSwapExactoutput() public {
```

```
        uint256 initialLiquidity = 100e18;
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), initialLiquidity);
        poolToken.approve(address(pool),initialLiquidity);

        pool.deposit({
            wethToDeposit: initialLiquidity,
            minimumLiquidityTokensToMint: 0,
            maximumPoolTokensToDeposit: initialLiquidity,
            deadline: uint64(block.timestamp)
        });
        vm.stopPrank();

        address someUser = makeAddr("SomeUser");
        uint256 UserInitialPoolTokenBalance = 11e18;
        poolToken.mint(someUser, UserInitialPoolTokenBalance);

        vm.startPrank(someUser);
        poolToken.approve(address(pool),type(uint256).max);
        pool.swapExactOutput(
            poolToken,
            weth,
            1 ether,
            uint64(block.timestamp)
        );
        assertLt(poolToken.balanceOf(someUser), 1 ether);
        vm.stopPrank();

        vm.startPrank(liquidityProvider);
        pool.withdraw(
            pool.balanceOf(address(liquidityProvider)),
            1,
            1,
            uint64(block.timestamp)
        );

        assertEq(weth.balanceOf(address(pool)),0);
        assertEq(poolToken.balanceOf(address(pool)),0);
    }
    function testUnknownOutputVaribleInSwapExactIntput() public {
        uint256 initialLiquidity = 100e18;
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), initialLiquidity);
        poolToken.approve(address(pool),initialLiquidity);

        pool.deposit({
            wethToDeposit: initialLiquidity,
            minimumLiquidityTokensToMint: 0,
            maximumPoolTokensToDeposit: initialLiquidity,
            deadline: uint64(block.timestamp)
        });
        vm.stopPrank();

        address someUser = makeAddr("SomeUser");
        uint256 UserInitialPoolTokenBalance = 11e18;
        poolToken.mint(someUser, UserInitialPoolTokenBalance);

        vm.startPrank(someUser);
        poolToken.approve(address(pool),type(uint256).max);
        uint256 output = pool.swapExactInput(
            poolToken,
            1 ether,
            weth,
            1 ether,
            uint64(block.timestamp)
        );
       assertEq(output, 1 ether);
        vm.stopPrank();
    }
```

**Recommended Mitigation:**

```
    function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
    // report-written This should be external
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (
            uint256 output)
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

-       uint256 outputAmount = getOutputAmountBasedOnInput(
             inputAmount,
             inputReserves,
             outputReserves
-        );

+        output = getOutputAmountBasedOnInput(
+            inputAmount,
+            inputReserves,
+            outputReserves
+        );

-        if (outputAmount < minOutputAmount) {
```

```
-             revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
-         }

+         if (output < minOutputAmount) {
+            revert TSwapPool__OutputTooLow(output, minOutputAmount);
+         }

-         _swap(inputToken, inputAmount, outputToken, outputAmount);
+         _swap(inputToken, inputAmount, outputToken, output);
      }
```

### Informationals

#### [I-1] `PoolFatcory::PoolFactory__PoolDoesNotExist` is not used and should be removed

```
- error PoolFactory__PoolDoesNotExist(address tokenAddress);
```

#### [I-2] `PoolFactory::Constructor` in Lacking Zero address checks

```
- constructor(address wethToken) {
+      if(wethToke == address(0)){
+         revert()
+     }
      i_wethToken = wethToken;
  }
```

#### [I-3] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`

```
-   string memory liquidityTokenSymbol = string.concat("ts", IERC20(tokenAddress).name());
+   string memory liquidityTokenSymbol = string.concat("ts", IERC20(tokenAddress).symbol());
```

#### [I-4] Event is missing `indexed` fields

**Description:** 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: 1307:61:31
- Found in src/TSwapPool.sol: 1897:108:10
- Found in src/TSwapPool.sol: 2010:110:10
- Found in src/TSwapPool.sol: 2125:116:10

### [I-5] `TSwapPool::Constructor` in Lacking Zero address checks

```
- constructor(address wethToken) {
+      if(wethToken == address(0)){
+         revert()
+       }
        i_wethToken = wethToken;
    }

     constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
+      if(i_wethToken == address(0) || i_wethToken == address(0)){
+         revert()
+       }
        i_wethToken = IERC20(wethToken);
        i_wethToken = IERC20(poolToken);
    }
```