



# TSwap Protocol Audit Report

Version 1.0

*Moon.io*

October 28, 2025

# Protocol Audit Report

Moon.io

March 7, 2023

Prepared by: Moon.io

Lead Auditors: Divyansh (ELO\_Anxiety)

## Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
- High
- Medium
- Low
- Informational

## 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](#)

## TSwap Pools

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.

For example: 1. User A has 10 USDC 2. They want to use it to buy DAI 3. They [swap](#) their 10 USDC -> WETH in the USDC/WETH pool 4. Then they [swap](#) their WETH -> DAI in the DAI/WETH pool

Every pool is a pair of [TOKEN X & WETH](#).

There are 2 functions users can call to swap tokens in the pool. - [swapExactInput](#) - [swapExactOutput](#)

We will talk about what those do in a little.

## Liquidity Providers

In order for the system to work, users have to provide liquidity, aka, "add tokens into the pool".

### Why would I want to add tokens to the pool?

The TSwap protocol accrues fees from users who make swaps. Every swap has a 0.3 fee, represented in [getInputAmountBasedOnOutput](#) and [getOutputAmountBasedOnInput](#). Each applies a 997 out of 1000 multiplier. That fee stays in the protocol.

When you deposit tokens into the protocol, you are rewarded with an LP token. You'll notice [TSwapPool](#) inherits the [ERC20](#) contract. This is because the [TSwapPool](#) gives out an ERC20 when

Liquidity Providers (LP)s deposit tokens. This represents their share of the pool, how much they put in. When users swap funds, 0.03% of the swap stays in the pool, netting LPs a small profit.

### LP Example

1. LP A adds 1,000 WETH & 1,000 USDC to the USDC/WETH pool
  1. They gain 1,000 LP tokens
2. LP B adds 500 WETH & 500 USDC to the USDC/WETH pool
  1. They gain 500 LP tokens
3. There are now 1,500 WETH & 1,500 USDC in the pool
4. User A swaps 100 USDC -> 100 WETH.
  1. The pool takes 0.3%, aka 0.3 USDC.
  2. The pool balance is now 1,400.3 WETH & 1,600 USDC
  3. aka: They send the pool 100 USDC, and the pool sends them 99.7 WETH

Note, in practice, the pool would have slightly different values than 1,400.3 WETH & 1,600 USDC due to the math below.

### Core Invariant

Our system works because the ratio of Token A & WETH will always stay the same. Well, for the most part. Since we add fees, our invariant technically increases.

$$x * y = k \quad x = \text{Token Balance X} \quad y = \text{Token Balance Y} \quad k = \text{The constant ratio between X \& Y}$$

### Variable Definitions

- $y$  = Token Balance Y
- $x$  = Token Balance X
- $x * y = k$
- $x * y = (x + \Delta x) * (y - \Delta y)$
- $\Delta x$  = Change of token balance X
- $\Delta y$  = Change of token balance Y
- $\beta = (\Delta y / y)$
- $\alpha = (\Delta x / x)$

### Final invariant equation without fees:

- $\Delta x = (\beta/(1 - \beta)) * x$
- $\Delta y = (\alpha/(1 + \alpha)) * y$

### Invariant with fees

- $\rho = \text{fee}$  (between 0 & 1, aka a percentage)
- $\gamma = (1 - \rho)$  (pronounced gamma)
- $\Delta x = (\beta/(1 - \beta)) * (1/\gamma) * x$
- $\Delta y = (\alpha\gamma/(1 + \alpha\gamma)) * y$  Our protocol should always follow this invariant in order to keep swapping correctly!

### Make a swap

After a pool has liquidity, there are 2 functions users can call to swap tokens in the pool. - `swapExactInput` - `swapExactOutput`

A user can either choose exactly how much to input (ie: I want to use 10 USDC to get however much WETH the market says it is), or they can choose exactly how much they want to get out (ie: I want to get 10 WETH from however much USDC the market says it is).

*This codebase is based loosely on Uniswap v1*

### Disclaimer

The Moon 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
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:

```
1 ./src/
2 #-- PoolFactory.sol
3 #-- 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

### Issues found

Severity	Number of issues found
High	4
Medium	2
Low	2
Info/Gas	10
Total	18

## Findings

### High

**[H-1] Incorrect Fee calculation in TSwapPool::getInputAmountBasedOnOutput causes protocol to take too many tokens from the users, resulting in lost fees.**

**Description:** The `getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens a user should deposit given an amount of output tokens. However, the function currently miscalculates the resulting amount. When calculating the fee, scales the amount by 10\_000 instead of 1\_000.

**Impact:** Protocol takes more fees than expected from users.

#### Proof of Code:

##### CODE

```

1   function testGetInputAmountBasedOnOutput() public {
2       // liquidity provider providing liquidity
3       vm.startPrank(liquidityProvider);
4       weth.approve(address(pool), 100e18);
5       poolToken.approve(address(pool), 100e18);
6       pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
7       vm.stopPrank();
8
9       // getting the reserves
10      vm.startPrank(user);
11      uint256 inputReserves = weth.balanceOf(address(pool));
12      uint256 outputReserves = poolToken.balanceOf(address(pool));
13
14      //assume that the user wants to get 10 pool tokens, for that how
15      // much input is needed will be returned by this function
16      uint256 outputTokenNeed = 10e18;

```

```

16         uint256 expectedInputNeeded = ((inputReserves * outputTokenNeed
17             ) *
18             1000) / ((outputReserves - outputTokenNeed) * 997);
19
20         // result returned by the function `getInputAmountBasedOnOutput
21
22         uint256 actualInputAmountByFunction = pool.
23             getInputAmountBasedOnOutput(
24                 outputTokenNeed,
25                 inputReserves,
26                 outputReserves
27             );
28             vm.stopPrank();
29
30         assert(actualInputAmountByFunction > expectedInputNeeded);
31     }

```

### Recommended Mitigation:

```

1     function getInputAmountBasedOnOutput(
2         uint256 outputAmount,
3         uint256 inputReserves,
4         uint256 outputReserves
5     )
6     public
7     pure
8     revertIfZero(outputAmount)
9     revertIfZero(outputReserves)
10    returns (uint256 inputAmount)
11    {
12 -     return ((inputReserves * outputAmount) * 10_000) /(
13 +     outputReserves - outputAmount) * 997);
14    }

```

## [H-2] Lack of slippage protection in TSwapPool::swapExactOutput causes users to potentially receive way fewer tokens

**Description:** The `swapExactOutput` function does not 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` should also specify a `maxInputAmount`.

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

**Proof of Concept:** 1. 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 = whatever 3. The function does not offer a `maxInputAmount` 4. As the transaction is pending in the mempool, the market changes! and the price moves HUGE -> 1WETH = 10,000USDC. 10x more than the user expected. 5. The transaction completes, but the user sends 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 a specific amount, and can predict how much they will spend on the protocol.

```

1      function swapExactOutput(
2          IERC20 inputToken,
3          IERC20 outputToken,
4          uint256 outputAmount,
5          uint256 maxInputAmount
6          uint64 deadline
7      )
8  )
9      public
10     revertIfZero(outputAmount)
11     revertIfDeadlinePassed(deadline)
12     returns (uint256 inputAmount)
13  {
14     uint256 inputReserves = inputToken.balanceOf(address(this));
15     uint256 outputReserves = outputToken.balanceOf(address(this));
16
17     inputAmount = getInputAmountBasedOnOutput(
18         outputAmount,
19         inputReserves,
20         outputReserves
21     );
22     if(inputAmount > maxInputAmount) {
23         revert();
24     }
25     _swap(inputToken, inputAmount, outputToken, outputAmount);
26 }
```

### [H-3] TSwapPool::sellPoolTokens mismatches 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 are 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, 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 protocol functionality.

#### Proof of Concept:

**Recommended Mitigation:** Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. NOTE-> This will also require changing the `sellPoolTokens` function to accept a new parameter (i.e. `minWethToReceive` to be passed to `swapExactInput`)

```

1      function sellPoolTokens(
2          uint256 poolTokenAmount,
3 +         uint256 minWethToReceive
4      ) external returns (uint256 wethAmount) {
5 -         return swapExactOutput(i_poolToken,i_wethToken,poolTokenAmount,
6 +         uint64(block.timestamp));
6 +         return swapExactInput(i_poolToken,poolTokenAmount,i_wethToken,
7             minWethToReceive,uint64(block.timestamp));
7     }

```

Additionaly, it might be wise to add a deadline to the function, as there is currently no deadline.

#### [H-4] In `TSwapPool::_swap` the extra tokens given to users after every `SWAP_COUNT_MAX` breaks the protocol invariant of $x * y = k$

**Description:** The protocol follows a strict invariant of  $x * y = k$ . Where -  $x$ : Balance of pool token -  $y$ : Balance of WETH -  $k$ : The constant product of 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 protocol funds will be drained .

The following highlighted code is responsible for this :

```

1      function _swap(
2          IERC20 inputToken,
3          uint256 inputAmount,
4          IERC20 outputToken,
5          uint256 outputAmount
6      ) private {
7          if (_isUnknown(inputToken) || _isUnknown(outputToken) ||
8              inputToken == outputToken) {
9              revert TSwapPool__InvalidToken();
10         }
11         swap_count++;
12         //FEE-ON-TRANSFER
13         if (swap_count >= SWAP_COUNT_MAX) {

```

```

13 @>         swap_count = 0;
14 @>         outputToken.safeTransfer(msg.sender, 1
15             _000_000_000_000_000_000);
16     }
17     emit Swap(msg.sender, inputToken, inputAmount, outputToken,
18             outputAmount);
19     inputToken.safeTransferFrom(msg.sender, address(this),
20                 inputAmount);
20     outputToken.safeTransfer(msg.sender, outputAmount);
}

```

**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.

More simply, the protocol's core invariant is broken.

**Proof of Concept:** 1. A user swaps 10 times, and collects the extra incentive of 1\_000\_000\_000\_000\_000 tokens. 2. That user continues to swap until all the protocol funds are drained.

#### Proof Of Code

```

1     function testInvariantBroken() public {
2         vm.startPrank(liquidityProvider);
3         weth.approve(address(pool), 100e18);
4         poolToken.approve(address(pool), 100e18);
5         pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
6         vm.stopPrank();
7
8         uint256 outputWeth = 1e17;
9
10        vm.startPrank(user);
11        poolToken.mint(user, 100e18);
12        poolToken.approve(address(pool), 100e18);
13        for (uint256 i = 0; i < 9; i++) {
14            pool.swapExactOutput(
15                poolToken,
16                weth,
17                outputWeth,
18                uint64(block.timestamp)
19            );
20        }
21        int256 startingY = int256(weth.balanceOf(address(pool)));
22        int256 expectedDeltaY = int256(-1) * int256(outputWeth);
23        pool.swapExactOutput(
24            poolToken,
25            weth,
26            outputWeth,
27            uint64(block.timestamp)
28        );
29        vm.stopPrank();
}

```

```

30         uint256 endingY = weth.balanceOf(address(pool));
31         int256 actualDeltaY = int256(endingY) - int256(startingY);
32
33         assertEq(actualDeltaY, expectedDeltaY);
34     }

```

**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 fees.

```

1   function _swap(
2       IERC20 inputToken,
3       uint256 inputAmount,
4       IERC20 outputToken,
5       uint256 outputAmount
6   ) private {
7       if (_isUnknown(inputToken) || _isUnknown(outputToken) ||
8           inputToken == outputToken) {
9           revert TSwapPool__InvalidToken();
10      swap_count++;
11      if (swap_count >= SWAP_COUNT_MAX) {
12          swap_count = 0;
13          outputToken.safeTransfer(msg.sender, 1
14          _000_000_000_000_000);
15      }
16      emit Swap(msg.sender, inputToken, inputAmount, outputToken,
17              outputAmount);
18
19  }

```

## Medium

**[M-1] TSwapPool::deposit is missing deadline check causing transactions to complete even after the deadline.**

**Description:** The `deposit` function accepts a deadline parameter, which according to documentation is “The deadline for the transaction to be completed by”. However, this parameter is never used .As a consequence, operations that add liquidity to the pool might be executed at unexpected times ,in market conditions where the deposit rate is unfavorable.

**Impact:** Transactions could be sent when market conditions are unfavorable to deposit ,even when adding a deadline parameter.

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

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

```

1   function deposit(
2       uint256 wethToDeposit,
3       uint256 minimumLiquidityTokensToMint,
4       uint256 maximumPoolTokensToDeposit,
5       uint64 deadline
6   )
7   external
8 +    revertIfDeadlinePassed(deadline)
9     revertIfZero(wethToDeposit)
10    returns (uint256 liquidityTokensToMint)

```

## [M-2] Rebase, fee-on-transfer, and ERC-777 tokens break protocol invariant

**Description:** The invariant of the protocol i.e.  $x * y = k$  is broken when weird ERC-20's are used in the protocol, some of which include the rebase, fee-on-transfer and erc-77 tokens.

Example of one such ERC-20 token :

### Fee on Transfer Token

```

1 // Copyright (C) 2020 d-xo
2 // SPDX-License-Identifier: AGPL-3.0-only
3
4 pragma solidity >=0.6.12;
5
6 import {ERC20} from "./ERC20.sol";
7
8 contract TransferFeeToken is ERC20 {
9
10     uint immutable fee;
11
12     // --- Init ---
13     constructor(uint _totalSupply, uint _fee) ERC20(_totalSupply)
14         public {
15             fee = _fee;
16         }
17
18     // --- Token ---
19     function transferFrom(address src, address dst, uint wad) override
20         public returns (bool) {
21             require(balanceOf[src] >= wad, "insufficient-balance");
22             if (src != msg.sender && allowance[src][msg.sender] != type(
23                 uint).max) {
24                 require(allowance[src][msg.sender] >= wad, "insufficient-
25                     allowance");

```

```

22         allowance[src][msg.sender] = sub(allowance[src][msg.sender]
23             , wad);
24     }
25
26     balanceOf[src] = sub(balanceOf[src], wad);
27     balanceOf[dst] = add(balanceOf[dst], sub(wad, fee));
28     balanceOf[address(0)] = add(balanceOf[address(0)], fee);
29
30     emit Transfer(src, dst, sub(wad, fee));
31     emit Transfer(src, address(0), fee);
32
33     return true;
34 }
```

**Impact:** The core invariant  $x * y = k$  is broken, which is extremely disruptive.

**Proof of Concept:** 1. `liquidity Provider` provides the initial liquidity to the fee, where a part of fee is burnt, hence the pool will receive (`poolTokenSended-fee`) amount of pool tokens in the pool. For example: 1. `100e18` weth and `101e18` poolToken is provided. 2. Suppose the fee is `1e18`. 3. Pool will receive `100e18` weth and `100e18` poolToken. 2. `user` tries to swap the poolToken with weth. 3. Suppose user wants to swap `2e18` poolToken with weth 4. User calls the function `swapExactInput` with input poolToken as `2e18`, expecting to receive 2eth back. 5. But the pool received only (`2e18 - fee`) = `1e18` and the user hence got only `1e18` weth. 6. Protocol invariant is broken, as `expectedDelta` is not equal to `actualDelta`.

PoC

```

1  function testWeirdERC20BreakInvariant() public {
2      assertEq(poolVul.getPoolToken(), address(poolTokenVul));
3      assertEq(poolVul.getWeth(), address(weth));
4
5      vm.startPrank(liquidityProvider);
6      poolTokenVul.transferFrom(address(this), liquidityProvider, 102
7          e18); //liquidityProvider will receive 101e18
8      assertEq(poolTokenVul.balanceOf(liquidityProvider), 101e18);
9      weth.approve(address(poolVul), 100e18);
10     poolTokenVul.approve(address(poolVul), 101e18);
11
12     poolVul.deposit(100e18, 100e18, 101e18, uint64(block.timestamp)
13         );
14     vm.stopPrank();
15
16     console.log(
17         "amount in poolVul",
18         poolTokenVul.balanceOf(address(poolVul))
19     ); // 100e18
20
21     uint256 inputPoolToken = 2e18;
```

```

1      vm.startPrank(user);
2      poolTokenVul.transferFrom(address(this), user, 10e18); //user
3          will get 9e18
4      poolTokenVul.approve(address(poolVul), inputPoolToken);
5      uint256 startingPoolTokenBalance = poolTokenVul.balanceOf(
6          address(poolVul)
7      );
8      uint256 expectedDeltaPoolToken = inputPoolToken;
9
10     poolVul.swapExactInput(
11         IERC20(address(poolTokenVul)),
12         inputPoolToken,
13         weth,
14         0,
15         uint64(block.timestamp)
16     );
17
18     uint256 endingPoolTokenBalance = poolTokenVul.balanceOf(
19         address(poolVul)
20     );
21     uint256 actualDeltaPoolToken = endingPoolTokenBalance -
22         startingPoolTokenBalance; // this will be 1e18 instead of 2
23             e18
24     vm.stopPrank();
25
26     assert(actualDeltaPoolToken != expectedDeltaPoolToken);
27     assertEq(actualDeltaPoolToken, expectedDeltaPoolToken - 1e18);
28
29     console.log("Invariant is broken (i.e. `x * y = k`)");
30 }

```

**Recommended Mitigation:** It is suggested to maintain a contract level allowlist of known good tokens.

## Low

**[L-1] Incorrect emmission of event LiquidityAdded inside the function `TSwapPool:::_addLiquidityMintAndTransfer` causing wrong output .**

**Description:** Inside the function `TSwapPool:::_addLiquidityMintAndTransfer`, order of the event `LiquidityAdded` is incorrect. According to the definition in documentation :

```

1     event LiquidityAdded(
2         address indexed liquidityProvider,
3         uint256 wethDeposited,
4         uint256 poolTokensDeposited
5     );

```

But in the function ,the order of `wethDeposited` and `poolTokensDeposited` is reversed :

Found in `src/TSwapPool.sol` Line:213

```

1   function _addLiquidityMintAndTransfer(
2       uint256 wethToDeposit,
3       uint256 poolTokensToDeposit,
4       uint256 liquidityTokensToMint
5   ) private {
6       _mint(msg.sender, liquidityTokensToMint);
7   @>     emit LiquidityAdded(msg.sender, poolTokensToDeposit,
8       wethToDeposit);
9
10      // Interactions
11      i_wethToken.safeTransferFrom(msg.sender, address(this),
12          wethToDeposit);
13      i_poolToken.safeTransferFrom(
14          msg.sender,
15          address(this),
16          poolTokensToDeposit
17      );
18  }

```

**Impact:** This is giving wrong output ,which can lead to errors in the frontend, hence can lead to frontend related bugs.

**Recommended Mitigation:** Change the order of 2nd and 3rd element in the event to ensure correct event emit.

```

1   function _addLiquidityMintAndTransfer(
2       uint256 wethToDeposit,
3       uint256 poolTokensToDeposit,
4       uint256 liquidityTokensToMint
5   ) private {
6       _mint(msg.sender, liquidityTokensToMint);
7   -     emit LiquidityAdded(msg.sender, poolTokensToDeposit,
8       wethToDeposit);
8 +     emit LiquidityAdded(msg.sender, wethToDeposit,
9         poolTokensToDeposit);
9
10      // Interactions
11      i_wethToken.safeTransferFrom(msg.sender, address(this),
12          wethToDeposit);
13      i_poolToken.safeTransferFrom(
14          msg.sender,
15          address(this),
16          poolTokensToDeposit
17      );
18  }

```

**[L-2] Default value returned by TSwapPool::swapExactInput results in incorrect return value given**

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

**Impact:** The return value will always be 0, giving incorrect information to the caller .

**Recommended Mitigation:**

```
1   function swapExactInput(
2       IERC20 inputToken,
3       uint256 inputAmount,
4       IERC20 outputToken,
5       uint256 minOutputAmount,
6       uint64 deadline
7   )
8     public
9     revertIfZero(inputAmount)
10    revertIfDeadlinePassed(deadline)
11    returns (
12        uint256 output
13    )
14  {
15     uint256 inputReserves = inputToken.balanceOf(address(this));
16     uint256 outputReserves = outputToken.balanceOf(address(this));
17
18 -     uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount,
19 +     output = getOutputAmountBasedOnInput(inputAmount, inputReserves,
19      outputReserves);
20
21 -     if (outputAmount < minOutputAmount) {
22 -         revert TSwapPool__OutputTooLow(outputAmount,
23 +     if (output < minOutputAmount) {
24 +         revert TSwapPool__OutputTooLow(output, minOutputAmount);
25     }
26
27 -     _swap(inputToken, inputAmount, outputToken, outputAmount);
28 +     _swap(inputToken, inputAmount, outputToken, output);
29
30 }
```

## Informational

### [I-1] Unused error:PoolFactory::PoolFactory\_\_PoolAlreadyExists(address tokenAddress) wasting unnecessary gas

**Description:** In the `PoolFactory:PoolFactory.sol` contract, the error `PoolFactory__PoolAlreadyExists` is not used anywhere in the contract. Therefore, creating unnecessary confusion and gas fee.

Found in `src/PoolFactory.sol` Line:24

```
1 @> error PoolFactory__PoolDoesNotExist(address tokenAddress);
```

**Recommended Mitigation:** Make sure to either remove the error completely or use it in the contract

### [I-2] Missing check for address (0) in the constructor of PoolFactory and TSwapPool contract

**Description:** In the constructor of `PoolFactory` contract, there should be a check for `address(0)` to ensure that `address i_wethToken` is initialized properly in the constructor.

2 Found Instances

- Found in `src/PoolFactory.sol` Line:43

```
1     constructor(address wethToken) {
2 @>     // @audit-info lacks a zero check
3     i_wethToken = wethToken;
4 }
```

- Found in `src/TSwapPool.sol` Line:77

```
1     constructor(
2         address poolToken,
3         address wethToken,
4         string memory liquidityTokenName,
5         string memory liquidityTokenSymbol
6     )
7     ERC20(liquidityTokenName, liquidityTokenSymbol)
8 {
9 @>     // @audit-info zero address check
10     i_wethToken = IERC20(wethToken);
11     i_poolToken = IERC20(poolToken);
12 }
```

**Recommended Mitigation:** Include a zero check before initializing the address of `i_wethToken` with `wethToken`.

```

1     constructor(address wethToken){
2 +         if(wethToken==address(0)){
3 +             revert();
4 +         }
5         i_wethToken=wethToken;
6     }

```

```

1     constructor(
2         address poolToken,
3         address wethToken,
4         string memory liquidityTokenName,
5         string memory liquidityTokenSymbol
6     )
7     ERC20(liquidityTokenName, liquidityTokenSymbol)
8     {
9 +         if(wethToken==address(0) || poolToken==address(0)){
10 +             revert();
11 +         }
12         i_wethToken = IERC20(wethToken);
13         i_poolToken = IERC20(poolToken);
14     }

```

**[I-3] Incorrect ERC-20 function call (i.e. `.name` in place of `.symbol`) in `PoolFactory::createPool` making the symbol name inconsistent.**

**Description:** In the `PoolFactory::createPool` function , for `liquidityTokenSymbol: IERC20(tokenAddress).name()` is used instead of `IERC20(tokenAddress).symbol()`, which is making the symbol name inconsistent .

Found in src/PoolFactory.sol Line:63

```

1     function createPool(address tokenAddress) external returns(address)
2     {
3         ...
4         ...
5         string memory liquidityTokenSymbol = string.concat(
6             "ts",
7             IERC20(tokenAddress).name()
8         );
9         ...
10    }

```

**Recommended Mitigation:** Use `IERC20(tokenAddress).symbol()` instead of `IERC20(tokenAddress).name()`

```

1      function createPool(address tokenAddress) external returns(address)
2      {
3          ..
4          string memory liquidityTokenSymbol = string.concat("ts",
5          IERC20(tokenAddress).name()
6          IERC20(tokenAddress).symbol()
7          );
8      ...
}
```

#### [I-4] Unnecessary element in TSwapPool::deposit revert statement

**Description:** In the `TSwapPool::deposit` function, there is a revert statement as follows:

Found in src/TSwapPool.sol Line:133

```

1      revert TSwapPool__WethDepositAmountTooLow(
2 @>      MINIMUM_WETH_LIQUIDITY,
3      wethToDeposit
4 );
```

Use of `MINIMUM_WETH_LIQUIDITY` is unnecessary as it's a constant and does not provide any relevant information.

**Recommended Mitigation:** Remove the `MINIMUM_WETH_LIQUIDITY` from the revert parameter

```

1      revert TSwapPool__WethDepositAmountTooLow(
2 -      MINIMUM_WETH_LIQUIDITY,
3      wethToDeposit
4 );
```

#### [I-5] Unused variable poolTokenReserves is declared inside TSwapPool::deposit wasting unnecessary gas

**Description:** There is a variable `poolTokenReserves` declared inside the `TSwapPool::deposit` function which is not used anywhere in the function, hence is not needed anywhere in the function and is wasting gas.

Found in src/TSwapPool.sol Line:140

```

1      function deposit(uint256 wethToDeposit,uint256
2      minimumLiquidityTokensToMint,uint256 maximumPoolTokensToDeposit,
3      uint64 deadline)
2      ...
3      uint256 wethReserves = i_wethToken.balanceOf(address(this))
```

```

4 @>      uint256 poolTokenReserves = i_poolToken.balanceOf(address(this))
5      );
6      uint256 poolTokensToDeposit = getPoolTokensToDepositBasedOnWeth
7          (
8              wethToDeposit
9          );
10     ...

```

**Recommended Mitigation:** Remove that line from the function directly.

```

1   function deposit(uint256 wethToDeposit,uint256
2       minimumLiquidityTokensToMint,uint256 maximumPoolTokensToDeposit,
3       uint64 deadline)
4   ...
5   -     uint256 wethReserves = i_wethToken.balanceOf(address(this))
6   -     uint256 poolTokenReserves = i_poolToken.balanceOf(address(this))
7   );
8     uint256 poolTokensToDeposit = getPoolTokensToDepositBasedOnWeth
9         (
10             wethToDeposit
11         );
12     ...

```

### [I-6] Value assignment of variable `liquidityTokensToMint` is after `_addLiquidityMintAndTransfer` which avoids the practice of CEI

**Description:** Inside the `TSwapPool::deposit` function, the variable `liquidityTokensToMint` is assigned after `_addLiquidityMintAndTransfer` which is not considered as the best practice.

Found in `src/TSwapPool.sol` Line:196

```

1   function deposit(uint256 wethToDeposit,uint256
2       minimumLiquidityTokensToMint,uint256 maximumPoolTokensToDeposit,
3       uint64 deadline)
4   ...
5   else {
6       // This will be the "initial" funding of the protocol. We
7       // are starting from blank here!
8       // We just have them send the tokens in, and we mint
9       // liquidity tokens based on the weth
10      _addLiquidityMintAndTransfer(
11          wethToDeposit,
12          maximumPoolTokensToDeposit,
13          wethToDeposit
14      );
15      @>      liquidityTokensToMint = wethToDeposit;
16  }

```

**Recommended Mitigation:** It is recommended to do the assignment of variable `liquidityTokensToMint` before calling this function `_addLiquidityMintAndTransfer`.

```

1      else {
2          // This will be the "initial" funding of the protocol. We
3          // are starting from blank here!
4          // We just have them send the tokens in, and we mint
5          // liquidity tokens based on the weth
6          liquidityTokensToMint = wethToDeposit;
7          _addLiquidityMintAndTransfer(
8              wethToDeposit,
9              maximumPoolTokensToDeposit,
10             wethToDeposit
11         );
10     liquidityTokensToMint = wethToDeposit;
11 }
```

#### [I-7] Direct use of numbers like 997 and 1000 in functions

**TSwapPool::getOutputAmountBasedOnInput and  
TSwapPool::getInputAmountBasedOnOutput making the code less readable and  
increases risk of errors.**

**Description:** In the functions `TSwapPool::getOutputAmountBasedOnInput` and `TSwapPool::getInputAmountBasedOnOutput`, direct numbers like 997, 1000 and 10000 are used ,which are causing confusion and making the code less readable.

#### 2 Found Instances

Found in src/TSwapPool.sol Line:294

```

1      function getOutputAmountBasedOnInput(
2          uint256 inputAmount,
3          uint256 inputReserves,
4          uint256 outputReserves
5      )
6          public
7          pure
8          revertIfZero(inputAmount)
9          revertIfZero(outputReserves)
10         returns (uint256 outputAmount)
11     {
12     @>     uint256 inputAmountMinusFee = inputAmount * 997;
13     @>     uint256 numerator = inputAmountMinusFee * outputReserves;
14     @>     uint256 denominator = (inputReserves * 1000) +
15         inputAmountMinusFee;
16         return numerator / denominator;
16 }
```

Found in src/TSwapPool.sol Line:316

```

1      function getInputAmountBasedOnOutput(
2          uint256 outputAmount,
3          uint256 inputReserves,
4          uint256 outputReserves
5      )
6      public
7      pure
8      revertIfZero(outputAmount)
9      revertIfZero(outputReserves)
10     returns (uint256 inputAmount)
11    {
12      return
13     @>     ((inputReserves * outputAmount) * 10000) /
14     @>     ((outputReserves - outputAmount) * 997);
15    }

```

**Recommended Mitigation:** Define these numbers as `constant` and then use them in the function, which will make the code readable and will decrease the chance of silly errors or typos .

#### [I-8] Missing natspec in TSwapPool::swapExactInput, making it difficult for user to understand the function

**Description:** In the `swapExactInput` function, there is no natspec which is not the best practice to do, for other developers and even protocol users, it's essential to know what the function do. By providing natspec, this gets clear.

**Recommended Mitigation:** Put natspec in the function for better documentation.

#### [I-9] TSwapPool::swapExactInput is declared public instead of external, wasting gas.

**Description:** The function `swapExactInput` is declared `public`, but should be declared `external` as it's not use anywhere in the contract, hence only external addresses can call it.

```

1      function swapExactInput(IERC20 inputToken,uint256 inputAmount,
2                               IERC20 outputToken,uint256 minOutputAmount,uint64 deadline)
3     @>     public revertIfZero(inputAmount) revertIfDeadlinePassed(
4     deadline) returns (uint256 output)
5     {
6         uint256 inputReserves = inputToken.balanceOf(address(this));
7         uint256 outputReserves = outputToken.balanceOf(address(this));
8
9         uint256 outputAmount = getOutputAmountBasedOnInput(
10             inputAmount,
11             inputReserves,
12             outputReserves,
13             deadline,
14             minOutputAmount);
15     }

```

```

10         outputReserves
11     );
12
13     if (outputAmount < minOutputAmount) {
14         revert TSwapPool__OutputTooLow(outputAmount,
15                                         minOutputAmount);
16     }
17
18     _swap(inputToken, inputAmount, outputToken, outputAmount);
19 }
```

**Impact:** Gas is getting wasted.

**Recommended Mitigation:** Change the `public` keyword with `external`.

```

1   function swapExactInput(IERC20 inputToken,uint256 inputAmount,
2 -     IERC20 outputToken,uint256 minOutputAmount,uint64 deadline)
3 +     public revertIfZero(inputAmount) revertIfDeadlinePassed(
4     deadline) returns (uint256 output)
5     external revertIfZero(inputAmount) revertIfDeadlinePassed(
6     deadline) returns (uint256 output)
7     {
8         uint256 inputReserves = inputToken.balanceOf(address(this));
9         uint256 outputReserves = outputToken.balanceOf(address(this));
10
11         uint256 outputAmount = getOutputAmountBasedOnInput(
12             inputAmount,
13             inputReserves,
14             outputReserves
15         );
16
17         if (outputAmount < minOutputAmount) {
18             revert TSwapPool__OutputTooLow(outputAmount,
19                                             minOutputAmount);
20         }
21
22         _swap(inputToken, inputAmount, outputToken, outputAmount);
23     }
```

#### [I-10] Missing `deadline` parameter in `TSwapPool::swapExactOutput` function's natspec

**Description:** In the `swapExactOutput` function, `deadline` parameter is missing in the natspec of the function.

**Recommended Mitigation:** Add natspec for `deadline` parameter in the function.

```

1  /*
2   * @notice figures out how much you need to input based on how much
3   * output you want to receive.
```

```
4      *
5      * Example: You say "I want 10 output WETH, and my input is DAI"
6      * The function will figure out how much DAI you need to input to
7      * get 10 WETH
8      * And then execute the swap
9      * @param inputToken ERC20 token to pull from caller
10     * @param outputToken ERC20 token to send to caller
11    * @param outputAmount The exact amount of tokens to send to caller
12    * @param deadline The deadline for the transaction to be completed
13    * by
14    */
15   function swapExactOutput(
16     IERC20 inputToken,
17     IERC20 outputToken,
18     uint256 outputAmount,
19     uint64 deadline
20   )
21   public
22   revertIfZero(outputAmount)
23   revertIfDeadlinePassed(deadline)
24   returns (uint256 inputAmount)
25   {
26     uint256 inputReserves = inputToken.balanceOf(address(this));
27     uint256 outputReserves = outputToken.balanceOf(address(this));
28
29     inputAmount = getInputAmountBasedOnOutput(
30       outputAmount,
31       inputReserves,
32       outputReserves
33     );
34     _swap(inputToken, inputAmount, outputToken, outputAmount);
35 }
```