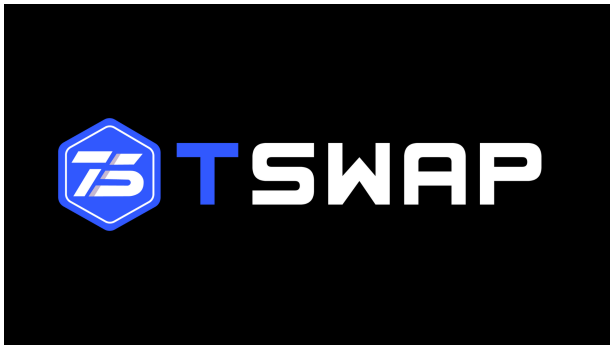

T-Swap Audit Report

Version 1.0

Prakash Yadav

September 3, 2025

T-Swap Audit Report

Prakash Yadav

September 3, 2025

Prepared by: Prakash Yadav Lead Auditors: - xxxxxxx

Table of Contents

Protocol Summary

T-Swap is a decentralized exchange (DEX) protocol built on Ethereum, implementing an Automated Market Maker (AMM) model similar to Uniswap. It facilitates permissionless token swaps using liquidity pools, where users can trade ERC20 tokens against Wrapped Ether (WETH) without requiring a centralized intermediary.

Key Components

- **PoolFactory**: A factory contract responsible for creating and managing individual liquidity pools. It deploys new TSwapPool instances for each ERC20 token paired with WETH, ensuring each token has its own dedicated pool.
- **TSwapPool**: The core pool contract that handles liquidity provision, token swaps, and withdrawals for a specific ERC20/WETH pair. Each pool is an ERC20 token itself, representing liquidity provider (LP) shares.

Core Functionalities

- **Liquidity Provision**: Users can deposit WETH and the paired ERC20 token into a pool to provide liquidity. In return, they receive LP tokens proportional to their contribution, which entitle them to a share of trading fees.
- **Token Swaps**: Users can swap tokens in two modes:
  - `swapExactInput`: Specify exact input amount, receive at least minimum output.
  - `swapExactOutput`: Specify exact output amount, pay at most maximum input. Swaps incorporate a 0.3% fee, distributed to liquidity providers.
- **Liquidity Withdrawal**: LP token holders can burn their tokens to withdraw their proportional share of the pool's assets.
- **Pricing Mechanism**: Uses the constant product formula (x * y = k), where x and y are the reserves of the two tokens in the pool. This ensures the product of reserves remains constant, providing automatic price discovery.
- **Incentives**: Includes a mechanism to reward frequent traders with extra tokens every 10 swaps, though this feature has been identified as potentially breaking the protocol's core invariant.
- **Security Features**: Implements transaction deadlines to mitigate Miner Extractable Value (MEV) attacks and includes slippage protection in swap functions.

The protocol aims to provide efficient, decentralized trading while maintaining security through mathematical invariants and user protections.

Disclaimer

Prakash Yadav 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

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda

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

Issues found

Category	No. of Issues
High	04
Medium	01
Low	02
Informational	09
Total	16

Findings

High

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

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

Impact: It can exhaust high fees from users we can lead to dis-engagement from user due to high fees.

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

[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` function should specify a `maxInputAmount`.

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

Proof of Concept: 1. The price should of 1 WETH right now is 1000 USDC. 2. User inputs a `TSwapPool::swapExactOutput` looking for 1 WETH 1. inputToken=USDC 2. outputToken=WETH 3. outputAmount=1 4. deadline= Given(Whatever). 3. The function does not offer a maxInput amount. 4. As the transaction is pending in the mempool, the market changes and the price moves HUGE -> 1 WETH is now 10,000 USDC. 10x more than the user expected. 5. The transaction completes,but the user sent the protocol 10,000 USDC instead of the expected 1,000 USDC.

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

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

```
9           revertIfZero(outputAmount)
10          revertIfDeadlinePassed(deadline)
11          returns (uint256 inputAmount)
12      {
13          uint256 inputReserves = inputToken.balanceOf(address(this));
14          uint256 outputReserves = outputToken.balanceOf(address(this));
15
16          inputAmount = getInputAmountBasedOnOutput(outputAmount,
                inputReserves, outputReserves);
17  +       if(inputAmount > maxInputAmount){
18  +          revert();
19  +       }
20
21  -        _swap(inputToken, inputAmount, outputToken, outputAmount);
22  +        _swap(inputToken, inputAmount, outputToken, maxInputToken,
        outputAmount);
23      }
```

[H-3]: `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to receive the incorrect amount of tokens.

Description: The `sellPoolToken` function is intended to allow users to easily sell and 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 amounts.

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 id a severe disruption of protocol functionality.

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

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

Additionally there need to add a deadline to the function, as there is currently no deadline .(MEV Attack)

[H-4]: In `SwapPool::_swap` the extra token given to users after every `swapCount` breaks the protocol invariant of `x * y=k`.

Description: The protocol follows a strict invariant of `x * y= k`. Where: - `x`: The balance of the pool token. - `y`: The 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 block of code is responsible for the issue:

```
1  function _swap(IERC20 inputToken, uint256 inputAmount, IERC20
      outputToken, uint256 outputAmount) private {
2          if (_isUnknown(inputToken) || _isUnknown(outputToken) ||
              inputToken == outputToken) {
3              revert TSwapPool__InvalidToken();
4          }
5
6          swap_count++;
7          if (swap_count >= SWAP_COUNT_MAX) {
8              swap_count = 0;
9              outputToken.safeTransfer(msg.sender, 1
                   _000_000_000_000_000_000);
10         }
11         emit Swap(msg.sender, inputToken, inputAmount, outputToken,
              outputAmount);
12
13         inputToken.safeTransferFrom(msg.sender, address(this),
              inputAmount);
14         outputToken.safeTransfer(msg.sender, outputAmount);
15     }
```

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. It basically means that 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_000` . 2. The user continues to swap until all the protocol funds are drained.

PoC

Place the following code in `TSwapPool.t.sol`.

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

        vm.startPrank(user);
        uint256 expected = 9e18;
        poolToken.approve(address(pool), 10e18);
        pool.swapExactInput(poolToken, 10e18, weth, expected, uint64(
            block.timestamp));
        vm.stopPrank();

        vm.startPrank(liquidityProvider);
        pool.approve(address(pool), 100e18);
        pool.withdraw(100e18, 90e18, 100e18, uint64(block.timestamp));
        assertEq(pool.totalSupply(), 0);
        assert(weth.balanceOf(liquidityProvider) + poolToken.balanceOf(
            liquidityProvider) > 400e18);
    }
```

If need additional so add Invariant.t.sol.

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.20;

import {Test} from "forge-std/Test.sol";
import {StdInvariant} from "forge-std/StdInvariant.sol";
import {ERC20Mock} from "../mocks/ERC20Mock.sol";
import {PoolFactory} from "../../src/PoolFactory.sol";
import {TSwapPool} from "../../src/TSwapPool.sol";
import {Handler} from "./Handler.t.sol";


contract Invariant is  StdInvariant,Test {
    ERC20Mock poolToken;
    ERC20Mock weth;
    PoolFactory factory;
    TSwapPool pool;
    Handler handler;

    uint256 constant STARTING_X = 100e18; //ERC20
    uint256 constant STARTING_Y = 50e18; //WETH
    function setUp() public{
        poolToken = new ERC20Mock();
        weth = new ERC20Mock();
        factory = new PoolFactory(address(weth));
        pool = TSwapPool(factory.createPool(address(poolToken)));
```

```
        // Initial x and y balances
        poolToken.mint(address(this),uint256(STARTING_X));
        weth.mint(address(this),uint256(STARTING_Y));

        poolToken.approve(address(pool),type(uint256).max);
        weth.approve(address(pool),type(uint256).max);

        //Deposit into pool
        pool.deposit(uint256(STARTING_Y),uint256(STARTING_Y),uint256(
            STARTING_X),uint64(block.timestamp));
        handler=new Handler(pool);
        bytes4[] memory selectors=new bytes4[](2);
        selectors[0]=handler.swapPoolTokenForWethOnOutputWeth.selector;
        selectors[1]=handler.deposit.selector;

        targetSelector(FuzzSelector({addr:address(handler),selectors:
            selectors}));
        targetContract(address(handler));
    }

    function statefulFuzz_constantProductFormula() external view {
        assert(handler.actualDeltaX() == handler.expectedDeltaX());
    }
}
```

And Code for Handler.t.sol too:

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.20;

import {Test,console2} from "forge-std/Test.sol";
import {TSwapPool} from "../../src/TSwapPool.sol";
import {ERC20Mock} from "../mocks/ERC20Mock.sol";

contract Handler is Test{
    TSwapPool pool;
    ERC20Mock weth;
    ERC20Mock poolToken;

    address liquidityProvider =makeAddr("lp");
    address swapper = makeAddr("swapper");

    int256 startingY;
    int256 startingX;
    int256 public expectedDeltaY;
    int256 public expectedDeltaX;
    int256 public actualDeltaY;
    int256 public actualDeltaX;

    constructor(TSwapPool _pool){
```

```
25          pool = _pool;
26          weth = ERC20Mock(_pool.getWeth());
27          poolToken = ERC20Mock(_pool.getPoolToken());
28      }
29      function swapPoolTokenForWethOnOutputWeth(uint256 outputWeth)
            public{
30          outputWeth=bound(outputWeth,1,type(uint256).max);
31          if(outputWeth>=weth.balanceOf(address(pool))){
32              return;
33          }
34          uint256 poolTokenAmount=pool.getInputAmountBasedOnOutput(
                outputWeth,poolToken.balanceOf(address(pool)),weth.balanceOf
                (address(pool)));
35          if(poolTokenAmount>=type(uint64).max){
36              return;
37          }
38          if(poolToken.balanceOf(swapper)<poolTokenAmount){
39              poolToken.mint(swapper,poolTokenAmount-poolToken.balanceOf(
                    swapper)+1);
40          }
41          startingY=int256(weth.balanceOf(swapper));
42          startingX=int256(poolToken.balanceOf(swapper));
43          expectedDeltaY=int256(outputWeth);
44          expectedDeltaX=-int256(poolTokenAmount);
45
46          vm.startPrank(swapper);
47          poolToken.approve(address(pool),type(uint256).max);
48          pool.swapExactOutput(poolToken,weth,outputWeth,uint64(block.
                timestamp));
49          vm.stopPrank();
50
51          //actual
52          uint256 endingY=weth.balanceOf(swapper);
53          uint256 endingX=poolToken.balanceOf(swapper);
54          actualDeltaY=int256(endingY)-startingY;
55          actualDeltaX=int256(endingX)-startingX;
56      }
57
58      function deposit(uint256 wethAmount) public {
59          uint256 minWeth = TSwapPool(pool).getMinimumWethDepositAmount()
                ;
60          wethAmount = bound(wethAmount, minWeth, type(uint64).max);
61          uint256 poolTokenAmount = pool.
                getPoolTokensToDepositBasedOnWeth(wethAmount);
62          //deposit
63          vm.startPrank(liquidityProvider);
64          weth.mint(liquidityProvider, wethAmount);
65          poolToken.mint(liquidityProvider, poolTokenAmount);
66          startingY = int256(weth.balanceOf(liquidityProvider));
67          startingX = int256(poolToken.balanceOf(liquidityProvider));
68          expectedDeltaY = -int256(wethAmount);
```

```
69          expectedDeltaX = -int256(poolTokenAmount);
70
71          weth.approve(address(pool), type(uint256).max);
72          poolToken.approve(address(pool), type(uint256).max);
73
74          pool.deposit(wethAmount, 0, poolTokenAmount, uint64(block.
               timestamp));
75          vm.stopPrank();
76          //actual
77          uint256 endingY = weth.balanceOf(liquidityProvider);
78          uint256 endingX = poolToken.balanceOf(liquidityProvider);
79          actualDeltaY = int256(endingY) - startingY;
80          actualDeltaX = int256(endingX) - startingX;
81      }
82  }
```

Recommended Mitigation: Remove the extra incentive. If we want to keep this in ,we should account for change in the x * y = k protocol invrinat. Or , we should set aside tokens in the same way we do with fees.

```
1   function _swap(IERC20 inputToken, uint256 inputAmount, IERC20
       outputToken, uint256 outputAmount) private {
2           if (_isUnknown(inputToken) || _isUnknown(outputToken) ||
               inputToken == outputToken) {
3               revert TSwapPool__InvalidToken();
4           }
5
6   -        swap_count++;
7   -       if (swap_count >= SWAP_COUNT_MAX) {
8   -            swap_count = 0;
9   -            outputToken.safeTransfer(msg.sender, 1
       _000_000_000_000_000_000);
10  -        }
11          emit Swap(msg.sender, inputToken, inputAmount, outputToken,
               outputAmount);
12
13          inputToken.safeTransferFrom(msg.sender, address(this),
               inputAmount);
14          outputToken.safeTransfer(msg.sender, outputAmount);
```

Medium

[M-1]: `TSwapPool::deposit` is not used deadline check in actual method and it is in deposit function but not being used can cause transaction to complete its proceeding even after the deadline.

Description: The `deposit` function accepts a deadline parameter, which according to the documentation is "/// @param deadline The deadline for the transaction to be completed by". However, this parameter is never used. As a consequences, operations that add liquidity to the pool might be executed at unexpected times, in market conditions where the deposit rate is unfavourable.

Impact: Transaction could be sent when market condition are unfavourable 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.

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
        if (wethToDeposit < MINIMUM_WETH_LIQUIDITY) {
            revert TSwapPool__WethDepositAmountTooLow(
                MINIMUM_WETH_LIQUIDITY, wethToDeposit);
        }
        if (totalLiquidityTokenSupply() > 0) {
            uint256 wethReserves = i_wethToken.balanceOf(address(this))
                ;
            uint256 poolTokenReserves = i_poolToken.balanceOf(address(
                this));
            uint256 poolTokensToDeposit =
                getPoolTokensToDepositBasedOnWeth(wethToDeposit);
            if (maximumPoolTokensToDeposit < poolTokensToDeposit) {
                revert TSwapPool__MaxPoolTokenDepositTooHigh(
                    maximumPoolTokensToDeposit, poolTokensToDeposit);
            }
            liquidityTokensToMint = wethToDeposit *
                totalLiquidityTokenSupply() / wethReserves;
            if (liquidityTokensToMint < minimumLiquidityTokensToMint) {
                revert TSwapPool__MinLiquidityTokensToMintTooLow(
```

```
                        minimumLiquidityTokensToMint, liquidityTokensToMint)
                        ;
25              }
26              _addLiquidityMintAndTransfer(wethToDeposit,
                    poolTokensToDeposit, liquidityTokensToMint);
27          } else {
28              _addLiquidityMintAndTransfer(wethToDeposit,
                    maximumPoolTokensToDeposit, wethToDeposit);
29              liquidityTokensToMint = wethToDeposit;
30          }
31      }
```

Low

[L-1]: `TSwapPool::LiquidityAdded` event has parameter out of order.

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTrans` function, it logs values is in an incorrect order.The `poolTokenToDeposit` value should go in the third parameter position,whereas the `wethToDeposit` value go to second.

Impact:: Event emission is incorrect,leading to off-chain function potentially malfunctioning.

Recommended Mitigation:

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

[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 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:

```
function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
        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);
+       uint256 output = getOutputAmountBasedOnInput(inputAmount, inputReserves, outputReserves);

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

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

[L-1]: `TSwapPool::LiquidityAdded` event has parameter out of order.

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTran` function, it logs values is in an incorrect order.The `poolTokenToDeposit` value should go in the

third parameter position,whereas the `wethToDeposit` value go to second.

Impact:: Event emission is incorrect,leading to off-chain function potentially malfunctioning.

Recommended Mitigation:

```
function _addLiquidityMintAndTransfer(
        uint256 wethToDeposit,
        uint256 poolTokensToDeposit,
        uint256 liquidityTokensToMint
    )
        private
    {
        _mint(msg.sender, liquidityTokensToMint);
-        emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
+        emit LiquidityAdded(msg.sender, wethToDeposit,, poolTokensToDeposit);
        i_wethToken.safeTransferFrom(msg.sender, address(this), wethToDeposit);
        i_poolToken.safeTransferFrom(msg.sender, address(this), poolTokensToDeposit);
    }
```

[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 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:

```
function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 output)
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));
```

```
15
16 -        uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount
       , inputReserves, outputReserves);
17 +        uint256 output = getOutputAmountBasedOnInput(inputAmount,
       inputReserves, outputReserves);
18
19 -        if (outputAmount < minOutputAmount) {
20 -            revert TSwapPool__OutputTooLow(outputAmount,
       minOutputAmount);
21 -        }
22 +        if (output < minOutputAmount) {
23 +            revert TSwapPool__OutputTooLow(outputAmount,
       minOutputAmount);
24 +        }
25
26 -        _swap(inputToken, inputAmount, outputToken, outputAmount);
27 +        _swap(inputToken, inputAmount, outputToken, output);
28      }
```

Informational

Informational

[I-1]: `PoolFactory::PoolFactory__PoolDoesNotExist` is not used in T-swap pool factory and should be removed.

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

[I-2]: Lacking zero address checks.

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

[I-3]: `PoolFactory::createPool` should use `.symbol()` instead of `.name()` in string memory liquidityTokenSymbol.

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

```
2 + 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 Line: 35

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

- Found in src/TSwapPool.sol Line: 43

```
1     event LiquidityAdded(address indexed liquidityProvider,
          uint256 wethDeposited, uint256 poolTokensDeposited);
```

- Found in src/TSwapPool.sol Line: 44

```
1     event LiquidityRemoved(address indexed liquidityProvider,
          uint256 wethWithdrawn, uint256 poolTokensWithdrawn);
```

- Found in src/TSwapPool.sol Line: 45

```
1     event Swap(address indexed swapper, IERC20 tokenIn, uint256
          amountTokenIn, IERC20 tokenOut, uint256 amountTokenOut);
```

[I-5]: `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: 247

```
1     function swapExactInput(
```

[I-6]: 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: 227

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

- Found in src/TSwapPool.sol Line: 244

```
1         return ((inputReserves * outputAmount) * 10000) / ((
              outputReserves - outputAmount) * 997);
```

- Found in src/TSwapPool.sol Line: 358

```
1             1e18, i_wethToken.balanceOf(address(this)),
                  i_poolToken.balanceOf(address(this))
```

- Found in src/TSwapPool.sol Line: 364

```
1             1e18, i_poolToken.balanceOf(address(this)),
                  i_wethToken.balanceOf(address(this))
```

[I-8]: 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

```
1 pragma solidity 0.8.20;
```

- Found in src/TSwapPool.sol Line: 15

```
1 pragma solidity 0.8.20;
```

[I-8]: 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: 36

```
1     uint256 private constant MINIMUM_WETH_LIQUIDITY = 1
          _000_000_000;
```

- Found in src/TSwapPool.sol Line: 244

```
1         return ((inputReserves * outputAmount) * 10000) / ((
              outputReserves - outputAmount) * 997);
```

- Found in src/TSwapPool.sol Line: 315

```
1             outputToken.safeTransfer(msg.sender, 1
                  _000_000_000_000_000_000);
```

[I-9]: Unused Custom Error

it is recommended that the definition be removed when custom error is unused

- Found in src/PoolFactory.sol Line: 22

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