T-Swap Audit Report

Version 1.0

Trashpirate.io

November 18, 2024

T-Swap Audit Report

Trashpirate.io

November 18, 2024

Prepared by: Trashpirate
Lead Auditors:
- Nadina Oates

Table of Contents

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

Disclaimer

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

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

This audit report was prepared as part of a security tutorial created by Patrick Collins (Cyfrin Updraft).

Issues found

Severity	Number of issues found
High	4
Medium	2
Low	2
Info	9
Gas	0
Total	16

Findings

High

[H-1] Wrong fee precision in the function `TSwapPool::getInputAmountBasedOnOutput` results in 10 times inflated fee amount

Description

The precision value to calculate the fee amount the function `TSwapPool::getInputAmountBasedOnOutput` is `10000` instead of `1000`. This results in the fee amount being inflated by a factor of 10 compared to the fee taken in `TSwapPool::getOutputAmountBasedOnInput`.

1 Found Instances

- Found in `src/TSwapPool.sol` Line: 294

```
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 }
```

Impact

The function `TSwapPool::getInputAmountBasedOnOutput` subtracts the wrong fee amount.

Proof of Concepts

The function `TSwapPool::getInputAmountBasedOnOutput` returns 10 times the amount than the correct formula `inputAmount = ((inputReserves * outputAmount)* 1000)/ ((outputReserves - outputAmount)* 997)`.

Code

Place following code into `TSwapPool.t.sol`:

```
1  ```solidity
```

```
function testGetInputAmountBasedOnOutput() public {
    uint256 inputReserves = 100e18;
    uint256 outputReserves = 100e18;
    uint256 outputAmount = 10e18;

    uint256 expectedInputAmount = ((inputReserves * outputAmount) *
        1000) / ((outputReserves - outputAmount) * 997);
    uint256 actualInputAmount = pool.getInputAmountBasedOnOutput(
        outputAmount, inputReserves, outputReserves);

    assertEq(expectedInputAmount, actualInputAmount);
}
```

Recommended mitigation

Change the precision value to `1000` in the function `TSwapPool::getInputAmountBasedOnOutput`.

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

[H-2] Missing input parameter `minInputAmount` in the function `TSwapPool::swapExactOutput` can lead to front-running attacks

Description

The function `TSwapPool::swapExactOutput` does not include the input parameter `maxInputAmount` which can lead to front-running attacks. The `maxInputAmount` parameter is used to specify the maximum amount of input tokens that the user is willing to swap. If the `maxInputAmount` is not specified, the user can be front-run by another user or a malicious actor and result in more input tokens than expected (user sells token at a lower price than expected).

1 Found Instances

- Found in src/TSwapPool.sol Line: 352

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

    inputAmount = getInputAmountBasedOnOutput(
@>
        outputAmount,
        inputReserves,
        outputReserves
    );

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

Impact

User can be front-run by another user and result less input tokens than expected.

Proof of Concepts

1. User wants to swap 11 pool tokens for 10 weth tokens
2. Malicious actor manipulates the price of the token
3. User receives pays 18 tokens instead of 11

Code

Place following code into TSwapPool.t.sol:

```
```solidity
function testSwapExactOutput() public {
    uint256 outputAmount = 10e18;

    // liquidity provider deposits
    vm.startPrank(liquidityProvider);
    weth.approve(address(pool), 100e18);
    poolToken.approve(address(pool), 100e18);
    pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
    vm.stopPrank();

```
```

```
12      // set up user
13      poolToken.mint(user, 200e18);
14
15      uint256 initialPoolTokenBalance = poolToken.balanceOf(user);
16      uint256 initialWethBalance = weth.balanceOf(user);
17
18      // user does regular swap
19      vm.startPrank(user);
20      poolToken.approve(address(pool), type(uint256).max);
21      pool.swapExactOutput(poolToken, weth, outputAmount, uint64(block.
            timestamp));
22      vm.stopPrank();
23
24      uint256 finalPoolTokenBalance = poolToken.balanceOf(user);
25      uint256 tokenDifference = initialPoolTokenBalance -
            finalPoolTokenBalance;
26      console.log("PoolToken Difference: ", tokenDifference);
27
28      address malicousUser = makeAddr("malicousUser");
29      poolToken.mint(malicousUser, 200e18);
30
31      // malicous user does swap just before user
32      vm.startPrank(malicousUser);
33      poolToken.approve(address(pool), type(uint256).max);
34      pool.swapExactOutput(poolToken, weth, outputAmount, uint64(block.
            timestamp));
35      vm.stopPrank();
36
37      // user does swap
38      vm.startPrank(user);
39      poolToken.approve(address(pool), type(uint256).max);
40      pool.swapExactOutput(poolToken, weth, outputAmount, uint64(block.
            timestamp));
41      vm.stopPrank();
42
43      uint256 finalPoolTokenBalanceAfterMEV = poolToken.balanceOf(user);
44      uint256 tokenDifferenceAfterMEV = finalPoolTokenBalance -
            finalPoolTokenBalanceAfterMEV;
45
46      assertGt(tokenDifferenceAfterMEV, tokenDifference);
47      console.log("PoolToken Difference After MEV: ",
            tokenDifferenceAfterMEV);
48  }
49  ```
```

Recommended mitigation

Add imput parameter `maxInputAmount` to the function `TSwapPool::swapExactOutput` so the function reverts if a specified token amount (`maxInputAmount`) is exceeded.

```
1  function swapExactOutput(
```

```
        IERC20 inputToken,
+       uint256 maxInputAmount,
        IERC20 outputToken,
        uint256 outputAmount,
        uint64 deadline
    )
        public
        revertIfZero(outputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 inputAmount)
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

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

[H-3] False function call in `TSwapPool::sellPoolTokens` function leads to wrong output amount

Description

Instead of calling the function `swapExactOutput` in the function `TSwapPool::sellPoolTokens`, the function `swapExactInput` is called. This results in the wrong output amount being calculated and returned.

1 Found Instances

- Found in `src/TSwapPool.sol` Line: 369

```
 function sellPoolTokens(
    uint256 poolTokenAmount
) external returns (uint256 wethAmount) {
    return
@>      swapExactOutput(
            i_poolToken,
            i_wethToken,
            poolTokenAmount,
            uint64(block.timestamp)
        );
}
```

Impact

Swap logic of the function `TSwapPool::sellPoolTokens` is incorrect and returns the wrong output amount.

Proof of Concepts

1. user is using `TSwapPool::swapExactInput` using the exact `tokenAmount` they want to sell 2. user receives 9.066 WETH tokens and sells 10.00PoolTokens 3. user is using `TSwapPool::sellPoolTokens` using the exact `tokenAmount` they want to sell 4. user receives 10 WETH tokens and sells 13.63 PoolTokens

Code

Place following code into `TSwapPool.t.sol`:

```
```solidity
function testSellPoolToken() public {
    uint256 tokenAmount = 10e18;
    uint256 expectedWeth = 9e18;

    // liquidity provider deposits
    vm.startPrank(liquidityProvider);
    weth.approve(address(pool), 100e18);
    poolToken.approve(address(pool), 100e18);
    pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
    vm.stopPrank();

    // set up user
    poolToken.mint(user, 200e18);
    vm.prank(user);
    poolToken.approve(address(pool), type(uint256).max);

    uint256 initialPoolTokenBalance = poolToken.balanceOf(user);
    uint256 initialWethBalance = weth.balanceOf(user);

    // user does regular swap using the swapExactInput function
    vm.prank(user);
    pool.swapExactInput(poolToken, tokenAmount, weth, expectedWeth,
        uint64(block.timestamp));

    uint256 intermediatePoolTokenBalance = poolToken.balanceOf(user);
    uint256 intermediateWethBalance = weth.balanceOf(user);
    uint256 tokensSold1 = initialPoolTokenBalance -
        intermediatePoolTokenBalance;
    uint256 wethReceived1 = intermediateWethBalance -
        initialWethBalance;
    // tokensSold1 = 10.000000000000000000 PoolToken
    // wethReceived1 = 9.066108938801491315 WETH

    // user swap with helper function
    vm.prank(user);
```
```

```
34      pool.sellPoolTokens(tokenAmount);
35      uint256 tokensSold2 = intermediatePoolTokenBalance - poolToken.
            balanceOf(user);
36      uint256 wethReceived2 = weth.balanceOf(user) -
            intermediateWethBalance;
37      // tokensSold2 = 13.632236326745931084 PoolToken
38      // wethReceived2 = 10.000000000000000000 WETH
39
40      assertEq(tokensSold1, tokenAmount);
41      assertEq(tokensSold2, tokenAmount);
42      assertEq(wethReceived1, wethReceived2);
43  }
44
45  ```
```

Recommended mitigation

Replace the function call `swapExactOutput` with `swapExactInput` in the function `TSwapPool::sellPoolTokens`. A slippage parameter such as `minOutputAmount` should be added to the function signature.

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

[H-4] In `TSwapPool::_swap` the extra tokens 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` is the amount of pool token - `y` is the amount of WETH - `k` is the constant product value of the two balances

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

1 Found Instances

- Found in `src/TSwapPool.sol` Line: 400

```
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 funds by doing many swaps of low amounts and collecting the extra incentive given out by the protocol. This means the protocol's core invariant is broken.

Proof of Concepts

User swaps 10 times a small amount of WETH and the protocol's invariant is broken.

Code

Place following code into `TSwapPool.t.sol`:

```
1  ```solidity
2  function testInvariantBroken() public {
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      vm.startPrank(user);
10     poolToken.approve(address(pool), 10e18);
11
12     uint256 outputWeth = 10e15;
13     int256 startingY = int256(weth.balanceOf(address(pool)));
14     int256 expectedDeltaY = int256(-1) * int256(outputWeth);
15
16     uint256 numberOfSwaps = 10;
17     vm.startPrank(user);
18     poolToken.approve(address(pool), type(uint256).max);
19     for (uint256 index = 0; index < numberOfSwaps; index++) {
20         pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
```

```
                timestamp));
21      }
22      vm.stopPrank();
23
24      uint256 endingY = weth.balanceOf(address(pool));
25      int256 actualDeltaY = int256(endingY) - int256(startingY);
26
27      assertEq(actualDeltaY, expectedDeltaY);
28  }
29  ```
```

Recommended mitigation

Several options are available to mitigate this issue: 1. Remove the extra incentive 2. Account for change in the `x * y = k` protocol invariant 3. Process extra incentive the same way as the protocol 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);
15      }
```

Medium

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

Description

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

1 Found Instances

- Found in `src/TSwapPool.sol` Line: 117

```
 function deposit(
    uint256 wethToDeposit,
    uint256 minimumLiquidityTokensToMint,
    uint256 maximumPoolTokensToDeposit,
@>  uint64 deadline
)
    external
    revertIfZero(wethToDeposit)
    returns (uint256 liquidityTokensToMint)
{}
```

Impact

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

Proof of Concepts

The `deadline` parameter is unused.

Compiler Output

Following compiler warning indicates unused `deadline` parameter:

```
```bash
Warning (5667): Unused function parameter. Remove or comment out the
    variable name to silence this warning.
--> src/TSwapPool.sol:99:9:
|
99 |         uint64 deadline
|         ^^^^^^^^^^^^^^^
```
```

Recommended mitigation

Consider following change to the function:

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

[M-2] Rebase, fee-on-transfer, and ERC777 tokens break protocol invariant

Description

The protocol follows a strict invariant of `x * y = k`. Where: - `x` is the amount of pool token - `y` is the amount of WETH - `k` is the constant product value of the two balances

This means whenever the balances change in the protocol, the ratio between the two amounts should remain constant. However, this is broken when the token amount is manipulated on transfer as it is the case for ERC20 tokens that have transfer/swap fees because it is not accounted for in the `TSwapPool::_swap` function.

1 Found Instances

- Found in `src/TSwapPool.sol` Line: 385

```
1   function _swap(...){...}
```

Impact

A user could maliciously drain the protocol funds by doing many swaps with a poorly designed ERC20 token. This means the protocol's core invariant is broken.

Proof of Concepts

User swaps an ERC20 with fees on transfer multiple times and breaks the protocol invariant.

Code

Place following code into `TSwapPool.t.sol`:

```
1  ```solidity
2  function testStrangeERC20() public {
3      vm.startPrank(liquidityProvider);
4      ERC20FeeOnTransferMock strangeToken = new ERC20FeeOnTransferMock();
5      TSwapPool strangePool = new TSwapPool(address(strangeToken),
           address(weth), "LTokenA", "LA");
6      weth.approve(address(strangePool), 100e18);
7      strangeToken.approve(address(strangePool), 100e18);
8      strangePool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp)
           );
9      strangeToken.transfer(user, 100e18);
10     vm.stopPrank();
11
12     vm.startPrank(user);
13     strangeToken.approve(address(strangePool), 10e18);
14
15     uint256 outputWeth = 10e15;
16     int256 startingY = int256(weth.balanceOf(address(strangePool)));
17     int256 expectedDeltaY = int256(-1) * int256(outputWeth);
18
```

```
19      uint256 numberOfSwaps = 2;
20      vm.startPrank(user);
21      strangeToken.approve(address(strangePool), type(uint256).max);
22      for (uint256 index = 0; index < numberOfSwaps; index++) {
23          strangePool.swapExactOutput(strangeToken, weth, outputWeth,
                uint64(block.timestamp));
24      }
25
26      vm.stopPrank();
27
28      uint256 endingY = weth.balanceOf(address(strangePool));
29      int256 actualDeltaY = int256(endingY) - int256(startingY);
30
31      assertEq(actualDeltaY, expectedDeltaY);
32  }
33  ```
```

Recommended mitigation

1. Prevent ERC20 tokens with fees, rebase, or ERC777 2. Account for change in the `x * y = k` protocol invariant

Low

`### [L-1] Event parameters in`TSwapPool::_addLiquidityMintAndTransfer‘ function are incorrect resulting in wrong event logs

Description

The event definition `TSwapPool::LiquidityAdded` indicates that the first parameter is the `liquidityProvider` address, the second parameter is the `wethDeposited` amount, and the third parameter is the `poolTokensDeposited` amount. However, the event is emitted with the parameters in the wrong order - `poolTokenDeposit` in second, and `wethDeposit` in third position. This results in the event logs being incorrect.

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

1 Found Instances

- Found in src/TSwapPool.sol Line: 196

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

Recommended mitigation Swap event paramters in second and third position.

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

[L-2] Unused return value in `TSwapPool::swapExactInput` function is unused and should be removed

Description

The return value of the `TSwapPool::swapExactInput` function is not used and will always return zero. This could cause confusion as a value is expected based on the function defintion.

1 Found Instances

- Found in `src/TSwapPool.sol` Line: 308

```
function swapExactInput(
    IERC20 inputToken,
    uint256 inputAmount,
    IERC20 outputToken,
    uint256 minOutputAmount,
    uint64 deadline
)
    public
    revertIfZero(inputAmount)
    revertIfDeadlinePassed(deadline)
    returns (
@>      uint256 output
    )
{}
```

Impact

The return value of the `TSwapPool::swapExactInput` function is not used and will always return zero possibly causing confusion or disrupt functionality that depends on the return value.

Proof of Concepts

1. Liquidity is provided
2. User swaps tokens
3. The function `TSwapPool::swapExactInput` always returns 0 regardless of the input value

Code

Place following code into `TSwapPool.t.sol`:

```
```solidity
function testSwapExactInputAlwaysReturnsZero(uint256 tokenAmount)
    public {
    tokenAmount = bound(tokenAmount, 1, 100e18);
    uint256 wethAmount = 100e18;

    // set up liquidity pool
    vm.startPrank(liquidityProvider);
    weth.approve(address(pool), type(uint256).max);
    poolToken.approve(address(pool), type(uint256).max);
    pool.deposit(wethAmount, wethAmount, 2 * tokenAmount, uint64(block.
        timestamp));
    vm.stopPrank();

    // set up user
    poolToken.mint(user, tokenAmount);
    vm.startPrank(user);
    poolToken.approve(address(pool), type(uint256).max);

    // swap
    uint256 output = pool.swapExactInput(poolToken, tokenAmount, weth,
        0, uint64(block.timestamp));
    assertEq(output, 0);
}
```
```

Recommended mitigation

There are two options to mitigate this issue:

1. Remove the return value from the function definition.
2. Rename the return value from `output` to `outputAmount`.

```
    function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (
-           uint256 output
+           uint256 outputAmount
        )
    {}
```

Informational

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

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

[I-2] Wrong naming of `liquidityTokenSymbol` in `PoolFactory::createPool` can lead to confusion and illegibility

Description

For the naming of the `liquidityTokenSymbol` in the `PoolFactory::createPool` function, the `IERC20(tokenAddress).name()` is used instead of `IERC20(tokenAddress).symbol()`. This can lead to confusion and illegibility of the codebase.

```
function createPool(address tokenAddress) external returns (address) {
        if (s_pools[tokenAddress] != address(0)) {
            revert PoolFactory__PoolAlreadyExists(tokenAddress);
        }
        string memory liquidityTokenName = string.concat("T-Swap ",
            IERC20(tokenAddress).name());
+       string memory liquidityTokenSymbol = string.concat("ts", IERC20
    (tokenAddress).symbol());
-       string memory liquidityTokenSymbol = string.concat("ts", IERC20
    (tokenAddress).name());
        TSwapPool tPool = new TSwapPool(tokenAddress, i_wethToken,
            liquidityTokenName, liquidityTokenSymbol);
        s_pools[tokenAddress] = address(tPool);

        s_tokens[address(tPool)] = tokenAddress;
        emit PoolCreated(tokenAddress, address(tPool));
        return address(tPool);
    }
```

[I-3] 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.

4 Found Instances

- Found in src/PoolFactory.sol Line: 35

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

- Found in src/TSwapPool.sol Line: 52

```
1       event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
1       event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
1       event Swap(
```

[I-4] Missing zero checks can lead to false initialzation of immutable variables in constructor

Description

When initializing immutable address variables in the constructor it is recommended to check for zero address to avoid false initialization that cannot be later.

2 Found Instances

- Found in src/PoolFactory.sol Line: 41

```
1       i_wethToken = wethToken;
```

- Found in src/TSwapPool.sol Line: 96

```
1       i_wethToken = IERC20(wethToken);
2       i_poolToken = IERC20(poolToken);
```

Recommended mitigation

Add zero address check to the `PoolFactory::constructor` and `TSwapPool::constructor` functions.

Example:

```
1       constructor(address wethToken) {
2   +       if(wethToken == address(0)) {
3   +           revert PoolFactory__ZeroAddress(wethToken);
4   +       }
5           i_wethToken = IERC20(wethToken);
6       }
```

[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.

1 Found Instances

- Found in src/TSwapPool.sol Line: 298

```
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.

4 Found Instances

- Found in src/TSwapPool.sol Line: 276

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

- Found in src/TSwapPool.sol Line: 295

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

- Found in src/TSwapPool.sol Line: 454

```
1                  1e18,
```

- Found in src/TSwapPool.sol Line: 463

```
1                  1e18,
```

[I-7] State variable `TSwapPool::poolTokenReserves` is not used and should be removed

1 Found Instances

- Found in src/TSwapPool.sol Line: 131

```
1      uint256 poolTokenReserves = i_poolToken.balanceOf(address(this
           ));
```

[I-8] Error parameter `TSwapPool::MINIMUM_WETH_LIQUIDITY` is constant and can be removed

1 Found Instances

- Found in src/TSwapPool.sol Line: 125

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

[I-9] The function `TSwapPool::_addLiquidityMintAndTransfer` contains external calls and therefor should be used in CEI (Check-Effects-Interactions) pattern

1 Found Instances

- Found in src/TSwapPool.sol Line: 177

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