Protocol Audit Report

Version 1.0

Akshat

August 1, 2024

Protocol Audit Report

Akshat

August 1 , 2024

Prepared by: Akshat Lead Auditors: - Akshat

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).

Disclaimer

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

Risk Classification

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

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

Audit Details

- 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

I enjoyed doing this audit as I learned a lot of defi and fuzz testing.

Issues found

Severity	Number of issues found
High	4
Medium	2
Low	3
Gas	3
Info	8
Total	20

Findings

High

[H-1] `TSwapPool::_swap` function rewards user with 1 token every 10 swaps , which breaks the protocol invariant , severely breaking swapping functionality of the protocol

Description: `TSwapPool::_swap` function does the following : “Every 10 swaps, we give the caller an extra token as an extra incentive to keep trading on T-Swap.”

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

```
4         }
```

Basically the user gets 1 `outputToken` every 10 swaps. But the invariant of swapping in this protocol follows Constant Product AMM , i.e. ,`x*y = k`, but if either x or y (which represent the 2 tokens of the pool) decrease by 1 , then this equality breaks , breaking protocol functionality.

Impact: Invariant of the protocol breaks.

Also a malicious user may do a lot of swaps and drain the protocol's balance.

Proof of Concept: This bug can be found by stateful fuzz testing. But I have converted the results from this fuzz testing into a unit test , so it is easier to understand and incorporate into your test suite

Proof of Code

Add the following test to `TSwapPool.t.sol`

```
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.approve(address(pool), type(uint256).max);
12        poolToken.mint(user, 100e18);
13        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
14        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
15        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
16        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
17        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
18        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
19        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
20        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
21        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp)); // 9 swaps
22 
23        int256 startingY = int256(weth.balanceOf(address(pool)));
24        int256 expectedDeltaY = int256(-1) * int256(outputWeth);
25 
```

```
26          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp)); // 10th swap breaks invariant
27          vm.stopPrank();
28
29          uint256 endingY = weth.balanceOf(address(pool));
30          int256 actualDeltaY = int256(endingY) - int256(startingY);
31          assertEq(actualDeltaY, expectedDeltaY);
32      }
```

Recommended Mitigation: Remove this 'rewarding-the-user-every-10-swaps' functionality

```
1  -    uint256 private swap_count = 0;
2  -    uint256 private constant SWAP_COUNT_MAX = 10;
3       .
4       .
5       .
6
7       /**
8        * @notice Swaps a given amount of input for a given amount of
             output tokens.
9  -     * @dev Every 10 swaps, we give the caller an extra token as an
       extra incentive to keep trading on T-Swap.
10       * @param inputToken ERC20 token to pull from caller
11       * @param inputAmount Amount of tokens to pull from caller
12       * @param outputToken ERC20 token to send to caller
13       * @param outputAmount Amount of tokens to send to caller
14       */
15      function _swap(
16          .
17          .
18          .
19 -        swap_count++;
20 -        if (swap_count >= SWAP_COUNT_MAX) {
21 -            swap_count = 0;
22 -            outputToken.safeTransfer(msg.sender, 1
       _000_000_000_000_000_000);
23 -        }
```

[H-2] `TSwapPool::getInputAmountBasedOnOutput` incorrectly calculates fees , charging more fees than it should

Description: `TSwapPool::getInputAmountBasedOnOutput` function is for the user to specify the output they want , and to get the amount of input they would need to give to get that much output. But the calculation done is incorrect and is currently charging `90.03%` fees instead of `0.3%` !! This is happening as the function is scaling the output by 10_000 instead of 1_000.

```
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: `TSwapPool::getInputAmountBasedOnOutput` is used inside `TSwapPool::swapExactOutput` , so whenever a user calls `TSwapPool::swapExactOutput` function to swap tokens , they will end up giving 90% fees.

Proof of Concept: 1. Liquidity Provider deposits 100 weth and 100 poolTokens 2. User wants to swap and get 10 weth 3. According to 0.3% fee the input amount should be around 11.14 poolTokens , but since the formula is incorrect , hence the input amount is 111.4 poolTokens , which is HUGE !

PoC

Place the following test into your `TSwapPool.t.sol` test suite

```
1       function test_getInputAmountBasedOnOutput_IncorrectlyCalculatesFees
            () public
2       {
3
4           weth.mint(user, 190e18);
5           poolToken.mint(user, 190e18); // to make user balance in both
                tokens = 200
6
7           vm.startPrank(liquidityProvider);
8           weth.approve(address(pool), 100e18);
9           poolToken.approve(address(pool), 100e18);
10          pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
11          vm.stopPrank();
12
13          vm.startPrank(user);
14          poolToken.approve(address(pool), 120e18);
15          pool.swapExactOutput(poolToken, weth, 10e18, uint64(block.
                timestamp));
16          // for 10 weth output
17          // expected(for 0.3% fee) = 11.14 poolTokens as input
18          // actual (for 90.03% fee) = 111.4 poolTokens
19          vm.stopPrank();
20          // To prove protocol is charging 90.03% fee , we will show that
                111.4 poolTokens were deducted from user balance instead of
```

```
                11.14 poolTokens
21          assert(poolToken.balanceOf(user) < 886e17); // a little more
                than 111.4 poolTokens were deducted hence a little less than
                 88.6 were left (200 - 111.4)
22          // console.log(poolToken.balanceOf(user)); //
                88.554552546528474312
23          // console.log(poolToken.balanceOf(address(pool))) ; //
                211.445447453471525688
24          assert(poolToken.balanceOf(address(pool)) > 2114e17); // it
                will have little more than 100 + 111.4 = 211.4 (with 18 DP)
25      }
```

Recommended Mitigation: Change the formula and scale by 1000 instead of 10000

```
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  +           ((inputReserves * outputAmount) * 1000) /
15              ((outputReserves - outputAmount) * 997);
16      }
```

[H-3] `TSwapPool::swapExactOutput` doesn't have a `maxInputAmount` parameter , causing user to pay huge amounts if market isn't favourable for that swap (Lack of slippage protection)

Description: `TSwapPool::swapExactOutput` function is for a user who wants to swap to get the number of input tokens they have to send to get a particular amount of output tokens. The protocol calculates it and returns this value. But if the market is unfavourable , this amount may be too high and it may not be preferred to swap at this rate.

Impact: The user may have to send a large amount of tokens to buy a small amount of tokens.

Proof of Concept: 1. Let , pool contains 1000 weth and 10 poolTokens 2. User wants to swap some weth for 5 poolTokens 3. Users ends up paying over 10_000 weth for 5 poolTokens!! - Keep in mind that `TSwapPool::swapExactOutput` also has a bug which scales the input amount needed by 10 times , but even without that bug , user would need to pay over 1_000 weth for 5 poolTokens , which

is HUGE.(This is Huge due to market conditions , nothing wrong with the protocol , its just how the protocol mechanism is designed to change exchange rates based on funds in liquidity pools , which is completely fine)

PoC

Place the following test into your `TSwapPool.t.sol` test suite

```
    function test_swapExactOutput_HugeInputForSmallOutput() public
    {

        weth.mint(liquidityProvider, 800e18); // now lp has 1000 weth
        weth.mint(user, 10030e18); // now user has 10_040 weth

        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 1000e18);
        poolToken.approve(address(pool), 10e18);
        pool.deposit(1000e18, 1000e18, 10e18, uint64(block.timestamp));
        vm.stopPrank();

        console.log(weth.balanceOf(user)); // 10040.000000000000000000
           weth
        console.log(poolToken.balanceOf(user)); //
           10.000000000000000000 poolToken
        console.log(weth.balanceOf(address(pool)));  //
           1000.000000000000000000 weth
        console.log(poolToken.balanceOf(address(pool))); //
           10.000000000000000000 poolToken

        vm.startPrank(user);
        weth.approve(address(pool), 10031e18);
        pool.swapExactOutput(weth, poolToken, 5e18, uint64(block.
           timestamp));
        vm.stopPrank();

        console.log(weth.balanceOf(user)); // 9.909729187562688065 weth
        console.log(poolToken.balanceOf(user)); //
           15.000000000000000000 poolToken
        console.log(weth.balanceOf(address(pool))); //
           11030.090270812437311935 weth
        console.log(poolToken.balanceOf(address(pool))); //
           5.000000000000000000 poolToken

        assert(weth.balanceOf(user) < 10e18); // started with 10_040
           weth , and just to take out 5 poolTokens , had to pay 10_030
            weth, which is HUGE!!
    }
```

Recommended Mitigation: Add a `maxInputAmount` parameter , which specifies max number of tokens the user is willing to send as input to get the desired number of output tokens , and if the

required input tokens exceed this value , we can just revert , protecting users from unfavourable market conditions like the one shown above.

```
+   error TSwapPool__InputTooHigh(uint256 actual, uint256 max)
    .
    .
    .


    function swapExactOutput(
        IERC20 inputToken,
        IERC20 outputToken,
+       uint256 maxInputAmount,
        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 TSwapPool__InputTooHigh(inputAmount,maxInputAmount)
+       }

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

[H-4] `TSwapPool::sellPoolTokens` function incorrectly calls `swapExactOutput` , causing un-favourable behaviour

Description: `TSwapPool::sellPoolTokens` function is supposed to be a “wrapper function to facilitate users selling pool tokens in exchange of WETH” . It calls `swapExactOutput` with wrong arguments. `swapExactOutput` accepts four arguments

```
    function swapExactOutput(
        IERC20 inputToken,
```

```
3           IERC20 outputToken,
4   =>      uint256 outputAmount,
5           uint64 deadline
6       )
```

and `TSwapPool::sellPoolTokens` function calls `swapExactOutput` as follows

```
1       function sellPoolTokens(
2           uint256 poolTokenAmount
3       ) external returns (uint256 wethAmount) {
4           return
5               swapExactOutput(
6                   i_poolToken,
7                   i_wethToken,
8   =>              poolTokenAmount,
9                   uint64(block.timestamp)
10              );
11      }
```

The problematic line being `poolTokenAmount` being mentioned as `outputAmount` which is not at all the intended functionality of `TSwapPool::sellPoolTokens`.

Impact: `TSwapPool::sellPoolTokens` function will not behave as it is supposed to , and users will end up buying `poolTokenAmount` number of weth.

Recommended Mitigation: `TSwapPool::sellPoolTokens` can call the `swapExactInput` function instead.

```
1       function sellPoolTokens(
2   -       uint256 poolTokenAmount
3   +       uint256 poolTokenAmount,
4   +       uint256 minWethAmount // minimum amount of weth user is
        expecting to get
5       ) external returns (uint256 wethAmount) {
6           return
7   -           swapExactOutput(
8   -               i_poolToken,
9   -               i_wethToken,
10  -               poolTokenAmount,
11  -               uint64(block.timestamp)
12  -           );
13  +           swapExactInput(
14  +               i_poolToken,
15  +               poolTokenAmount,
16  +               i_wethToken,
17  +               minWethAmount,
18  +               uint64(block.timestamp)
19  +           )
20      }
```

Medium

[M-1] `TSwapPool::deposit` has a unused parameter , `deadline` , meaning transactions will go through even after deadline specified by the Liquidity Provider

Description: `TSwapPool::deposit` has `deadline` as one of its parameters , but it is not used anywhere in the function

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

    }
```

Impact: The LP (Liquidity Provider) who sent their transaction to deposit funds into the pool may have their transaction may be executed at unexpected times where market conditions are unfavourable for depositing.

Recommended Mitigation: Add a timestamp check on the deadline , mnaking sure transaction only goes through before the deadline , else reverts.

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

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

Description: The T-Swap protocol assumes all tokens behave like standard ERC20 tokens. However, it doesn't account for non-standard tokens such as rebasing tokens, fee-on-transfer tokens, and ERC777

tokens. These tokens can manipulate balances in unexpected ways, breaking the core invariant of the protocol: x * y = k (constant product formula). - Rebasing tokens: These tokens can change the balance of holders without transfers, affecting the pool's balance unexpectedly. - Fee-on-transfer tokens: These tokens deduct a fee on each transfer, resulting in fewer tokens received than sent. - ERC777 tokens: These tokens have hooks that can execute code before and after transfers, potentially manipulating balances or reentering the protocol.

Impact: - Severe disruption of the protocol's core functionality - Potential loss of funds for liquidity providers and traders - Manipulation of exchange rates and liquidity pool balances - Broken invariants leading to incorrect price calculations and swaps

Low

[L-1] Constructor of `PoolFactory` lacks a zero-check on address of `i_wethToken` . Same for constructor of `PoolFactory`

Description: Constructor of `PoolFactory` sets the address of `i_wethToken` but doesnt check whether the address is non-zero or not

```
1      constructor(address wethToken) {
2  =>      i_wethToken = wethToken;
3      }
```

Similar in constructor of `PoolFactory`

```
1      constructor(
2          address poolToken,
3          address wethToken,
4          string memory liquidityTokenName,
5          string memory liquidityTokenSymbol
6      ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
7  =>      i_wethToken = IERC20(wethToken);
8  =>      i_poolToken = IERC20(poolToken);
9      }
```

Impact: The address of `i_wethToken` and/or `poolToken` might be mistakenly be assigned to `address(0)` which will cause all the tokens being sent to this address to be burnt , severely breaking protocol functionality

Recommended Mitigation: Incorporate a zero-address check in the constructor

```
1
2  +    error PoolFactory__InvalidAddress();
3       .
```

```
4       .
5       .
6
7
8       constructor(address wethToken) {
9   +       if(wethToken != address(0)){
10  +           i_wethToken = wethToken;
11  +       }
12  +       else{
13  +           revert PoolFactory__InvalidAddress();
14  +       }
15  -       i_wethToken = wethToken;
16      }
```

```
1
2   +   error PoolFactory__InvalidAddress();
3       .
4       .
5       .
6       constructor(
7           address poolToken,
8           address wethToken,
9           string memory liquidityTokenName,
10          string memory liquidityTokenSymbol
11      ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
12  +       if(wethToken != address(0)){
13  +           i_wethToken = IERC20(wethToken);
14  +       }
15  +       else{
16  +           revert PoolFactory__InvalidAddress();
17  +       }
18  +       if(poolToken != address(0)){
19  +           i_poolToken = IERC20(poolToken);
20  +       }
21  +       else{
22  +           revert PoolFactory__InvalidAddress();
23  +       }
24  -       i_wethToken = IERC20(wethToken);
25  -       i_poolToken = IERC20(poolToken);
26      }
```

[L-2] `TSwapPool::_addLiquidityMintAndTransfer` emits a event wrongly

Description:

`TSwapPool::_addLiquidityMintAndTransfer` emits the following event

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

But as we can see from the following event-declaration , the amount of weth deposited is the 2nd param , and poolTokens deposited the 3rd , and not the other way around

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

Impact: Incorrect event information causes confusion and may lead to some serious issues/bugs

Recommended Mitigation: Emit the event as follows

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

[L-3] `TSwapPool::swapExactInput` function is supposed to return the output numbers of tokens , but it is never returned, causing confusion and incorrect information being given.

Description: `TSwapPool::swapExactInput` function is supposed to return the output numbers of tokens. While it is declared in function declaration , but it is never updated inside the function body nor is explicity returned , causing default value , i.e. 0 , to be returned always.

Impact: Always 0 is returned , causing incorrect information to be sent to the user.

Recommended Mitigation: Make the following change

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

-       uint256 outputAmount = getOutputAmountBasedOnInput(
+       outputAmount = getOutputAmountBasedOnInput(
            inputAmount,
```

```
20              inputReserves,
21              outputReserves
22          );
23
24          if (outputAmount < minOutputAmount) {
25              revert TSwapPool__OutputTooLow(outputAmount,
                    minOutputAmount);
26          }
27
28          _swap(inputToken, inputAmount, outputToken, outputAmount);
29      }
```

Gas

[G-1] Unused event in `PoolFactory` should be removed

The following error is not used anywhere so should be removed.

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

[G-2] `TSwapPool::swapExactInput` function is never used inside the contract , so should be marked `external`

`TSwapPool::swapExactInput` function is currently declared as a **`public`** function , but is intended to be used only outside the contract , so it should be marked as `external` as it would save gas in deploying the contract.

[G-3] `TSwapPool::totalLiquidityTokenSupply` function is never used inside the contract , so should be marked `external`

`TSwapPool::totalLiquidityTokenSupply` function is currently declared as a **`public`** function , but is intended to be used only outside the contract , so it should be marked as `external` as it would save gas in deploying the contract.

Informational

[I-1] Events should have indexed fields

Events with less than or equal to 3 parameters , should have all params as indexed , and events with more than 3 params , should have 3 params as indexed. Indexing the parameters makes the protocol more transparent and makes off-chain monitoring easier.

Found Instances: - `PoolFactory` - event PoolCreated(address tokenAddress, address poolAddress); - `TSwapPool` - event LiquidityAdded(address indexed liquidityProvider, uint256 wethDeposited, uint256 poolTokensDeposited); - event LiquidityRemoved(address indexed liquidityProvider, uint256 wethWithdrawn, uint256 poolTokensWithdrawn); - event Swap(address indexed swapper, IERC20 tokenIn, uint256 amountTokenIn, IERC20 tokenOut, uint256 amountTokenOut);

[I-2] `PoolFactory::liquidityTokenSymbol` uses `.name()` property of ERC20 to create a symbol, instead `.symbol()` should be used

In `PoolFactory::createPool` function , `PoolFactory::liquidityTokenSymbol` is meant to be a symbol of the Liquidity Token that will be given to Liquidity Providers, and it is currently concatinating `ts` and name of the ERC20 (the `poolToken`). The name might be too big , hence symbol should be used instead

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

[I-3] Really big number literals can be replaced by scientific notation

1.

```
1 -   uint256 private constant MINIMUM_WETH_LIQUIDITY = 1_000_000_000;
2 +   uint256 private constant MINIMUM_WETH_LIQUIDITY = 1e9;
```

2.

```
1 -   outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000);
2 +   outputToken.safeTransfer(msg.sender, 1e18);
```

[I-4] `TSwapPool__WethDepositAmountTooLow` event emits a constant varaible as one of its parameters , which is not preferred

```
    error TSwapPool__WethDepositAmountTooLow(
        uint256 minimumWethDeposit,
        uint256 wethToDeposit
    );
    .
    .
    .
    function deposit
        .
        .
        .
        revert TSwapPool__WethDepositAmountTooLow(
=>              MINIMUM_WETH_LIQUIDITY,
                wethToDeposit
            );
```

`MINIMUM_WETH_LIQUIDITY` is a constant variable as can easily be accessed from the contract's bytecode hence we may not emit it in the error each time.

Instead just do :

```
    error TSwapPool__WethDepositAmountTooLow(
-       uint256 minimumWethDeposit,
        uint256 wethToDeposit
    );
    .
    .
    .
    function deposit
        .
        .
        .
        revert TSwapPool__WethDepositAmountTooLow(
-               MINIMUM_WETH_LIQUIDITY,
                wethToDeposit
            );
```

[I-5] `TSwapPool::deposit` function contains a redundant line which isn't used anywhere in the function so should be removed

```
    function deposit
    .
    .
    .
```

```
5
6       if (totalLiquidityTokenSupply() > 0) {
7           .
8           .
9           .
10  -       uint256 poolTokenReserves = i_poolToken.balanceOf(address(this)
        );
```

[I-6] `TSwapPool::getOutputAmountBasedOnInput` and `TSwapPool::getInputAmountBasedOnOutput` functions make use of number literals , instead they should be declared public constant variables and then used. Similar use of 'magic numbers' found in `TSwapPool::getPriceOfOneWethInPoolTokens` and `TSwapPool::getPriceOfOnePoolTokenInWeth`

Instances: - `getOutputAmountBasedOnInput`

```
1   =>      uint256 inputAmountMinusFee = inputAmount * 997;
2           uint256 numerator = inputAmountMinusFee * outputReserves;
3   =>      uint256 denominator = (inputReserves * 1000) +
        inputAmountMinusFee;
4           return numerator / denominator;
```

- `getInputAmountBasedOnOutput`

```
1       return
2   =>          ((inputReserves * outputAmount) * 10000) /
3   =>          ((outputReserves - outputAmount) * 997);
```

- `getPriceOfOneWethInPoolTokens`

```
1       function getPriceOfOneWethInPoolTokens() external view returns (
            uint256) {
2           return
3               getOutputAmountBasedOnInput(
4   =>              1e18,
5                   i_wethToken.balanceOf(address(this)),
6                   i_poolToken.balanceOf(address(this))
7               );
8       }
```

- `getPriceOfOnePoolTokenInWeth`

```
1       function getPriceOfOnePoolTokenInWeth() external view returns (
            uint256) {
2           return
3               getOutputAmountBasedOnInput(
```

```
=>                  1e18,
                    i_poolToken.balanceOf(address(this)),
                    i_wethToken.balanceOf(address(this))
                );
    }
```

[I-7] `TSwapPool::swapExactInput` is missing natspec

`TSwapPool::swapExactInput` is one of the most important functions of the protocol , and without a proper documentation or natspec , it may become confusing to understand or use this function.

Please add proper documentation to this function.

[I-8] `TSwapPool::deposit` function isn't following CEI

```
    function deposit(
        .
        .
        .
        } else {
            // This will be the "initial" funding of the protocol. We
               are starting from blank here!
            // We just have them send the tokens in, and we mint
               liquidity tokens based on the weth
            _addLiquidityMintAndTransfer(
                wethToDeposit,
                maximumPoolTokensToDeposit,
                wethToDeposit
            );
=>          liquidityTokensToMint = wethToDeposit;
        }
```

The `TSwapPool::_addLiquidityMintAndTransfer` makes external calls and we should update any state variables (here , `liquidityTokensToMint` , even though it is not a state variable but still) before making any external calls . It is good practice to follow Checks-Effects-Interactions(CEI).

Make the following change:

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

```
                maximumPoolTokensToDeposit,
                wethToDeposit
            );
-           liquidityTokensToMint = wethToDeposit;
        }
```