TSwap Protocol Audit Report

Version 1.0

cosminmarian53

April 25, 2025

TSwap Protocol Audit Report

cosminmarian53

April 25, 2025

Prepared by: cosminmarian53

Table of Contents

- Medium
  - [M-1]`TSwapPool::deposit` is missing deadline check causing transactions to complete even after the deadline has passed
  - [M-2] Rebase, fee-on-transfer, and ERC-777 tokens break protocol invariant
- Low
  - [L-1] In `TSwapPool::_addLiquidityMintAndTransfer` the `emit LiquidityAdded` event is not initialized correctly/not called with the proper parameters
  - [L-2] Default value returned value by `TSwapPool::swapExactInput` results in incorrect return value given
  - [L-3] Public Function Not Used Internally
- Informational
  - [I-1]`PoolFactory::PoolFactory__PoolDoesNotExist(address tokenAddress)` is not used and should be removed
  - [I-2] `PoolFactory::constructor` lacks zero address check
  - [I-3] `PoolFactory::createPool::liquidityTokenSymbol` should use `.symbol()` instead of `.name()`
  - [I-4] `TSwapPool` events should be indexed if there are more than 3 parameters
  - [I-5] `TSwapPool::constructor` lacks zero address check
  - [I-6] 'TSwapPool::getOutputAmountBasedOnInput' uses magic numbers instead of constants or immutables
  - [I-7] In `TSwapPool::deposit`, the variable `liquidityTokensToMint` should be updated before the external call
  - [I-8] `TSwapPool::swapExactOutput` and `TSwapPool::swapExactInput` have no natspec
- Gas
  - [G-1] In `TSwapPool::deposit` the variable `poolTokenReserves` is never used

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 main auditor. cosminmarian53 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: 1ec3c30253423eb4199827f59cf564cc575b46db

Scope

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

- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum

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

Engaging in the audit of this project has been an invaluable learning opportunity. As an aspiring auditor just beginning my journey, I've gained hands-on experience with fuzz-testing methodologies, honed my approach to specifying and verifying critical invariants, and deepened my insight into the most prevalent attack vectors that threaten DeFi protocols and the broader Web3 ecosystem. This process has not only expanded my technical skill set but also given me a richer appreciation for the intricate security challenges inherent in decentralized applications.

Issues found

Severity	Number of issues found
High	4
Medium	2
Low	3
Info	8
Gas	1
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 designed to work backwards from a desired output—given the number of tokens you want to receive, it should tell you exactly how many tokens you need to deposit. In practice, however, it returns the wrong value. The mistake lies in the fee calculation: instead of applying a multiplier of 1_000 to determine the fee, the code is using 10_000. As a result, every time you call this function, it over-scales the fee by a factor of ten and returns an inflated deposit requirement.

Impact:

Protocol takes more fees than expected from users.

Proof of Code:

The following test shows how the `getInputAmountBasedOnOutput` takes too many tokens from the user.

Add this to the testing suite:

testFlawedSwapExactOutput

```
    function testFlawedSwapExactOutput() public {
        uint256 initialLiquidity = 100e18;
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), initialLiquidity);
        poolToken.approve(address(pool), initialLiquidity);
        pool.deposit(
            initialLiquidity,
            0,
            initialLiquidity,
            uint64(block.timestamp)
        );
        vm.stopPrank();

        // User has 11 pool tokens
        address someUser = makeAddr("someUser");
        uint256 userInitialPoolTokenBalance = 11e18;
        poolToken.mint(someUser, userInitialPoolTokenBalance);

        vm.startPrank(someUser);
        poolToken.approve(address(pool), userInitialPoolTokenBalance);
```

```
21          pool.swapExactOutput(poolToken, weth, 1 ether, uint64(block.
               timestamp));
22          //  Initial liquidity is 1:1 so the use should have paid  ~1
               pool token
23          //  However, it spent much more than that
24          assertLt(poolToken.balanceOf(someUser), 1 ether);
25          vm.stopPrank();
26
27          vm.startPrank(liquidityProvider);
28          pool.withdraw(
29              pool.balanceOf(liquidityProvider),
30              1 /*minWethToWithdraw*/,
31              1/*minPoolTokensToWithdraw*/,
32              uint64(block.timestamp)
33          );
34          vm.stopPrank();
35          //Assert
36          assertEq(weth.balanceOf(address(pool)), 0);
37          assertEq(poolToken.balanceOf(address(pool)), 0);
38      }
```

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
13 -           return ((inputReserves * outputAmount) * 10_000) /
14 -           ((outputReserves - outputAmount) * 997);
15 +           return ((inputReserves * outputAmount) * 1_000) /
16 +           ((outputReserves - outputAmount) * 997);
17     }
```

[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:

Proof of Concept:

1. The price of 1 WETH right now is 1,000 USDC
2. User inputs a swapExactOutput looking for 1 WETH
    i. inputToken = USDC
    ii. outputToken = WETH
    iii. outputAmount = 1
    iv. deadline = 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 only has to spend up to a specific amount, and can predict how much they will spend on the protocol.

```
1      function swapExactOutput(
2          IERC20 inputToken,
3  +       uint256 maxInputAmount,
4  .
5  .
6  .
7          inputAmount = getInputAmountBasedOnOutput(outputAmount,
               inputReserves, outputReserves);
8  +       if(inputAmount > maxInputAmount){
9  +           revert(); //eventually, use a custom error here for
       verbosity
10 +       }
11         _swap(inputToken, inputAmount, outputToken, outputAmount);
```

[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're willing to sell in the `poolTokenAmount` parameter. However, the function currently miscalculates the swapped amount.

This is due to the fact that the `swapExactOutput` function is called, 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.

Recommended Mitigation:

Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require changing the `sellPoolTokens` function to accept a new parameter (ie `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, uint64(block.timestamp));
6  +        return swapExactInput(i_poolToken, poolTokenAmount,
       i_wethToken, minWethToReceive, uint64(block.timestamp));
7      }
```

Additionally, 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 `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 the two balances

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

The following block of code is responsible for the issue.

```
1  swap_count++;
2  if (swap_count >= SWAP_COUNT_MAX) {
3    swap_count = 0;
4    outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000);
5  }
```

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

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

Proof of Concept:

1. A user swaps 10 times, and collects the extra incentive of `1_000_000_000_000_000_000` tokens
2. That user continues to swap until all the protocol funds are drained

The following code proves that the invariant is broken. Add it to `TSwapPool.t.sol`:

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

        uint256 outputWeth = 1e17;

        vm.startPrank(user);
        poolToken.approve(address(pool), type(uint256).max);
        poolToken.mint(user, 100e18);
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
```

```
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );

        int256 startingX = int256(weth.balanceOf(address(pool)));
        int256 expectedDeltaX = int256(outputWeth) * -1;

        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        vm.stopPrank();

        int256 endingX = int256(weth.balanceOf(address(pool)));
        int256 actualDeltaX = int256(endingX) - int256(startingX);

        assertEq(actualDeltaX, expectedDeltaX);
    }
```

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.

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

Medium

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

Description: The `deposit` function 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 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 not optimal to deposit, even when adding a deadline parameter.

Proof of Concept: The `deadline` parameter is not used.(will soon add more data to back this up)

Recommended Mitigation:

```
  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 ERC-777 tokens break protocol invariant

Description: In the `_swap` function, we have a fee-on-transfer. This issue breaks the protocol invariant and could potentially lead to funds being stolen. I recommned looking at the WeirdERC20's fee-on-transfer Github repository.

Impact: This could lead to balancer pools being drained of all their funds. As an example, mentioned in the Github repository linked above, `The STA transfer fee was used to drain $500k from several balancer pools`.

Proof of Concept: We can use the test from [H-4] to prove again, that the invariant breaks.

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

        uint256 outputWeth = 1e17;

        vm.startPrank(user);
        poolToken.approve(address(pool), type(uint256).max);
        poolToken.mint(user, 100e18);
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
```

```
46              outputWeth,
47              uint64(block.timestamp)
48          );
49          pool.swapExactOutput(
50              poolToken,
51              weth,
52              outputWeth,
53              uint64(block.timestamp)
54          );
55          pool.swapExactOutput(
56              poolToken,
57              weth,
58              outputWeth,
59              uint64(block.timestamp)
60          );
61          pool.swapExactOutput(
62              poolToken,
63              weth,
64              outputWeth,
65              uint64(block.timestamp)
66          );
67
68          int256 startingX = int256(weth.balanceOf(address(pool)));
69          int256 expectedDeltaX = int256(outputWeth) * -1;
70
71          pool.swapExactOutput(
72              poolToken,
73              weth,
74              outputWeth,
75              uint64(block.timestamp)
76          );
77          vm.stopPrank();
78
79          int256 endingX = int256(weth.balanceOf(address(pool)));
80          int256 actualDeltaX = int256(endingX) - int256(startingX);
81
82          assertEq(actualDeltaX, expectedDeltaX);
83      }
```

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 -         swap_count++;
2 -         // Fee-on-transfer
3 -         if (swap_count >= SWAP_COUNT_MAX) {
4 -             swap_count = 0;
5 -             outputToken.safeTransfer(msg.sender, 1
      _000_000_000_000_000_000);
6 -         }
```

Low

[L-1] In `TSwapPool::_addLiquidityMintAndTransfer` the `emit LiquidityAdded` event is not initialized correctly/not called with the proper parameters

Description:

When the event `TSwapPool::LiquidityAdded` is emitted in `TSwapPool::_addLiquidityMintAndTrans` it is not called with the proper parameters, and it logs values in an incorrect order.

Impact:

This might return/display wrong information, misleading the user.

Proof Of Concept:

The event is emitted like this:

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

However, it is declared:

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

Recommended Mitigation: The `poolTokensToDeposit` value should go in the third parameter position, whereas the `wethToDeposit` value should go second. Change the order of the parameters, so that the emitted event returns the correct information to the user.

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

[L-2] Default value returned value 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.

Proof of Concept:

Add the following to the test suite:

Code

```
    function testSwapExactInputReturnsZero() public {
         uint256 initialLiquidity = 100e18;
         vm.startPrank(liquidityProvider);
         weth.approve(address(pool), initialLiquidity);
         poolToken.approve(address(pool), initialLiquidity);
         pool.deposit(
             initialLiquidity,
             0,
             initialLiquidity,
             uint64(block.timestamp)
         );
         vm.stopPrank();

         // User has 11 pool tokens
         address someUser = makeAddr("someUser");
         uint256 userInitialPoolTokenBalance = 11e18;
         poolToken.mint(someUser, userInitialPoolTokenBalance);

         vm.startPrank(someUser);
         poolToken.approve(address(pool), userInitialPoolTokenBalance);
         uint256 amountOut = pool.swapExactInput(
             poolToken,
             userInitialPoolTokenBalance,
             weth,
             0,
             uint64(block.timestamp)
         );
         assertEq(amountOut, 0);
     }
```

Recommended Mitigation:

```
{
    uint256 inputReserves = inputToken.balanceOf(address(this));
    uint256 outputReserves = outputToken.balanceOf(address(this));

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

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

```
        minOutputAmount);
12      }
13
14  -         _swap(inputToken, inputAmount, outputToken, outputAmount);
15  +         _swap(inputToken, inputAmount, outputToken, output);
16  }
17  }
```

[L-3] Public Function Not Used Internally

If a function is marked public but is not used internally, consider marking it as `external`.

2 Found Instances

- Found in src/TSwapPool.sol Line: 247

```
1       function swapExactInput(
```

- Found in src/TSwapPool.sol Line: 460

```
1       function totalLiquidityTokenSupply() public view returns
```

Informational

[I-1]`PoolFactory::PoolFactory__PoolDoesNotExist(address tokenAddress)` is not used and should be removed

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

[I-2] `PoolFactory::constructor` lacks zero address check

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

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

Description

The memory variable `liquidityTokenSymbol` calls the wrong function when concatenating strings.

Example

Instead of having let's say "tsWETH" we would have "tsWrappedEther" or something similar.

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

[I-4] `TSwapPool` events should be indexed if there are more than 3 parameters

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

[I-5] `TSwapPool::constructor` lacks zero address check

```
    constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
+       if((poolToken==0)||(wethToken==0))
+       {
+           revert(); //maybe consider adding a custom error for these issues.
+       }
        i_wethToken = IERC20(wethToken);
        i_poolToken = IERC20(poolToken);
    }
```

[I-6] 'TSwapPool::getOutputAmountBasedOnInput' uses magic numbers instead of constants or immutables

Found instances:

```
- uint256 inputAmountMinusFee = inputAmount * 997;
+ uint256 inputAmountMinusFee = inputAmount * PRECISION_MINUS_FEE; // or something like that, the choice is up to you how you call these variables but make sure they are suggestive
- uint256 denominator = (inputReserves * 1000) + inputAmountMinusFee;
+ uint256 denominator = (inputReserves * PRECISION_WITH_FEE) + inputAmountMinusFee;
```

This can help track or reuse such numbers, making the code more readable.

[I-7] In `TSwapPool::deposit`, the variable `liquidityTokensToMint` should be updated before the external call

Description:

The variable `liquidityTokensToMint` is updated after an external call, which does not follow CEI.

Impact:

This disrupts the readability of the code and is not a best practice.

Proof of Concept:

It happens in the following context:

```
_addLiquidityMintAndTransfer(
  wethToDeposit,
  maximumPoolTokensToDeposit,
  wethToDeposit
);
liquidityTokensToMint = wethToDeposit;
```

Recommended Mitigation:

Consider updating the variable before the external call, so that it follows CEI.

Code

```
+   liquidityTokensToMint = wethToDeposit;
    _addLiquidityMintAndTransfer(
                wethToDeposit,
                maximumPoolTokensToDeposit,
```

```
                wethToDeposit
            );
-   liquidityTokensToMint = wethToDeposit;
```

[I-8] `TSwapPool::swapExactOutput` and `TSwapPool::swapExactInput` have no natspec

Description: The `swapExactOutput` and `swapExactInput` have no natspecs. This makes the code harder to read and to understand by other developers.

Impact: This lowers the readability of the code, and does not follow best practices.

Recommended Mitigation:

Add natspecs to both functions.(The natspecs bellow are simply examples, they may require to be adjusted accordingly) For `swapExactOutput`:

```
+    /**
+     * @notice  .
+     * @dev     .
+     * @param   inputToken  .
+     * @param   outputToken  .
+     * @param   outputAmount  .
+     */
 function swapExactOutput(
```

For `swapExactInput`:

```
+    /**
+     * @notice  .
+     * @dev     .
+     * @param   inputToken  .
+     * @param   inputAmount  .
+     * @param   outputToken  .
+     * @param   minOutputAmount  .
+     * @param   deadline  .
+     */
    function swapExactInput(
```

Gas

[G-1] In `TSwapPool::deposit` the variable `poolTokenReserves` is never used

Description:

The variable `poolTokenReserves` is never used.

Impact:

This can lead to a gas cost increase, while small it can make a difference in fees.

Recommended Mitigation:

Remove the unused variable:

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