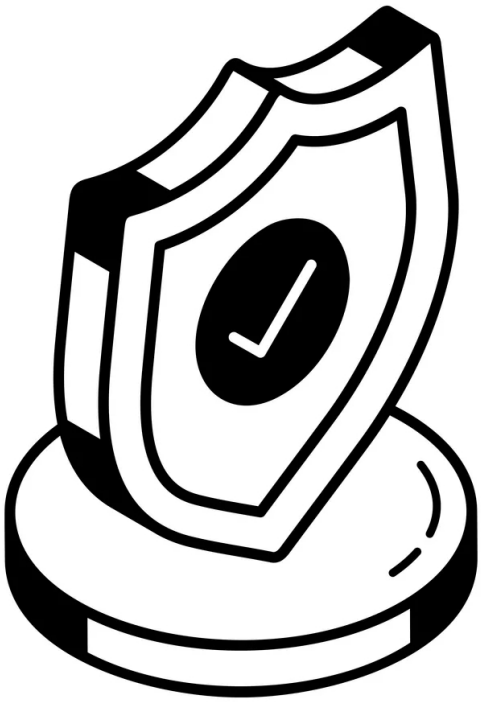

TSwap Audit Report

Version 1.0

YANPITRIK

July 22, 2024

Protocol Audit Report

Yan Pitrik

July 22, 2024

Prepared by: Yan Pitrik

Lead Security Researcher:

- Yan Pitrik

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

TSwap Pools

The protocol starts as simply a `PoolFactory` contract. This contract is used to create new “pools” of tokens. It helps make sure every pool token uses the correct logic. But all the magic is in each `TSwapPool` contract.

You can think of each `TSwapPool` contract as it’s own exchange between exactly 2 assets. Any ERC20 and the WETH token. These pools allow users to permissionlessly swap between an ERC20 that has a pool and WETH. Once enough pools are created, users can easily “hop” between supported ERC20s.

For example:

1. User A has 10 USDC
2. They want to use it to buy DAI
3. They `swap` their 10 USDC -> WETH in the USDC/WETH pool
4. Then they `swap` their WETH -> DAI in the DAI/WETH pool

Every pool is a pair of `TOKEN X` & `WETH`.

There are 2 functions users can call to swap tokens in the pool.

- `swapExactInput`
- `swapExactOutput`

We will talk about what those do in a little.

Liquidity Providers

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

Why would I want to add tokens to the pool?

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

When you deposit tokens into the protocol, you are rewarded with an LP token. You’ll notice `TSwapPool` inherits the `ERC20` contract. This is because the `TSwapPool` gives out an ERC20 when

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

LP Example

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

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

Core Invariant

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

```
x * y = k
```

- x = Token Balance X
- y = Token Balance Y
- k = The constant ratio between X & Y

Our protocol should always follow this invariant in order to keep swapping correctly!

Make a swap

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

- `swapExactInput`
- `swapExactOutput`

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

Disclaimer

The author 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

Audit Details

Commit Hash:

```
e643a8d4c2c802490976b538dd009b351b1c8dda
```

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

Issues found

Severity	Number of issues
HIGH	4
MEDIUM	2
LOW	1
INFO/GAS	8
——-	——————-
TOTAL	15

Findings

HIGHs

[H-01] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` function causes input value is too high

Description: The `getInputAmountBasedOnOutput` is intended to calculate the amount of tokens a user should deposit given the amount of output tokens. Hovewer, the function returns incorrect value, because of miscalculationg fees.

Impact: Users are charged too many input tokens, protocol takes more fees than expected.

Proof of concept:

add this function to `TSwapPool.t.sol` test file

Show code

```
function test_swapExactOutputMiscalculatesInputAmount() public {
        // add liquidity

        address user2 = makeAddr("user2");
        poolToken.mint(user2, 12e18);

        console.log("User weth balance before: ", weth.balanceOf(user2));
        console.log("User poolToken balance before: ", poolToken.balanceOf(user2));

        console.log("Protocol weth balance before: ", weth.balanceOf(address(pool)));
        console.log("Protocol poolToken balance before: ", poolToken.balanceOf(address(pool)));

        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        // set initial liquidity to 1:1
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        vm.startPrank(user2);
        poolToken.approve(address(pool), 12e18);
        // user should have paid approx 1e18 of pool tokens
        pool.swapExactOutput(poolToken, weth, 1e18, uint64(block.timestamp));
        vm.stopPrank();

        console.log("User weth balance after: ", weth.balanceOf(user2));
        console.log("User poolToken balance after: ", poolToken.balanceOf(user2));

        console.log("Protocol weth balance after: ", weth.balanceOf(address(pool)));
        console.log("Protocol poolToken balance after: ", poolToken.balanceOf(address(pool)));

        assertLt(poolToken.balanceOf(user2), 5e18); // user was charged too much of poolToken

        // LP could withdraw all tokens
        vm.startPrank(liquidityProvider); // just "prank" is not enough, there will bee two tx (balanceof and withdraw)
        pool.withdraw(pool.balanceOf(liquidityProvider), 1, 1, uint64(block.timestamp));

        assertEq(weth.balanceOf(address(pool)), 0);
```

```
39          assertEq(poolToken.balanceOf(address(pool)), 0);
40      }
```

and then run following command: `forge test --mt test_swapExactOutputMiscalculatesInputAmo -vvv`

Recommended Mitigation:

```
1 -     return ((inputReserves * outputAmount) * 10000) / ((outputReserves
      - outputAmount) * 997);
2 +     return ((inputReserves * outputAmount) * 1000) / ((outputReserves -
       outputAmount) * 997);
```

[H-02] Lack of maximum input amount check inside `TSwapPool::swapExactOutput` function could cause user to be charged too much.

Description: No slippage protection in `swapExactOutput` function. For example, in `swapExactInput` function is specified `minOutputAmount`, so in `swapExactOutput` there should be `maxInputAmount` specified.

Impact: If the market conditions change before the TX processes, the user could get much worse swap and be charged for too high input amount of tokens.

Proof of Concept:

1. The price of 1WETH is for example 4000 USDC
2. User calls `swapExactOutput` function:
    1. `inputToken` USDC
    2. `outputToken` WETH
    3. outputAmount = 1e18
    4. deadLine = actual `block.timestamp`
3. Function does not have `maxInputAmount` parameter
4. TX is pending in a mempool and price of 1WETH changes and it is now 6000USDC.
5. TX completes and user sent protocol 2000 of USDC more than expected.

Recommended Mitigation: It is partly mitigated by users approval of how much USDC could protocol spend on behalf of themselfs. But if there is no limit, you have to add the `maxInputAmount` function parameter and then check for it when calculates required input amount.

```
1 +     error TSwapPool__maxInputExceeded(uint256,uint256);
2
3 function swapExactOutput(
4         IERC20 inputToken,
```

```
5           IERC20 outputToken,
6           uint256 outputAmount,
7   +       uint256 maxInputAmount
8           .
9           .
10          .
11          inputAmount = getInputAmountBasedOnOutput(outputAmount,
               inputReserves, outputReserves);
12  +       if (inputAmount > maxInputAmount) revert
        TSwapPool__maxInputExceeded(inputAmount,maxInputAmount);
13
14          _swap(inputToken, inputAmount, outputToken, outputAmount);
15      }
```

[H-03] The function `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to receive the incorrect amount of tokens

Description: The `sellPoolTokens` function should be used to easily sell pool tokens and receive WETH tokens in exchange. The `poolTokenAmount` parameter indicates how many pool tokens user is willing to sell. However, function miscalculates the swapped amount, because there is `swapExactOutput` function called inside this function, whereas the `swapExactInput` should be called instead, because user specified amount of input tokens and not output.

Impact: User will swap the wrong amount of tokens and could loose money.

Proof of Concept: The test function below proves that user gets exact amount of WETH token as the `poolTokenAmount` input, instead he gets charged by that amount of pool tokens. In the same time, due to the incorrect fee calculation inside the `getInputAmountBasedOnOutput` function, users is charged too much pool tokens.

add this function to `TSwapPool.t.sol` test file

Show code

```
1  function test_sellPollTokensIsWrong() external {
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 pooltokenAmount = 5e18;
9          uint256 inputReserves = poolToken.balanceOf(address(pool));
10         uint256 outputReserves = weth.balanceOf(address(pool));
11         uint256 expectedAmount = pool.getOutputAmountBasedOnInput(
              pooltokenAmount, inputReserves, outputReserves);
12         uint256 userWethBalanceBefore = weth.balanceOf(user);
```

```
13          vm.startPrank(user);
14          poolToken.approve(address(pool), type(uint256).max); // insted
                there will be insufficient allowance
15          poolToken.mint(user, 50e18); // instead there is insufficient
                balance (because of baf fee calculation)
16          uint256 userPooltokenBalanceBefore = poolToken.balanceOf(user);
17          console.log("Users pooltokenbalance before: ",
                userPooltokenBalanceBefore);
18          console.log("Users wethbalance before: ", userWethBalanceBefore
                );
19
20          uint256 actualAmount = pool.sellPoolTokens(pooltokenAmount);
21          vm.stopPrank();
22
23          console.log("Expected amount: ", expectedAmount);
24          console.log("Actual amount: ", actualAmount);
25
26          uint256 userWethBalanceAfter = weth.balanceOf(user);
27          uint256 userPooltokenBalanceAfter = poolToken.balanceOf(user);
28          console.log("Users pooltokenbalance after: ",
                userPooltokenBalanceAfter);
29          console.log("Users wethbalance after: ", userWethBalanceAfter);
30
31          assertEq(userWethBalanceBefore + pooltokenAmount,
                userWethBalanceAfter); // users gets 5 WETH instead of sell
                5
32              // WETH
33          assertNotEq(actualAmount, expectedAmount);
34          assertLt(userPooltokenBalanceAfter, 10 ether); // user is
                charged too much pool tokens
35      }
```

and then run following command: `forge test --mt test_sellPollTokensIsWrong -vvv`

Recommended Mitigation: Chenge the implementation to use `swapExactInput` function instead of `swapExactOutput`. You have to add new input parameter to `sellPoloTokens` function to be able to set the minimum output amount.

Additionaly, consider to add the deadline parameter to the function, so users could specify it, instead of actually hard coded block.timestamp value.

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

```
        , minWethOutputAmount, uint64(block.timestamp));
8       }
```

[H-04] Extra tokens transfer inside `TSwapPool::_swap` function breaks protocol invariant `x * y = k`

Description: The `_swap` function is very important part of protocol and its responsible for all token transfers during the swapping methods. There is an extra token transfer inside the function, which is executed every 10 swaps and everytime this condition is met, additional 1e18 amount of output token is transfered together with original swap. The protocol follows an invariant of `x * y = k`. Where:

- `x` is the balance of the pool token
- `y` is the balance of the weth token
- `k` is the constant product of this two balances

This means, that whenever the balances change in the protocol, the ration between them should remain constant. However, this is brokendue to the extra incentive in the `_swap` function. meaning that over time the protocol funds will be drained.

Following block of code is responsible for the issue described above:

```
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: This transfer has a high impact on the protocol invariant and breaks the constant product flow. An malicious user could drain the protocol doing a lot of swaps and collectin the extra incentive given out by the protocol.

Proof of Concept:

1. User swaps 10 times and collects the extra incentive of 1e18 output tokens.
2. User can continue to swap untill all protocol funds are drained.

add this function to `TSwapPool.t.sol` test file

Show code

```
1   function test_invariantBreaking() external {
2           // add liquidity
3           vm.startPrank(liquidityProvider);
4           weth.approve(address(pool), 100e18);
5           poolToken.approve(address(pool), 100e18);
```

```
6           pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
7           vm.stopPrank();
8
9           uint256 wethAmount = 1e15; // user could deposit relatively
                small amount every time
10
11          vm.startPrank(user);
12          //poolToken.mint(user, 10e18);
13          poolToken.approve(address(pool), 20e18);
14
15          // doing first 9 swaps
16          pool.swapExactOutput(poolToken, weth, wethAmount, uint64(block.
                timestamp));
17          pool.swapExactOutput(poolToken, weth, wethAmount, uint64(block.
                timestamp));
18          pool.swapExactOutput(poolToken, weth, wethAmount, uint64(block.
                timestamp));
19          pool.swapExactOutput(poolToken, weth, wethAmount, uint64(block.
                timestamp));
20          pool.swapExactOutput(poolToken, weth, wethAmount, uint64(block.
                timestamp));
21          pool.swapExactOutput(poolToken, weth, wethAmount, uint64(block.
                timestamp));
22          pool.swapExactOutput(poolToken, weth, wethAmount, uint64(block.
                timestamp));
23          pool.swapExactOutput(poolToken, weth, wethAmount, uint64(block.
                timestamp));
24          pool.swapExactOutput(poolToken, weth, wethAmount, uint64(block.
                timestamp));
25
26          int256 startingY = int256(weth.balanceOf(address(pool)));
27
28          // this swap should break invariant with extra 1WETH transfer
29          pool.swapExactOutput(poolToken, weth, wethAmount, uint64(block.
                timestamp));
30
31          vm.stopPrank();
32
33          int256 expectDeltaY = int256(wethAmount) * int256(-1);
34
35          uint256 endingY = weth.balanceOf(address(pool));
36          int256 deltaY = int256(endingY) - int256(startingY);
37
38          assertEq(deltaY, expectDeltaY);
39      }
```

and then run following command: `forge test --mt test_invariantBreaking -vvv`

Recommended Mitigation:

Remove the part where user is rewarded with extra tokens. If you want to keep this in, you should

account for the change in the `x * y = k` invariant.

```
    function _swap(IERC20 inputToken, uint256 inputAmount, IERC20
        outputToken, uint256 outputAmount) private {
        if (_isUnknown(inputToken) || _isUnknown(outputToken) ||
            inputToken == outputToken) {
            revert TSwapPool__InvalidToken();
        }

-       swap_count++;
-       if (swap_count >= SWAP_COUNT_MAX) {
-           swap_count = 0;
-           outputToken.safeTransfer(msg.sender, 1
    _000_000_000_000_000_000); // @audit   literals in code
-       }
        emit Swap(msg.sender, inputToken, inputAmount, outputToken,
            outputAmount);

        inputToken.safeTransferFrom(msg.sender, address(this),
            inputAmount);
        outputToken.safeTransfer(msg.sender, outputAmount);
    }
```

MEDIUMs

[M-01] `TSwapPool::deposit` function missing check for `deadline` parameter causing transactions to complete even after dedadline passed.

Description: The `deposit` function has `deadline` parameter, which should be the timestamp limit, when transaction is no longer valid if its exceeded. Hovewer, there is no check for that inside a function, so even the deadline passes, the transaction should be executed.

Impact: If there will be another transaction ordered in the block before our transaction, and that transaction will have huge impact on the ratio of poolToken/weth pair, it could affected final amount of liquidity tokens minted. The minted amount of tokens should ends to be rapidly lower that it will be within our deadline limit. In the other words, the transactions could be sent when market conditions are unfavorable to deposit, even when adding a deadline parameter.

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

Recommended Mitigation: Apply the `revertIfDeadlinePassed` modifier on the `deposit` function.

```
     function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
```

```
4           uint256 maximumPoolTokensToDeposit,
5           uint64 deadline
6       )
7           external
8           revertIfZero(wethToDeposit)
9   +       revertIfDeadlinePassed(deadline)
10          returns (uint256 liquidityTokensToMint)
11      {
12          .
13          .
```

[M-02] ERC777, rebase, fee-on-transfer tokens breaks protocol invariant

Description: The protocol follows an invariant of `x * y = k`. Where:

- `x` is the balance of the pool token
- `y` is the balance of the weth token
- `k` is the constant product of this two balances

This means, that whenever the balances change in the protocol, the ration between them should remain constant. However, this will be broken if some of "weird" ERC20s will be used within the protocol.

Impact: Invariant will be broken.

Proof of Concept:

Recommended Mitigation: Do not use "weird" ERC20 tokens.

LOWs

[L-01] In the `TSwapPool::_addLiquidityMintAndTransfer` function there is incorrect emission of event `TSWapPool::LiquidityAdded`

Description: Emission of the `LiquidityAdded` event is incorrect, due to the bad order of fiels in the event. This causing event to emit the incorrect information about protocol.

Impact: When some off-chain logic depends on the event emission from protocol, it could get incorrect information.

Recommended Mitigation:

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

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

INFORMATIONALs

[I-01] `PoolFactory::PoolFactory__PoolDoesNotExist` error is not used

Description: There is declaration of `PoolFactory__PoolDoesNotExist` error inside contract, but its never used anywhere and should be removed.

Recommended Mitigation:

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

[I-02] Missing events 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: 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-03] Missing zero address checks could cause undesirable behavior

```
+   error PoolFactory__ZeroAddressInput();
    constructor(address wethToken) {
+       if(wethToken == address(0)) {
+           revert PoolFactory__ZeroAddressInput();
+       }
        i_wethToken = wethToken;
    }
```

```
+   error TSwapPool__ZeroAddressInput();
    constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    )
        ERC20(liquidityTokenName, liquidityTokenSymbol)
    {
+       if(wethToken == address(0) || poolToken == address(0)) {
+           revert TSwapPool__ZeroAddressInput();
+       }
        i_wethToken = IERC20(wethToken);
        i_poolToken = IERC20(poolToken);
    }
```

[I-04] `PoolFactory::liquidityTokenSymbol` should use `.symbol()` instead of `.name()`

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

[I-05] The `poolTokenReserves` variable in `TSwapPool::deposit` function is not used and should be removed

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

[I-06] The `TSwapPool::deposit` function should follow CEI pattern

Even `liquidityTokensToMint` is not the state variable, the change of it should be placed before the external call

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

[I-07] 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: 231

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

- Found in src/TSwapPool.sol Line: 248

```
        return ((inputReserves * outputAmount) * 10000) / ((
            outputReserves - outputAmount) * 997); // @audit  Big
```

- Found in src/TSwapPool.sol Line: 379

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

- Found in src/TSwapPool.sol Line: 385

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

[I-08] Function `TSwapPool::swapExactInput` returns default value, resultion in returning zero value

Description: There is named return value declared in `swapExactInput` function declaration, but its never assigned and due to that function returns 0.

Impact: Function gives incorrect information to the user.

Proof of Concept:

Recommended Mitigation:

```
    function swapExactInput(
        IERC20 inputToken,
```

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