



Protocol Audit Report

Version 1.0

Pratik

September 6, 2025

Protocol Audit Report

Pratik

September 6, 2025

Prepared by: [Pratik] Lead Auditors: - Pratik Tikare

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
 - Scope
 - Roles
- Executive Summary
 - Issues found
- Findings
 - High
 - * [H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users, resulting in lost fees.
 - * [H-2] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially receive way fewer tokens.
 - * [H-3] `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to receive the incorrect amount of tokens.
 - * [H-4] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` breaks the protocol invariant of $x * y = k$.

- Medium
 - * [M-1] `TSwapPool::deposit` is missing deadline check causing transactions to complete even after the deadline
- Low
 - * [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order.
 - * [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given.
- Informational
 - * [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed
 - * [I-2] Lacking zero address check
 - * [I-3] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`

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

I have made all effort to find as many vulnerabilities in the code in the given time period, but hold no responsibilities for the findings provided in this document. A security audit by me 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	

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

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

Audit Details

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda

Scope

- In Scope:

```
1 ./src/  
2 #-- PoolFactory.sol  
3 #-- 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 is a part of the Cyfrin Updraft's security and auditing course.

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Info	3
Total	10

Findings

High

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

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

Impact: Protocol takes more fees than expected from users.

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 -        return ((inputReserves * outputAmount) * 10000) / ((  
13 +        return ((inputReserves * outputAmount) * 1000) / ((  
14        outputReserves - outputAmount) * 997);  
        outputReserves - outputAmount) * 997);  
    }
```

[H-2] Lack of slippage protection in TSwapPool::swapExactOutput causes users to potentially receive way fewer tokens.

Description: The `swapExactOutput` function does not include any sort of slippage protection. This function is similar to what is done in `TSwapPool::swapExactInput`, where the function specifies a `minOutputAmount`, the `swapExactOutput` function should specify a `maxInputAmount`.

Impact: If market condition changes before the transaction processes, the user could get a much worse swap.

Proof of Concept: 1. The price of 1 WETH is 1,000 USDC 2. User inputs a `swapExactOutput` looking for 1 WETH 1. inputToken = USDC 2. outputToken = WETH 3. outputAmount = 1 4. deadline = block.timestamp 3. The function does not offer a max input 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.

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,  
8              inputReserves, outputReserves);  
8      +      if(inputAmount > maxInputAmount){  
9      +          revert();  
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 token 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 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,  
6         poolTokenAmount, uint64(block.timestamp));  
7     +     return swapExactInput(i_poolToken, poolTokenAmount, i_wethToken,  
8         minWethToReceive, uint64(block.timestamp));  
9     }
```

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 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 will be drained.

The following block of code is responsible for the issue.

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

More 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. The user continues to swap until all the protocol funds are drained.

Proof Of Code

Place the following into `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));
22        vm.stopPrank();
23
24        int256 startingY = int256(weth.balanceOf(address(pool)));
25        int256 expectedDeltaY = int256(-1) * int256(outputWeth);
26
27        uint256 endingY = weth.balanceOf(address(pool));
28
29        int256 actualDeltaY = int256(endingY) - int256(startingY);
30
31        assertEq(actualDeltaY, expectedDeltaY);
32    }
```

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



```
4 - }
```

Medium

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

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 unfavorable to deposit, even when adding a deadline parameter.

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

Recommended Mitigation: Consider making the following change to the function.

```
1     function deposit(  
2         uint256 wethToDeposit,  
3         uint256 minimumLiquidityTokensToMint,  
4         uint256 maximumPoolTokensToDeposit,  
5         uint64 deadline  
6     )  
7     external  
8 +     revertIfDeadlinePassed(deadline)  
9     revertIfZero(wethToDeposit)  
10    returns (uint256 liquidityTokensToMint)  
11    {
```

Low

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

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

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

Recommended Mitigation:

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

[L-2] Default value returned by TSwapPool : : swapExactInput results in incorrect return value given.

Description: The `swapExactInput` function is expected to return the actual amount of tokens bought by the caller. However, while it declares the named return value `output` it is never assigned a value, nor uses an explicit return statement.

Impact: The return value will always be 0. giving incorrect information to the caller.

Recommended Mitigation:

```
1     {
2         uint256 inputReserves = inputToken.balanceOf(address(this));
3         uint256 outputReserves = outputToken.balanceOf(address(this));
4
5 -         uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount,
6 +         output = getOutputAmountBasedOnInput(inputAmount, inputReserves
    , outputReserves);
7
8
9 -         if (outputAmount < minOutputAmount) {
10 -             revert TSwapPool__OutputTooLow(outputAmount,
    minOutputAmount);
11 -         }
12
13 +         if (output < minOutputAmount) {
14 +             revert TSwapPool__OutputTooLow(outputAmount,
    minOutputAmount);
15 +         }
16
17 -         _swap(inputToken, inputAmount, outputToken, outputAmount);
18 +         _swap(inputToken, inputAmount, outputToken, output);
19     }
```

Informational

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

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

[I-2] Lacking zero address check

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

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

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