



# **TSwap Audit Report**

Version 1.0

January 10, 2025

# Protocol Audit Report

pindarev

January 10, 2025

Prepared and Audit by: pindarev

## 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 lost fees
    - \* [H-2] No 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 transaction to complete even after the deadline is set
- Low
  - \* [L-1] `TSwapPool::_addLiquidityMintAndTransfer` emits event `TSwapPool::LiquidityAdded` out of order
  - \* [L-2] Function `TSwap::swapExactInput` is returning wrong value
- Informationals
  - \* [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` event is not used and should be removed
  - \* [I-2] Lacking zero address checks
  - \* [I-3] Wrong string concat for `PoolFactory::createPool` calling ERC20 `name` instead of `symbol`

## 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 pindarev 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

### Scope

### Roles

## Executive Summary

### Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Info	2
Total	9

## Findings

### High

#### [H-1] Incorrect Fee calculation in TswapPool::getInputAmountBasedOnOutput causes protocol to take too many tokens from users, resulting lost fees

**Description** The `getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens a user should deposit given an amount 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 is being too expensive and 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  
13 -         ((inputReserves * outputAmount) * 10000) /  
14 +         ((inputReserves * outputAmount) * 1000) /  
15         ((outputReserves - outputAmount) * 997);  
16 }
```

#### [H-2] No 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::swapExactOutput`, where the function specifies a `minOutputAmount`, the `swapExactOutput` function should specify `maxInputAmount`.

**Impact** If market conditions change before the transaction processes, the user could get a much worse swap.

**Proof of Concepts** 1. The price of 1 WETH right now is 1,000 USDC 2. User inputs a `swapExactOutput` looking for 1 WETH 1. inputToken = USDC 2. outputToken = WETH 3. outputAmount = 1 4. 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(  
8              outputAmount,  
9              inputReserves,  
10             outputReserves  
11         );  
12 +     if(inputAmount > maxInputAmount) {  
13 +         revert();  
14 +     }  
15  
16     _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 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 (`minWethToReceive` to be passed to `swapExactInput`)

```
1      function sellPoolTokens(  

```

```
2         uint256 poolTokenAmount,
3 +         uint256 minWethToReceive
4     ) external returns (uint256 wethAmount) {
5         return
6 -         swapExactOutput(
7 -             i_poolToken,
8 -             i_wethToken,
9 -             poolTokenAmount,
10 -            uint64(block.timestamp)
11 -        );
12 +         swapExactInput(
13 +             i_poolToken,
14 +             poolTokenAmount,
15 +             i_wethToken,
16 +             minWethToReceive,
17 +             uint64(block.timestamp)
18 +        );
19     }
```

**[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 follow 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
5                                     _000_000_000_000_000_000);
6         }
```

**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

Proof Of Code

Place the following into `TSwapPool.t.sol`.

```
1
2     function testInvariantBroken() public {
3         vm.startPrank(liquidityProvider);
4         weth.approve(address(pool), 100e18);
5         poolToken.approve(address(pool), 100e18);
6         pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
7         vm.stopPrank();
8
9         uint256 outputWeth = 1e17;
10
11        vm.startPrank(user);
12        poolToken.approve(address(pool), type(uint256).max);
13        poolToken.mint(user, 100e18);
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        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
23
24        int256 startingY = int256(weth.balanceOf(address(pool)));
25        int256 expectedDeltaY = int256(-1) * int256(outputWeth);
26
27        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
28        vm.stopPrank();
29
30        uint256 endingY = weth.balanceOf(address(pool));
31        int256 actualDeltaY = int256(endingY) - int256(startingY);
32        assertEq(actualDeltaY, expectedDeltaY);
33    }
```

**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
6 -             _000_000_000_000_000_000);
7 -         }
```

## Medium

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

**Description** The `deposit` function accepts a deadline parameter, which according to the documentation is “@param deadline 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 deposit rate is unfavorable.

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

**Proof of Concepts** The `deadline` parameter is unused.

**Recommended mitigation** Consider making following change to the function.

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

## Low

### [L-1] TSwapPool::\_addLiquidityMintAndTransfer emits event TSwapPool::LiquidityAdded out of order

**Description** When the `LiquidityAdded` event is emitted, it logs value in an incorrect order. The `poolTokensToDeposit` value should go in the third parameter position, whereas the

`wethToDeposit` value should go second.

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

**Recommended mitigation** Swap the order of variables.

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

## [L-2] Function `TSwap::swapExactInput` is returning wrong value

**Description** The `swapExactInput` 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     function swapExactInput(
2         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 (uint256 output)
12    {
13        uint256 inputReserves = inputToken.balanceOf(address(this));
14        uint256 outputReserves = outputToken.balanceOf(address(this));
15
16 +        output = getOutputAmountBasedOnInput(
17 -        uint256 outputAmount = getOutputAmountBasedOnInput(
18            inputAmount,
19            inputReserves,
20            outputReserves
21        );
22
23 -        if (outputAmount < minOutputAmount) {
24 -            revert TSwapPool__OutputTooLow(outputAmount,
minOutputAmount);
25 -        }
26 +        if (output < minOutputAmount) {
27 +            revert TSwapPool__OutputTooLow(output, minOutputAmount);
```

```
28 +     }
29
30
31 -     _swap(inputToken, inputAmount, outputToken, outputAmount);
32 +     _swap(inputToken, inputAmount, outputToken, output);
33 }
```

## Informationals

**[I-1] PoolFactory::PoolFactory\_\_PoolDoesNotExist event is not used and should be removed**

### Recommended mitigation

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

**[I-2] Lacking zero address checks**

### Recommended mitigation

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

**[I-3] Wrong string concat for PoolFactory::createPool calling ERC20 name instead of symbol**

**Description** `PoolFactory::liquidityTokenSymbol` is `memory` variable creating token symbol for liquidity pool. Function should concat “ts” to the actual ERC20 token symbol, but instead of calling the function `symbol` on ERC20 it’s calling `name`

### Recommended mitigation

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