T-Swap Protocol Audit Report

Version 1.0

Hugo H.

November 24, 2025

T-Swap Protocol Audit Report

Hugo H.

November 21, 2025

Lead Auditors: - Hugo H.

Table of Contents

- Medium
    * [M-1] `TSwapPool::deposit` is missing deadline check causing transactions to complete even after the deadline.
    * [M-2] Rebase, fee-on-transfer, and ERC777 tokens break protocol invariant
- Low
    * [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order causing event to emit incorrect information
    * [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
- Informational
    * [I-1] `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.

Disclaimer

The Hugo H. 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 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		
	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

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 found
High	4
Medium	2
Low	2
Info	3

Severity	Number of issues found
Total	11

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.

Proof of code Put below coding in TSwapPool.t.sol. You sill see that transaction revert as the amount to be transferred is much more bigger than expected (See amountTokenIn in debugging).

```
1      function testGetInputAmountBasedOnOutput() 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          vm.startPrank(user);
9          poolToken.approve(address(pool), 10e18);
10         // After we swap, there will be ~110 tokenA, and ~91 WETH
11         // 100 * 100 = 10,000
12         // 110 * ~91 = 10,000
13      // uint256 expected = 9e18;
14
15         vm.expectRevert();
16         pool.swapExactOutput(poolToken, weth, 9e18, uint64(block.
           timestamp));
17     }
```

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

[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 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 = N/A 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 to 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
8  inputAmount = getInputAmountBasedOnOutput(
9          outputAmount,
10         inputReserves,
11         outputReserves
```

```
12              );
13
14  +           if (inputAmount > maxInputAmount) {
15  +               revert();
16  +           }
17
18              _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.

Proof of Concepts

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
 6  -             swapExactOutput(
 7  -                 i_poolToken,
 8  -                 i_wethToken,
 9  -                 poolTokenAmount,
10  -                 uint64(block.timestamp)
11  -             );
12  +         return
13  +             swapExactInput(
14  +                 i_poolToken,
15  +                 poolTokenAmount,
16  +                 i_wethToken,
```

```
17  +                    minWethToReceive,
18  +                    uint64(block.timestamp)
19  +                );
20        }
```

Additionally, it might be wise to add a deadline to the function, as there is currently no deadline. (MEV later)

[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 Concepts

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       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
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));
27            vm.stopPrank();
28
29            // actual
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 -        if (swap_count >= SWAP_COUNT_MAX) {
3 -            swap_count = 0;
4 -            outputToken.safeTransfer(msg.sender, 1
      _000_000_000_000_000_000);
5 -        }
```

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 Concepts 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      {
```

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

//Finding...

Low

[L-1] `TSwapPool::LiquidityAdded` event has parameters out of order causing event to emit incorrect information

Description When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTrans` function, it logs values 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

```
1  -emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
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(
 6  -             inputAmount,
 7  -             inputReserves,
 8  -             outputReserves
 9  -         );
10  +         output = getOutputAmountBasedOnInput(
11  +             inputAmount,
12  +             inputReserves,
13  +             outputReserves
14  +         );
15
16  -         if (outputAmount < minOutputAmount) {
17  -             revert TSwapPool__OutputTooLow(outputAmount,
     minOutputAmount);
18  -         }
19  +         if (output < minOutputAmount) {
20  +             revert TSwapPool__OutputTooLow(outputAmount,
     minOutputAmount);
21  +         }
22
23  -         _swap(inputToken, inputAmount, outputToken, outputAmount);
24            _swap(inputToken, inputAmount, outputToken, output);
25        }
```

Informational

[I-1] `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      }
```

Also below:

```
1  +      if (wethToken == address(0) || poolToken == address(0)) {
2  +          revert();
3  +      }
4          i_wethToken = IERC20(wethToken);
5          i_poolToken = IERC20(poolToken);
```

[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());
```