TSwap Audit Report

Prepared by: thomasbtho

Date: December 17, 2025

Table of Contents

Protocol Summary

TSwap is a DEX protocol, similar to Uniswap. It is an automated market maker. It allows users to swap tokens for WETH and WETH for tokens.

Disclaimer

thomasbtho 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

The CodeHawks severity matrix is used to determine severity. See the documentation for more details.

Audit Details

The findings described in this document correspond to the following 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 found
High	4
Medium	1
Low	2
Gas	1
Info	9
Total	17

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 calcultate 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 takes more fees than expected from users.

Proof of Concept:

Code

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

```
    function testSwapExactOutputCostsTooMuchFees() public {
        uint256 initialLiquidity = 100e18;

        // 1. Liquidity provider deposits
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), initialLiquidity);
        poolToken.approve(address(pool), initialLiquidity);
        pool.deposit(initialLiquidity, 0, initialLiquidity,
uint64(block.timestamp));
        vm.stopPrank();

        uint256 wethExactOutput = 5e17; // 0.5 WETH
        uint256 poolTokenReserves = poolToken.balanceOf(address(pool));
        uint256 wethReserves = weth.balanceOf(address(pool));
        uint256 correctPoolTokenAmount =
            ((poolTokenReserves * wethExactOutput) * 1000) / ((wethReserves
- wethExactOutput) * 997);

        // 2. User wants to swap `poolToken` to get exactly 0.5 WETH
        vm.startPrank(user);
        poolToken.approve(address(pool), type(uint256).max);
        uint256 actualPoolTokenAmount = pool.swapExactOutput(poolToken,
weth, wethExactOutput, uint64(block.timestamp));
        vm.stopPrank();

        // 3. User has paid 10x more fees
        assertEq(actualPoolTokenAmount, correctPoolTokenAmount * 10);
    }
```

Recommended Mitigation:

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

```
+   return ((inputReserves * outputAmount) * 1000) / ((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 conditions change before the transaction processes, the user could get a much worse swap.

Proof of Concept:

1. The price of WETH is `1000` USDC
2. User inputs a `swapExactOutput` looking for `1` WETH
   3. inputToken = USDC
   4. outputToken = WETH
   5. outputAmount = 1
   6. deadline = whatever
3. The function does not offer a `maxInputAmount`
4. As the transaction is pending in the mempool, the market changes! And the price moves HUGE -> `1` WETH is now `10000` USDC. 10x more than the user expected
5. The transaction completes, but the user sent the protocol `10000` USDC instead of the expected `1000` USDC

Recommended Mitigation:

Add a `uint256 maxInputAmount` parameter so the user only has to spend up to a specific amount, and can predict how much they will spend on the protocol.

```
    function swapExactOutput(
        IERC20 inputToken,
+       uint256 maxInputAmount,

        // ...

        inputAmount = getInputAmountBasedOnOutput(outputAmount,
inputReserves, outputReserves);
+       if(inputAmount > maxInputAmount) {
+           revert();
+       }

        _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` functions 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 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.

```
    function sellPoolTokens(uint256 poolTokenAmount) external returns
(uint256 wethAmount) {
@>      return swapExactOutput(i_poolToken, i_wethToken, poolTokenAmount,
uint64(block.timestamp));
    }
```

Impact:

Users will swap the wrong amount of tokens, which is a severe disruption of protocol functionality.

Proof of Concept:

Code:

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

```
    function testSellPoolTokensSwapsTheWrongAmountOfTokens() public {
        uint256 initialLiquidity = 100e18;

        // 1. Liquidity provider deposits
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), initialLiquidity);
        poolToken.approve(address(pool), initialLiquidity);
        pool.deposit(initialLiquidity, 0, initialLiquidity,
uint64(block.timestamp));
        vm.stopPrank();

        // 2. User wants to sell 0.5 pool tokens
        uint256 poolTokenAmountToSell = 5e17;
        uint256 poolTokenReserves = poolToken.balanceOf(address(pool));
        uint256 wethReserves = weth.balanceOf(address(pool));
        uint256 expectedWethAmountToReceive =
            pool.getOutputAmountBasedOnInput(poolTokenAmountToSell,
poolTokenReserves, wethReserves);

        uint256 startingWethUserBalance = weth.balanceOf(user);

        vm.startPrank(user);
        poolToken.approve(address(pool), type(uint256).max);
        pool.sellPoolTokens(poolTokenAmountToSell);
        vm.stopPrank();

        // 3. User receives the wrong amount
        uint256 actualWethUserBalance = weth.balanceOf(user);
        uint256 expectedWethUserBalance = startingWethUserBalance +
expectedWethAmountToReceive;

        assert(actualWethUserBalance != expectedWethUserBalance);
    }
```

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`.

```
    function sellPoolTokens(
        uint256 poolTokenAmount,
+       minWethToReceive
        ) external returns (uint256 wethAmount) {
-       return swapExactOutput(i_poolToken, i_wethToken, poolTokenAmount,
uint64(block.timestamp));
+       return swapExactInput(i_poolToken, poolTokenAmount, i_wethToken,
minWethToReceive, uint64(block.timestamp));
    }
```

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 `SWAP_COUNT_MAX` breaks the procotol 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.

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

Impact:

A user could maliciously drain the protocol of funds by doing a lot of swaps and collecting the extra incentive gien 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 `1e18` tokens
2. That user continues to swap until all the protocol funds are drained

Code:

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

```
    function testInvariantBroken() public {
        uint256 initialLiquidity = 100e18;

        // 1. Liquidity provider deposits
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), initialLiquidity);
        poolToken.approve(address(pool), initialLiquidity);
        pool.deposit(initialLiquidity, 0, initialLiquidity,
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));

        uint256 startingY = weth.balanceOf(address(pool));
        int256 expectedDeltaY = int256(-1) * int256(outputWeth);

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

        uint256 endingY = weth.balanceOf(address(pool));
        int256 actualDeltaY = int256(endingY) - int256(startingY);
        assertNotEq(actualDeltaY, expectedDeltaY);
    }
```

Recommended Mitigation:

Remove the extra incentive mechanism. If you want to keep this in, you should account for that change in the protocol invariant. Or, you should set aside tokens in the same way you do with fees.

```diff
-   uint256 private swap_count = 0;

    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);
-       }
        emit Swap(msg.sender, inputToken, inputAmount, outputToken, outputAmount);

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

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.

```diff
    function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
        revertIfZero(wethToDeposit)
+       revertIfDeadlinePassed(deadline)
        returns (uint256 liquidityTokensToMint)
    {
```

Low

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

Description:

When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTransfer` 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.

```
event LiquidityAdded(address indexed liquidityProvider, uint256
wethDeposited, uint256 poolTokensDeposited);

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

Impact:

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

Recommended Mitigation:

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

Proof of Concept:

Code

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

```
    function testSwapExactInputAlwaysReturnsZero() public {
        uint256 initialLiquidity = 100e18;

        // 1. Liquidity provider deposits
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), initialLiquidity);
        poolToken.approve(address(pool), initialLiquidity);
        pool.deposit(initialLiquidity, 0, initialLiquidity,
uint64(block.timestamp));
        vm.stopPrank();

        // 2. User wants to swap exactly 5 `poolToken` to get WETH
        uint256 inputAmount = 5e18;
        uint256 poolTokenReserves = poolToken.balanceOf(address(pool));
        uint256 wethReserves = weth.balanceOf(address(pool));
        uint256 minOutputAmount =
pool.getOutputAmountBasedOnInput(inputAmount, poolTokenReserves,
wethReserves);

        uint256 startingPoolTokenUserBalance = poolToken.balanceOf(user);
        uint256 startingWethUserBalance = weth.balanceOf(user);
        vm.startPrank(user);
        poolToken.approve(address(pool), type(uint256).max);
        uint256 outputAmount =
            pool.swapExactInput(poolToken, inputAmount, weth,
minOutputAmount, uint64(block.timestamp));
        vm.stopPrank();

        // 3. User has less poolToken and more WETH, but the function
returned 0
        uint256 endingPoolTokenUserBalance = poolToken.balanceOf(user);
        uint256 endingWethUserBalance = weth.balanceOf(user);
        uint256 expectedPoolTokenUserBalance = startingPoolTokenUserBalance
- inputAmount;

        assertEq(endingPoolTokenUserBalance, expectedPoolTokenUserBalance);
        assertGt(endingWethUserBalance, startingWethUserBalance);
        assertEq(outputAmount, 0);
    }
```

Recommended Mitigation:

Use the correct return value

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

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

-        _swap(inputToken, inputAmount, outputToken, outputAmount);
+        _swap(inputToken, inputAmount, outputToken, output);
```

Gas

[G-1] Unused `poolTokenReserves` local variable in `TSwapPool::deposit` should be removed

Informational

[I-1] Unused error `PoolFactory__PoolDoesNotExist` should be removed

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

[I-2] Unchecked zero-adresses assignment

2 found instances

- `PoolFactory::constructor`

```
    constructor(address wethToken) {
@>      i_wethToken = wethToken;
    }
```

- `TSwapPool::constructor`

```
    constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    )
        ERC20(liquidityTokenName, liquidityTokenSymbol)
    {
@>      i_wethToken = IERC20(wethToken);
@>      i_poolToken = IERC20(poolToken);
    }
```

[I-3] In `PoolFactory::createPool` the liquidity token symbol uses the token's name instead of its symbol

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

[I-4] `TSwapPool::MINIMUM_WETH_LIQUIDITY` constant value should not be emitted in events

[I-5] `TSwapPool::deposit` does not follow CEI

[I-6] Magic numbers

Found instances

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

-
  ```
  uint256 denominator = (inputReserves * 1000) + inputAmountMinusFee;
  ```

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

[I-7] Missing natspec on the `TSwapPool::swapExactInput` function

[I-8] Missing `deadline` parameter in the natspec of `TSwapPool::swapExactOutput`

[I-9] The `TSwapPool::totalLiquidityTokenSupply` function visibility should be external