TSwap Protocol Security Report 5/8/2025

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutputAmount` causes protocol to take too many token from user resulting in lost fees

Description: The `TSwapPool::getInputAmountBasedOnOutputAmount` is attended to calculate the amount tokens a user should deposit given an amount of output tokens. The function miscalculates the 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:

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
        return
-           ((inputReserves * outputAmount) * 10000) /
+           ((inputReserves * outputAmount) * 1000) /
            ((outputReserves - outputAmount) * 997);
    }
```

[H-2] Lack of slippage protection in `TSwapPool::swapExactOutput` causes user to potentially recieve fewer tokens

Description: The `swapExactOutput` function does not include slippage protection. The function is similar to `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 1 WETH right now is 1000 USDC 2. User inputs a `swapExactOutput` looking for 1 WETH (a) inputToken = USDC (b) outputToken = WETH (c) outputAmount = 1 (d) deadline = whenever 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 to 1 WETH = 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: A `maxInputAmount` so the user only has to spend a certain amount and can predict how much they can spend on the protocol.

```
    function swapExactOutput(
        IERC20 inputToken,
+       uint256 maxInputAmount,
    )
.
.
.
        inputAmount = getInputAmountBasedOnOutput(
            outputAmount,
            inputReserves,
            outputReserves
        );
+       if(inputAmount > maxInputAmount) {
+          revert
+       }
```

[H-3] `TSwapPool::sellPoolTokens` mistmatches input and output tokens causing users to recieve the incorrect amount of tokens

Description: The `sellPoolTokens` function is intended to allow users to easily sell pool tokens and recieve WETH in exchange. Users indiciate how many pool tokens they're willing to sell in the `poolTokenAmount` parameter. The function currently miscalculates the swapped amount.

The `swapExactOutput` function where the `swapExactInput` should be called. User specify the exact amount of input tokens not output tokens.

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

Recommended Mitigation: Change the implementation to use `swapExactInput` instead of `swapExactOutput`. This would require changing the `sellPoolTokens` function to accept a new parameter (`minWethToRecieve` to be passed to `swapExactInput`)

```
    function sellPoolTokens(
        uint256 poolTokenAmount,
+       uint256 minWethToRecieve,
    ) external returns (uint256 wethAmount) {
        return
-            swapExactOutput(
-                i_poolToken,
```

```
-                 i_wethToken,
-                 poolTokenAmount,
-                 uint64(block.timestamp)
-             );
+             swapExactInput(
+                i_poolToken,
+                poolTokenAmount,
+                i_wethToken,
+                poolTokenAmount,
+                minWethToRecieve,
+                uint64(block.timestamp)
+             );

    }
```

[H-4] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` breaks the invariant of `x * y = k`

Description: The protocol follows a strict invariant of `x * y = k` where: - `x`: is the balance of the pool token - `y`: is the balance of WETH - `k`: The constant product of the two balances

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

Whenever the balances change in the protocol the ration between the two amounts should remain constant represented by `k`. `k` is broken due to the incentives offered every 10 swaps. Over time the protocol funds will be drained.

Impact: A user could drain the protocol by doing many swaps and collecting the extra incentive given out by the protocol. The protocols core invariant is broken.

Proof of Concept: 1. A user swaps 10 times and collect the incentive tokens 2. User continues to swap until the protocol funds are drained

Proof of Code: Copy the following snippet into `TSwapPool.t.sol`

```
function testInvariantBroken() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, 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));

        int256 startingY = int256(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);
        assertEq(actualDeltaY, expectedDeltaY);
    }
```

Recommended Mitigation: Remove the extra incentive. If you want to keep this in account for the change in the `x * y = k` invariant. Or set aside tokens the same way the protocol does with fees.

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". The deadline parameter is never used. Operations that add liquidity to the pool might be executed at unexpected times in market conditions where the deposit rate is unfavourable.

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.

```
    function deposit(
```

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

Low

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

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTransfe` function it logs values in an incorrect order. The `poolTokensToDeposit` value should go in the third parameter position. The `wethToDeposit` value should go second.

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

Recommended Mitigation:

```
- 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. While it declares the named return value `output` it is never assigned a value nor uses an explict return statement.

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

Recommended Mitigation:

```
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

-       uint256 outputAmount = getOutputAmountBasedOnInput(
+       output = getOutputAmountBasedOnInput(
            inputAmount,
            inputReserves,
```

```
            outputReserves
        );

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



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

Informational

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

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

[I-2] Lacking zero address checks

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

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

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