High

[H-1] TITLE `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 @///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 the deposit rate in 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 deadline function

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

[H-2] Incorrect fee calculation in `TSwapPool:: getInputAmountBasedOnOutput` causes protocol to take too many tokens

from users, resulting in lost fees.

Description: The `getInputAmountBasedOnOutput` function is inteded 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 1000.

Impact: Protocol takes more fees than expected from users.

Proof of Concept:

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

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.

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

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

[H-4] 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 miscalculaes 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 protcol functionality.

Proof of Concept:

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,
+       uint256 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));
    }
```

[H-5] 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.

```
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 given out by the protocol.

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 untill all the protocol funds are drained

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.

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

Low

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

Description: When the `LiquidityAdded` event is emitted in the `TswapPool::_addLiquidityMintAndTransfer` , 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.

Proof of Concept:

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. However, while it declares the named return value `ouput` it is never assigned a value, nor uses an explict return statement.

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

Recommended Mitigation:

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

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

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

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

Informationals

[I-1] `PoolFactory_PoolDoesNotExist` is not used and should be removed

[I-2] Lacking zero address checks

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

        i_wethToken = IERC20(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());
```

[I-4] Event is missing `indexed` fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

▶ 4 Found Instances

- Found in src/PoolFactory.sol Line: 35

```
    event PoolCreated(address tokenAddress, address poolAddress);
```

- Found in src/TSwapPool.sol Line: 52

```
    event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
    event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
    event Swap(
```