[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnWeth` causes protocol to take too many tokens from the user, resulting in lost fees

Description: The `getInputAmountBasedOnWeth` function is intented 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 10000 instead of 1000.

Impact: Protocol takes more fees than expected from users.

Recommended Mitigation:

```
-    ((inputReserves * outputAmount) * 10000)
+    ((inputReserves * outputAmount) * 1000)
```

[H-2] `TSwapPool::deposit` is missing deadline checking 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 dealine 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,

          // @audit-high deadline not being used
          uint64 deadline
      )
          external
          revertIfZero(wethToDeposit)
+         revertIfDeadlinePassed(deadline)
          returns (uint256 liquidityTokensToMint)
      {
```

[H-3] Lack of slippage protecttion in `tSwapPool::swapExactOutput` causes users to potentially recieve

Description: The `swapExactOutput` function does not include any sort of slippage protection. This function is similar to what is done in `TSwapPool::swapExactInput`, wherethe 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 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 int the mempool, the market changes and the price moves HUGE -> 1 WETH is now 10,000 USDC. 10x more tahn 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 imclude 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,
        IERC20 outputToken,
        uint256 outputAmount,
        uint64 deadline
    )

     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 recieve 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 Concept:

▶ POC

```
    function testSellPoolTokens() public{
         vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        vm.startPrank(user);
        poolToken.approve(address(pool), 10e18);

        uint256 output = pool.swapExactOutput(weth, poolToken, 10e18,
uint64(block.timestamp));
        console.log("Output from swapExactOutput:", output );
    }
```

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) {
        // @audit this is wrong
        return
-
swapExactOutput(i_poolToken,i_wethToken,poolTokenAmount,uint64(block.timest
amp));
+             swapExactInput(i_poolToken,poolTokenAmount,i_wethToken,
minWethToRecevive,uint64(block.timestamp));
    }
```

[H-5] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` breaks the protpcol 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

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

Impact: The 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, the protocol's core invariant is broken

Proof Of Concept: A user swaps 10 times and collects the extra incentive. That user continues to swap untill all the funds in the protocol is draimed

▶ POC

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

        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));
        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 incentives or we should set aside tokens in the same way we do fees

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

[M-1] Rebase, fee on transfer, and ERC777 tokens break protocol invariant

/// finding...

[L-1] `TSwapPool::LiquidityAdded` event parameter is out of order

Description: when the `TSwapPool::LiquidityAdded` event is emitted in the `TSwapPool::addLiquidityMintAndTransfer` function, it logs values in an incorrect order. The `poolTokensToDeposit` value should go in the thired parameter position, where as the `wethToDeposit` value should go second

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

Recommended Mitigation:

```
    _mint(msg.sender, liquidityTokensToMint);
-   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 `output` it is never assigned a value nor uses an

explicit return statement.

Impact: The return value will always be zero, giving an incorrect output to the caller.

Proof Of Concept:

▶ POC

```
    function testSwapExactInputReturnValue() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        vm.startPrank(user);
        poolToken.approve(address(pool), 10e18);

        uint256 expected = 0;
        uint256 output = pool.swapExactInput(poolToken, 10e18, weth,
expected, uint64(block.timestamp));
        //console.log("Output from swapExactInput:", output );
        assertEq(output, expected);
    }
```

Recommended Mitigation:

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

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

```
    }
```

[I-1] `PoolFactory::PoolFactory_PoolDoesNotExist` is not used and should be reduced

```
- 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::liquidityTokenSymbol` 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] Lacking zero address checks

```
   constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    ) ERC20(liquidityTokenName, liquidityTokenSymbol) {

+         if(wethToken == address(0)){
+            revert
+        }
+         if(poolToken == address(0)){
+            revert
+        }

        i_wethToken = IERC20(wethToken);
```

```
            i_poolToken = IERC20(poolToken);
        }
```