# 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 the 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 is unfavourable.

**Impact :** Transactions could be sent when market conditions are unfavourable to deposit, even when adding a deadline parameter.

**Proof of Concept :** The `deadline` parameter is unused.

Place the test inside `TSwapPool.t.sol` .

1. The deadline is set to 0 which means that the deadline has passed.
2. The liquidityProvider can still deposit weth and poolToken into the pool.

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

     uint64 deadlineLate = uint64(0);

     pool.deposit(100e18,100e18,100e18,deadlineLate);


     assertEq(pool.balanceOf(liquidityProvider),100e18);
     assertEq(weth.balanceOf(liquidityProvider),100e18);
     assertEq(poolToken.balanceOf(liquidityProvider),100e18);

     assertEq(weth.balanceOf(address(pool)), 100e18);
     assertEq(poolToken.balanceOf(address(pool)), 100e18);
     vm.stopPrank();
  }
```

**Recommended Mitigation :** Consider making the following change to the function.

```
function deposit(
     uint256 wethToDeposit,
     uint256 minimumLiquidityTokensToMint,
     uint256 maximumPoolTokensToDeposit,
     uint64 deadline
 )
```

```
        external
        revertIfZero(wethToDeposit)
+        revertIfDeadlinePassed(deadline)
        returns (uint256 liquidityTokensToMint)
```

---

## [H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users, resulting in last fees.

**Description :** The `getInputAmountBasedOnOutput` function is intended to calculate 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 10000 instead of 1000.

**Impact :** Takes more fees than expected from users.

**Proof Of Code :**

1. Place this in your `TSwapPool.t.sol` .

```
       function testWrongInputAmount() public view {
        uint256 outputAmount = 10e18;
        uint256 outputReserves = 200e18;
        uint256 inputReserves = 100e18;

        uint256 expectedInputAmount = ((inputReserves * outputAmount) *
1000) / ((outputReserves - outputAmount) * 997);
        uint256 actualInputAmount =
pool.getInputAmountBasedOnOutput(outputAmount, inputReserves,
outputReserves);

        assertEq(expectedInputAmount, actualInputAmount);
    }
```

2. The assertion fails.

**Recommended Mitigation :**

```
    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) * 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 1 WETH right is now 1000 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 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 10,000 USDC instead of the expected 1,000 USDC .

PoC

**Recommended Mitigation:** We should include a `maxInputAmount` so the user only has to spend upto 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);
         _swap(inputToken, inputAmount, outputToken, outputAmount);
+        if(inputAmount > maxInputAmount){
+         revert();
+        }
```

---

### [H-3] `TSwapPool::sellTokens` 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 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( i.e `minWethToReceive` to be passed to `swapExactInput`)

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

    }
```

Additionaly it might be wise an deadline to the function , as there is currently no deadline .

---

### [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 pool tokens
- `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.

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 `1_000_000_000_000_000_000` tokens
2. The user continues to swap until all the protocol funds are drained.

▶ Proof Of Code

Place the following 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); //po

        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 , 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++;
-        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 parameter out of order causing event to emit incorrect information

**Description:** When the `LiquidityAdded` event is emitted in the `_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.

**Impact:** Event emission is incorrect, leading to offchain 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. However , while it declares the named return value `output` it is never asssigned a value, nor used a explicit 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 (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);
    }
```

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

**Description:**

**Impact:**

**Proof of Concept:**

**Recommended Mitigation:**