[s-#] TITLE (rOOT cAUSE + iMPACT)

Description:

Impact:

Proof of Concept:

Recommended Mitigation:

HIGHS

[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 calculate the amount of tokens a user should deposit given an amount of tokens of output token. 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

```
        function testFlawedSwapExactOutput() public {
            uint256 initialLiquidity = 100e18;
            vm.startPrank(liquidityProvider);
            weth.approve(address(pool), initialLiquidity);
            poolToken.approve(address(pool), initialLiquidity);

            pool.deposit({
                wethToDeposit: initialLiquidity,
                minimumLiquidityTokensToMint: 0,
                maximumPoolTokensToDeposit: initialLiquidity,
                deadline: uint64(block.timestamp)
            });
            vm.stopPrank();

            // User has 11 pool tokens
            address someUser = makeAddr("someUser");
            uint256 userInitialPoolTokenBalance = 11e18;
            poolToken.mint(someUser, userInitialPoolTokenBalance);
            vm.startPrank(someUser);

            // Users buys 1 WETH from the pool, paying with pool tokens
            poolToken.approve(address(pool), type(uint256).max);
            pool.swapExactOutput(
                poolToken,
                weth,
```

```
                1 ether,
                uint64(block.timestamp)
        );

    // Initial liquidity was 1:1, so user should have paid ~1 pool token
    // However, it spent much more than that. The user started with 11
tokens, and now only has less than 1.
    assertLt(poolToken.balanceOf(someUser), 1 ether);
    vm.stopPrank();

    // The liquidity provider can rug all funds from the pool now,
    // including those deposited by user.
    vm.startPrank(liquidityProvider);
    pool.withdraw(
        pool.balanceOf(liquidityProvider),
        1, // minWethToWithdraw
        1, // minPoolTokensToWithdraw
        uint64(block.timestamp)
    );

    assertEq(weth.balanceOf(address(pool)), 0);
    assertEq(poolToken.balanceOf(address(pool)), 0);
}
```

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 fo slippage protector 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::swapExactOInput`, where the function specifies a `minOutputAmount`, the `swapExactOutput` function should specify a `maxInputAmount`.

Impact: If martket conditions change before the transaction is completed, the user could get a much worse swap.

Proof of Concept: write yours.

1. The price of 1 WETH right now is 1,000 USDC
2. User input 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 amlount
4. As the transaction is pending in the mempoool, the market changes! and the price moves HUGE -> 1 WETH is now 10,000 USDC. 10x more than the user expected.
5. The transacyion completes, but the user sent the protocol 10,000 USDC instead of the expected 1,000 USDC,

Recommended Mitigation: We should include a `maxInputAmount` so the user only has to spend up to a specific amount, and can predict howmuch 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` mismacyhes 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 token they are 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 shouldbe called, because users specify the exact amount of input tokens, not

output.

Impact: Users will swap the wrong amount of token 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 chaning the `swapExactOutput`. Note that this would also require changing the `sellPoolToken` 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)
            );
    }
```

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 `swapCount` breaks the protocol invariant of `x * y = k`

Description: The protocol follows a strick 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 od swaps and collecting the extra incentive given out by the protocol.

Most simplyput, the protocol core invariant is broken.

Proof of Concept:

1. A user swap 10 times, and collect the extra incentive of `1_000_000_000_000_000 tokens.
2. The user contirnues 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);   //this is where we do the swap.
        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));


        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 mechanism. If you want to keep this in, we should account for th changes in the x * y = k protocol invariant. Or 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);
-       }
```

MEDIUMS

[M-1] `TSwapPool::deposit` is missing deadline check causing the transaction to complete even after the deadline

Description: The `deposit` function accepts a dealine 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, operation that adds liquidity to the pool might be executed at unexpected times, in marketconditions where the deposit rate is unfavorable.

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.

Recommended Mitigation: Consider making the following change to the function as below.

```
    function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint, //LP token -> if empty we
can pick hence STARTING_Y
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

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

Description:

Impact:

Proof of Concept:

Recommended Mitigation:

LOWS

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

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAnsTransfer` 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 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 token bought by the caller. However, while it declares the named 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: write mine

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, outputAmount);
    }
```

INFORMATIONALS

[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());
```

[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.

- 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(
```

[I-5] Use if Magic number is Discouraged

It can be confusing to see numbe literal in a codebade, and it's much more readeable if the numbers are given a name.

Impact: A little typo can cause for a wrong figure to to be written, one might mean to write 1000 and mistkenly write 10000.

Examples:

```
        uint256 inputAmountMinusFee = inputAmount * 997;
        uint256 denominator = (inputReserves * 1000) +
inputAmountMinusFee;
```

Instead, you could use:

```
    uint256 public constant REMAINDER = 997;
    uint256 public constnt PRECISION = 1000;

    uint256 inputAmountMinusFee = inputAmount * REMAINDER;
        uint256 denominator = (inputReserves * PRECISION) +
inputAmountMinusFee;
```

[I-6] No natspec written for `TSwapPool::swapExactInput` to explain what exactly the function is supposed to do.

Description: The natspec provides rich documentatio for the contract and for each function explaining what each function is supposed to do, it makes for easy unsderstanding of the code base outside the original programmers who wrote the Smart contract

Impact: Without the natspecc we can only guess what the function is meant to do and if there is a mistake it will be near impossible to detect hearby likely putting the entire protocol at risk.

Recommended Mitigation: Write the well documented natspec of `TSwapPool::swapExactInput`.

[I-7] `TSwapPool::swapExactInput` should be external rather than public for gas optimization

Let `TSwapPool::swapExactInput` be external rather than public as pexternal function utilizes less gas fees.

```
    function swapExactInput(
        IERC20 inputToken,  // e input token to swap / sell ie: DAI
        uint256 inputAmount, // e amount of input token to sell ie: DAOI
        IERC20 outputToken, // e output token to buy / buy ie: WETH
        uint256 minOutputAmount, // e minimum output amount expected to
receive
        uint64 deadline // e deadline for when the transaction should
expire.
    )
-       public
+       external
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (
            uint256 output)
```