High

[H-1] The function `TSwapPool:getInputAmountBasedOnOutput`, do not charged the user with a 0,03% of fee as expected, instead it charged to the user with a 91.3% of fee

Description

The maths on the function `TSwapPool:getInputAmountBasedOnOutput` are wrong. Why are bad?

because the function is multiplicating by 10_000 not by a 1_000 as expected. so it is charged with a 91.3% of fee instead of a 0,03% of fee.

```
@>     ((inputReserves * outputAmount) * 10000) / ((outputReserves - outputAmount) * 997);
```

Impact

The user is charge with a lot of fees. Intead of be charged with a 0,03%, he is charged with 91.3% of fee. making imposible to use the protocol. Example:

Input reserves and output reserves are identified by IR and OR

IR = 100e18 OR = 100e18

Bob 100 (want 100 as output) ((IR * OR) * 10000) / ((OR - 100) * 997) What charged bob with a 91.3% of fee. not with a 0,03% of fee. If the 10000 was a 1000 the fee would be 0,03%

Proof of Concept

Paste this test in `TSwapPool.t.sol`

Code

```javascript

function testFeeCalculation() 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 outputAmount = 9e18;
    uint256 inputReserves = 100e18;
    uint256 outputReserves = 100e18;
```

```
    uint256 calculatedInputAmount = pool.getInputAmountBasedOnOutput(outputAmount, inputRese

    uint256 expectedInputAmount = ((inputReserves * outputAmount) * 1000) / ((outputReserves

    console.log("Calculated Input Amount:", calculatedInputAmount);
    console.log("Expected Input Amount:", expectedInputAmount);
}
```

```

### Recommendation

Replace the 10000 with a 1000, this way the fee will be 0,03% instead of 91.3%

## [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 functions is similar to waht is done in `TSwapPool::swapExactInput`, where the function specifies a `minOutputAmount`, the `swapExactOutput` function should specify a `maxInputAmount`.

### Impact

If market condition changes before the transaction processes, the user may receive way fewer tokens than they expected.

### Proof of Concept

1. The price of 1 WETH right now is 1,000 USDC
2. User inputs a `swapExactOutput`
    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 in the mempool, the market changes! And the price moves HUGE -> 1 WETH is now 10,000 USDC. 10x more than the user expected.
```

5. The transaction completes, but the user sent the protool 10,000 USDC instead of the expected 1000 USDC.

——————Write POC ——————————————————————————————-

Recommendation

We should include a max input amount so the user only has to spend up to a specific amount, and can predict 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-3] `TSwapPool:sellPoolTokens` mismatches input and output tokens causing users to recieve the incorrect amoutn of tokens

Description

The `TSwapPool: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 int the `poolTokenAmount` parameter. Howerver 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 thaht should be called. Because users specify the exact amount of input toknes, not output.

Impact

Users will swap the wrong amount of tokens, which is a severe disruption to the protocol.

Proof of Concept

Paste this test in `TSwapPool.t.sol`

Code

```javascript
    function testSellPoolTokensGiveTheIncorrectAmount() public {


    console.log("BalanceOfUserWethB", weth.balanceOf(user));
    console.log("BalanceOfUserPoolB", pool.balanceOf(user));
    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.mint(user, 100e18);
    poolToken.approve(address(pool), 100e18);

    uint256 expected = 9e18;

    pool.sellPoolTokens(1e18);

    vm.stopPrank();

    console.log("BalanceOfUserWethA", weth.balanceOf(user));
    console.log("BalanceOfUserPoolA", pool.balanceOf(user));




}
```

Recommendation

Consider chaning the implemntations 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 minWeethToReceive
    ) external returns (uint256 wethAmount) {
        return
-            swapExactOutput(i_poolToken,i_wethToken,poolTokenAmount,uint64(block.timestamp)
+            swapExactInput(i_poolToken,poolTokenAmount,i_wethToken,minWeethToReceive,uint64
    }
```

Additionally it might be wise to add a deadline to the function as there is currently no deadline. (MEV later)

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

Description

The protocols 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 extras incentive in the `_swao` function. Meaning that over time the protocol funds will be drained.

The follow block of code is reponsible 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 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. Thath user continues to swap until all the protocol funds are drained.

Profe 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;
        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));
        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);
    }
```

Recommendation

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

Medium

[M-1] On the funtion `TSwapPool:deposit` the patameter `deadline` is never used, causing that the function won't revert if the deadline passed by the user is completed.

Description

The function `TSwapPool:deposit` never use the parameter `deadline`.

Impact

The function ignores the deadline so the transaction won't revert if the deadline passed by the user is completed.

Proof of Concept

1. A user deposit and set a deadline of the next block
2. The user interact or do wheatever he want
3. The deadline pass but the function don't revert, The deadline is ignored

Recommendation

Use the modifier `revertIfDeadlinePassed` with the deadline passed by the user to prevent this.

[M-2] Rebase, fee-on-transfer, and ERC-777 tokens break protocol invariant

Description

If i try to swap a token that has a fee-on-transfer, the protocol invariant is broken. x * y = k , isn't true any more, because, beeing a fee of transfer, the ratio of tokens change, changing also the invariant. the tokens ratio should be constant.

Impact

The protocol invariant is broken.

Proof of Concept

1. A user deposit 1000 tokens of WETH and 1000 tokens of poolToken
2. Another user swap 1000 tokens of poolToken for WETH
3. The invariant is broken (becasue the token have a fee-on-transfer, changing the ratio of tokens, what should be constant)

Here you have a PoC where you can see that the user recives more WETH than expected, because of the fee-on-transfer

Profe of Code

```javascript

    function testFeeOnTransferErc20BreaksInvariant() public {
    vm.startPrank(liquidityProvider);
    YeildERC20 tokenA = new YeildERC20();
    pool = new TSwapPool(address(tokenA), address(weth), "LTokenA", "LA");
    weth.approve(address(pool), 100e18);
    tokenA.approve(address(pool), 100e18);
    pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
    console.log("BalanceOfUserWethB", weth.balanceOf(user));
    console.log("BalanceOfUserPoolB", tokenA.balanceOf(user));
    vm.stopPrank();

    vm.startPrank(user);
    tokenA.mint(user, 100e18);
    tokenA.approve(address(pool), 100e18);
    uint256 expected = 9e18;

    pool.swapExactInput(tokenA, 10e18, weth, expected, uint64(block.timestamp));
    assert(weth.balanceOf(user) >= expected);

    console.log("BalanceOfUserWethA", weth.balanceOf(user));
    console.log("BalanceOfUserPoolA", tokenA.balanceOf(user));
}
```

Recommendation

Make some check to avoid ERC777 Tokens or fee-on-transfer tokens. Or try to implement a way to handle this without breking the invariant.

Low

[L-1] The event `LiquidityAdded` emited on `TSwapPool:_addLiquidityMintAndTransfer` is backwards, causing a wrong implementation of the event

Description

The event `LiquidityAdded` is emited on `TSwapPool:_addLiquidityMintAndTransfer` with the parameters inverted.

Impact

The event is emited with the parameters inverted, causing a wrong implementation of the event and interpretation of the event by the front-end or any other tool that use the event, like oracles.

Proof of Concept

The event `LiquidityAdded` is emited on `TSwapPool:_addLiquidityMintAndTransfer` with the parameters `liquidityProvider`, `wethDeposited`, `poolTokensDeposited`. but on the function, the parameters are inverted, the parameters are `liquidityProvider`, `poolTokensDeposited`, `wethDeposited`.

here is the event declaration:

```
event LiquidityAdded(
    address indexed liquidityProvider,
    uint256 wethDeposited,
    uint256 poolTokensDeposited
);
```

And here is the implementations, with the parameters inverted:

```
emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
```

Recommendation

Change the parameters of the event `LiquidityAdded` to `liquidityProvider`, `wethDeposited`, `poolTokensDeposited`.

[L-2] The function `TSwapPool:swapExactOutput` says that returns an uint256 output but this uint256 will be always 0 because they never give it a value

Description

The function `TSwapPool:swapExactInput` says that returns an uint256 output. This uint256 are never given a value, so they will always be 0.

Impact

The function `TSwapPool:swapExactInput` will always return 0.

Proof of Concept

Paste this test into `TSwapPool.t.sol`.

Code

```javascript
    function testSwapExactOutputAlwaysReturnZero() 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 = 9e18;

    uint256 alwaysZero = pool.swapExactInput(poolToken, 10e18, weth, expected, uint64(block.

    vm.stopPrank();

    assertEq(alwaysZero, 0);

}
```

Recommendation

I have some recomendations to solve this problem:

1. Give a value to output
2. RRemove the return of the function

Informational

[I-1] The `PoolFactory:PoolFactory__PoolDoesNotExist(address tokenAddress)` error is never used

Description

```
@>    error PoolFactory__PoolDoesNotExist(address tokenAddress);
```

This error is never used in the contract

Recommendation

Remove this line

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

[I-2] Nothing checks that a bad user pass zero address in the contrscrutor of the `PoolFactory` and `TSwapPool`, If someone pass zero address, the contract won't be functional

Description

There aren't any check to prevent that the parameter that pass the user isn't zero

Impact

The addresses will be zero. That affects directly to the functionality of the contract

Recommendation

Add some checks to prevent that the user pass zero address

[I-3] The parameters of the events should be indexed if there are thre or less

Description

The parameters of the events, if the event have 3 or less parameters should be indexed, this way, them are easeali accessible. If the event have more than 3 parameters, the 3 more important should be indexed

Recommendation

Add the `indexed` keyword before the name of the parameter on the events, you can do it 3 time per event.

[I-4] Unsed variable on `TSwapPool:deposit`, causign a gas waste.

Description

The variable `poolTokenReserves`, created on line 194, is never used.

Impact

How this variables access to storage and is never used, it causes a gas waste.

Recommendation

Remove the variable `poolTokenReserves` of the function `deposit`

```
-     uint256 poolTokenReserves = i_poolToken.balanceOf(address(this));
+
```

[I-5] Magic numbers should be avoided

Description

On the function `TSwapPool:getOutputAmountBasedOnInput` and ‘`TSwapPool:getInputAmountBasedOnOutput` the numbers 997 and 1000 are magic numbers.

Recommendation

These numbers should be replaced by constant variables

[I-6] The function `TSwapPool:swapExactInput` is public, but it is not used in the contract, beeing a gas waste

Description

The function `TSwapPool:swapExactInput` is public, but it is not used in the contract.

```
    function swapExactInput(
        IERC20 inputToken,
```

```
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
@>        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 output)
    {}
```

Impact

Public functions are more gas expensive than external functions, so, having a public function that is not used in the contract, is a gas waste

Recommendation

Mark the function `TSwapPool:swapExactInput` as external

[I-7] The function `TSwapPool:swapExactInput` do not have documentation

[I-8] Missing natspec documentation for parameter deadline on function `TSwapPool:swapExactOutput`