T-Swap

PoolFactory.sol ✅

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users, and 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 tokens. However the function currently miscalculates the resulting amount. When calculating the fees, it scales the amount by 10_000 instead of 1_000.

Impact:

Protocol takes more fees than expected from users.

Recommended Mitigation:

```
- ((inputReserves * outputAmount) * 10000) / ((outputReserves - ou
+ ((inputReserves * outputAmount) * 1000) / ((outputReserves - out
```

[H-3] Lack of slippage protection `TSwapPool::swapExactOutput` causes the users to potentially receive way fewer tokens

Description:

The `swapExactOutput` function does not include any sort of slippage protection. This function is similar 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 WETH is 1,000 USDC
2. Users inputs a `swapExactOutput` looking for 1 WETH
    a. inputToken = USDC
    b. outputToken = WETH
    c. outputAmount = 1
    d. 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 protocol 10,000 USDC instead of the expected 1,000 USDC

Recommended Mitigation:

We should include a `maxInputAmount` so the user only have 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 > maxOutputAmount) {
```

```
+         revert();
+     }

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

[H-4] `TSwapPool::sellPoolTokens` mismatches input and output tokens causing user 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 (ie `minWethToReceive` to be passed to `swapExactInput`)

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

Additionally, it might be wise to add a deadline to this function as there is currently no deadline.

[H-5] In `TSwapPoool::_swap` , the extra tokens given to the user 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 ration between the two amounts should remains 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 of swap 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. That user continue to swap until all the protocol funds are drained

▼ Proof of Code

Paste 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);
    for (uint i = 0; i < 9; i++) {
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
    }
```

```
        int256 startingY = int256(weth.balanceOf(address(pool)));
        int256 expectedDeltaY = 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 the change in the `x * y = k` protocol invariant. Or, we should set aside tokens in the same way we do it with the fees.

```
- swap_count++;
- if (swap_count >= SWAP_COUNT_MAX) {
-     swap_count = 0;
-     outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_0
- }
```

[M-1] `TSwapPoool::deposit` is missing `deadline` check causing transactions to complete even after 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 the market conditions are unfavorable to deposit, even when adding a deadline parameter.

Proof of Concept:

The `deadline` parameter is unused.

Recommended Mitigation:

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

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

Description:

When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTransfer` function, it logs values in an incorrect order. The `poolTokensDeposit` 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 malfunctionning.

Recommended Mitigation:

```
- emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDepos
+ emit LiquidityAdded(msg.sender, wethToDeposit, poolTokensToDepos
```

[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 user an explicit return statement.

Impact:

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

Recommended Mitigation:

```
function swapExactInput(
    IERC20 inputToken,
    uint256 inputAmount,
    IERC20 outputToken,
    uint256 minOutputAmount,
    uint64 deadline // @audit - info should be external
)
    public
    revertIfZero(inputAmount)
    revertIfDeadlinePassed(deadline)
    returns (
        uint256 output // @audit-low: protocol give wrong output
    )
{
    uint256 inputReserves = inputToken.balanceOf(address(this));
    uint256 outputReserves = outputToken.balanceOf(address(this));

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

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

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

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` us bit used and should be removed

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

[I-2] Lacking zero address checking

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

[I-3] `PoolFactory::createFactory` should use `.symbol()` instead of `.name()`

```
string memory liquidityTokenSymbol = string.concat(
    "ts",
-    IERC20(tokenAddress).name()
+    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](src/PoolFactory.sol#L35)

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

- Found in src/TSwapPool.sol [Line: 52](src/TSwapPool.sol#L52)

```
event LiquidityAdded(
```

- Found in src/TSwapPool.sol [Line: 57](src/TSwapPool.sol#L57)

```
event LiquidityRemoved(
```

- Found in src/TSwapPool.sol [Line: 62](src/TSwapPool.sol#L62)

```
event Swap(
```

[S-#] TITLE (Root Cause + Impact)

Description:

Impact:

Proof of Concept:

Recommended Mitigation: