High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` casues ptotocol takes too many toknes from users, resulting in loss of fees.

Description: The `getInputAmountBasedOnOutput` function is intended to calcaulate amount of tokens user should depoit given an amount tokens as output tokens.However the function currently miscalcualtion the resulting amount.When calculating the fee , it scales the amount by 10_000 instead of 1_00

Impact: Ptotocol takes more fees than expected from users.

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)
+         ((inputReserves * outputAmount) * 1000) /((outputReserves - outputAmount) * 997);
    }
```

[H-2] Lack of slippage protection in `TSwapPool::swapExactOutput` function causes potentinally recive way fewe 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 specify the `minOutputAmount`,the `swapExactOutput` function shiuild specify the `maxInputAmount`.

Impact: If market conditions change before transaction processes, the user could get much worse swap.

Recommended-Mitigation: We should include a `maxInputAmount` so the user only has to spend upto specific amount,can predict how much they will spend on the protocol.

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 in the mempool, the market chanegs!.And the price move HUGE -> 1 WETH is now 10,000 USDC. 10x more than the user expected.
5. The transaction complete, but the user sent the protocal 10,000 USDC instead of expected 1,000 USDC

```
    function swapExactOutput(
        IERC20 inputToken,
        IERC20 outputToken,
        uint256 outputAmount,
+       uint256 maxInputAmouont
        uint64 deadline
    )
        public
        revertIfZero(outputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 inputAmount)
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

        inputAmount = getInputAmountBasedOnOutput(
            outputAmount,
            inputReserves,
            outputReserves
        );

+        if(inputAmount > maxInputAmouont){
+            revert()
+        }
        _swap(inputToken, inputAmount, outputToken, outputAmount);
    }
```

[H-3] `swapExactOutput` function called within `TSwapPool::sellPoolTokens` function arguments passed inccorectly, resulting expected outcome will not achivable(will not receive expected tokens).

Description: The `sellPoolTokens` function intendedly allow users to sell tokens anda recieve WETH tokens in exchange.Users indicate how many pool tokens they are willing to sell or exchange in this `poolTOkenAmount` parameter.However, the function currenlty miscalculate the swapped amount

Impact: Users will swap the wrong amount of tokens , which is severe disruption of protocol functionlity.

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

```
function sellPoolTokens(
    uint256 poolTokenAmount,
+   uint256 minWethToReceive
  ) external returns (uint256 wethAmount) {
-    return swapExactOutput(i_poolToken,i_wethToken,poolTokenAmount,i_wethToken,uint64(
+    return swapExactOutput(i_poolToken,poolTokenAmoun,i_wethToken,minWethToReceive,uint

  }
```

[H-4] `TSwapPool::_swap` function extra tokens given to users after evry `swapCount` breaks the protocol invariant `x*y=k`

Description: The Protocal follows strict invariant of `x * y = k` - `x` : The balance of the pooltoken - `y` : The balance of WETH - `k` : The constant product of the balance

This means whenever balances change in protocal, the ratio between the two amounts should remain constant, hence the `k`. However this is broken extra incentive in the `_swap` function.Meaning that over the time protocol funds will be drained.

The followinf code is responsible for this 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 maliciously drain the protocol of funds by doing lot of swaps and collecting the extra incentive given out by the protocol.

Most simple put, the protocols core invarinat is borken.

Prrof Of Concept: 1. A user swaps 10 times and collects extra incentievs of `1_000_000_000_000_000_000` tokens 2. That user continous to swaps until protocol funds get drained.

Proof Of Code

```
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 expectedDeltY=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,expectedDeltY);
    }
```

Recommended-Mitigation: Remove the extra inceintive.If you want to keep this in, we should account for the change in the x * y = k protocol invariant. Or we shouuld 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] `TSwapPoop::deposit` function is missing deadline check causing transactions to complete after deadline reached also.

Description: `deposit` function has paramer called `deadline` but there no validation, with lack of this validation trasaction will execute after deadline crossed also. Due to this operation addliquidity to the pool might be executed at unexpected times, in market condition where the deposit rate is unfavorable

Impact: Transaction could be sent when market conditions is unfavorbale for deposit,even when adding a deadline parameter without validation.

Prof of Concept: The `deadline` parameter is unused.

Recommended Mitigation: Consider making following change to the function.

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

Low

[L-1] `TSwapPool::_addLiquidityMintAndTransfer` function have `LiquidityAdded` event parameters out of order.

Description: When `LiquidityAdded` event emitted in in the `TSwapPool::_addLiquidityMintAndTransfer` function , values logs will emit in incorrect oreder. Where as `poolTokensToDeposit` should go third position paramter like wise `wethToDeposit` should come 2nd position.

Impact: Event emission is incorrect where as 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` resuslts incorrect return value given.

Description: `swapExactInput` function is expected to return the actaul amount of tokens bought by caller.However while its declares the named return value `output` it is never assigned as value,nor uses an 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);
```

Gas

[G-1] `PoolFactory::createPool` function should use `abi.encodePacked` for moe gas effient

Description: Using `string.concat` will be not more gas efficeint patter instead we can use `abi.encodePacked` **Impact:** Leads to more estimation costs

Prof of Concept:

```
string memory liquidityTokenName = string.concat("T-Swap ", IERC20(tokenAddress).name());
```

- Gas cost is nearly `50-100 gas/bytes`

```
string memory liquidityTokenName = string.(abi.encodePackekd("T-Swap ", IERC20(tokenAddress)
```

- Gas cost is nearly `30-60 gas/bytes` -Nealry will save `20-40 gas/byte`

Recommended Mitigation: Instead of using `string.concat` use `string(abi.encodePacked)`

Code

```
-    string memory liquidityTokenName = string.concat("T-Swap ", IERC20(tokenAddress).name()
+    string memory liquidityTokenName = string.(abi.encodePackekd("T-Swap ", IERC20(tokenAdd
```

Informationals

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` not used and should be removed

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

[I-2] `PoolFactory::constructor` Lacking zero address validation

```
constructor(address wethToken) {
        //@auidt - low zero address validation missing
+       if(wethToken == address(0)){
+            revert zeroAddressAreNotValid();
+        }
        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] `TSwapPool::MINIMUM_WETH_LIQUIDITY` constant varibales we can declare `10e9` instead `1_000_000_000`

```
- uint256 private constant MINIMUM_WETH_LIQUIDITY = 1_000_000_000;
+ uint256 private constant MINIMUM_WETH_LIQUIDITY = 1e9;
```

[I-5] `TSwapPool::LiquidityAdded` event should have `indexed` key for better filteration especially if we have more than 3 arguments

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

[I-6] `TSwapPool::deposit` function its not suggestable to emit constant variales everytime

```
if (wethToDeposit < MINIMUM_WETH_LIQUIDITY) {
            revert TSwapPool__WethDepositAmountTooLow(
-               MINIMUM_WETH_LIQUIDITY,
                wethToDeposit
            );
        }
```

[I-6] `TSwapPool` **contract magical numbers casuing messy tracking**

```
+   uint256 private constant POOL_PERCENTAGE=997
+   uint256 private constant POOL_PRECESION=1000

-    uint256 inputAmountMinusFee = inputAmount * 997;
+    uint256 inputAmountMinusFee = inputAmount * POOL_PERCENTAGE;

        uint256 numerator = inputAmountMinusFee * outputReserves;
-        uint256 denominator = (inputReserves * 1000) + inputAmountMinusFee;
+        uint256 denominator = (inputReserves * POOL_PRECESION) + inputAmountMinusFee;
        return numerator / denominator;
```