T-Swap Audit Report

Version 1.0

palmcivet.eth

January 24, 2024

Protocol Audit Report

palmcivet.eth

24 January 2024

Prepared by: palmcivet.eth

Table of Contents

- Medium
  - [M-1] `TSwapPool::deposit` is missing `deadline` check, causing transactions to complete even after the deadline
  - [M-2] Rebase, fee-on-transfer, and ERC-777 tokens break protocol invariant
- Low
  - [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order, causing event to emit incorrect information
  - [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
- Informational
  - [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` error is not used, wasting gas
  - [I-2] Constructors lacks zero address check
  - [I-3] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`
  - [I-4] Events should be indexed if there are more than 3 parameters
  - [I-5] `TSwapPool::MINIMUM_WETH_LIQUIDITY` is a constant and therefore doesn't need to be emitted
  - [I-6] Unused local variable in `TSwapPool::deposit` function can be removed
  - [I-7] Following CEI (Checks, Effects, Interactions) is recommended
  - [I-8] Constants should be defined and used instead of "magic numbers" / literals
  - [I-9] Functions not used internally could be marked external
  - [I-10] `TSwapPool::swapExactInput` function is missing natspec
  - [I-11] `TSwapPoo::swapExactOutput` function is missing deadline parameter in natspec
  - [I-12] `TSwapPool::totalLiquidityTokenSupply` function should be marked external

Protocol Summary

This project is meant to be a permissionless way for users to swap assets between each other at a fair price. You can think of T-Swap as a decentralized asset/token exchange (DEX). T-Swap is known as an Automated Market Maker (AMM) because it doesn't use a normal "order book" style exchange, instead it uses "Pools" of an asset. It is similar to Uniswap. To understand Uniswap, please watch this video: Uniswap Explained

Disclaimer

The palmcivet.eth team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

The findings described in this document correspond to the following commit hash:

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda

Scope

```
./src/
#-- PoolFactory.sol
#-- TSwapPool.sol
```

Roles

- Liquidity Provider: The user who provides liquidity to the protocol in exchange for LP tokens.
- Trader: The user who uses the protocol to swap between tokens.

Executive Summary

I enjoyed reviewing this project and learnt a lot about the process. Thanks, Patrick.

Issues found

Severity	Number of issues found
High	4
Medium	2
Low	2
Informational	12
Total	20

Findings

High

[H-1] 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 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 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. That user continues to swap until all the protocol funds are drained

Code

Place the following into `TSwapPool.t.sol`

```
    function test_invariant_breaks() 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);

        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
```

```
27          vm.stopPrank();
28
29          uint256 endingY = weth.balanceOf(address(pool));
30          int256 actualDeltaY = int256(endingY) - int256(startingY);
31          assertEq(actualDeltaY, expectedDeltaY);
32      }
```

Recommended Mitigation: Consider removing 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 is done for the fees.

```
1 -         swap_count++;
2 -         if (swap_count >= SWAP_COUNT_MAX) {
3 -             swap_count = 0;
4 -             outputToken.safeTransfer(msg.sender, 1
      _000_000_000_000_000_000);
5 -         }
```

[H-2] Incorrect fee calculationin `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 output tokens. 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

Place the following into `TSwapPool.t.sol`

```
1  function test_getInputAmountBasedOnOutput_takes_higher_fee() public {
2          vm.startPrank(user);
3
4          // Define input reserves, output reserves, and output amount
5          uint256 inputReserves = 10000; // Example value
6          uint256 outputReserves = 5000; // Example value
7          uint256 outputAmount = 1000; // Example amount
8
9          // Call the function with these values
10         uint256 actualInputAmount = pool.getInputAmountBasedOnOutput(
               outputAmount, inputReserves, outputReserves);
11         // Calculate the expected input amount with correct fee (0.3%)
12         uint256 expectedInputAmount = ((inputReserves * outputAmount) *
               1000) / ((outputReserves - outputAmount) * 997);
```

```
13
14          // Assert that the actual input amount is higher than the
                expected amount
15          assertGt(actualInputAmount, expectedInputAmount);
16          console.log("Expected input amount:", expectedInputAmount);
17          console.log("Actual input amount:", actualInputAmount);
18          vm.stopPrank();
19
20          vm.startPrank(liquidityProvider);
21          weth.approve(address(pool), 100e18);
22          poolToken.approve(address(pool), 100e18);
23          pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
24          vm.stopPrank();
25
26          // Simulate a user swapping to demonstrate the loss
27          vm.startPrank(user);
28          IERC20 inputToken = IERC20(address(poolToken)); // Assuming
                poolToken is the input token
29          IERC20 outputToken = IERC20(address(weth)); // Assuming WETH is
                 the output token
30          uint64 deadline = uint64(block.timestamp + 1 hours); // Set a
                deadline for the swap
31
32          // Record user's token balances before the swap
33          uint256 userInitialInputTokenBalance = inputToken.balanceOf(
                user);
34          uint256 userInitialOutputTokenBalance = outputToken.balanceOf(
                user);
35
36          // Perform the swap
37          poolToken.approve(address(pool), 100e18);
38          pool.swapExactOutput(inputToken, outputToken, outputAmount,
                deadline);
39
40          // Check user's token balances after the swap
41          uint256 userFinalInputTokenBalance = inputToken.balanceOf(user)
                ;
42          uint256 userFinalOutputTokenBalance = outputToken.balanceOf(
                user);
43
44          // Calculate the amount of input tokens spent by the user
45          uint256 inputTokensSpent = userInitialInputTokenBalance -
                userFinalInputTokenBalance;
46          uint256 outputTokensReceived = userFinalOutputTokenBalance -
                userInitialOutputTokenBalance;
47
48          // Assert that the user received the correct output amount
49          assertEq(outputTokensReceived, outputAmount);
50          // Assert that the user spent more input tokens than the
                expected amount
51          assertGt(inputTokensSpent, expectedInputAmount);
```

```
52
53          console.log("Output tokens received:", outputTokensReceived);
54          console.log("Input tokens spent:", inputTokensSpent);
55          vm.stopPrank();
56      }
```

Recommended Mitigation:

```
1  function getInputAmountBasedOnOutput(
2          uint256 outputAmount,
3          uint256 inputReserves,
4          uint256 outputReserves
5      )
6          public
7          pure
8          revertIfZero(outputAmount)
9          revertIfZero(outputReserves)
10         returns (uint256 inputAmount)
11     {
12 -       return ((inputReserves * outputAmount) * 10_000) / ((
       outputReserves - outputAmount) * 997);
13 +       return ((inputReserves * outputAmount) * 1_000) / ((
       outputReserves - outputAmount) * 997);
14     }
```

[H-3] 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. 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 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

Code

Place the following into `TSwapPool.t.sol`

```
function test_swapExactOutput_has_no_slippage_protection() public {
        poolToken.mint(user, 10000e18);
        vm.startPrank(liquidityProvider);
        weth.mint(address(pool), 50e18);
        poolToken.mint(address(pool), 5000e18);
        vm.stopPrank();

        vm.startPrank(user);
        poolToken.approve(address(pool), 10000e18);

        uint256 outputAmount = 1e18; // User wants 1 WETH
        IERC20 inputToken = IERC20(address(poolToken));
        IERC20 outputToken = IERC20(address(weth));
        uint64 deadline = uint64(block.timestamp + 1 hours);

        // Record user's input token balance before the swap
        uint256 userInitialInputTokenBalance = inputToken.balanceOf(
            user);

        // Perform the swap
        pool.swapExactOutput(inputToken, outputToken, outputAmount,
            deadline);

        // Check user's input token balance after the swap
        uint256 userFinalInputTokenBalance = inputToken.balanceOf(user)
            ;

        // Calculate the amount of input tokens spent by the user
        uint256 inputTokensSpent = userInitialInputTokenBalance -
            userFinalInputTokenBalance;

        // Assert that the user spent significantly more input tokens
            due to lack of slippage protection
        // The expected amount should be significantly lower than the
            actual amount spent
        uint256 expectedInputAmount = 1000e18; // Example expected
            amount (needs to be realistic based on initial pool
             // reserves)
        assertGt(inputTokensSpent, expectedInputAmount);

        console.log("Expected input amount:", expectedInputAmount);
```

```
35          console.log("Actual input amount:", inputTokensSpent);
36
37          vm.stopPrank();
38      }
```

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

```
1       function swapExactOutput(
2           IERC20 inputToken,
3   +       uint256 maxInputAmount,
4   .
5   .
6   .
7       inputAmount = getInputAmountBasedOnOutput(outputAmount,
            inputReserves, outputReserves);
8   +   if(inputAmount > maxInputAmount) revert();
9       _swap(inputToken, inputAmount, outputToken, outputAmount);
```

[H-4] `TSwapPool::sellPoolTokens` 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 wheras 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:

Code

Place the following into `TSwapPool.t.sol`

```
1   function test_sellPoolTokens_mismatches_input_and_output() public {
2           // Mint additional tokens to the user and the pool
3           poolToken.mint(user, 10000e18);
4           uint256 poolTokenAmountToSell = 10e18; // Example amount of
                pool tokens to sell
5           poolToken.mint(user, poolTokenAmountToSell);
6
```

```
7           // Adjust pool reserves to create a more extreme condition
8           vm.startPrank(liquidityProvider);
9           weth.mint(address(pool), 100e18); // Example WETH reserve
10          poolToken.mint(address(pool), 1000e18); // Example poolToken
                reserve, creating a high slippage scenario
11          vm.stopPrank();
12
13          vm.startPrank(user);
14          poolToken.approve(address(pool), type(uint256).max);
15
16          // Record user's WETH balance before selling pool tokens
17          uint256 userInitialWethBalance = weth.balanceOf(user);
18
19          // User sells pool tokens
20          pool.sellPoolTokens(poolTokenAmountToSell);
21
22          // Record user's WETH balance after selling pool tokens
23          uint256 userFinalWethBalance = weth.balanceOf(user);
24
25          // Calculate the amount of WETH received
26          uint256 wethReceived = userFinalWethBalance -
                userInitialWethBalance;
27
28          // Calculate expected WETH if swapExactInput was used
29          uint256 expectedWethWithExactInput = pool.
                getOutputAmountBasedOnInput(
30              poolTokenAmountToSell, poolToken.balanceOf(address(pool)),
                    weth.balanceOf(address(pool))
31          );
32
33          // Assert that the amount of WETH received is not what would be
                expected from swapExactInput
34          assertNotEq(
35              wethReceived,
36              expectedWethWithExactInput,
37              "User received an amount of WETH equivalent to
                    swapExactInput, which is not expected."
38          );
39
40          console.log("Expected WETH with swapExactInput:",
                expectedWethWithExactInput);
41          console.log("Actual WETH received:", wethReceived);
42
43          vm.stopPrank();
44      }
```

Recommended Mitigation: Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require changing the `sellPoolTokens` to accept a new parameter (ie `minWethToReceive` to be passed to `swapExactInput`).

```
-    function sellPoolTokens(uint256 poolTokenAmount) external returns (uint256 wethAmount) {
+    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. (MEV later in course)

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 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 market conditions are unfavorable to deposit, even when adding a deadline parameter.

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

```
Warning (5667): Unused function parameter. Remove or comment out the variable name to silence this warning.
   --> src/TSwapPool.sol:107:9:
    |
107 |         uint64 deadline
    |         ^^^^^^^^^^^^^^^
```

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

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

```
8           revertIfZero(wethToDeposit)
9   +       revertIfDeadlinePassed(deadline)
10          returns (uint256 liquidityTokensToMint)
11      {
```

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

Description: Rebase tokens will break the protocol invariant because of their inherent mint and burn functionality to maintain their stable prices, which could be manipulated to adjust the supply. This dynamic supply change can disrupt the pool's balance without corresponding swaps, breaking the invariant.

Fee-on-transfer tokens will break the protocol invariant as discussed in H-1. Tokens that deduct a fee on transfer effectively remove a portion of the tokens from the pool during each swap. This alters the pool's balance and disrupts the constant product invariant.

ERC-777 tokens introduce potential reentrancy risks due to their hooks that allow additional logic during transfer operations. While not directly impacting the constant product formula, they can lead to unforeseen vulnerabilities and manipulation within the protocol. Please see Consensys' UniswapV1 audit for more info.

Impact: The use of such tokens could enable malicious actors to drain funds from the protocol or manipulate prices. With rebase tokens, the changing supply can be exploited to create imbalances. Fee-on-transfer tokens can steadily deplete the pool's reserves. ERC-777 tokens introduce additional vectors for attack such as reentrancy, potentially compromising the integrity of the protocol.

Proof of Concept:

- Rebase tokens could be used in a series of swaps to artificially inflate the token amount in the pool, then rebase to a lower supply, effectively removing more value than contributed.
- Fee-on-transfer tokens can be repeatedly swapped in and out of the pool. Each transaction would erode the pool's balance, leading to a gradual drain of resources.
- ERC-777 tokens can potentially exploit callback functions to perform reentrancy attacks, manipulating the swap process or extracting funds.

Recommended Mitigation: Consider restricting token types by implementing checks to ensure only standard ERC20 tokens without these functionalities are allowed.

Low

[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::_addLiquidityMintAndTrans` function, it logs the 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 tokens bought by the caller. However, while it declares the named return 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:

Code

Place the following into `TSwapPool.t.sol`

```
function test_swapExactInput_returns_zero() public {
        uint256 zero = 0;

        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        IERC20 inputToken = IERC20(address(poolToken)); // Assuming poolToken is the input token
        uint256 inputAmount = 1000; // Example amount
        IERC20 outputToken = IERC20(address(weth)); // Assuming WETH is the output token
```

```
13          uint256 minOutputAmount = 100; // Example amount
14          uint64 deadline = uint64(block.timestamp + 1 hours); // Set a
                deadline for the swap
15
16          vm.startPrank(user);
17          poolToken.approve(address(pool), 100e18);
18
19          uint256 returnedValue = pool.swapExactInput(inputToken,
                inputAmount, outputToken, minOutputAmount, deadline);
20
21          vm.stopPrank();
22
23          assertEq(zero, returnedValue);
24      }
```

Recommended Mitigation:

```
1       {
2           uint256 inputReserves = inputToken.balanceOf(address(this));
3           uint256 outputReserves = outputToken.balanceOf(address(this));
4
5   -       uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount,
        inputReserves, outputReserves);
6   +       output = getOutputAmountBasedOnInput(inputAmount, inputReserves
        , outputReserves);
7
8   -       if (outputAmount < minOutputAmount) {
9   -           revert TSwapPool__OutputTooLow(outputAmount,
        minOutputAmount);
10  +       if (output < minOutputAmount) {
11  +           revert TSwapPool__OutputTooLow(output, minOutputAmount);
12          }
13
14  -       _swap(inputToken, inputAmount, outputToken, outputAmount);
15  +       _swap(inputToken, inputAmount, outputToken, output);
16      }
```

Informational

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` error is not used, wasting gas

Description: The error `PoolFactory__PoolDoesNotExist` is never used in the contract.

Impact: This costs additional gas to deploy.

Recommended Mitigation: Remove the error from the codebase.

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

[I-2] Constructors lacks zero address check

```
constructor(address wethToken) {
+       if(wethToken == address(0)) revert PoolFactory__InvalidAddress
    ();
        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] Events should be indexed if there are more than 3 parameters

Description: 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: 43

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

- Found in src/TSwapPool.sol Line: 44

```
    event LiquidityRemoved(address indexed liquidityProvider,
        uint256 wethWithdrawn, uint256 poolTokensWithdrawn);
```

- Found in src/TSwapPool.sol Line: 45

```
1      event Swap(address indexed swapper, IERC20 tokenIn, uint256
           amountTokenIn, IERC20 tokenOut, uint256 amountTokenOut);
```

```
1 -    event Swap(address indexed swapper, IERC20 tokenIn, uint256
      amountTokenIn, IERC20 tokenOut, uint256 amountTokenOut);
2 +    event Swap(address indexed swapper, IERC20 indexed tokenIn,
      uint256 amountTokenIn, IERC20 indexed tokenOut, uint256
      amountTokenOut);
```

[I-5] `TSwapPool::MINIMUM_WETH_LIQUIDITY` is a constant and therefore doesn't need to be emitted

```
1 -    error TSwapPool__WethDepositAmountTooLow(uint256 minimumWethDeposit
      , uint256 wethToDeposit);
2 +    error TSwapPool__WethDepositAmountTooLow(uint256 wethToDeposit);
```

```
1 if (wethToDeposit < MINIMUM_WETH_LIQUIDITY) {
2 -            revert TSwapPool__WethDepositAmountTooLow(
      MINIMUM_WETH_LIQUIDITY, wethToDeposit);
3 +            revert TSwapPool__WethDepositAmountTooLow(wethToDeposit);
4          }
```

[I-6] Unused local variable in `TSwapPool::deposit` function can be removed

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

[I-7] Following CEI (Checks, Effects, Interactions) is recommended

```
1 -    _addLiquidityMintAndTransfer(wethToDeposit,
      maximumPoolTokensToDeposit, wethToDeposit);
2
3      liquidityTokensToMint = wethToDeposit;
4 +    _addLiquidityMintAndTransfer(wethToDeposit,
      maximumPoolTokensToDeposit, wethToDeposit);
```

[I-8] Constants should be defined and used instead of "magic numbers" / literals

Using constants instead of literals will make the code more readable.

- Found in src/TSwapPool.sol Line: 229

```
1         uint256 inputAmountMinusFee = inputAmount * 997;
```

- Found in src/TSwapPool.sol Line: 231

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

- Found in src/TSwapPool.sol Line: 246

```
1         return ((inputReserves * outputAmount) * 10000) / ((
              outputReserves - outputAmount) * 997);
```

- Found in src/TSwapPool.sol Line: 331

```
1             outputToken.safeTransfer(msg.sender, 1
                  _000_000_000_000_000_000);
```

- Found in src/TSwapPool.sol Line: 374

```
1             1e18, i_wethToken.balanceOf(address(this)),
                  i_poolToken.balanceOf(address(this))
```

- Found in src/TSwapPool.sol Line: 380

```
1             1e18, i_poolToken.balanceOf(address(this)),
                  i_wethToken.balanceOf(address(this))
```

[I-9] Functions not used internally could be marked external

- Found in src/TSwapPool.sol Line: 249

```
1     function swapExactInput(
```

[I-10] `TSwapPool::swapExactInput` function is missing natspec

[I-11] `TSwapPoo::swapExactOutput` function is missing deadline parameter in natspec

```
1 /*
2      * @notice figures out how much you need to input based on how much
3      * output you want to receive.
4      * @param inputToken ERC20 token to pull from caller
5      * @param outputToken ERC20 token to send to caller
6      * @param outputAmount The exact amount of tokens to send to caller
7      */
8     function swapExactOutput(
```

```
        IERC20 inputToken,
        IERC20 outputToken,
        uint256 outputAmount,
        uint64 deadline
    )
```

[I-12] `TSwapPool::totalLiquidityTokenSupply` function should be marked external

```
function totalLiquidityTokenSupply() public view returns (uint256) {
        return totalSupply();
    }
```