TSwap Protocol Audit Report

Version 1.0

kidhapelmzy

June 3, 2025

Protocol Audit Report

kidhapelmzy

June 2, 2025

Prepared by: kidhapelmzy Lead Auditors: - kidhapelmzy [CMG]

Table of Contents

- Lows
  - [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
- Informationals
  - [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed
  - [I-2] Lacking zero address checks
  - [I-3] `PoolFactory::createPool` should use `.symbol()` instead od `.name()` for `liquidityTokenSymbol`

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The YOUR_NAME_HERE 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

Scope

```
#-- e643a8d4c2c802490976b538dd009b351b1c8dda
```

Roles

- Liquidity Providers: Users who have liquidity deposited into the pools. Their shares are represented by the LP ERC20 tokens. They gain a 0.3% fee every time a swap is made.
- Users: Users who want to swap tokens.

Issues found

Severity	Number of issues found
High	3
Medium	1
Low	2
Info	3
Total	9

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getOutputAmountBasedOnInput` causes protocol to take too many tokens from users, resulting in lost fees

Description: The `getOutputAmountBasedOnInput` function is intended to calculate the amount of tokens a user should deposit given an amount of tokens of output tokens. However, the

function currectly miscalculates the resulting amount. When calculating the fee, it scales the amount by 10_000 instead of 1_000.

Impact: Protocols takes more fees than intended from users.

Proof of Concept: To test this, include the following code in the `TSwapPool.t.sol` file:

```
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, pays with poolToken
        poolToken.approve(address(pool), type(uint256).max);
        pool.swapExactOutput(poolToken, weth, 1 ether, uint64(block.
            timestamp));

        // Initially 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 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 deposityed by user.
        vm.startPrank(liquidityProvider);
        pool.withdraw(
            pool.balanceOf(liquidityProvider),
            1, // minwethToWithdraw
            1, // minPoolTokensToWithdwar
            uint64(block.timestamp)
        );

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

```
42      }
```

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
13  -           ((inputReserves * outputAmount) * 10000) / ((outputReserves
        - outputAmount) * 997);}
14  +           ((inputReserves * outputAmount) * 1000) / ((outputReserves
       - outputAmount) * 997);}
```

[H-2] Lack of slippage protection in `TSwapPool::swapExactOutput` cause users to potentilaly recieve way fewer tokens.

Description: The `swapExactOutput` function does not include any sort of slippage protection. This functiom 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 would get a much worse swap.

Proof of Concept: 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 max Input amount 4. As the transaction is pending in the mempool, the market changes! And the price moves HUGE -> 1 WETH is noe 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 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
8  .
9
10 .
11         inputAmount = getInputAmountBasedOnOutput(
12                 outputAmount,
13                 inputReserves,
14                 outputReserves
15             );
16 +           if(inputAmount > maxInputAmount) {
17                 revert();
18 }
```

[H-3] `TSwapPool::sellPoolTokens` mismatched input and output token amount causing users to recieve the incorrect amount of tokens

Description: The `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 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(`minWethToRecieve`) to be passed to `swapExactInput`

```
1  function sellPoolTokens(
2          uint256 poolTokenAmount,
3  +       uint256 minWethToRecieve,
4      ) external returns (uint256 wethAmount) {
5          return
6  -           swapExactOutput(i_poolToken, i_wethToken, poolTokenAmount,
       uint64(block.timestamp));
7  +           swapExactOutput(i_poolToken, poolTokenAmount, i_wethToken,
       minWethToRecieve, uint64(block.timestamp));
8      }
```

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

```

### [H-4] In `TSwapPool::_swap` the extra token 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 balance change in the protocol, the ratio between the two accounts 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 fund will be drained.

The following block of code is responsible for the issue.

```javascript
        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 simplyput, the protocol's core invariant is broken.

Proof of Concept: 1. A user swaps 10 times and collect the extra incentives of `1_000_000_000_000_000_000` tokens 2. That user continues 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);
        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)
        );
```

```
62          pool.swapExactOutput(
63              poolToken,
64              weth,
65              outputWeth,
66              uint64(block.timestamp)
67          );
68
69          int256 starting__Y = int56(weth.balanceOf(address(pool)));
70          int256 expectedDelta__Y = int256(-1) * int256(outputWeth);
71
72          pool.swapExactOutput(
73              poolToken,
74              weth,
75              outputWeth,
76              uint64(block.timestamp)
77          );
78          vm.stopPrank();
79
80          uint256 ending__Y = weth.balanceOf(address(pool));
81
82          int256 actualDelta__Y = int256(ending__Y) - int256(starting__Y)
                ;
83          assertEq(actualDelta__Y, expectedDelta__Y);
84      }
```

Recommended Mitigation: Remove the extra incentive mechanism. If you want to keep this is, 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 for 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  -         }
```

Medium

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

Description: The `deposit` function acccepts a deadline parameter, which according to the documentation is "The deadline for the transaction to be completed by". However, this parameter is not 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 unfavourable.

Impact: Transaction couls be sent when marketconditions are unfavorableto 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

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

Lows

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

Description: When the `liquidityAdded` event is emmited in the `TSwapPool::_addLiquidityMintAndTra` function, it logs values in an incorrect order. The `poolTokenToDeposit` 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 name 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:

Recommended Mitigation:

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 od `.name()` for `liquidityTokenSymbol`

```
-         string memory liquidityTokenSymbol =string.concat("ts",IERC20(tokenAddress).name());

+          string memory liquidityTokenSymbol = string.concat("ts", IERC20(tokenAddress).symbol());
```