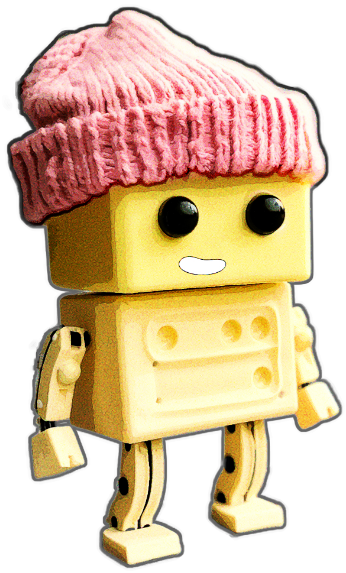

TSwap Initial Audit Report

Version 0.1

Cyfrin.io

April 11, 2025

Protocol Audit Report

Akoja

April 11th, 2025

TSwap Audit Report

Prepared by: Akoja Lead Auditors: - Abraham akoja

See table

- Lows
  * [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order causing an incorrect info to be emitted.
  * [L-2] Default value returned by `TSwapPool::swapExactInput` function results in incorrect return value given
- High
  * [H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users, resulting in lost fees.
  * [H-2] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially recieve way fewer tokens.
  * [H-3] `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to recieve the incorrect amount of tokens
  * [H-4] In `TSwapPool::_swap` the extr tokens given to users after every `swapcount` breaks the protocol invariant of `x*y= K`. where:

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.

Disclaimer

I made all effort to find as many vulnerabilities in the code in the given time period, but hold no responsibilities for the findings provided in this document. A security audit by me 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

		Impact		
	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

Commit Hash

```
e643a8d4c2c802490976b538dd009b351b1c8dda
```

Scope

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

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.

Executive Summary

i found 8 vunerabilities, 4 highs and one informational, 2 lows and 1 medium. total time spent was an a couple of hours.

Issues found

Severity	Nunber of issues found
high	4
Medium	1
Low	2
Info	1
Total	8

Findings

Informationals

[I-1] Lacking Zero Address checks

Description:

```
constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
 +      if(wethToken == address(0)){
 +       revert();
 +   }

 +      if(poolToken == address(0)){
 +       revert();
 +   }
        i_wethToken = IERC20(wethToken);
        i_poolToken = IERC20(poolToken);
    }
```

[I-2] `PoolFactory::createPool` should use `.symbol()` instead od `.name()`

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

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

[I-3] the `PoolFactory__PoolDoesNotExist` within `PoolFactory` contract is not used.

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

Medium

[M-1] `TSwapPool::deposit` is missing deadline check causing transactions to complete even after the deadline exceeds

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 times, in market conditions where the deposit rate is unfavourable.

Impact: Transactions could be sent when market conditions are unfavourable to deposit, even when adding a deadline parameter.

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

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

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

Lows

[L-1] `TSwapPool::LiquidityAdded` event has parameters out of order causing an incorrect info to be emitted.

Description: when the `liquidityAdded` event is emmitted in the `TSwapPool::_addLiquidityMintAndTra` function, it logs values in an incorrect order. The `poolTokensToDeposit` value should go in the third parameter position, whereass 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` function 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 would always be zero, giving incorrect info to the caller.

Proof of Concept:

```
function testSwapExactInputReturnsZero() public{
        testDeposit();
        vm.startPrank(liquidityProvider);
          uint256 initialLiquidity = 100e18;
           weth.approve(address(pool), initialLiquidity);
        poolToken.approve(address(pool), initialLiquidity);
        uint256 test = pool.swapExactInput(poolToken, initialLiquidity,
            weth, 1,  uint64(block.timestamp));
        assertEq(test, 0);
        console.log("test value",test);
    }
```

Recommended Mitigation:

```
{
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

        uint256 outputAmount = getOutputAmountBasedOnInput(
            inputAmount,
            inputReserves,
            outputReserves
        );

        if (outputAmount < minOutputAmount) {
            revert TSwapPool__OutputTooLow(outputAmount,
                minOutputAmount);
        }

        _swap(inputToken, inputAmount, outputToken, outputAmount);

+           return outputAmount;
        }
```

High

[H-1] Incorrect fee calculation in `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 tokens 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:

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);

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

As a result, user swapping tokens via the `swapExactOutput` function will pay more tokens than expected for thier trades. this is risky for users that provide infinite allowance to the `TSwapPool` contract. Moreover, note that the issue is worsened by the fact that the `swapExactOutput` function does not allow users to specify a maximum of input tokens, as is describe in another issue in this report. it is worth nothing that the tokens paid by users are not loast but rather can be swiftly taken by liquidity providers. therefore, this contract could be used to trick users, have them swap funds at unfavourable rates and finally rug all the liquidity from the pool. 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);
        console.log("beforeswap",poolToken.balanceOf(someUser));

        // user buys 1 weth from the pool paying with pool tokens
        poolToken.approve(address(pool), type(uint256).max);
        pool.swapExactOutput(
            poolToken,
            weth,
            1 ether,
            uint64(block.timestamp)
        );

        // initial liquidity is 1:1 so users should have paid ~1 pool
            token
        // however it spent much more than that the user started with
            11 tokens and now only has less
        assertLt(poolToken.balanceOf(someUser), 1 ether);
        vm.stopPrank();



        // the liquidity provider can rug all the funds from the pool
            now
        // including those deposited by the user.
        vm.startPrank(liquidityProvider);
        pool.withdraw(
            pool.balanceOf(liquidityProvider),
            1,//minWethToWithdraw
            1,//minPoolTokenToWithdraw
            uint64(block.timestamp)
        );

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

```
            assertEq(poolToken.balanceOf(address(pool)),0);
             console.log("afterswap",poolToken.balanceOf(someUser));

        }
```

[H-2] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially recieve 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. usdt is 1,000 usdc for 1 2. user inputs a `swapExactOutput` looking for WETH a. inputToken = USDC b. outputToken = WETH c. outputAmount = 1 d. deadline = watever 3. the function does not offer a maxInput amount 4. as the transaction is pending in the mempool, the market changes and prices move HUGE -> Weth is now 10,000 usdc. 10x more than the user expected 5. the transaction completes but the user sent the protocol 10k usdc instead of the expected 1k.

Recommended Mitigation:

```
    function swapExactOutput(
            IERC20 inputToken,
            IERC20 outputToken,
            uint256 outputAmount,
+           uint256 maxInputAmount,
            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 > maxInputAmount){
+                revert();
+           }

```

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

[H-3] `TSwapPool::sellPoolTokens` mismatches input and output tokens 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 are 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 `minWethToRecieve` to be passed `swapExactInput`

```
   function sellPoolTokens(
          uint256 poolTokenAmount,
+         uint256 minWethToRecieve,
      ) external returns (uint256 wethAmount) {

-       return swapExactOutput(
-               i_poolToken,
-               i_wethToken,
-                poolTokenAmount,
-                uint64(block.timestamp)
-           );

+       return swapExactInput(
+               i_poolToken,
+               poolTokenAmount,
+                i_wethToken ,
+               minWethToRecieve,
+                uint64(block.timestamp)
+           );
      }
```

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

[H-4] In `TSwapPool::_swap` the extr tokens given to users after every `swapcount` breaks the protocol 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 incentives int he `_swap` function. Meaning that over time the protocol funds will be drained

Description: A user could could maliciosly drain the protocol of funds by doing a lot of swaps and collecting the extra incentive given out by the protocol.

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 doin a lot of swaps and collecting the extra incentive given out by the protocol.

```
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 startingX = int256(poolToken.balanceOf(address(pool)))
       ;

        // expectedDeltaX = int256(poolTokenAmount);

         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)
            );

          int256 startingY = int256(weth.balanceOf(address(pool)));
          int256 expectedDeltaY = 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, expectedDeltaY);
        }
```

Proof of Concept: 1. A user swaps 10 times and coleects the extra incentive of 1`_000_000_000_000_000_000` tokens 2. That user continues to swap untill 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;
      //    int256 startingX = int256(poolToken.balanceOf(address(pool)))
           ;

            // expectedDeltaX = int256(poolTokenAmount);

             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)
            );

           int256 startingY = int256(weth.balanceOf(address(pool)));
           int256 expectedDeltaY = 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, expectedDeltaY);
        }
```

Recommended Mitigation: remove the extra incentive mechanism unless you want to keep this then you 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);
-        }
```