# TSwapPool Audit Report

Version 0.1

*Cyfrin.io*

February 17, 2024

# Protocol Audit Report

amaqkkg

February 17, 2024

Prepared by: amaqkkg

Lead Auditors:

- amaqkkg

## Table of Contents

    * [H-3] `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to receive incorrect amount of tokens
    * [H-4] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` breaks the protocol invariant of `x * y = k`
  - Medium
    * [M-1] `TSwapPool::deposit` is missing deadline check causing transaction to complete even after the deadline
  - Low
    * [L-1] `TSwapPool::LiquidityAdded` event has parameter 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` is not used and should be removed
    * [I-2] `PoolFactory::constructor` and `TSwapPool::constructor` lacking zero address check
    * [I-3] `PoolFactory::createPool` should use .symbol() instead of .name()
    * [I-4] Event is missing `indexed` fields

## 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 amaqkkg 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 |
| Likelihood | High | H | H/M | M |
| | 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

Part of cyfrin updraft security research audit course.

### Issues found

| Severity | Number of issues found |
| --- | --- |
| High | 4 |
| Medium | 1 |
| Low | 2 |
| Info | 4 |
| Total | 11 |

## Findings

### High

#### [H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` function making the protocol take too many tokens from users

**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 miscalcualtes 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:**

1. Price of 1 weth is 1 poolToken
2. A user want to get `100_000` weth by swapping poolToken
3. The user use the `swapExactOutput`
4. Instead of sending `100_300` poolToken (0.3% fee), the user send `1_003_009` poolToken to swap `100_000` worth of weth

Proof Of Code

add this to `TSwapPool.t.sol`:

```
    function testGetInputAmountBasedOnOutputHighFee() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
```

```
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        vm.startPrank(user);
        uint256 initialUserWethBalance = weth.balanceOf(user);
        uint256 initialUserPoolTokenBalance = poolToken.balanceOf(
            address(user)
        );

        uint256 swapAmount = 100_000;

        poolToken.approve(address(pool), type(uint256).max);
        pool.swapExactOutput(
            poolToken,
            weth,
            swapAmount,
            uint64(block.timestamp + 1)
        );
        uint256 actualUserWethBalanceAfterSwap = weth.balanceOf(address
            (user));
        uint256 amountWethGotFromSwap = actualUserWethBalanceAfterSwap
            -
            initialUserWethBalance;

        uint256 actualUserPoolTokenBalanceAfterSwap = poolToken.
            balanceOf(
            address(user)
        );

        uint256 amountPoolTokenSpendForSwap =
            initialUserPoolTokenBalance -
            actualUserPoolTokenBalanceAfterSwap;

        console.log(
            "[*] how much user gain weth from swap: ",
            amountWethGotFromSwap
        );
        console.log(
            "[*] how much user paid poolToken for swap: ",
            amountPoolTokenSpendForSwap
        );
    }
```

you will get input from console log:

```
Logs:
  [*] how much user gain weth from swap:  100_000
  [*] how much user paid poolToken for swap:  1_003_009
```

the result above shown that user paid a tremendous amount (1_003_009) of poolToken just to swap

100_000 weth. as we know that the protocol intended to take 0.3% fee every swap, the normal amount should be 100_300 poolToken used with the current liquidity setUp on the test suite.

**Recommended Mitigation:**

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
        return
-           ((inputReserves * outputAmount) * 10000) /
+           ((inputReserves * outputAmount) * 1000) /
            ((outputReserves - outputAmount) * 997);
    }
```

### [H-2] Missing 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 much worse swap.

**Proof of Concept:**

1. The price of weth 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.

**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.

```
    function swapExactOutput(
        IERC20 inputToken,
+       uint256 maxInputAmount,
.
.
.
        inputAmount = getInputAmountBasedOnOutput(
            outputAmount,
            inputReserves,
            outputReserves
        );
+       if(inputAmount > maxInputAmount) {
+           revert();
+       }
```

### [H-3] `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to receive incorrect amount of tokens

**Description:** The `sellPoolTokens` function is intended to allow users to easily sell pool tokens and receive weth in exchange. User indicate how many pool tokens they are willing to sel 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(
+           swapExactOutput(
                i_poolToken,
-               i_wethToken,
+               poolTokenAmount,
+               i_wethToken,
-               poolTokenAmount,
+               minWethToReceive,
                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 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 token
- `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 overtime 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 can 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 extra incentive 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 = 1e16;
        int256 startingY = int256(weth.balanceOf(address(pool)));
        int256 expectedDeltaY = int256(-1) * int256(outputWeth);

        vm.startPrank(user);
        poolToken.approve(address(pool), type(uint256).max);
        for (uint256 i = 0; i < 10; ++i) {
            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. If you want to keep this in, we should account for the change in the `x * y = k` protocol. 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)
    ;
-   }
```

## Medium

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

**Description:** The `deposit` function accepts a deadline parameter, which according to the documentation "deadline The deadline for the transaction to be completed by". However, this parameter is never used. As a consequence, operation 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.

**Recommended Mitigation:** Consider to making the 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::LiquidityAdded` event has parameter out of order causing event to emit incorrect information

**Description:** WHen the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTran` function, it logs values in an incorrect order. The `poolTokensToDeposit` value should go second.

**Impact:** Event emission is incorrect, leading to off-chain function 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.

**Recommended Mitigation:**

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

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

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

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

## Informational

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

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

### [I-2] `PoolFactory::constructor` and `TSwapPool::constructor` lacking zero address check

PoolFactory:

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

TSwapPool:

```
    constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
+       if(poolToken == address(0) || wethToken == address(0)) {
+           revert();
+       }
        i_wethToken = IERC20(wethToken);
        i_poolToken = IERC20(poolToken);
    }
```

### [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] 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.

- Found in src/PoolFactory.sol Line: 35

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

- Found in src/TSwapPool.sol Line: 52

```
    event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
1       event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
1       event Swap(
```