Protocol Audit Report

Version 1.0

Cyfrin.io

September 22, 2025

Protocol Audit Report

Cyfrin.io

September 22, 2025

Prepared by: YoYiL Lead Auditors: - YoYiL

Table of Contents

  - Medium
    * [M-1] Missing deadline enforcement in `TSwapPool::deposit` allows execution past intended expiry
  - Low
    * [L-1] Parameter ordering issue in `TSwapPool::LiquidityAdded` event
    * [L-2] `TSwapPool::swapExactInput` returns an unintended default value
  - Informational/Non-Crits
    * [I-1] Unused error `PoolFactory::PoolFactory__PoolDoesNotExist` should be eliminated
    * [I-2] Missing zero-address validation
    * [I-3] `PoolFactory::createPool` should rely on `.symbol()` instead of `.name()`
    * [I-4] Events lack `indexed` parameters

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 YoYiL 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

Link to Repo to be audited	
Commit hash	f426f57731208727addc20adb72cb7f5bf29dc03
Number of Contracts in Scope	2
Total SLOC for contracts in scope	374
Complexity Score	174
How many external protocols does the code interact with	Many ERC20s
Overall test coverage for code under audit	40.91%

Scope

```
src/PoolFactory.sol
src/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

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Info	4
Gas	0
Total	11

Findings

High

[H-1] Fee miscalculation in `TSwapPool::getInputAmountBasedOnOutput` overcharges users

Description: The `getInputAmountBasedOnOutput` routine is supposed to compute how many input tokens are required for a desired output. The formula currently multiplies by `10_000` instead of `1_000` when applying the fee-adjusted scaling, inflating the computed input amount.

Impact: Users pay more input tokens than the fee model intends, effectively extracting excessive value from traders.

Recommended Mitigation:

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
```

```
     {
-        return ((inputReserves * outputAmount) * 10_000) / ((
    outputReserves - outputAmount) * 997);
+        return ((inputReserves * outputAmount) * 1_000) / ((
    outputReserves - outputAmount) * 997);
     }
```

[H-2] Absent slippage ceiling in `TSwapPool::swapExactOutput` risks excessive input consumption

Description: Unlike `swapExactInput` (which uses a `minOutputAmount`), the `swapExactOutput` function offers no `maxInputAmount` guard. If reserve ratios shift before confirmation, the caller may end up spending far more input tokens than anticipated.

Impact: Users can suffer large unfavorable fills if market conditions move sharply (e.g., during volatility or MEV reordering).

Proof of Concept (scenario): 1. 1 WETH initially equals 1,000 USDC. 2. User calls `swapExactOutput` targeting 1 WETH (outputAmount = 1). 3. No limit on input specified. 4. While pending, price moves to 1 WETH = 10,000 USDC. 5. Transaction executes; user pays 10,000 USDC instead of the expected 1,000 USDC.

Recommended Mitigation: Introduce a `maxInputAmount` parameter and revert when exceeded.

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

[H-3] Incorrect function choice in `TSwapPool::sellPoolTokens` causes unintended swap semantics

Description: The `sellPoolTokens` function is intended to let users provide an exact amount of pool tokens and receive WETH. Presently it calls `swapExactOutput`, which targets a desired output amount rather than consuming a fixed input. Since users define the pool token amount they are supplying, `swapExactInput` is the appropriate primitive.

Impact: The executed swap does not align with user intent, leading to incorrect token flow and potentially unexpected pricing outcomes.

Proof of Concept: The function forwards `poolTokenAmount` into a call that interprets it as an output rather than the exact input being sold.

Recommended Mitigation: Switch to `swapExactInput` and add a slippage parameter such as `minWethToReceive`. Also consider introducing a deadline parameter since one is currently absent.

```
    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));
    }
```

[H-4] Reward distribution in `TSwapPool::_swap` breaks constant product invariant (`x * y = k`)

Description: The AMM relies on maintaining the invariant `x * y = k`, where: - x = pool token reserves - y = WETH reserves - k = constant product

A periodic bonus inside `_swap` transfers extra output tokens every time `swap_count` reaches `SWAP_COUNT_MAX`. This unsolicited token emission alters reserves without corresponding input, violating the invariant and enabling strategic draining over repeated swaps.

Problematic segment:

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

Impact: Attackers (or even normal users) can repeatedly trigger the bonus, siphoning value. Over time, reserves diverge from the pricing formula, undermining integrity and fairness.

Proof of Concept (high-level): 1. Execute enough swaps to trigger the bonus repeatedly. 2. Collect the extra `1_000_000_000_000_000_000` tokens multiple times. 3. Reserve balances no longer reflect the expected constant product progression. 4. Funds can be gradually extracted.

(Provided test demonstrates invariant expectations failing when bonus triggers.)

Recommended Mitigation: Eliminate (or redesign) the periodic reward so it does not directly distort reserves. If incentives are desired, isolate them via a separate emissions mechanism or pre-funded reward pool tracked outside the invariant.

```
-         swap_count++;
-         // Fee-on-transfer
-         if (swap_count >= SWAP_COUNT_MAX) {
-             swap_count = 0;
-             outputToken.safeTransfer(msg.sender, 1
    _000_000_000_000_000_000);
-         }
```

If retaining a reward system: - Use accounting that deducts from a dedicated incentive budget. - Avoid modifying pool reserves unpredictably. - Ensure any emissions are transparently reflected in pricing logic or excluded from invariant calculations.

Medium

[M-1] Missing deadline enforcement in `TSwapPool::deposit` allows execution past intended expiry

Description: The `deposit` function includes a `deadline` argument described as the timestamp by which the transaction must finalize, yet this value is never referenced. As a result, liquidity additions can still settle after the user's intended validity window, potentially in unfavorable price conditions.

Impact: Users cannot rely on the deadline parameter to protect them from adverse execution timing (e.g. price movements or MEV-induced delays).

Proof of Concept: The `deadline` variable is accepted but unused in the function body.

Recommended Mitigation:

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint, // LP tokens -> if empty,
            we can pick 100% (100% == 17 tokens)
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

Low

[L-1] Parameter ordering issue in `TSwapPool::LiquidityAdded` event

Description: In the `TSwapPool::_addLiquidityMintAndTransfer` function, the `LiquidityAdded` event arguments are emitted in the wrong sequence. The value for `poolTokensToDeposit` should be the third argument, and `wethToDeposit` should appear as the second argument.

Impact: Off-chain consumers relying on event parameter positions may process incorrect data, potentially causing analytical or indexing discrepancies.

Recommended Mitigation:

```
- emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
+ emit LiquidityAdded(msg.sender, wethToDeposit, poolTokensToDeposit);
```

[L-2] `TSwapPool::swapExactInput` returns an unintended default value

Description: The `swapExactInput` function is meant to return the actual number of output tokens received. Although a named return variable `ouput` (note the misspelling) is declared, it is never assigned nor is there an explicit return statement. As a result, the function always returns zero.

Impact: Callers will consistently receive an incorrect return value (0), which can break integrations or lead to faulty assumptions about swap results.

Recommended Mitigation:

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

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

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

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

Informational/Non-Crits

[I-1] Unused error `PoolFactory::PoolFactory__PoolDoesNotExist` should be eliminated

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

This custom error is declared but never referenced anywhere in the codebase. Unused declarations add noise and should be pruned to keep the contract surface minimal.

[I-2] Missing zero-address validation

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

A defensive check prevents deployment with an invalid WETH token address, mitigating misconfiguration or attack vectors involving the zero address.

[I-3] `PoolFactory::createPool` should rely on `.symbol()` instead of `.name()`

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

Token symbols are shorter, more conventional for derivative naming, and more consistently implemented than full token names, reducing string size and potential inconsistencies.

[I-4] Events lack `indexed` parameters

Adding `indexed` to appropriate event parameters enables efficient off-chain filtering. While each indexed topic increases gas cost slightly, these events are infrequent enough that indexing up to three fields is typically warranted. If an event contains fewer than three parameters, all should normally be indexed.

Occurrences: - src/TSwapPool.sol: line 44 - src/PoolFactory.sol: line 37 - src/TSwapPool.sol: line 46 - src/TSwapPool.sol: line 43

Recommended Mitigation: - Ensure up to three most query-relevant fields per event are marked `indexed`. - For events with one or two fields, index all of them.