TSwap Protocol Audit Report

Version 1.0

Reina Baz

January 7, 2025

TSwap Audit Report

Reina Baz

January 7, 2025

Prepared by: Reina Baz

Table of Contents

    * [H-5] In `tSwapPool::_swap` the extra tokens given to the users after `swap_count` breaks the protocol invariant of `x * y = k`

  - Lows
    * [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order.
    * [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 of `.name()`
    * [I-4] Event is missing `indexed` fields
    * [I-5] Define and use `constant` variables instead of using literals
    * [I-6] **`public`** functions not used internally could 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 Reina 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
- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum
- Tokens:
- Any ERC20 token

Scope

- In 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.

Issues found

Severity	Number of issues found
High	5
Medium	0
Low	2
Info	6
Total	13

Findings

Highs

[H-1] `TSwapPool::deposit` is missing a deadline check causing transactions to complete even after the deadline.

Description The `deposit` function accepts a deadline parameter, which according to the documentation is "@param deadline 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 might be unfavorable.

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

Proof of Concepts 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)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

[H-2] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causing the protocol to take too many tokens from the 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 fees, it scales the amount by 10_000 instead of 1_000.

Impact Protocol takes more fees than expected from the users.

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

[H-3] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially receive way fewer tokens.

Description The `swapExactOutput` function does not contain any sort of slippage protection. This function is similar to what is done in `TSwapPool:swapExactInput`, where the function specifies the `minOutputAmount`, the `swapExactOutput` should specify a `maxInputAmount`.

Impact If market conditions change before the transaction happens, the user could get a much worse swap.

Proof of Concepts

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

[H-4] `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 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, whereas the `swapExactInput` function should be called. Because users specify the exact amount of input not output.

Impact Users will swap the wrong amount of tokens, which is a severe disruption of the protocol's severity.

Proof of Concepts

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(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.

[H-5] In `tSwapPool::_swap` the extra tokens given to the users after `swap_count` breaks the protocol invariant of `x * y = k`

Description The protocol follows the 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 from funds by doing a lot of swaps and collecting the extra incentive given out by the protocol.

More simply put, the protocol's core invariant is broken.

Proof of Concepts 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 untill all the protocol funds are drained.

Proof Of Code

Place the follow test 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));
        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. 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 way we do with fees.

```
- swap_count++;
-         if (swap_count >= SWAP_COUNT_MAX) {
-             swap_count = 0;
```

```
4 -             outputToken.safeTransfer(msg.sender, 1
  _000_000_000_000_000_000);
5 -         }
```

Lows

[L-1] `TSwapPool::LiquidityAdded` event has parameters out of order.

Description When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTrans` function, it logs values in an incorrect order. The `poolTokensToDeposit` value should go in the third parameter position, wheres as the `wethToDeposit` value should go second.

Impact Event emission is incorrect, leading to off-chain functions potentially malfunctioning.

Recommended mitigation

```
1 -         emit LiquidityAdded(msg.sender, poolTokensToDeposit,
  wethToDeposit);
2 +         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 value `output` it is never assigned a value, nor uses an explicit return statement.

Impact The return value will be always be 0, giving incorrect information to the caller.

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
9 -         if (outputAmount < minOutputAmount) {
10 -            revert TSwapPool__OutputTooLow(outputAmount,
  minOutputAmount);
```

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

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

     }
```

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 == 0){
+        revert();
+    }
        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] 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.

4 Found Instances

- Found in src/PoolFactory.sol Line: 35

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

- Found in src/TSwapPool.sol Line: 52

```
1      event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
1      event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
1      event Swap(
```

[I-5] Define and use `constant` variables instead of using literals

If the same constant literal value is used multiple times, create a constant state variable and reference it throughout the contract.

4 Found Instances

- Found in src/TSwapPool.sol Line: 276

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

- Found in src/TSwapPool.sol Line: 295

```
1              ((outputReserves - outputAmount) * 997);
```

- Found in src/TSwapPool.sol Line: 455

```
1                  1e18,
```

- Found in src/TSwapPool.sol Line: 464

```
1                  1e18,
```

[I-6] `public` functions not used internally could be marked `external`

Instead of marking a function as **`public`**, consider marking it as `external` if it is not used internally.

1 Found Instances

- Found in src/TSwapPool.sol Line: 298

```
1      function swapExactInput(
```