TSwapPool Protocol Audit Report

Version 1.0

Rahber Ahmed

March 28, 2025

TSwapPool Audit Report

Rahber Ahmed

March 28,2025

Prepared by: Rahber Ahmed

Table of Contents

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 Rahber Ahmed 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

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda

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

- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum
- Tokens:
  - Any ERC20 token

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

The TSwapPool Audit Report presents a detailed security review of the TSwapPool smart contract and its associated components. The audit was conducted to identify vulnerabilities, assess the security risks, and recommend mitigations for the protocol. The scope of the audit covered PoolFactory.sol and TSwapPool.sol, focusing on Solidity security best practices, protocol logic, and potential attack vectors.

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Informational	5
Total	12

Findings

High

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

Description: the `getInputAmountBasedOnOutput` function is intended to calculate amount of token input based on output amount,Hiwever currently function miscalculates the resulting amount.When calculating the fee,it scales the amount by 10_000 instead of 1_000.

Impact: The protocol takes more fees from user than expected.

Proof Of Concept:

code

```
contract TestIncorrectFeeCalculation {
    TSwapPool public pool;

    constructor() {
        pool = new TSwapPool();
    }

    function testIncorrectCalculation() external pure returns (uint256
       incorrectAmount, uint256 correctAmount) {
        uint256 outputAmount = 100;
        uint256 inputReserves = 10000;
        uint256 outputReserves = 5000;

        // Incorrect calculation due to wrong scaling factor
        incorrectAmount = ((inputReserves * outputAmount) * 10000) / ((
            outputReserves - outputAmount) * 997);

        // Correct calculation using the right scaling factor
        correctAmount = ((inputReserves * outputAmount) * 1000) / ((
            outputReserves - outputAmount) * 997);
    }
}
```

Recommended Mitigation:

```
  function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {   //@audit-info:there are magic numbers
        return
-           ((inputReserves * outputAmount) * 10000) /
+           ((inputReserves * outputAmount) * 10000)

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

[H-2] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially receieve way fewer tokens

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

Impact: If market condition changes before the transaction process,user could get a much worse swap.

Proof Of Concept: 1.Say the price of 1 WETH=1,000 DAI 2.User wants to swap 1,000 DAI for 1 WETH,consider the following transaction through `swapExactOutput` 1. inputToken= DAI outputToken=WETH outputAmount=1 WETH deadline=uint64(block.timestamp) 2.As the transaction is pending in mempool,the market condition chnages and now the price of 1 WETH=10,000 DAI trnsaction completes but the user will have paid 10,000 DAI for 1 WETH as opposed to 1,000 DAI expected.

Recommended Mitigation: We should include `maxInputAmount` so the user only has to spend up to specific amount and can predict how much they will spend on the protocol.

```
function swapExactOutput(
        IERC20 inputToken,
+       uint256 maxInputAmount,
.
.
.
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));
+       if(inputAmount > maxInputAmount){
+           revert();
+       }
```

[H-3] TSwapPool::sellPoolToken mismatches input and output tokens causing user to receive incorrect amount of tokens.

Description: `sellPoolToken` allows users to sell pool tokens and receive WETH in return,user indicates how many tokens they are willing to exchange in `poolTokenAmount`,however the function miscalculates the swappped amount.

This is due to the fact that `swapExactOuput` function is being called,whereas the function `swapExactInput` should be called,because user specify the input tokens not the output.

Impact: Users will swap wrong tokens which is severe disruption of protocol's functionality.

Recommended Mitigation: Consider using `swapExactInput` instead of `swapExactOutput`.Note that this would require `sellPoolToken` to accept parameter `minWETHToReceive` to be passed to `swapExactInput`

```
    function sellPoolTokens(
        uint256 poolTokenAmount,
+       uint256 minETHToReceive
     ) external returns (uint256 wethAmount) {
        return
-      swapExactOutput(i_poolToken,i_wethToken,poolTokenAmount,uint64(
    block.timestamp));
+      swapExactInput(i_poolToken,i_wethToken,poolTokenAmount,
    minWETHToReceive,uint64(block.timestamp));
    }
```

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

[H-4] In TSwapPool::_swap extra tokens given to user after every swapCount breaks the invariant of x * y = k

Description: The protocol follows the strict invariant of `x * y = k` where

- `x`: the balance of pool token
- `y`: the balance of WETH
- `k`: the constant product of the two

This means whenever the balance of the two tokens changes in the protocol the ratio of the two must remain constant,however when user is given extra tokens after every `swapCount` this will make protocols funds drain over time.

the following block of code is responsible for this issue

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

Impact: The user maliciously could drain the protocol of funds by doing lot of swaps and collecting extra incentive given out by the protocol.

Proof Of Concept:

1. A user swaps 10 times and collects extra incentive of `1_000_000_000_000_000_000`
2. A user keeps repeating this untill all the fund of the protocol is drained.

put following test into your `TSwapPool.t.sol`

Proof Of Code

```
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` being given for users for swapping 10 times,or if you want to incentivize users for this then you should account for it in `x * y = k`.Or we should set aside tokens in a same way we do for 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 making user to make transaction even after deadline have passed.

Description: `TSwapPool::deposit` accepts deadline parameter which according to the documentation is ''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 execute at unexpected times,specially in market conditions where deposit rate is unfavourable.

Impact: Transactions could be sent when market conditions are unfavourable even after adding dealine parameter.

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

Recommended Mitigation: consider adding `deadline` check.

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline //@audit-High: deadline is not being used
    )
        external
+       revertIfDeadlinePassed(uint64 deadline)
```

```
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
        if (wethToDeposit < MINIMUM_WETH_LIQUIDITY) {
            revert TSwapPool__WethDepositAmountTooLow(
                MINIMUM_WETH_LIQUIDITY,
                wethToDeposit
            );
        }
```

Low

[L-1] `TSwapPool::LiquidityAdded` parameters are out of order causing information to emit incorrectly.

Description: When the `LiquidityAdded` is emitted in the `TSwapPool::_addLiquidityMintAndTransfer` function,it logs parameters of it in incorrect order `poolTokensToDeposit` should go in the third position,whereas `wethToDeposit` should go in the second position.

Impact: event emission is incorrect leading off-chain functions to potentially malfunctioning.

Recommended Mitigation:

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

[L-2] `TSwapPool::swapExactInput` returns default value of `uint256 output` resulting in incorrect return value.

Description: The function `swapExactInput` is expected to return the amount of token bought by caller but the `uint256 output` is never assigned any value,resulting in the return of default value of type uint256 which is zero.

Impact: The return value will always be 0 giving caller the wrong information.

Proof Of Concept:

Recommended Mitigation:

```
function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
```

```
    )
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 output)
    {

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





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

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

Informational

[I-1] `PoolFactory::constructor` does not have zero a address check.

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

[I-2] `PoolFactory:: error PoolFactory__PoolDoesNotExist(address tokenAddress)` is never used and hence should be removed

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

[I-3] `PoolFactory::CreatePool` this function has used `.name()` instead of using `.symbol()` hence should be changed.

```
- 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

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

- Found in src/TSwapPool.sol Line: 52

  ```
      event LiquidityAdded(
  ```

- Found in src/TSwapPool.sol Line: 57

  ```
      event LiquidityRemoved(
  ```

- Found in src/TSwapPool.sol Line: 62

  ```
      event Swap(
  ```

[I-5] `TSwapPool::deposit` function does not follow CEI(checks,effects and interactions) and hence `liquidityTokensToMint = wethToDeposit` should be moved up before `_addLiquidityMintAndTransfer(wethToDeposit,maximumPoolTokensToDeposit,wethToDeposit);`

```
+   liquidityTokensToMint = wethToDeposit;
    _addLiquidityMintAndTransfer(
                wethToDeposit,
                poolTokensToDeposit,
```

```
                    liquidityTokensToMint
                );
            } else {


                _addLiquidityMintAndTransfer(
                    wethToDeposit,
                    maximumPoolTokensToDeposit,
                    wethToDeposit
                );
-    liquidityTokensToMint = wethToDeposit;
```