TSwap Protocol Audit Report

Prepared by: Hamid

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

Ola Hamid 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 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
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

I 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

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda
- In Scope:

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

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

```
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.

Issues found

Severity	Number of issues found
High	5
Medium	1
Low	2
Info	11
Total	19

Findings

HIGH

[H-1] Incorect Fee calculated in the `TSwapPool:getInputAmountBasedOnOutput` cause the protocol to take too many fee token fee from the user resulting in lost Fees.

Description: The `TSwapPool:getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens a user should deposit given an amount of token of output tokens, however, the function currently miscalculate the resulting amount ,When calculating the fee, it scales the amount by 10_000 instead of 1_000.

Impact: Protocol takes more fee than user

PROOF OF CONCEPTS:

```
    function testGetInputAmountBasedOnOutput() public {
        // start the Lp
        vm.startPrank(liquidityProvider);
        // Approve the wETH
        weth.approve(address(pool), 100e18);
        // Approve poolToken
        poolToken.approve(address(pool), 100e18);
        // ACTUAL deposit the wETH and poolToken
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));

        console.log ("this is the total amount of WETH in pool ADDRESS",
weth.balanceOf(address(pool)));
        console.log ("this is the total amount of WETH in pool ADDRESS",
poolToken.balanceOf(address(pool)));

        uint l_OutputAmount = 5e18;
        uint l_InputReserve = weth.balanceOf(address(pool));
        uint l_OutputReserve = poolToken.balanceOf(address(pool));
        uint Precision1000 = 1000;
        uint precision997 = 997;

        uint num = ((l_InputReserve * l_OutputAmount) * Precision1000);
        uint dem = ((l_OutputReserve - l_OutputAmount) * precision997);
        uint actualOutPut = num/dem;
        uint remainActualBalanceInPool = (l_InputReserve - actualOutPut);

        console.log("this is the output of the actual outPut:",
actualOutPut);
        console.log("this is the remaining amount in pool after actual
swap:", remainActualBalanceInPool);

        vm.startPrank(user);
        poolToken.approve(address(pool),9e18 );
        pool.getInputAmountBasedOnOutput(l_OutputAmount, l_InputReserve,
l_OutputReserve);
        uint expectedOutput =
pool.getInputAmountBasedOnOutput(l_OutputAmount, l_InputReserve,
l_OutputReserve);
        vm.stopPrank();

        assert(actualOutPut != expectedOutput);
        // vm.expectRevert();
        // pool.getInputAmountBasedOnOutput(l_OutputAmount,
l_InputReserve, l_OutputReserve);
    }
```

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] `TSwapPool.sol:deposit` function is missing a deadline param being unused, causing the the transaction to be completed even after the deadline has passed.

Description: The `deposit` Function accepts a deadline param according to the contract and documentation "the deadline of the transaction to be completed, by", however this param is never used, as a consequence of the transaction, operators 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 unfavourable to deposit, even when adding a deadline parameter

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

Recommended Mitigation: Consider making the following change to the Function

```
  function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        // @audit unused deadline parameters.
        // impact high, severly disrupting the protocol
        // likelihood high
        uint64 deadline
    )
        external
        revertIfZero(wethToDeposit)
+       revertIfDeadLinePassed(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {}
```

[H-3] Lack 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 transaciton processes, the user could get a much worse swap.

Proof of Concept:

1. The price of 1 WETH right now is 1,000 USDC
2. User inputs aswapExactOutput 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 the incorrect amount of tokens

Description: The `sellPoolTokens` function is intended to allow users to easily 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 miscalculaes 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 protcol function ability

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(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. (MEV later)

[H-5] in `TSwapPool.sol::_swap` the extra tokens given to the users after every `swapCounts` breaks the protocol invariant of `x * y = k`

Description: The protocol follow a strict invariants of `x * y = k`, where

- `x`: the balance of the pool token
- `y`: the balance of the weth token
- `k`: the constan product of the two balance tokens

this means that when ever the balances change in the protocol, the raio between the two amount should remain constant, hence the `k`, how ever this is broken due to the extra insentive in the swap function in the swap function. meaning that over time the rotocol funds will be drained

The following block of code is responsible for the issue

```
        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);
        }
```

Impact: A useer could use malicioous draian the protocol by the doing alot of swaps and collecting the extra incentive given out by the protocol

Proof of Concept:

1. a user swas 10 times and collect the extra intentive of `1_000_000_000_000_000` tokens
2. the user continues until all funds are drained

place the following code in your unit tests

```
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));
        vm.stopPrank();


        int256 startingX = int256(weth.balanceOf(address(pool)));
        int256 expectingDeltaX = int256(-1) * int256(outputWETH);

        uint256 endingX = weth.balanceOf(address(pool));
        int256 actualDeltaX = int256(endingX) - int256(startingX);
        assertEq(actualDeltaX, expectingDeltaX);
    }
```

Recommended Mitigation: remove the extra intentive, if you wil like to keep this in, say it that you want to change the `x * y = z` protocol invariant. OR remove this

```
-         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);
-        }
```

MEDIUM

[M-1] Rebase, fee-on-transfer, and ERC777 tokens break protocol invariant

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

LOW

[L-1] `TSwapPool.sol:liquidityAdded` event has a param out of order causing event to emit incorrect information

Description: when the `TSwapPool.sol:liquidityAdded` emit out of the `_addLiquidityMintAndTransfer` function, it return wrong information due to the mismatch placement of the parameters in the events

Impact: Evenr emmission is incorrenct, leading to off-chain functions to malfunction

Recommended Mitigation:

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

[L-2] The uint256 `Output` param in the `TSwapPool.sol:swapExactInput` function is unused, Function returns Zero.

Description:

Impact:

Proof of Concept:

Recommended Mitigation:

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

Description: error message isn`t used at all and should be removed, causing a less cheaner code.

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

[I-2] `PoollFactory.sol:constructor` lacking zero address checks

Description: The `PoollFactory.sol:constructor:wETH` param in the constructor doesn't have a zero address check

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

[I-3] `PoolFactory:createPool` function should use `.symbol` instead of `.name`

Description: The `PoolFactory:createPool` function, has a logic to create a pool for tokens, there is a line that handles the symbols of the token to be created and it returns `.name` function instead of returning a `.symbol` function.

```
- 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: 36

  ```
      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] The `TSwapPool.sol:constructor` does not have a zero address checker.

Description: In The `TSwapPool.sol:constructor`, the `poolToken` and the `wethToken` params does not have a zero address checker.

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

[I-6] Deposit function not following CEI.

```
+           liquidityTokensToMint = wethToDeposit;
            _addLiquidityMintAndTransfer(
                wethToDeposit,
                maximumPoolTokensToDeposit,
                wethToDeposit
            );
            // @auditDone it will be better if this was before the add
liquidinty tranfer call to follow CEI
-            liquidityTokensToMint = wethToDeposit;
        }
```

[I-7] NO Magic numbers

Description: use constants and immutable when you have big numbers, instead of hard coding into function.

```
+ uint private constant Precision997 = 997;
//----code continuation----//
+ uint256 inputAmountMinusFee = inputAmount * Precision997;
- uint256 inputAmountMinusFee = inputAmount * 997;
```

```
-            ((inputReserves * outputAmount) * 10000) /
-            ((outputReserves - outputAmount) * 997);
+            ((inputReserves * outputAmount) * PRECISON10000) /
+            ((outputReserves - outputAmount) * PRECISION997);
```

I-8: `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

```
    function swapExactInput(
```

I-9: 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

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

- Found in src/TSwapPool.sol Line: 295

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

- Found in src/TSwapPool.sol Line: 454

```
                1e18,
```

- Found in src/TSwapPool.sol Line: 463

```
                1e18,
```

L-2: 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

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

- Found in src/TSwapPool.sol Line: 295

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

- Found in src/TSwapPool.sol Line: 454

```
                1e18,
```

- Found in src/TSwapPool.sol Line: 463

```
                1e18,
```

GAS

[G-1] Waste of Gas from local var, next line should be commented out.

Description: The line below in in the `deposit` function is not being used and should be commented out.

```
- uint256 poolTokenReserves = i_poolToken.balanceOf(address(this));
```