TSwap Protocol Audit Report

Version 1.0

mafa

October 28, 2025

Protocol Audit Report

mafa

Oct 28, 2025

Prepared by: mafa

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.

Disclaimer

The mafa 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: 1ec3c30253423eb4199827f59cf564cc575b46db ## Scope

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

Role

- 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	2
Low	2
Info	4
Total	12

Findings

High

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

Description: The `getInputAmountBasedOnOutput` 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 fee, it scales the anout by 10_000 instead of 1_000

Impact: Protocol takes more fees than expected from users.

Recommended Mitigation:

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
```

```
4           uint256 outputReserves
5       )
6           public
7           pure
8           revertIfZero(outputAmount)
9           revertIfZero(outputReserves)
10          returns (uint256 inputAmount)
11      {
12  -       return ((inputReserves * outputAmount) * 10000) / ((
        outputReserves - outputAmount) * 997);
13  +       return ((inputReserves * outputAmount) * 1000) / ((
        outputReserves - outputAmount) * 997);
14      }
```

[H-2] Lack of 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 a much worse swap.

Proof of Concept: 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. The function is pending in the mempool, the markt changes! And the price move to 1 weth = 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 amout, and can pridict how much they will spend on the protocol.

```
1       function swapExactOutput(
2           IERC20 inputToken,
3           IERC20 outputToken,
4           uint256 outputAmount,
5   +       uint256 maxInputAmount
6   .
7   .
8   .
9       {
10          uint256 inputReserves = inputToken.balanceOf(address(this));
11          uint256 outputReserves = outputToken.balanceOf(address(this));
12
```

```
13          inputAmount = getInputAmountBasedOnOutput(outputAmount,
                inputReserves, outputReserves);
14
15  +       if (inputAmount > maxInputAmount) {
16  +           revert();
17  +       }
18          _swap(inputToken, inputAmount, outputToken, outputAmount);
19      }
```

[H-3] `TSwapPool::sellPoolTokens` mismatching input and output tokens causing users to receive the incorret 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 protocol functionality

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`)

```
1       function sellPoolTokens(
2           uint256 poolTokenAmount
3   +       uint256 minWethToReceive)
4       external returns (uint256 wethAmount) {
5   -       return swapExactOutput(i_poolToken, i_wethToken, poolTokenAmount
        , uint64(block.timestamp));
6   +       return swapExactInput(i_poolToken, poolTokenAmount, i_wethToken
        , minWethToReceive, uint64(block.timestamp));
7       }
```

Additionally, it might be wise to add a deadline to the function, as there is currently no deadline

[H-4] In `TSwapPool::_swap` function the extra tokens given to users after every 10 `swapCount`breaks the protocol invariant of `x*y=K`

Description: The protocol follow s a strict invariant of `x*y=K`. Where: - `x`: The balance of the pool token - `y`: The balance of the WETH - `K`: The constant prodect of the two balance

This means, that whenever the balance change in the protocol, the ratio between the two amounts should remain constant. hence the `K`. However, this is broken due to teh extra incentive in the `_swap` function. Meaning that over time the protocol funfs will be drained.

The follow 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 of funds by doing a lot of swaps and collecting the extra incentive given out by the protocol. The invariant is broken

Proof of Concept: 1. A user swaps 10 times, and collect 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 following into `TSwap.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);
        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));
```

```
17          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
18          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
19          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
20          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
21
22          int256 startingY= int256(weth.balanceOf(address(pool)));
23          int256 expectedDeltaY= int256(outputWeth)*int256(-1);
24          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
25          vm.stopPrank();
26
27          uint256 endingY=weth.balanceOf(address(pool));
28          int256 actualDeltaY=int256(endingY)-int256(startingY);
29          assert(actualDeltaY == expectedDeltaY);
30
31      }
```

Recommended Mitigation: Remove the extra incentive machanism. 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

```
1  -        swap_count++;
2  -        if (swap_count >= SWAP_COUNT_MAX) {
3  -            swap_count = 0;
4  -            outputToken.safeTransfer(msg.sender, 1
       _000_000_000_000_000_000);
5  -        }
```

Medium

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

Description: The `deposite` function accepts a dealine parameter, which according to the dicumentation 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 be executed at unexpected times, in market conditions where the deposite rate is unfavorable.

Impact: Transaction could be sent when market conditions are unfavorable to deposite, even when adding a deadline parameter

Proof of Concept: The `dealine` parameter is unused

Recommended Mitigation:

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint, //LP Token to mint, if
            empty, can decide
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(uint64 deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

[M-2] Rebase, fee-on-transfer, and ERC777 tokens break protocol invariant

Description: Certain types of tokens, such as Rebase tokens, fee-on-transfer tokens, and ERC777 tokens, can break the core invariant of automated market makers (AMMs) like the x * y = k formula. Rebase tokens automatically adjust total supply across all holders, which can cause the pool's expected balances to diverge from actual on-chain balances. Fee-on-transfer tokens deduct a percentage on each transfer, so the pool may receive fewer tokens than expected

Impact: Liquidity pool imbalances can occur when pool reserves no longer match the expected invariant, creating a risk that attackers could exploit the discrepancy to manipulate prices and drain tokens.

Recommended Mitigation: Avoid adding Rebase, fee-on-transfer, or ERC777 tokens to AMM pools unless explicitly supported.Implement checks that account for actual token amounts received versus expected amounts.

Low

[L-1] `TSwapPool::LiquidityAdded` event has parameter out of order causing event to emit incorrect information

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

Impact: Event emission is incorrect, leading to off-chain functions 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 value returns `output`it is never assigned a value, nor uses an explict return statement.

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

Recommended Mitigation:

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

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

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

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

Information

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

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

[I-2] Lacking zero address checks

`PoolFactory::constructor`

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

`TSwapPool::constructor`

```
+       if(address(wethToken)==address(0) || address(poolToken)==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] `TSwapPool::getOutputAmountBasedOnInput` function should not use magic number

```
        uint256 inputAmountMinusFee = inputAmount * 997;
        uint256 numerator = inputAmountMinusFee * outputReserves;
        uint256 denominator = (inputReserves * 1000) + inputAmountMinusFee;
```

Instead , should use:

```
+       uint256 constant SWAP_FEE_NUMERATOR = 997;
+       uint256 constant SWAP_FEE_DENOMINATOR = 1000;

        uint256 inputAmountMinusFee = inputAmount * SWAP_FEE_NUMERATOR;
        uint256 numerator = inputAmountMinusFee * outputReserves;
        uint256 denominator = (inputReserves * SWAP_FEE_DENOMINATOR) + inputAmountMinusFee;
```