Protocol Audit Report

Version 1.0

Miriam Shaka

April 30, 2025

Protocol Audit Report

Miriam Shaka

April 30, 2025

Table of Contents

Protocol Summary

TSwap

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

TSwap Pools

The protocol starts as simply a `PoolFactory` contract. This contract is used to create new "pools" of tokens. It helps make sure every pool token uses the correct logic. But all the magic is in each `TSwapPool` contract.

You can think of each `TSwapPool` contract as it's own exchange between exactly 2 assets. Any ERC20 and the WETH token. These pools allow users to permissionlessly swap between an ERC20 that has a pool and WETH. Once enough pools are created, users can easily "hop" between supported ERC20s.

Liquidity Providers

In order for the system to work, users have to provide liquidity, aka, "add tokens into the pool".

Core Invariant

The pool maintains a mathematical invariant: `x * y = k`, where:

x = Token X balance

y = Token Y (WETH) balance

The protocol depends on adherence to this invariant for correct swap behavior.

Disclaimer

This report is provided on a best-effort basis. While significant effort was made to uncover vulnerabilities, this audit does not guarantee the complete absence of issues. The audit focuses on the Solidity smart contracts and does not constitute an endorsement of the business model.

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

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.

Executive Summary

This audit reviewed the TSwap protocol's core contracts: PoolFactory and TSwapPool. A total of 16 issues were identified, ranging from critical fee miscalculations to the absence of slippage controls. Most issues are solvable with minor code changes

Issues found

Severity	Number of issues found
High	4
Medium	2
Low	3
Info	7
Gas	0
Total	16

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocal to take too much tokens from users hence lost fees

Description: `getInputAmountBasedOnOutput` calculates the intrended amount of tokens a user should deposit given an amount of tokens of output tokens. Hoever this is miscalculated bty the function. When calculating the fees it scales the amount by 10_000 instead of 1_000

Impact: Protocal takes notre fees than expected

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);
```

```
14      }
```

[H-2] No slippage protection in `TSwapPool::swapExactOutput` function causing users to potentially

Description: `swapExactOutput` function lacks the slippage protection unlike to `TSwapPool::swapExactInput` which has specified a `maxInputAmount`

Impact: If market conditions change before th e 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
5. The function does not offer a maxInput amount
6. 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
7. 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.

```
1      function swapExactOutput(
2          IERC20 inputToken,
3  +       uint256 maxInputAmount,
4  .
5  .
6  .
7          inputAmount = getInputAmountBasedOnOutput(outputAmount,
               inputReserves, outputReserves);
8  +       if(inputAmount > maxInputAmount){
9  +           revert();
10 +       }
11         _swap(inputToken, inputAmount, outputToken, outputAmount);
```

[H-3] `TSwapPool::sellPoolTokens` returns the swapExactOutput out of order causing users to receive the incorrect amount of tokens

Description: The `sellPoolTokens` is meant to allow users to easily sell pool tokens and receive WEETH in exchange. Users indicate how many pool tokens they're willing to selll in the `poolTokenAmount` parameter . But the function miscalcultes 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.

Proof of Concept:

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

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));
    }
```

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

Description: Description: The protocol follows 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 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.

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

Proof of Concept: 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 until all the protocol funds are drained

```
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);

        for(int i = 0; i<9; i++){
            pool.swapExactOutput(poolToken, weth, outputWeth, uint64(
                block.timestamp));
        }

        int256 startingX = int256(weth.balanceOf(address(pool)));
        int256 expectedDeltaX = int256(outputWeth) * -1;

        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
        vm.stopPrank();

        int256 endingX = int256(weth.balanceOf(address(pool)));
        int256 actualDeltaX = int256(endingX) - int256(startingX);

        assertEq(actualDeltaX, expectedDeltaX);
    }
```

Recommended Mitigation: 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++;
-         // 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] `TSwapPool::deposit` is msising deadline check causing transactions to complete even after the deadline

Description: The `deposit` function accepts a deadline parameter, which according to the documentation is the deadline The deadline for the transaction to be completed by. But this parameter is never used which can lead adding liquidity to the pool can be called at unexpected times even when market condition is not favourable

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

Proof of Concept: `deadline` parameter is not used

Recommended Mitigation: Consider making changes to this function

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {...}
```

[M-2] Rebase, fee-on-transfer, and ERC-777 tokens break protocol invariant

Description:

Impact:

Proof of Concept:

Recommended Mitigation:

Low

[L-1] Public Function Not Used Internally

Description: If a function is marked public but is not used internally, consider marking it as `external`.

Impact:

Proof of Concept:

```
1     function swapExactInput(
```

Recommended Mitigation:

[L-2] Literal Instead of Constant

Description: 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.

Impact:

Proof of Concept:

```
1         uint256 inputAmountMinusFee = inputAmount * 997;
2         ...
3         uint256 denominator = (inputReserves * 1000) +
             inputAmountMinusFee;
```

Recommended Mitigation:

[L-3] Default return value is retunred by `TSwapPool::swapExactInput`

Description: The function is expected to return the actual amount of tokens by the caller. However thre variable `output is declared` it is ever assigned a value nor uses an explicit return statement.

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

Proof of Concept:

Recommended Mitigation:

```
{
    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

[I-1] `TSwapPool::_addLiquidityMintAndTransfer` `LiquidityAdded` emits events out of order

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTrans` function in the wrong order. `poolTokensToDeposit` and `wethToDeposit` should switch position

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

Proof of Concept:

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

Recommended Mitigation:

[I-2] `PoolFactory::error PoolFactory__PoolDoesNotExist(address tokenAddress);` not used and should be removed

[I-3] The `PoolFactory` constructor lacks a zero check

Description:

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

[I-4] `TSwapPool::deadline` not used and should be removed

[I-5] `TSwapPool::deposit` no need to emit `MINIMUM_WETH_LIQUIDITY` as its written in the code and can be easily refered

```
revert TSwapPool__WethDepositAmountTooLow(
    MINIMUM_WETH_LIQUIDITY,
    wethToDeposit
);
```

[I-6] `TSwapPool::deposit::poolTokenReserves` not used and should be removed

[I-7] `TSwapPool::deposit::swapExactOutput` info missing deadline param in nutspec