Protocol Audit Report

Version 1.0

ZophiaWong

October 23, 2025

T-Swap Audit Report

Zephyr Wong

Oct. 15, 2025

Prepared by: Zephyr Wong Lead Auditors: - Zephyr Wong

Table of Contents

Protocol Summary

This protocol 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

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

Scope

- Commit Hash: 1ec3c30253423eb4199827f59cf564cc575b46db
- In Scope:

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

- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum

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

Issues found

Severity	Number of issues found
High	4
Medium	2
Low	2
Info	5
Total	13

Findings

High

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

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

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

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] The absence of slippage protection in the `TSwapPool::swapExactOutput` function has the potential to result in users depositing more tokens to obtain exact output 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. Assuming the price of 1 WETH right now is 1,000 USDC.
2. A user call `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 that users only have to spend up to a specific amount and predict their spending on the protocol.

```
1  function swapExactOutput(
2          IERC20 inputToken,
```

```
3           IERC20 outputToken,
4           uint256 outputAmount,
5   +        uint256 maxInputAmount,
6           uint64 deadline
7       )
8           public
9           revertIfZero(outputAmount)
10          revertIfDeadlinePassed(deadline)
11          returns (uint256 inputAmount)
12      {
13          uint256 inputReserves = inputToken.balanceOf(address(this));
14          uint256 outputReserves = outputToken.balanceOf(address(this));
15
16          inputAmount = getInputAmountBasedOnOutput(outputAmount,
               inputReserves, outputReserves);
17  +        if(inputAmount > maxInputAmount) {
18  +            revert();
19  +        }
20          _swap(inputToken, inputAmount, outputToken, outputAmount);
21      }
```

[H-3] `TSwapPool::sellPoolTokens` mismatches the input and output tokens causing users to receive incorrect amount of tokens

Description: The `sellPoolTokens` function allows users to easily sell pool tokens and receive WETH in exchange. Users indicate how many pool tokens they wish to sell using the `poolTokenAmount` parameter. However, the function currently miscalculate the swapped amount. This is because the `swapExactOutput` is called, whereas the `swapExactInput` functions is the one that should be called. Because users specify exact amount of input tokens(pool tokens), not the output(WETH).

Impact: Users will swap 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 this would also require changing the `sellPoolTokens` to accept an extra parameter `minWethToReceive` to be passed to `swapExactInput`.

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

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

Description: The protocol follows a 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 balances

This means, whatever these two balances change in the protocol, their product should always remain constant, hence `k`. However, this is broken due to the extra incentive in `_swap` function.

The follow block of code is responsible for the issue.

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

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

Proof of Concept:

1. A user swap 10 times, and collect the extra incentive of `1_000_000_000_000_000_000` of tokens
2. That user continues to swap until all the protocol funds are drained.

PoC

Add the following into the `TSwapPool.t.sol`

```
    function testInvariantBroken() public {
        // deposit
        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 (uint256 i = 0; i < 9; i++) {
            pool.swapExactOutput(poolToken, weth, outputWeth, uint64(
                block.timestamp));
```

```
15          }
16          int256 startingX = int256(weth.balanceOf(address(pool)));
17          // 10th swap would get extra incentive
18          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
19
20          vm.stopPrank();
21          int256 endingX = int256(weth.balanceOf(address(pool)));
22          int256 expectedDeltaX = int256(-1) * int256(outputWeth);
23          int256 actualDeltaX = endingX - startingX;
24          int256 extraIncentive = 1_000_000_000_000_000_000;
25          assertEq(extraIncentive, expectedDeltaX - actualDeltaX);
26      }
```

Recommended Mitigation: Remove the extra incentive mechanism. If this is to be retained, it will be necessary to account for the change in the `x * y = k` protocol invariant. As an alternative solution, we could consider setting aside tokens in a manner consistent with our approach to fees.

```
1  -         swap_count++;
2  -         // Fee-on-transfer
3  -         if (swap_count >= SWAP_COUNT_MAX) {
4  -             swap_count = 0;
5  -             outputToken.safeTransfer(msg.sender, 1
       _000_000_000_000_000_000);
6  -         }
```

Medium

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

Description: The `deposit` function accepts a deadline parameter, which according to the documentation is "*The deadline for the transaction to be completed by*". However, the 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 is unfavorable.

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

Proof of Concept: The `deadline` is never used.

Recommended Mitigation:

```
1      function deposit(
2          uint256 wethToDeposit,
3          uint256 minimumLiquidityTokensToMint,
```

```
4           uint256 maximumPoolTokensToDeposit,
5           uint64 deadline
6       )
7           external
8   +        revertIfDeadlinePassed(deadline)
9           revertIfZero(wethToDeposit)
10          returns (uint256 liquidityTokensToMint)
```

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

Description:

Impact:

Proof of Concept:

PoC: rebase token

PoC: fee-on-transfer token

PoC: ERC-777 token

Recommended Mitigation:

- Rebase
  - a
- Fee-on-transfer
- ERC-777 token

Low

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

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTran`, it logs values in an incorrect order. The `poolTokensToDeposit` should go in the third parameter position, whereas `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] The default value returned by the `TSwapPool::swapExactInput` function is incorrect

Description: The `swapExactInput` function is expected to return the actual amount of tokens purchased by the caller. However, while the function does indeed declares the named return value, it does not actually assign a value to it or use an explicit return statement.

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

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] `PoolFactory::PoolFactory__PoolDoesNotExist` is never used and should be removed

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

[I-2] Lacking zero address check in `PoolFactory::constructor`

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

```
6       }
```

[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::constructor` Lacking zero address check - wethToken & poolToken

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

[I-5] `TSwapPool` events should be indexed

```
- event Swap(address indexed swapper, IERC20 tokenIn, uint256 amountTokenIn, IERC20 tokenOut, uint256 amountTokenOut);
+ event Swap(address indexed swapper, IERC20 indexed tokenIn, uint256 amountTokenIn, IERC20 indexed tokenOut, uint256 amountTokenOut);
```