Protocol Audit Report

Version 1.0

khal45

October 6, 2025

Protocol Audit Report

khal45

October 6, 2025

Prepared by: khal45 Lead Auditors:

- khal45

Table of Contents

Protocol Summary

T-Swap is 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

Khal45 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
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

```
./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

Over the course of the security review, khal45 engaged with the t-swap protocol to review it. In this period of time a total of 12 issues were found

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 protocol to take too many tokens from users, 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 output tokens. However, the function currently miscalculates the resulting amount. When calculating the fee, it scales the amount by 10_000 instead of 1_000.

Impact: Protocol takes more fees than expected from users

Proof of Concept: Add these variables to the `TSwapPool.t.sol` test suite

```
    uint256 wethAmountToDeposit = 100e18;
    uint256 poolTokenAmountToDeposit = 100e18;
    uint256 minimumLiquidityTokensToMint = 100e18;
    uint256 maximumPoolTokensToDeposit = 100e18;
    uint256 additionalPoolTokensToMint = 100e18;
```

Then add the following test

```
function test_IncorrectFeeCalculation() public {
        // first deposit tokens to the pool
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), wethAmountToDeposit);
        poolToken.approve(address(pool), poolTokenAmountToDeposit);
        pool.deposit(
            wethAmountToDeposit, minimumLiquidityTokensToMint,
               maximumPoolTokensToDeposit, uint64(block.timestamp)
        );
        vm.stopPrank();
```

```
10
11          vm.startPrank(user);
12          // mint the user some more tokens
13          poolToken.mint(user, additionalPoolTokensToMint);
14          uint256 totalUserBalance = poolToken.balanceOf(user);
15          // approve the pool to spend the user's entire balance
16          poolToken.approve(address(pool), totalUserBalance);
17
18          // swap 1 weth
19          uint256 outputAmount = 1e18;
20          // call `swapExactOutput` which calls `
                getInputAmountBasedOnOutput`
21          pool.swapExactOutput(poolToken, weth, outputAmount, uint64(
                block.timestamp));
22          // input and output reserves are needed to calculate the `
                inputAmountBasedOnWeth`
23          uint256 inputReserves = poolToken.balanceOf(address(pool));
24          uint256 outputReserves = weth.balanceOf(address(pool));
25          // what we should expect the input amount to be, see actual
                function for the intended calculation
26          uint256 expectedInputAmountBasedOnOutput =
27              ((inputReserves * outputAmount) * 1000) / ((outputReserves
                    - outputAmount) * 997);
28          // what the function actually returns due to the miscalculation
29          uint256 actualInputAmountBasedOnOutput =
30              pool.getInputAmountBasedOnOutput(outputAmount,
                    inputReserves, outputReserves);
31          // we see that the actual input amount is significantly greater
                than the expected input amount
32          console.log("expected input amount:",
                expectedInputAmountBasedOnOutput);
33          console.log("actual input amount:",
                actualInputAmountBasedOnOutput);
34          assertGt(actualInputAmountBasedOnOutput,
                expectedInputAmountBasedOnOutput);
35          vm.stopPrank();
36      }
```

Recommended Mitigation:

```
1  function getInputAmountBasedOnOutput(
2          uint256 outputAmount,
3          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) * 10_000) / ((
       outputReserves - outputAmount) * 997);
13 +         return ((inputReserves * outputAmount) * 1_000) / ((
       outputReserves - outputAmount) * 997);
14       }
```

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

Description: The `swapExactOutput` 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 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 = 20 minutes
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 sends 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              IERC20 outputToken,
5              uint256 outputAmount,
6              uint64 deadline
7      )
8  .
9  .
10 .
```

```
11      {
12          inputAmount = getInputAmountBasedOnOutput(outputAmount,
               inputReserves, outputReserves);
13  +       if(inputAmount > maxInputAmount) {
14                revert();
15           }
16           _swap(inputToken, inputAmount, outputToken, outputAmount);
17           }
```

[H-3] The `sellPoolTokens` function miscalculates amount of tokens bought

The `sellPoolTokens` is intended to allow users easily sell pool tokens and receive WETH in exchange. Users indicate how many pool tokens they're willing to sell using the `poolTokenAmount` parameter. However, the function currently miscalculates the swapped amount.

This is due to the fact that the `swapExactOutput` function is called, whereas the `swapExactInput` is the one that should be called. Because users specify the exact amount of input tokens - not output tokens.

Consider changing the implementation to use the `swapExactInput` function. Note that this would also require to change the `sellPoolTokens` function to accept a new parameter (e.g., `minWethToReceive`) to be passed down to `swapExactInput`.

```
1       function sellPoolTokens(
2           uint256 poolTokenAmount
3   +       uint256 minWethToReceive
4       ) external returns (uint256 wethAmount) {
5   -       return swapExactOutput(
6   +       return swapExactInput(
7               i_poolToken,
8               poolTokenAmount,
9               WETH_TOKEN,
10  +           minWethToReceive,
11              uint64(block.timestamp)
12          );
13      }
```

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

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

Impact: A user could maliciously drain the protocol of funds by doing a lot of swaps and collecting the extra incentives given out by the protocol. Most simply put, the protocol's core invariant is broken.

The following block of code is responsible for the issue.

```
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           }
```

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

Proof Of Code

Place the following into TSwapPool.t.sol

```
1     function testInvariantBroken() public {
2         vm.startPrank(liquidityProvider);
3         weth.approve(address(pool), 100e18);
4         poolToken.approve(address(pool), 100e18);
5         pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
6         vm.stopPrank();
7
8         uint256 outputWeth = 1e17;
9
10        vm.startPrank(user);
11        poolToken.approve(address(pool), type(uint256).max);
12        poolToken.mint(user, 100e18);
13        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
14        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
15        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
              timestamp));
16        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          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
22
23          int256 startingY = int256(weth.balanceOf(address(pool)));
24          int256 expectedDeltaY = int256(-1) * int256(outputWeth);
25
26          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
27          vm.stopPrank();
28
29          uint256 endingY = weth.balanceOf(address(pool));
30          int256 actualDeltaY = int256(endingY) - int256(startingY);
31          assertNotEq(actualDeltaY, expectedDeltaY);
32      }
```

Recommended Mitigation: Remove the extra incentive. 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 transactions to complete after the deadline

Description: The deposit function accepts a `deadline` parameter which according to the documentation is `/// @param 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 the market conditions where the deposit is unfavourable.

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 changes to the 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, ERC777, and centralized ERC20s can break core invariant

Description: The core invariant of the protocol is:

`x * y = k`. In practice though, the protocol takes fees and actually increases k. So we need to make sure `x * y = k` before fees are applied.

Impact: A user could maliciously drain the protocol of funds

Recommended Mitigation: Limit the number of tokens that can be swapped in the protocol to tokens that cannot break the protocol invariant

Low

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

Description: When the `LiquidityAdded` added event is emitted in the `TSwapPool::_addLiquidityMintAndTransfer` function, it logs values in an incorrect order. The `poolTokensToDeposit` value should go in the third parameter position, whereas the `wethToDeposit` 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 `TSwapPool::sWapExactInput` function is expected to return the actual amount of tokens bought by the caller. However, while it declares the named return value `output` it is never assigned a value, nor uses 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 (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] The error `PoolFactory::PoolFactory__PoolAlreadyExists` is not used and should be removed

The error `PoolFactory::PoolFactory__PoolAlreadyExists` exists in the `PoolFactory` contract but is not used anywhere in the contract. This is a waste of gas and should be removed.

```
 contract PoolFactory {
-    error PoolFactory__PoolAlreadyExists(address tokenAddress);
    error PoolFactory__PoolDoesNotExist(address tokenAddress);
```

[I-2] Lacks zero address checks

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

[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] 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.

- Found in src/PoolFactory.sol: 1307:61:31
- Found in src/TSwapPool.sol: 1897:108:10
- Found in src/TSwapPool.sol: 2010:110:10
- Found in src/TSwapPool.sol: 2125:116:10