TSwap Protocol Audit Report

Version 1.0

Ske

March 31, 2025

Protocol Audit Report

2ke

March 12, 2025

Prepared by: [2ke] Lead Security Researcher : - @theblack_2ke

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.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	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

The findings described in this document correspond the following commit hash: Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda

Scope

- In Scope: ./src/ #– PoolFactory.sol #– TSwapPool.sol

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

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	4
Medium	1
Low	2
Info	3
Total	10

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAMountBasedOnOutput` causes protocol to take too manytokens deom users, resilting in lost fees.

Description: The `getInputAMountBasedOnOutput` function is intended to calculate the amount of tokens a user should deposit given an amount of token of output token. 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: PoC

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);
    }
```

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

Description: The `swapExactOutput` function does not include any sort of slippage protection. This function is similiar to what is done in `TSwapPool::swapExactInput`, where the function specifies a `minOutputAmount`, The `swapExactOutput` function should specify a `maxInputAmount`

Impact: If market condition changes before the transaction processes, the user could get a much worse swap.

Proof of Concept: PoC 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. 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-3] 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 token and receive WETH in exchange. Users indicates how many pool tokens they're willing to sell in 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 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 functionality.

Proof of Concept: PoC

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: 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.

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 untill all the protocol funds are drained

Proof Of Code

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

```
    function testInvariantBreaks() 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;
        int256 startingY = int256(weth.balanceOf(address(pool)));
        int256 expectedDeltaY = int256(outputWeth);

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

        vm.stopPrank();

        uint256 endingY = weth.balanceOf(address(pool));
        int256 actualDeltaY = int256(endingY) - int256(startingY);

        assertEq(actualDeltaY, expectedDeltaY);
    }
```

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()` function is missing dealine check which can cause transactions to complete even after the deadline

Description: The `deposit()` function accepts a deadline parameter, which according to the doc is "`deadline` The deadline for the transaction to be completed by", However, this paramater is never used. As a consequence, operations that add liquidity to the pool might be executed at unexpected times, in market condition where the deposit rate is unfavorable.

Impact: Transaction could be sent when market conditions are unfavorable to deposit, even when adding a deadline param.

Proof of Concept: The `deadline` parameter is unused.

```
Warning (5667): Unused function parameter. Remove or comment out the variable name to silence this warning.
--> src/TSwapPool.sol:123:9:
    |
123 |         uint64 deadline
    |         ^^^^^^^^^^^^^^^
```

Recommended Mitigation: Consider making the following changes to the functions.

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

Low

[L-1] `TSwapPool::LiquidityAdded` event paramaeter out of order causing emmission of wrong information.

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

Impact: Event emmission 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 token 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.

Proof of Concept: PoC

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] Error variable `PoolFactory__PoolDoesNotExist` not used in `PoolFactor.sol` which is a waste of gas.

Description: In `PoolFactory.sol` an error is being called (`error PoolFactory__PoolDoesNotExist(address tokenAddress);`) but is never being used

Impact: The fact that `PoolFactory.sol::PoolFactory__PoolDoesNotExist(address tokenAddress)` is not used can lead to waste of gas, silent failures, etc…

Recommended Mitigation: Implement it in the `PoolFactory::getPool()` function:

```
    function getPool(address tokenAddress) external view returns (address) {
+       if (s_pools[tokenAddress] == address(0)) {
+           revert PoolFactory__PoolDoesNotExist(tokenAddress);
+       }
        return s_pools[tokenAddress];
    }
```

[I-2] Lacking zero address checks which can lead to contract malfunction

Description: In the `PoolFactory.sol` and `TSwapPool.sol`, inside the constructor their no check of `address(0)`.

Impact: The fact that this contract constructor lacks in zero addresses checks is really bad and can like too several things like: - Contract Misconfiguration : If `i_wethToken` is set to `address(0)`, the contract might not work at all because `WETH` is required for pool creation and swaps - Transaction Failures When Using WETH: If `i_wethToken == address(0)`, this will revert because `address(0)` does not implement `transfer()`.

Proof of Concept:

Recommended Mitigation: Add some checks to it as suggested

```
    constructor(address wethToken) {
+       require(wethToken != address(0), "PoolFactory: WETH token cannot be zero address");
        i_wethToken = wethToken;
    }
```

[I-3] Incorrect Function Usage (name() Instead of symbol()) Leading to Misleading Token Representation

Description: the function `PoolFactory::createPool()` is calling `IERC20(tokenAddress).name()` instead of `IERC20(tokenAddress).symbol()` when setting the symbol of the liquidity token.

Impact: the misleading of this 2 might bring to: - Incorrect Token Symbol – The liquidity pool token's symbol will be a concatenation of "ts" and the full token name instead of its expected short symbol. - User Confusion – Liquidity providers and traders might see unexpected or incorrect token representations, making it harder to identify assets.

Proof of Concept:

Recommended Mitigation:

```
-   string memory liquidityTokenSymbol = string.concat(
-          "ts",
-           IERC20(tokenAddress).name()
-   );

+    string memory liquidityTokenName = string.concat(
+        "T-Swap ",
+        IERC20(tokenAddress).symbol()
+     );
```