TSwap Audit Report

Version 1.0

37H3RN17Y2

October 12, 2024

TSwap Audit Report

37H3RN17Y2

October 12, 2024

Prepared by: [37H3RN17Y2] (https://github.com/37H3RN17Y2) Lead Security Researcher: - 37H3RN17Y2

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. To understand Uniswap, please watch this video: Uniswap Explained

Disclaimer

The 37H3RN17Y2 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
	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 to the following commit hash:

```
e643a8d4c2c802490976b538dd009b351b1c8dda
```

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

I loved auditing this codebase. Patrick is such a wizard at writing intentionally bad code.

Issues found

Severity	Number of issues found
High	4
Medium	2
Low	2
Info	12
Total	20

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput`, causing the protocol to take too much tokens from users

Description: The `getInputAmountBsaedOnOutput` 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 input 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.

Recommended Mitigation:

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

```
6           public
7           pure
8           revertIfZero(outputAmount)
9           revertIfZero(outputReserves)
10          returns (uint256 inputAmount)
11      {
12          return
13  -           ((inputReserves * outputAmount) * 10000) /
14  -           ((outputReserves - outputAmount) * 997);
15  +           ((inputReserves * outputAmount) * 1000) /
16  +           ((outputReserves - outputAmount) * 997);
17      }
```

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

Description: The `swapExactInput` function does not include any sort of slippage protection. This function is similar to what is done `TSwapPool::swapExactInput`, where the function specifies the `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 = whatever 3. The function does not offer a `maxInputAmount` 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.

```
1   +   error TSwapPool__InputTooHigh(uint256 actual, uint256 min);
2   .
3   .
4   .
5       function swapExactOutput(
6           IERC20 inputToken,
7   +       uint256 maxInputAmount,
8           IERC20 outputToken,
9           uint256 outputAmount,
10          uint64 deadline
11      )
12          public
13          revertIfZero(outputAmount)
```

```
14          revertIfDeadlinePassed(deadline)
15          returns (uint256 inputAmount)
16      {
17          uint256 inputReserves = inputToken.balanceOf(address(this));
18          uint256 outputReserves = outputToken.balanceOf(address(this));
19
20          inputAmount = getInputAmountBasedOnOutput(
21              outputAmount,
22              inputReserves,
23              outputReserves
24          );
25
26  +       if (inputAmount > maxInputAmount) {
27  +           revert TSwapPool__InputTooHigh(inputAmount, maxInputAmount)
        ;
28  +       }
29
30          _swap(inputToken, inputAmount, outputToken, outputAmount);
31      }
```

[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 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 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 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 (i.e. `minWethToReceive` to be passed to `swapExactInput`)

```
1       function sellPoolTokens(
2           uint256 poolTokenAmount
3   -   ) external returns (uint256 wethAmount) {
4   +   ) external returns (uint256 wethAmount, uint256 minWethToReceive) {
5           return
6   +           swapExactInput(
```

```
+                i_poolToken,
+                poolTokenAmount,
+                i_wethToken,
+                minWethToReceive,
+                uint64(block.timestamp)
+            );
-            swapExactOutput(
-                i_poolToken,
-                i_wethToken,
-                poolTokenAmount,
-                uint64(block.timestamp)
-            );
    }
```

[H-4] In `TSwapPool::_swap`, the extra tokens given to users after every `swapCount` breaks protocol invariant of `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 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 following 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.

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

Proof of Concept: 1. A user swaps 10, 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`.

```
    function testInvariantBroken() public {
        vm.startPrank(liquidityProvider);
        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           // calculation for swapping poolToken for WETH
9           uint256 outputWeth = 1e17;
10          uint256 poolTokenAmount = pool.getInputAmountBasedOnOutput(
                outputWeth, poolToken.balanceOf(address(pool)), weth.
                balanceOf(address(pool)));
11
12          // user makes 10 swaps, breaks invariant since protocol gives
                extra tokens out as incentive
13          vm.startPrank(user);
14          poolToken.approve(address(pool), type(uint256).max);
15          poolToken.mint(user, 100e18);
16          // make 9 swaps first
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          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
23          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
24          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
25          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
26
27          // 10th swap here
28          // calculate X & Y values before swap
29          // weth taken out of pool, so * -1
30          int256 expectedDeltaY = int256(-1) * int256(outputWeth);
31          int256 expectedDeltaX = int256(poolTokenAmount);
32          int256 startingY = int256(weth.balanceOf(address(pool)));
33          int256 startingX = int256(poolToken.balanceOf(address(pool)));
34
35          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
36          vm.stopPrank();
37
38          // calculate X & Y values after swap
39          int256 endingY = int256(weth.balanceOf(address(pool)));
40          int256 endingX = int256(poolToken.balanceOf(address(pool)));
41          int256 actualDeltaY = endingY - startingY;
```

```
42          int256 actualDeltaX = endingX - startingX;
43
44          assertEq(actualDeltaY, expectedDeltaY);
45          assertEq(actualDeltaX, expectedDeltaX);
46      }
```

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 for 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 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, 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 deposit rate is unfavourable (e.g. MEV frontrunning attacks causing bad price).

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.

```
1      function deposit(
2          uint256 wethToDeposit,
3          uint256 minimumLiquidityTokensToMint, // LP tokens, if empty (
               pool initialize), pick 0, 100% (== 17 tokens)
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: The core invariant of the protocol is: x * y = k. Assets are transferred using with the assumption that the amount transferred, is the amount received. However, rebase tokens and fee-on-transfer tokens do not follow this behaviour. Additionally, ERC-777 tokens allows making reentrancy calls on transferFrom(address from, address to, uint256 amount).

Impact: These tokens have the ability to mess with the protocol's accounting if in use. This is because the transfer functions aren't optimized to handle them, and as a result can lead to situations in which the pools' asset balance will be way less than the amount expected and tracked. Here, the protocol will incur extra costs to cover for these situations. Also, the tokens received from airdrops and positive rebases can be lost forever as there's no way to retrieve them.

Recommended Mitigation: Before any token transfer, both in and out of the protocol, recommend checking the contract balance before and after, and registering the difference as the amount sent. This helps handle fee-on-transfer tokens.

For the rebasing tokens and variable balances, a system of balance tracking and excess token sweep functions can be implemented to periodically skim the excess tokens from the contrascts to prevent them from being lost.

Alternatively, explicitely blocklisting these token types to prevent them from being made pool assets.

Reference: Pashov Reference: Consensys

Low

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

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndrans` 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:

```
1 -   emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit)
      ;
2 +   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 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(
+       uint256 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] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed

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

[I-2] Lacking zero address checks in constructor of `PoolFactory` and `TSwapPool` contracts

`PoolFactory` contract constructor

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

`TSwapPool` contract constructor

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

[I-3] `liquidityTokenSymbol` in `PoolFactory::createPool` function 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` contract events are 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.

4 Found Instances

- Found in src/PoolFactory.sol Line: 35

```
event PoolCreated(address tokenAddress, address poolAddress);
```

- Found in src/TSwapPool.sol Line: 52

```
event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
event Swap(
```

[I-5] `MINIMUM_WETH_LIQUIDITY` in `TSwapPool` contract is a constant and therefore not required to be emitted in events to save gas

```
        error TSwapPool__WethDepositAmountTooLow(
-           uint256 minimumWethDeposit,
            uint256 wethToDeposit
        );
        .
        .
        .
        if (wethToDeposit < MINIMUM_WETH_LIQUIDITY) {
            revert TSwapPool__WethDepositAmountTooLow(
-               MINIMUM_WETH_LIQUIDITY,
                wethToDeposit
            );
        }
```

[I-6] `poolTokenReserves` variable in `TSwapPool` is not used and should be removed to save contract deployment gas cost

```
-   uint256 poolTokenReserves = i_poolToken.balanceOf(address(this));
```

[I-7] Best practice to follow Checks, Effects, Interactions (CEI) pattern in `TSwapPool::deposit` function

```
        } else {
            // This will be the "initial" funding of the protocol. We
                are starting from blank here!
```

```
3               // We just have them send the tokens in, and we mint
                   liquidity tokens based on the weth
4   +           liquidityTokensToMint = wethToDeposit;
5               _addLiquidityMintAndTransfer(
6                   wethToDeposit,
7                   maximumPoolTokensToDeposit,
8                   wethToDeposit
9               );
10  -           liquidityTokensToMint = wethToDeposit;
11          }
```

[I-8] Magic numbers shoud be avoided in `TSwapPool::getOutputAmountBasedOnInput` and `TSwapPool::getInputAmountBasedOnOuput`. Constant variables should be used instead

```
1   +   uint256 private constant SWAP_FEE = 997;
2   +   uint256 private constant SWAP_FEE_PRECISION = 1000;
3   .
4   .
5   .
6   -       uint256 inputAmountMinusFee = inputAmount * 997;
7   +       uint256 inputAmountMinusFee = inputAmount * SWAP_FEE;
8           uint256 numerator = inputAmountMinusFee * outputReserves;
9   -       uint256 denominator = (inputReserves * 1000) +
        inputAmountMinusFee;
10  +       uint256 denominator = (inputReserves * SWAP_FEE_PRECISION) +
        inputAmountMinusFee;
11  .
12  .
13  .
14  -           ((inputReserves * outputAmount) * 10000) /
15  -           ((outputReserves - outputAmount) * 997);
16  +           ((inputReserves * outputAmount) * SWAP_FEE_PRECISION) /
17  +           ((outputReserves - outputAmount) * SWAP_FEE);
```

[I-9] `TSwapPool::swapExactInput` does not have natspec

`TSwapPool::swapExactInput` is an important function and detailed natspec should be provided.

[I-10] The visibility of `TSwapPool::swapExactInput` should be external as it is not used internally in any other functions of the contract

[I-11] `TSwapPool::swapExactOutput` does not have natspec for the parameter `deadline`

[I-12] The visibility of `TSwapPool::totalLiquidityTokenSupply` should be external as it is not used internally in any other functions of the contract