T-Swap Audit Report

Version 1.0

scooljinstitute.com

February 4, 2024

T-Swap Audit Report

Scoolj Institutte

Feburary 04, 2024

Prepared by: Scoolj Institute Lead Auditors: - Scoolj, Oluwajuwonlo Joseph - Patrick Collins

Table of Contents

    * [H-5] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` breaks the protocol invariant of `x * y = k`

- Low
  - [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order
  - [L-2] Default value returned by `TSwapPool::swapExactInput` result in incorrect value return given
- Informational
  - [I-1] : `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed
  - [I-2] Lacking zero address checks
  - [I-3]`PoolFactory::createPool` should use `.symbol()` instead of `.name()`
  - [I-4] Event is missing `indexed` fields

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 Scoolj Institute 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
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

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

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

Actors / 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 is an interesting protocol but required security review as recommended in this report.

Issues found

Severity	Number of issues Found
High	4
Medium	0
Low	2
Info	4
Gas	0
Total	10

Findings

High

[H-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.

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

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

Recommended Mitigation: Consider making the following change to the function.

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

```
11          }
```

[H-2] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOuput` causes the protocol to take too many tokens from users, resulting in lost fees

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

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
13  -           ((inputReserves * outputAmount) * 10000) /
14  +           ((inputReserves * outputAmount) * 1000) /
15              ((outputReserves - outputAmount) * 997);
16      }
```

[H-3] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially receive way fewer 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 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 1 WETH right now is 1,000 USDC 2. User input 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.

```
1
2       function swapExactOutput(
3           IERC20 inputToken,
4   +       uint256 maxInputAmount
5   .
6   .
7   .
8         inputAmount = getInputAmountBasedOnOutput(
9               outputAmount,
10              inputReserves,
11              outputReserves
12          );
13
14  +        if(inputAmount > maxInputAmount){
15  +            revert();
16  +        }
17
18          _swap(inputToken, inputAmount, outputToken, outputAmount);
```

[H-4] `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 token they're willing to sell in the `poolTokenAmount` parameter. However, the function currently miscalcutes 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 tokkens, which is a severe disruption of protocol functionality

Proof of Concept:

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(uint256 poolTokenAmount) external returns(
        uint256 wethAmount) {
```

```
-         return swapExactOutput(i_poolToken, i_wethToken,
    poolTokenAmount, uint64(block.timestamp));

+    function sellPoolTokens(uint256 poolTokenAmount, uint256
    minWethToReceive) external returns(uint256 wethAmount) {
+         return swapExactInput(i_poolToken, i_wethToken,
    poolTokenAmount, uint64(block.timestamp));
     }

Additionally, It might be wise to add a deadline to the function , as
    there is currently no deadline.
```

[H-5] 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 theprotocol, the ratio between the two amounts should 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 ccode os 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 untill all the protocol funds are drained

Proof Of Code

Place the following `TSwapPool.t.sol`

```
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);
        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));

        int256 startingY =  int256(weth.balanceOf(address(pool)));
        int256 expectedDeltaY = int256(-1) * int256(outputWeth);
        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++;
-         if (swap_count >= SWAP_COUNT_MAX) {
-             swap_count = 0;
-             outputToken.safeTransfer(msg.sender, 1
   _000_000_000_000_000_000);
-         }
```

Low

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

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

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

[L-2] Default value returned by `TSwapPool::swapExactInput` result in incorrect value return given

Description: The `swapExactInput` function is expected to return the actual amount of token bought by the caller. However, while it declare the `output` but was never assign a value nor used in an explicit return statement.

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

Proof of Concept:

Recommended Mitigation:

```
 function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 output)
    {
        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] : `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed

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

[I-2] Lacking 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 Line: 35

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

- Found in src/TSwapPool.sol Line: 52

```
1       event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
1       event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
1       event Swap(
```