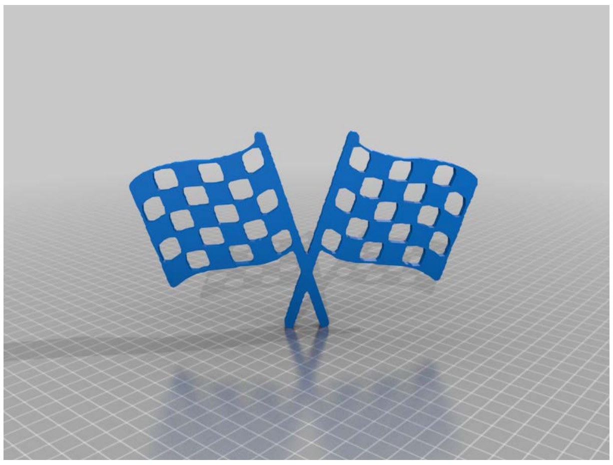

# TSwap Protocol Audit Report

Version 1.0

*Cyfrin.io*

April 24, 2024

# TSwap Protocol Audit Report

Samuel Troy Dominguez

April, 21, 2024

Prepared by: Samuel Troy Dominguez Lead Auditors: - Samuel Troy Dominguez

## Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
  - Findings
  - HIGH
    - [H-1] `TSwapPool::getInputAmountBasedOnOutput` uses the wrong value for calculating fee's, ending up with charging the user 10x as much as it should for fee's.
    - [H-2] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially receive way fewer tokens.
    - [H-3] `TSwapPool::sellPoolTokens` calls the wrong function to complete the swap of tokens, causing users to receive the incorrect amount of tokens.
    - [H-4] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` breaks the protocol invariant of `x * y = k`.

- MEDIUM
  - [M-1] `TSwapPool::deposit` is missing deadline check causing transactions to complete even after the deadline.
- LOW
  - [L-1] `TSwapPool::LiquidityAdded` event is emmiting the wrong information
  - [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return `output` value given.
- Informationals
  - [I-1] `PoolFactory::PoolFactory_PoolDoesNotExist` is never used in the code and should be removed.
  - [I-2] `PoolFactory::constructor` is lacking zero address checks and `TSwapPool::constructor`is lacking zero address checks also.
  - [I-3] `PoolFactory::createPool` should use `.symbol()` instead of using `.name()`
  - [I-4] Event is missing `indexed` fields
  - [I-5] `TSwapPool::deposit` should be rearranged to follow CEI
  - [I-6] Constants should be defined and used instead of literals
  - [I-7] `TSwapPool::swapExactInput` should be changed from `public` to `external` to save gas, `public` is unnecessary and wastes gas in this situation.
  - [I-8] `TSwapPool::totalLiquidityTokenSupply` should be changed from `public` to `external` to save gas
- Gas
  - [G-1] `TSwapPool::poolTokenReserves` in the `TSwapPool::deposit` function is never used and is retrieved later in the function by calling a different function. It should be removed

## Protocol Summary

Protocol does X, Y, Z

## Disclaimer

Samuel Troy Dominguez 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

### Scope

### Roles

## Executive Summary

### Issues found

| Severity | Number of issues found |
|---|---|
| High | 4 |
| Medium | 1 |
| Low | 2 |
| Info | 8 |
| Gas | 1 |
| Total | 16 |

## Findings

## HIGH

### [H-1] `TSwapPool::getInputAmountBasedOnOutput` uses the wrong value for calculating fee's, ending up with charging the user 10x as much as it should for fee's.

**DESCRIPTION** the `getInputAmountBasedOnOutput` function is intended to calculate and deliver the value for the amount of input tokens the user must deposit to be able to receive the amount of output tokens they are requesting. The function miscalulates the calculated `input` value because it multiplies by 10_000 instead of 1_000 in its calculation equation.

**IMPACT** The protocol takes too many fee's from the user.

**PROOF OF CONCEPT** Add this test to `TSwapPool.t.sol`

This test calls `swapExactOutput` which calls and uses `getInputAmountBasedOnOutput` to calculate the amount of input tokens the user needs to deposit/spend for their desired output amount

```
function testFlawedSwapExactOutput() public {
        uint256 initialLiquidity = 100e18;

        vm.startPrank(liquidityProvider);
        weth.approve(address(tPool), initialLiquidity);
        poolToken.approve(address(tPool), initialLiquidity);


        tPool.deposit({wethToDeposit: initialLiquidity,
           minimumLiquidityTokensToMint: 0,
           maximumPoolTokensToDeposit: initialLiquidity,
            deadline: uint64(block.timestamp)
        });

        vm.stopPrank();

        // user has 11 pool tokens
        address someUser = makeAddr("someUser");
        uint256 userInitialPoolTokenBalance = 11e18;
        pooltoken.mint(someUser, userInitialPoolTokenBalance);
        vm.startPrank(someUser);
        // user buys 1 weth from the pool, payingwith poolToken
        poolToken.approve(address(tPool), type(uint256).max);
        tPool.swapExactOutput(poolToken, weth, 1 ether, uint64(
           block.timestamp));

        // Initial Liquidity was 1:1 so user should have paid 1
           poolToken
        // However, it spent much more than that. User started with
            11 tokens but now has less than 10
        assertLt(poolToken.balanceOf(someUser), 1 ether);
        vm.stopPrank();

        // the liquidity provider can rug all funds from the pool
           now,
        // including those deposited by the user
        vm.startPrank(liquidityProvider);
        tPool.withdraw(
            tPool.balanceOf(liquidityProvider),
            1, //minWethToWithdraw
            1, // minPoolTokens to witdraw
            uint64(block.timestamp)
        )

        assertEq(weth.balanceOf(address(tPool)), 0);
        assertEq(poolToken.balanceOf(address(tPool)), 0);

    }
```

**RECOMMENDED MITIGATION**

```
 function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
        return
-            ((inputReserves * outputAmount) * 10000) /
+            ((inputReserves * outputAmount) * 10000) /
             ((outputReserves - outputAmount) * 997);
    }
```

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

**DESCRIPTION** The `swapExactOutput` function does not include any 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 markete conditions change before the 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 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** This function should include a `maxInputAmount` so the user can be sure to only spend up to a decided maximum amount, and can predict how much they will spend with the protocol.

```
        function swapExactOutput(
            IERC20 inputToken,
+            uint256 maxInputAmount,
            IERC20 outputToken,
            uint256 outputAmount,
            uint64 deadline
        )
.
.
.
        inputAmount = getInputAmountBasedOnOutput(
            outputAmount,
            inputReserves,
            outputReserves
        );
        _swap(inputToken, inputAmount, outputToken, outputAmount);
+       if (inputAmount > maxInputAmount) {
+            revert();
+        }
```

### [H-3] `TSwapPool::sellPoolTokens` calls the wrong function to complete the swap of 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. It does not do as intended. Users are instructed to input the amount of pool tokens that they are willing to sell in the `poolTokenAmount` paramater and get the amount `wethAmount` returned to them. This function does not correctly do this, it calls the function that is to be used when the user inputs the amount of `output` (WETH) it is trying to receive, the result is a completely different value that expected.

The function calls `swapExactOutput` which is to be used when the user specifies the amount of `output` (WETH) tokens that they are attempting to receive, and the function determines the correct amount of `input` tokens necessary for that transaction. This is opposite of what `sellPoolTokens` is intended to do. `sellPoolTokens` needs to call `swapExactInput` because `poolTokens` are the `input` token and the protocol needs to determine the correct amount of `output` (WETH) for the users specified `poolTokenAmount`.

**IMPACT** Users will swap the wrong amount of tokens, which is a severe disruption of protocol functionality.

**PROOF OF CONCEPT**

*! Write POC HERE !*

**RECOMMENDED MITIGATION**

Change the function to use `swapExactInput` instead of using `swapExactOutput`. Note that this will also requier changing the `sellPoolTokens` function to accept a new input paramater `minWethToReceive` (minOutputAmount) to be passed in to `swapExactInput`.

```
    function sellPoolTokens(
        uint256 poolTokenAmount
+        uint256 minWethToReceive
    ) external returns (uint256 wethAmount) {
        return
-            swapExactOutput(
-                i_poolToken,
-                i_wethToken,
-                poolTokenAmount,
-                uint64(block.timestamp)
-            );
+            swapExactInput(
+                IERC20 i_poolToken,
+                uint256 poolTokenAmount,
+                IERC20 i_wethToken,
+                uint256 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 in `TSwapPool` - `y`: The balance of WETH in `TSwapPool` - `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 incentive given out by the protocol.

More simply put, the protcols 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 into `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);
        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));

        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++;
-          // 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` is missing deadline check causing transactions to complete even after the deadline.

**DESCRIPTION** The `deposit` function accepts a `deadline` as a paramater, which according to the documentation is - “The deadline for the transaction to be completed by”. However, this paramater is never used. The negative result is that operations that add liquidity to the pool can be executed at unexpected times, and in market conditions where the deposit rate is unfavorable.

**IMPACT** Transactions can be sent when market conditions are unfavorable to deposit, even when a user adds a `deadline` paramater to avoid this situatio.

**PROOF OF CONCEPT** The `deadline` paramater 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)
```

## LOW

### [L-1] `TSwapPool::LiquidityAdded` event is emmiting the wrong information

**DESCRIPTION** The event in `TSwapPool::_addLiquidityMintAndTransfer` function is emitting the information in the wrong order, this isnt a big deal in this protocol, but if it were to ever incorporate oracles - it could become a major problem. `pooltokensToDeposit` should go after `wethToDeposit`.

**IMPACT** The protocol will deliver wrong information and the values for the tokens will be wrong in the emitted event. The user or anything that uses this emitted information will be reading / using the wrong information and values.

**PROOF OF CONCEPT** This is the `LiquidityAdded` event and its paramaters:

```
    event LiquidityAdded(
        address indexed liquidityProvider,
        uint256 wethDeposited,
        uint256 poolTokensDeposited
    );
```

**RECOMMENDED MITIGATION** Change the order of the event arguments that are passed in so that it matches the correct values that it is representing.

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

### [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return `output` 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.

**PROOF OF CONCEPT** * could write a test for this function to show that no matter the value that we swap, it will always return 0.

**RECOMMENDED MITIGATION**

```
    {
        uint256 inputReserves = inputToken.balanceOf(address(this))
           ;
        uint256 outputReserves = outputToken.balanceOf(address(this
           ));

-         uint256 outputAmount = getOutputAmountBasedOnInput(
-             inputAmount,
-             inputReserves,
-             outputReserves
-         );
+         uint256 output = getOutputAmountBasedOnInput(
+             inputAmount,
+             inputReserves,
+             outputReserves
+         );

-         if (outputAmount < minOutputAmount) {
-             revert TSwapPool__OutputTooLow(outputAmount,
    minOutputAmount);
-         }
+         if (output < minOutputAmount) {
+             revert TSwapPool__OutputTooLow(outputAmount,
    minOutputAmount);
+         }
-         _swap(inputToken, inputAmount, outputToken, outputAmount);
+         _swap(inputToken, inputAmount, outputToken, outputAmount);
     }
```

## Informationals

### [I-1] `PoolFactory::PoolFactory_PoolDoesNotExist` is never used in the code and should be removed.

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

### [I-2] `PoolFactory::constructor` is lacking zero address checks and `TSwapPool::constructor`is lacking zero address checks also.

```
     constructor(address wethToken) {
+         if (wethToken == address(0)) {
```

```
+               revert();
+           }
        i_wethToken = wethToken;
    }
```

```
    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] `PoolFactory::createPool` should use `.symbol()` instead of using `.name()`

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

  ```
      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] `TSwapPool::deposit` should be rearranged to follow CEI

**DESCRIPTION** it would be better if this was placed before `_addLiquidityMintAndTransfer` call to follow CEI because as `_addLiquidityMintAndTransfer` makes an external call and can lead to problems like reentrancy, BUT `liquidityTokensToMint` is not a state variable so it should be fine.

```
        else {
            // This will be the "initial" funding of the protocol.
               We are starting from blank here!
            // We just have them send the tokens in, and we mint
               liquidity tokens based on the weth

+             liquidityTokensToMint = wethToDeposit;

            _addLiquidityMintAndTransfer(
                wethToDeposit,
                maximumPoolTokensToDeposit,
                wethToDeposit
            );

-             liquidityTokensToMint = wethToDeposit;
        }
```

### [I-6] Constants should be defined and used instead of literals

- Found in src/TSwapPool.sol Line: 276

```
        uint256 inputAmountMinusFee = inputAmount * 997;
```

- Found in src/TSwapPool.sol Line: 278

```
        uint256 denominator = (inputReserves * 1000) +
            inputAmountMinusFee;
```

- Found in src/TSwapPool.sol Line: 294

```
            ((inputReserves * outputAmount) * 10000) /
```

- Found in src/TSwapPool.sol Line: 295

```
                ((outputReserves - outputAmount) * 997);
```

- Found in src/TSwapPool.sol Line: 402

```
                outputToken.safeTransfer(msg.sender, 1
                    _000_000_000_000_000_000);
```

- Found in src/TSwapPool.sol Line: 454

```
                    1e18,
```

- Found in src/TSwapPool.sol Line: 463

```
                    1e18,
```

#### [I-7] `TSwapPool::swapExactInput` should be changed from `public` to `external` to save gas, `public` is unnecessary and wastes gas in this situation.

#### [I-8] `TSwapPool::totalLiquidityTokenSupply` should be changed from `public` to `external` to save gas

### Gas

#### [G-1] `TSwapPool::poolTokenReserves` in the `TSwapPool::deposit` function is never used and is retrieved later in the function by calling a different function. It should be removed

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