

# **TSwap Audit Report**

Version 1.0

*Frank.io*

# TSwap Audit Report

Frank Ozoalor

Feb 20, 2024

Prepared by: Frank, Security Researcher:

## Table of Contents

- Table of Contents
- Protocol Summary
- TSwap Pools
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
  - Issues found
- Findings
  - High
    - \* [H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causing protocol to take too many tokens from users.
    - \* [H-2] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially receive way fewer tokens
    - \* [H-3] `TSwap::sellPoolTokens` mismatches input and output 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 has parameters out of order
  - \* [L-2] Default value returned by `TSwap::swapExactInput` results in incorrect return value given
- Informationals
  - \* [I-1] `PoolFactory::PoolFactory_PoolDoesNotExist` is not used and should be removed
  - \* [I-2] Lacking zero address check in the constructor
  - \* [I-3] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`
  - \* [I-4] Use of Magic Numbers should be avoided

## Protocol Summary

This 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](#)

## TSwap Pools

The protocol starts as simply a PoolFactory contract. This contract is used to create new "pools" of tokens. It helps make sure every pool token uses the correct logic. But all the magic is in each TSwapPool contract.

You can think of each TSwapPool contract as it's own exchange between exactly 2 assets. Any ERC20 and the WETH token. These pools allow users to permissionlessly swap between an ERC20 that has a pool and WETH. Once enough pools are created, users can easily "hop" between supported ERC20s.

## Disclaimer

This audit 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. An audit by Frank 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

## Audit Details

### Scope

```
1 ./src/  
2 #-- PoolFactory.sol  
3 #-- TSwapPool.sol  
4  
5 - Solc Version: 0.8.20  
6 - Chain(s) to deploy contract to: Ethereum
```

### 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

Severity	Number of issues found
Meduim	1
Low	2
Gas	1
Informational	4
Total	12

## Findings

### High

**[H-1] Incorrect fee calculation in TSwapPool::getInputAmountBasedOnOutput causing protocol to take too many tokens from users.**

**Description** The `getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens users 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** Copy and paste the following code in `TSwapPool.t.sol`

POC

```
1  function testFlawedSwapExactOutput() public {
2      uint256 initialLiquidity = 100e18;
3      vm.startPrank(LiquidityProvider);
4      weth.approve(address(pool), initialLiquidity);
5      poolToken.approve(address(pool), initialLiquidity);
6
7      pool.deposit({
8          wethToDeposit: initialLiquidity,
9          minimumLiquidityTokensToMint: 0,
10         maximumPoolTokensToDeposit: initialLiquidity,
11         deadline: uint64(block.timestamp)
12     });
13     vm.stopPrank();
14
15     //user has 11 pool tokens
16     address userA = makeAddr("userA");
```

```
17     uint256 userInitialPoolTokenBalance = 11e18;
18     poolToken.mint(userA, userInitialPoolTokenBalance);
19     vm.startPrank(userA);
20
21     console.log("starting balance", poolToken.balanceOf(userA));
22
23     //User buys 1 WETH from the pool, paying with pool tokens
24     poolToken.approve(address(pool), type(uint256).max);
25     pool.swapExactOutput(poolToken, weth, 1 ether, uint64(block.
        timestamp));
26
27     //Initial Liquidity was 1:1, so user should have paid ~1 pool
        token
28     // However, it spent much more than that. The user started
        with 11 tokens and now only has less than that 10
29     assertLt(poolToken.balanceOf(userA), 1 ether);
30     console.log("closing balance", poolToken.balanceOf(userA));
31     vm.stopPrank();
32
33     //The liquidity provider can rug all funds from the pool now
34     //Including those deposited by user.
35     vm.startPrank(liquidityProvider);
36     console.log(
37         "previous balance of liquidity provider",
38         pool.balanceOf(liquidityProvider)
39     );
40     pool.withdraw(
41         pool.balanceOf(liquidityProvider),
42         1,
43         1,
44         uint64(block.timestamp)
45     );
46     console.log(
47         "after withdraw balance of liquidity provider",
48         weth.balanceOf(liquidityProvider)
49     );
50     assertEq(weth.balanceOf(address(pool)), 0);
51     assertEq(poolToken.balanceOf(address(pool)), 0);
52 }
```

### Recommended Mitigation

```
1
2     function getInputAmountBasedOnOutput(
3         uint256 outputAmount,
4         uint256 inputReserves,
5         uint256 outputReserves
6     )
7     public
8     pure
9     revertIfZero(outputAmount)
```

```
10     revertIfZero(outputReserves)
11     returns (uint256 inputAmount)
12     {
13         return
14 -         (((inputReserves * outputAmount) * 10000) /          ((
15 +         ((inputReserves * outputAmount) * 1000) /          ((
16         outputReserves - outputAmount) * 997);
17         outputReserves - outputAmount) * 997);
18     }
```

## [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 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 the market conditions change before the transaction processes, the user could get a much worse swap.

### Proof of Concept

1. Assuming the price of 1WETH right now is 1,000 USDC
2. User calls a `swapExactOutput` looking for 1 WETH
  - inputToken = USDC
  - outputToken = WETH
  - outputAmount = 1
  - deadline = uint64(block.timestamp)
3. The function does not offer a `maxInputAmount`
4. As the transaction is pending in the mempool, the market changes! and the price moves -> 1 WETH is now 10,000 USDC. Which is 10x more than the user expected
5. The transaction complete, 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 predict how much they will spend on the protocol.

```
1     function swapExactInput(
2         IERC20 inputToken,
3 +         uint256 maxInputAmount
4     .
5     .
6     .
```

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

### [H-3] TSwap::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 are 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 and not output.

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

**Proof of Concept** Paste the following code in TSwapPool.t.sol

POC

```
1     function test_sellPoolTokens() public {  
2         vm.startPrank(liquidityProvider);  
3         weth.approve(address(pool), type(uint256).max);  
4         poolToken.approve(address(pool), type(uint256).max);  
5         pool.deposit(50e18, 50e18, 50e18, uint64(block.timestamp));  
6         uint256 poolTokenAmountToSell = 7e18;  
7         uint256 amountReturned = pool.sellPoolTokens(  
8             poolTokenAmountToSell);  
9         uint256 expected = 8e18;  
10        /**  
11        users will be depositing way too many tokens for little ether,  
12        if user wants to sell 7 pooltokens, at max 8 pool token is  
13        meant to be taken from the user (to cover for fees)  
14        But due to the wrong scaling in `SwapExactOutput()` function  
15        called in the sellPoolTokens(),  
16        protocol takes 81 pool tokens from user and returns only 7  
17        ether back to the user  
18        */  
19    }
```



```
14         */
15         assert(amountreturned > expected);
16         vm.stopPrank();
17     }
```

**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 `minWethToRecieve` to be passed to `swapExactInput`

```
1  function sellPoolTokens(
2      uint256 poolTokenAmount
3  +    uint256 minWethToRecieve
4  ) external returns (uint256 wethAmount) {
5  -    return swapExactOutput(i_poolToken,i_wethToken,    poolTokenAmount,
6  +    return swapExactInput(i_poolToken, poolTokenAmount,i_wethToken,
7  +    minWethToRecieve, uint64(block.timestamp));
7  + }
```

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

#### [H-4] In `TSwapPool : :_swap` the extra tokens given to users after every `swapCount` breaks the protocol invariant of $x \cdot y = k$

**Description** The protocol follows a strict invariant of  $x \cdot y = k$ . Where:

- $x$ : The balance of the pool tokens
- $y$ : 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

```
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  }
```

**Impact** A user could maliciously drain the protocol funds by doing alot of swaps and collecting the extra incentive given out by the protocol. Therefore breaking the protocol's core invariant

#### Proof of Concept

1. A user swaps 10 times and collects the extra incentive of 1\_000\_000\_000\_000\_000\_000
2. That user continues to swap until all the funds in the protocol are drained.

Paste the following code in `TSwapPool.t.sol`

POC

```
1 function test_InvariantBroken() 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
14    pool.swapExactOutput(
15        poolToken,
16        weth,
17        outputWeth,
18        uint64(block.timestamp)
19    );
20
21    pool.swapExactOutput(
22        poolToken,
23        weth,
24        outputWeth,
25        uint64(block.timestamp)
26    );
27
28    pool.swapExactOutput(
29        poolToken,
30        weth,
31        outputWeth,
32        uint64(block.timestamp)
33    );
34
35    pool.swapExactOutput(
36        poolToken,
37        weth,
38        outputWeth,
39        uint64(block.timestamp)
40    );
41
42    pool.swapExactOutput(
43        poolToken,
44        weth,
```

```
45         outputWeth,  
46         uint64(block.timestamp)  
47     );  
48  
49     pool.swapExactOutput(  
50         poolToken,  
51         weth,  
52         outputWeth,  
53         uint64(block.timestamp)  
54     );  
55  
56     pool.swapExactOutput(  
57         poolToken,  
58         weth,  
59         outputWeth,  
60         uint64(block.timestamp)  
61     );  
62  
63     pool.swapExactOutput(  
64         poolToken,  
65         weth,  
66         outputWeth,  
67         uint64(block.timestamp)  
68     );  
69  
70     pool.swapExactOutput(  
71         poolToken,  
72         weth,  
73         outputWeth,  
74         uint64(block.timestamp)  
75     );  
76  
77     int256 startingY = int256(weth.balanceOf(address(pool)));  
78     int256 expectedDeltaY = int256(-1) * int256(outputWeth);  
79  
80     pool.swapExactOutput(  
81         poolToken,  
82         weth,  
83         outputWeth,  
84         uint64(block.timestamp)  
85     );  
86     vm.stopPrank();  
87  
88     uint256 endingY = weth.balanceOf(address(pool));  
89     int256 actualDeltaY = int256(endingY) - int256(startingY);  
90     assertEq(actualDeltaY, expectedDeltaY);  
91 }
```

**Recommended Mitigation** Remove the extra incentive mechanism. if you want to keep this, 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 - //Fee-on-transfer
3 -     if (swap_count >= SWAP_COUNT_MAX) {
4 -         swap_count = 0;
5 -         outputToken.safeTransfer(msg.sender 1
6 -             _000_000_000_000_000_000);
7 -     }
```

## 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 “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 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 change to the function

```
1     function deposit(
2         uint256 wethToDeposit,
3         uint256 minimumLiquidityTokensToMint,
4         uint256 maximumPoolTokensToDeposit,
5         uint64 deadline
6     )
7     external
8 +     revertIfDeadlinePassed(uint64 deadline)
9     revertIfZero(wethToDeposit)
10    returns (uint256 liquidityTokensToMint)
```

## 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

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

#### [L-2] Default value returned by TSwap::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.

#### Proof of Concept

#### Recommended Mitigation

```
1      {
2          uint256 inputReserves = inputToken.balanceOf(address(this));
3          uint256 outputReserves = outputToken.balanceOf(address(this));
4
5 -         uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount,
6 +         uint256 output = getOutputAmountBasedOnInput(inputAmount,
7             inputReserves, outputReserves);
8             inputReserves, outputReserve);
9
10 -        if (outputAmount < minOutputAmount) {revert
11 +        if (output < minOutputAmount) {revert TSwapPool__OutputTooLow(
12             TSwapPool__OutputTooLow(outputAmount, minOutputAmount); }
13             outputAmount, minOutputAmount); }
14
15 -        _swap(inputToken, inputAmount, outputToken, outputAmount);
16 +        _swap(inputToken, inputAmount, outputToken, output);
17     }
```

#### Informationals

#### [I-1] PoolFactory::PoolFactory\_PoolDoesNotExist is not used and should be removed

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

**[I-2] Lacking zero address check in the constructor**

```
1   constructor(address wethToken) {  
2 +   if(wethToken == address(0)){  
3 +     revert()  
4 +   }  
5     i_wethToken = wethToken;  
6   }
```

**[I-3] PoolFactory::createPool should use .symbol() instead of .name()**

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

**[I-4] Use of Magic Numbers should be avoided**

```
1 +   uint256 constant MULTIPLIER = 997;  
2 +   uint256 constant SCALING_FACTOR = 1000;  
3   return  
4 -   (((inputReserves * outputAmount) * 1000) / ((outputReserves -  
    outputAmount) * 997);  
5 +   (((inputReserves * outputAmount) * SCALING_FACTOR) / ((  
    outputReserves - outputAmount) * MULTIPLIER);
```