



TSwap Audit Report

Version 1.0

Cyfrin.io

July 17, 2024

TSwap Audit Report

Antonio Rodríguez-Ynyesto Sánchez

17th of July, 2024

Prepared by: Cyfrin Lead Auditors: - Antonio Rodríguez-Ynyesto

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
 - Scope
 - Roles
- Executive Summary
 - Issues found
- Findings
 - High
 - * [H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users, resulting in higher fees
 - * [H-2] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially receive way fewer tokens
 - * [H-3] `TSwapPool::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 $x * y = k$

- * [H-5] Rebase, fee-on-transfer and ERC-777 tokens break protocol invariant
- Medium
 - * [M-1] `TSwapPool::deposit` is missing deadline. Transactions sent after deadline will not revert
- Low
 - * [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order
 - * [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
- Informational
 - * [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed
 - * [I-2] Lacking zero address check
 - * [I-3] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`

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

Antonio Rodríguez-Ynyesto has made 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

Scope

Roles

Executive Summary

Issues found

Severity	No. of Issues
High	5
Medium	2
Low	3
Info	9
Total	19

Findings

High

[H-1] Incorrect fee calculation in TSwapPool::getInputAmountBasedOnOutput causes protocol to take too many tokens from users, resulting in higher fees

Description: The `getInputAmountBasedOnOutput` 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 resulting amount. When calculating the fee, it scales the amount by 10,000 instead of 1,000.

Impact: Protocol charges a higher fee than expected from users.

Proof of Concept: Add the following test to TSwapPool.t.sol and check that it passes, showing that the amount of poolTokens spent to get 1 WETH is much larger than should be (about 10x):

```
1      function testSwapExactOutputChargesEnormousFee() external {
2          // Crate the pool
3          uint256 initialLiquidity = 100 ether;
4          vm.startPrank(LiquidityProvider);
5          poolToken.approve(address(pool), initialLiquidity);
6          weth.approve(address(pool), initialLiquidity);
7          pool.deposit({
8              wethToDeposit: initialLiquidity,
9              minimumLiquidityTokensToMint: 0,
10             maximumPoolTokensToDeposit: initialLiquidity,
11             deadline: uint64(block.timestamp)
12         });
13         vm.stopPrank();
14
15         // Give the swapper a starting poolToken balance
16         address swapper = makeAddr("swapper");
17         uint256 swapperInitialPoolTokenBalance = 20 ether;
18         poolToken.mint(swapper, swapperInitialPoolTokenBalance);
19
20         // Swapper tries to swap poolTokens for 1 WETH (which should
21             cost about 1 poolToken, since the original ratio is 1:1)
22         vm.startPrank(swapper);
23         poolToken.approve(address(pool), type(uint256).max);
24         pool.swapExactOutput({
25             inputToken: poolToken,
26             outputToken: weth,
27             outputAmount: 1 ether,
28             deadline: uint64(block.timestamp)
29         });
30         vm.stopPrank();
```

```
31      // Check that the swapper has spent over 10 poolTokens instead
32      // of approximately 1 that was expected
33      assertLt(poolToken.balanceOf(swapper), 10 ether);
34  }
```

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

[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 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 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.

```
1  function swapExactOutput(
2      IERC20 inputToken,
```

```
3 +     uint256 maxInputAmount,
4 .
5 .
6 .
7     inputAmount = getInputAmountBasedOnOutput(outputAmount,
8         inputReserves, outputReserves);
8 +     if(inputAmount > maxInputAmount){
9 +         revert();
10 +     }
11     _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 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.

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 (ie `minWethToReceive` to be passed to `swapExactInput`)

```
1     function sellPoolTokens(
2         uint256 poolTokenAmount,
3 +         uint256 minWethToReceive,
4         ) external returns (uint256 wethAmount) {
5 -         return swapExactOutput(i_poolToken, i_wethToken,
6 +         return swapExactInput(i_poolToken, poolTokenAmount,
7         i_wethToken, minWethToReceive, uint64(block.timestamp));
8     }
```

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

[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.

```
1      swap_count++;
2      if (swap_count >= SWAP_COUNT_MAX) {
3          swap_count = 0;
4          outputToken.safeTransfer(msg.sender, 1
5                                   _000_000_000_000_000_000);
6      }
```

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 into `TSwapPool.t.sol`.

```
1
2      function testInvariantBroken() public {
3          vm.startPrank(liquidityProvider);
4          weth.approve(address(pool), 100e18);
5          poolToken.approve(address(pool), 100e18);
6          pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
7          vm.stopPrank();
8
9          uint256 outputWeth = 1e17;
10
11         vm.startPrank(user);
12         poolToken.approve(address(pool), type(uint256).max);
13         poolToken.mint(user, 100e18);
14         pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
15         pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
16         pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
17     }
```



```
17     pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.  
18         timestamp));  
19     pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.  
20         timestamp));  
21     pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.  
22         timestamp));  
23  
24     int256 startingY = int256(weth.balanceOf(address(pool)));  
25     int256 expectedDeltaY = int256(-1) * int256(outputWeth);  
26  
27     pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.  
28         timestamp));  
29     vm.stopPrank();  
30  
31     uint256 endingY = weth.balanceOf(address(pool));  
32     int256 actualDeltaY = int256(endingY) - int256(startingY);  
33     assertEq(actualDeltaY, expectedDeltaY);  
34 }
```

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.

```
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 -     }
```

[H-5] Rebase, fee-on-transfer and ERC-777 tokens break protocol invariant

Description:

Impact:

Proof of Concept:

Recommended Mitigation:

Medium

[M-1] TSwapPool::deposit is missing deadline. Transactions sent after deadline will not revert

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 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      // @audit high deadline not being used  
6      // if someone sets a deadline, let's say, next block, they  
7      // could still deposit after it has passed!!  
8      // IMPACT: HIGH a deposit expected to fail will go through.  
9      // This is a severe disruption of functionality  
10     // Likelyhood: HIGH - always the case!  
11     uint64 deadline  
12 )  
13 +   external  
14   revertIfDeadlinePassed(deadline)  
15   revertIfZero(wethToDeposit)  
16   returns (uint256 liquidityTokensToMint)  
17 {
```

Low

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

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

```
1      {
2          uint256 inputReserves = inputToken.balanceOf(address(this));
3          uint256 outputReserves = outputToken.balanceOf(address(this));
4
5 -         uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount
6 +         , inputReserves, outputReserves);
7         output = getOutputAmountBasedOnInput(inputAmount,
8         inputReserves, outputReserves);
9
10 -        if (outputAmount < minOutputAmount) {
11 -            revert TSwapPool__OutputTooLow(outputAmount,
12 minOutputAmount);
13 +        if (output < minOutputAmount) {
14 +            revert TSwapPool__OutputTooLow(outputAmount,
15 minOutputAmount);
16     }
17
18 -        _swap(inputToken, inputAmount, outputToken, outputAmount);
19 +        _swap(inputToken, inputAmount, outputToken, output);
20     }
```

Informational

[I-1] PoolFactory::PoolFactory__PoolDoesNotExist is not used and should be removed

Recommended Mitigation:

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

[I-2] Lacking zero address check**Recommended Mitigation:**

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

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

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