TSwap Audit Report

Version 1.0

Frank.io

February 24, 2024

TSwap Audit Report

Frank Ozoalor

Feb 2, 2024

Prepared by: Frank, Security Researcher:

Table of Contents

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

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

- Solc Version: 0.8.20
- 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

```
function testFlawedSwapExactOutput() public {
        uint256 initialLiquidity = 100e18;
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), initialLiquidity);
        poolToken.approve(address(pool), initialLiquidity);

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

        //user has 11 pool tokens
        address userA = makeAddr("userA");
```

```
        uint256 userInitialPoolTokenBalance = 11e18;
        poolToken.mint(userA, userInitialPoolTokenBalance);
        vm.startPrank(userA);

        console.log("starting balance", poolToken.balanceOf(userA));

        //User buys 1 WETH from the pool, paying with pool tokens
        poolToken.approve(address(pool), type(uint256).max);
        pool.swapExactOutput(poolToken, weth, 1 ether, uint64(block.
            timestamp));

        //Initial Liquidity was 1:1, so user should have paid ~1 pool
            token
        // However, it spent much  more than that. The user started
            with 11 tokens and now only has less than that 10
        assertLt(poolToken.balanceOf(userA), 1 ether);
        console.log("closing balance", poolToken.balanceOf(userA));
        vm.stopPrank();

        //The liquidity provider can rug all funds from the pool now
        //Including those deposited by user.
        vm.startPrank(liquidityProvider);
        console.log(
            "previous balance of liquidity provider",
            pool.balanceOf(liquidityProvider)
        );
        pool.withdraw(
            pool.balanceOf(liquidityProvider),
            1,
            1,
            uint64(block.timestamp)
        );
        console.log(
            "after withdraw balance of liquidity provider",
            weth.balanceOf(liquidityProvider)
        );
        assertEq(weth.balanceOf(address(pool)), 0);
        assertEq(poolToken.balanceOf(address(pool)), 0);
    }
```

Recommended Mitigation

```

    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
```

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

[H-2] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentiallu recieve 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 recieve the incorrect amount of tokens

Description The `sellPoolTokens` function is intended to allow users to easily sell pool tokens and recieve 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(
              poolTokenAmountToSell);
8         uint256 expected = 8e18;
9         /**
10        users will be depositing way too many tokens for little ether,
11        if user wants to sell 7 pooltokens, at max 8 pool token is
              meant to be taken from the user (to cover for fees)
12        But due to the wrong scaling in `SwapExactOutput()` function
              called in the sellPoolTokens(),
13        protocol takes 81 pool tokens from user and returns only 7
              ether back to the user
```

```
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,
       uint64(block.timestamp));}
6  + return swapExactInput(i_poolToken, poolTokenAmount,i_wethToken,
       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*y = k`

Description The protocol follows a strict invariant of `x * 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

```
function test_InvariantBroken() 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, 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 compelete 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

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

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

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

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

-    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, output);
    }
```

Informationals

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

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

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

```
    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] Use of Magic Numbers should be avioded

```
+  uint256 constant MULTIPLIER = 997;
+  uint256 constant SCALING_FACTOR = 1000;
    return
-         ((inputReserves * outputAmount) * 1000) / ((outputReserves - outputAmount) * 997);
+         ((inputReserves * outputAmount) * SCALING_FACTOR) / ((outputReserves - outputAmount) * MULTIPLIER);
```