TSwap Audit Report

Version 1.0

Manish Roy

March 17, 2024

Lead Security Researcher: `Manish Roy`

Index

* [I-2] Lacking zero address checks
* [I-3] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`
* [I-4] `TSwapPool::MINIMUM_WETH_DEPOSIT` is a constant, and might not need to be emitted
* [I-5] No natspec for important function - `TSwapPool::swapExactInput`
* [I-6] The else block in the `TSwapPool::deposit` function does not follow CEI and may cause unwanted issues
* [I-7] Provide better error for `TSwapPool::getInputAmountBasedOnOutput`
* [I-8] Provide better error for `TSwapPool::getInputAmountBasedOnOutput`
* [I-9] Explicit type declaration missing at `TSwapPool::swapExactOutput` function
* [I-10] Magic numbers found - can be tricky sometimes
* [I-11] Event is missing `indexed` fields

Disclaimer

I, the security researcher, have 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 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.

Protocol Summary

TSwap 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

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

The `CodeHawks severity matrix` has been leveraged to determine severity of the bugs. See the documentation for more details.

Audit Details

The findings described in this document correspond to the following commit hash:

`55d1e086ed0917fd055b14f63099c2342eb6b86a`

in the following repository:

https://github.com/Cyfrin/5-t-swap-audit

Scope

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

Issues found

Severity	No. of Issues
High	4
Medium	2
Low	2
Gas	2
Informational	11
Total	**21**

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes 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 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.

Recommended Mitigation:

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
-       return ((inputReserves * outputAmount) * 10_000) / ((outputReserves - outputAmount) * 997);
+       return ((inputReserves * outputAmount) * 1_000) / ((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 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. Consider, the price of 1 WETH right now is 1,000 USDC
2. User inputs a `swapExactOutput` looking for 1 WETH
    - inputToken = USDC
    - outputToken = WETH
    - outputAmount = 1
    - 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,
               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 miscalculaes 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.

Proof of Concept:

1. Consider, the price of 1 WETH right now is 1,000 USDC and the pool has 10,000 WETHs and 10,000 USDCs in reserve
2. User hits `sellPoolTokens` with the following values:
    - poolTokenAmount = 1000
    - minWethToReceive = 1
3. Expected return should be ~1 as `wethAmount`
4. Instead, internally the following calculations occur:

```
1       swapExactOutput(inputToken=USDC, outputToken=WETH,
           outputAmount=1000, deadline=uint64(block.timestamp))
```

which returns

```
1       // ((inputReserves * outputAmount) * 10000) / ((
           outputReserves - outputAmount) * 997);
2       ((10000 * 1000) * 10000) / ((10000 - 1000) * 997); // =>
           11144.544745347153
```

Impact: PoC shows an error case and can have severe issues if the `outputReserves` was less than the `outputAmount`. If nothing breaks, users can still swap the **wrong** amount of tokens, which is a severe disruption of protocol functionality.

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`)

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

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

[H-4] In `TSwapPool::_swap` function 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.

```
        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 until 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));
```

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

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
      _000_000_000_000_000_000);
6 -         }
```

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 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 unfavorable.

Impact: Transactions could be sent when market conditions are unfavorable 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, // LP tokens -> if empty,
            we can pick 100% (100% == 17 tokens)
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

[M-2] Rebase, fee-on-transfer and ERC777 tokens break protocol invariant

Description: These tokens have a weird "baked-in" functionality that breaks our protocol The fee on transfer in this case with the liquidityTokens, pays out so much that the x * y = k formula breaks.

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 `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 `ouput` it is never assigned a value, nor uses an explict return statement.

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

Recommended Mitigation:

```
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));
-       uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount, inputReserves, outputReserves);
+       output = getOutputAmountBasedOnInput(inputAmount, inputReserves, outputReserves);
-       if (output < minOutputAmount) {
-           revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
+       if (output < minOutputAmount) {
+           revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
        }
-       _swap(inputToken, inputAmount, outputToken, outputAmount);
+       _swap(inputToken, inputAmount, outputToken, output);
    }
```

Gas

[G-1] Unused variable `poolTokenReserves` in `TSwapPool::deposit` function - carries gas weightage

```
1      if (totalLiquidityTokenSupply() > 0) {
2              uint256 wethReserves = i_wethToken.balanceOf(address(this))
                   ;
3  -           uint256 poolTokenReserves = i_poolToken.balanceOf(address(
       this));
```

[G-2] Use of `public` visibility modifier where `external` can be used

Functions to consider:

- `TSwapPool::swapExactInput`
- `TSwapPool::totalLiquidityTokenSupply`

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] `TSwapPool::MINIMUM_WETH_DEPOSIT` is a constant, and might not need to be emitted

```
-       error TSwapPool__WethDepositAmountTooLow(uint256 minimumWethDeposit, uint256 wethToDeposit);
+       error TSwapPool__WethDepositAmountTooLow(uint256 wethToDeposit);
```

```
-       revert TSwapPool__WethDepositAmountTooLow(MINIMUM_WETH_LIQUIDITY, wethToDeposit);
+       revert TSwapPool__WethDepositAmountTooLow(wethToDeposit);
```

[I-5] No natspec for important function - `TSwapPool::swapExactInput`

Recommended Mitigation: Natspecs for important functions are crucial for well-rounded quality assurance.

[I-6] The else block in the `TSwapPool::deposit` function does not follow CEI and may cause unwanted issues

```
    } 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-7] Provide better error for `TSwapPool::getInputAmountBasedOnOutput`

When `outputReserves` & `outputAmount` are the same, `TSwapPool::getInputAmountBasedOnOutput` will revert with arithmetic divide by zero. This is not very helpful for users. Consider adding a custom error message.

```
+       error TSwapPool__CannotRemoveEntireBalance();
.
.
.
+       if (outputReserves, outputAmount) {revert TSwapPool__CannotRemoveEntireBalance();}
        return (inputReserves * outputAmount) / ((outputReserves - outputAmount));
```

[I-8] Provide better error for `TSwapPool::getInputAmountBasedOnOutput`

When `outputReserves` & `outputAmount` are the same, `TSwapPool::getInputAmountBasedOnOutput` will revert with arithmetic divide by zero. This is not very helpful for users. Consider adding a custom error message.

```
+       error TSwapPool__CannotRemoveEntireBalance();
.
.
.
+       if (outputReserves, outputAmount) {revert TSwapPool__CannotRemoveEntireBalance();}
        return (inputReserves * outputAmount) / ((outputReserves - outputAmount));
```

[I-9] Explicit type declaration missing at `TSwapPool::swapExactOutput` function

```
- inputAmount = getInputAmountBasedOnOutput(outputAmount, inputReserves, outputReserves);
+ uint256 inputAmount = getInputAmountBasedOnOutput(outputAmount, inputReserves, outputReserves);
```

[I-10] Magic numbers found - can be tricky sometimes

- Found in src/TSwapPool.sol::getOutputAmountBasedOnInput
- Found in src/TSwapPool.sol::getInputAmountBasedOnOutput
- Found in src/TSwapPool.sol::_swap
- Found in src/TSwapPool.sol::getPriceOfOneWethInPoolTokens
- Found in src/TSwapPool.sol::getPriceOfOnePoolTokenInWeth

Recommended Mitigation: Consider using constants and immutables wherever possible.

[I-11] 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/TSwapPool.sol: Line: 44
- Found in src/PoolFactory.sol: Line: 37
- Found in src/TSwapPool.sol: Line: 46
- Found in src/TSwapPool.sol: Line: 43