T-Swap Audit Report

Version 1.0

Luo Yingjie

October 11, 2024

T-Swap Audit Report

Luo Yingjie

October 11, 2024

Prepared by: Luo Yingjie Lead Auditors:

- Luo Yingjie

Table of Contents

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

The Luo Yingjie team 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
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

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda

Scope

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

- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum
- Tokens:
  - Any ERC20 token

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.

Executive Summary

Add some notes of how the audit went, types of issues found, etc.

We spend X hours with Y auditors using Z tools, etc.

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Info	4
Gas Optimizations	0
Total	11

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
```

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

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

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 rate than expected.

Proof of Concept:

1. The price of 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 maxInputAmount
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 send the protocol 10,000 USDC for 1 WETH instead of 1,000 USDC.

Recommended Mitigation:

We should include a `maxInputAmount` so the user only has to spend up to a specified amount, and can predict how much they will spend on the protocol.

```
function swapExactOutput(
        IERC20 inputToken,
        IERC20 outputToken,
        uint256 outputAmount,
        uint64 deadline,
+       uint256 maxInputAmount
    )
    .
    .
    .
    inputAmount = getInputAmountBasedOnOutput(
            outputAmount,
            inputReserves,
            outputReserves
        );

+       if(inputAmount > maxInputAmount){
+          revert();
+       }


        _swap(inputToken, inputAmount, outputToken, outputAmount);
```

[H-3] `TSwapPool::sellPoolTokens` mismatch 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 swapper 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 tokens.

Impact:

Users will 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 the `swapExactInput` function.

```
function sellPoolTokens(
```

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

[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 the WETH token
- `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.

```
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 funds by calling the `_swap` function multiple times.

Most importantly, this breaks the invariant of the protocol, which could lead to unexpected behavior.

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.

PoC

Place the following code in the `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)
```

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

Recommended Mitigation:

Medium

[M-1] `TSwapPool::deposit` is missing deadline check causing transaction to complete even after the deadline

Description:

Remove the extra incentives mechanism. If you want to keep this in, we should account for the change in the x * y = k invariant. Or we should set aside tokens in the same way we do with fees.

```
-       swap_count++;
-       if (swap_count >= SWAP_COUNT_MAX) {
-           swap_count = 0;
-           outputToken.safeTransfer(msg.sender, 1
    _000_000_000_000_000_000);
-       }
```

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, operators that add liquidity to the pool might be executed at unexpected times, in market conditions where the deposit rate is unfavorable.

Impact:

Transaction 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,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
        .
        .
        .
    }
```

Low

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

Description:

When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTransfer` 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 `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:

```
function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (
            uint256 output
        )
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

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

```
+        output = getOutputAmountBasedOnInput(inputAmount,inputReserves,outputReserves);
-        if (outputAmount < minOutputAmount) {revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);}
+        if (output < minOutputAmount) {revert TSwapPool__OutputTooLow(output, minOutputAmount);}

-        _swap(inputToken, inputAmount, outputToken, outputAmount);
+        _swap(inputToken, inputAmount, outputToken, output);
     }
```

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]: 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.

4 Found Instances

- Found in src/PoolFactory.sol Line: 35

```
1       event PoolCreated(address tokenAddress, address poolAddress);
```

- Found in src/TSwapPool.sol Line: 52

```
1       event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
1       event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
1       event Swap(
```