Protocol Audit Report

Version 1.0

Ayoub Kroim

June 22, 2024

T-Swap Audit Report

Ayoub Kroim

June 22, 2024

Prepared by: Ayoub Kroim

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
- Executive Summary
  - Issues found
- Findings
  - [H-1] Incorrect fee calculating in `TSwapPool::getInputAmountBasedOnOutput` causes protocal to take too many token from users, resulting in lost fee.
  - [H-2] Lack of slippage protaction in `TSwapPool::swapExactOutput` causes users to may have a WAY WORSE swap.
  - [H-3] `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to receive incorrect amount of tokens.
  - [H-4] In `TSwapPool::_swap` the extra tokens given to users after every `swap_count` breaks the protocol invariant of `x * y = k`
  - Medium

    * [M-1] `TSwapPool:deposit` deadline checks is missing causing transaction to complete even after a deadline passed.
- Low
    * [L-1] `TSwapPool::LiquidityAdded` event has parameter out of order.
    * [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given.
- Informationals
    * [I-1] `PoolFactory:PoolFactory__PoolDoesNotExist` is unused and should be removed.
    * [I-2] Lacking of zero address checks.
    * [I-3] `PoolFactory:creatPool` shoud use `.symbol()` instite 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)

Disclaimer

Ayoub Kroim 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
	High	H	H/M	M
Likelihood	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

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

Executive Summary

Issues found

Severtity	Number of issues found
High	4
Meduim	1
Low	2
Info	9
Total	16

Findings

[H-1] Incorrect fee calculating in `TSwapPool::getInputAmountBasedOnOutput` causes protocal to take too many token from users, resulting in lost fee.

Description: the `getInputAmountBasedOnOutput` function is intended to calculate the amount of token a user should deposit based on output token. However, the function currently miscalculates the resulting amount. When calculating the fee, it scales the amount by 10_000 insted 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) * 10000) / ((
    outputReserves - outputAmount) * 997);
+       return ((inputReserves * outputAmount) * 1000) / ((
    outputReserves - outputAmount) * 997);
    }
```

[H-2] Lack of slippage protaction in `TSwapPool::swapExactOutput` causes users to may have a WAY WORSE swap.

Description: The `swapExactOutput` function does not include any sort of slippage protection. This function is similar to what is done in `TSwapPool::swapExactOutput`, where the function specifies a `minOutputAmount`, the function `swapExactOutput` should also specify a `maxInputAmount`.

Impact: If market conditions change before the transaction processes, the user get wrose swap.

Recommended Mitigation: We should include `maxInputAmount` so the user only have to spend up to specific amount, and can predict how much they will spend in protocol.

```

    function swapExactOutput(
        IERC20 inputToken,
        IERC20 outputToken,
        uint256 outputAmount,
+       uint256 maxInputAmount,
        uint64 deadline

        .
        .
        .

        inputAmount = getInputAmountBasedOnOutput(outputAmount,
            inputReserves, outputReserves);
+       if(inputAmount > maxInputAmount){
+           revert();
+         }
         _swap(inputToken, inputAmount, outputToken, outputAmount);
```

[H-3] TSwapPool::sellPoolTokens mismatches input and output tokens causing users to receive incorrect amount of tokens.

Description: the `seelPoolTokens` 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 fucntion currently miscalculates the swapped amount.

This is due to the fact that the `swapExactOutput` function is called, whereas the `swapExactInput` fucntion is the one that should be called. Because users specify the exact amount of input token, not output.

Impact: Users will swap the wrong amount to tokens, which severe disruption of protcol functionality.

Recommended Mitigation: Consider changing the implemetation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require changing the `sellPoolTokens` fucntion 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));
```

Additinally, 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 swap_count 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 untill all the protocol funds are drained.

Proof of code

Place the following into `TSwapPoolTest`

```
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);
        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 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.

```
-          swap_count++;
-          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` deadline checks is missing causing transaction to complete even after a deadline passed.

Description: the `deposit` function in accepts a deadline parameter, which according to the documentation is "The deadline for the transaction to be completed by". However, this parameter is never been used. As a consequence, the operationrs 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 send to the pool when market condition is unfavorable to deposit, enen when adding a deadline parameter.

Proof of Concept: deadline is not used.

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 `LiquidityAdded` event is emitted in `TSwapPool::_addLiquidityMintAndTransfer` function, it logs value in incorrect order. the `poolTokensToDeposit` value should go in the thrid parameter position, whereas the `wethToDeposit` value should go second.

Impact: Event emission in 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 exepected 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 explict 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)
```

```diff
        revertIfDeadlinePassed(deadline)
        returns (
-           uint256 output
+           uint256 outputAmount

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

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

        if (outputAmount < minOutputAmount) {
            revert TSwapPool__OutputTooLow(outputAmount,
                minOutputAmount);
        }

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

Informationals

[I-1] `PoolFactory:PoolFactory__PoolDoesNotExist` is unused and should be removed.

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

[I-2] Lacking of zero address checks.

```diff
    constructor(address wethToken) {
+        if(wethToken == address(0)){
+            revert();
+        }
        i_wethToken = wethToken;
    }
```

[I-3] `PoolFactory:creatPool` shoud use `.symbol()` instite of `.name()`

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

- Found in src/TSwapPool.sol: Line: 109 "Because"MINIMUM_WETH_LIQUIDITY" is constant is nor required to be emitted."
- Found in src/TSwapPool.sol: Line: 114 "unused variable"
- Found in src/TSwapPool.sol: Line: 145 "it will be better if `_addLiquidityMintAndTransfer` was before `_addLiquidityMintAndTransfer` to follow CEI(checks-effects-interaction).
- Found in src/TSwapPool.sol: Line: 271 "you need natspec"
- Found in src/TSwapPool.sol: Line: 279 "Should be external"
- Found in src/TSwapPool.sol: LIne: 386 "Should be external"