T-Swap Audit Report

Version 1.0

Zkillua.io

T-Swap Audit Report

Zkillua.io

Jan 5th, 2024

Prepared by: [Zkillua]

Table of Contents

Table of Contents

Protocol Summary

T-Swap as a decentralized asset/token exchange (DEX). It is a permissionless way for users to swap assets between each other at a fair price

Disclaimer

The Zkillua 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

Scope

- Code Base: https://github.com/Cyfrin/5-t-swap-audit
- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda
- In Scope:

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

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	5
Medium	0
Low	2
Info	0
Gas Optimizations	0

Severity	Number of issues found
Total	7

Findings

High

[H-1] TSwapPool::_swap breaks Invariant x*y=k

Description: The _swap function is designed to give an extra token for every ten transactions made, this breaks the invariant x*y=k. x: The balance of the pool token y: The balance of WETH k: The constant product of the two balances

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.

Proof of Concept:

Code

```
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        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.

[H-2] TSwapPool::getInputAmountBasedOnOutput Incorrectly calculates the input amount.

Description: getInputAmountBasedOnOutput function is used to calculate the input value for a given output value taking fee into account. However it incorrectly calculates the value by multiplying with 10000 instead of 1000 .

Impact: : Protocol takes more fees than expected from users.

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) * 10_000) / ((
        outputReserves - outputAmount) * 997);
13  +           return ((inputReserves * outputAmount) * 1_000) / ((
        outputReserves - outputAmount) * 997);
14      }
```

[H-3] 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 transaciton processes, the user could get a much worse swap.

Proof of Concept: The user wants to swap USDC for 1Weth at then market prices of 1Weth=100USDC, swapExactOutput function is called to calculate the amount of usdc needed for 1 Weth. When transaction is submitted and waiting in mempool for execution, if market changes and ratio becomes 1Weth=1000USDC, user ends up paying 10x more than what he expects.

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.

```
    function swapExactOutput(
        IERC20 inputToken,
+       uint256 maxInputAmount,
.
.
.
        inputAmount = getInputAmountBasedOnOutput(outputAmount,
            inputReserves, outputReserves);
+       if(inputAmount > maxInputAmount){
+           revert();
+       }
        _swap(inputToken, inputAmount, outputToken, outputAmount);
```

[H-4] `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.

Impact: Users will swap the wrong amount of tokens, which is a severe disruption of protcol 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));
    }
```

[H-5] `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, operationrs 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)
    {
```

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);
    }
```