Protocol Audit Report

Version 1.0

Cyfrin.io

June 10, 2024

TSwap Audit Report

Pedro.io

June 10, 2024

Prepared by: [Pedro] Lead Auditors: - Pedro

Table of Contents

- Medium
  - [M-1] `TSwapPool::deposit` is missing deadline check causing transaction to complete event after the deadline
- Low
  - [L-1] `TSwapPool::LiquidityAdded`event has parameters out of order, causing event to emit incorrect information
  - [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
- Informationals
  - [I-1] `PoolFactory__PoolDoesNotExist` is not used and should be removed
  - [I-2] Lacking zero address checks
  - [I-3] `PoolFactory::createPool` should used `.symbol()` instead of `.name()`
  - [I-4] `event Swap` should have Indexed parameters

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The YOUR_NAME_HERE 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
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L

	Impact			
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Scope Details

- Commit Hash:
- In Scope:

Scope

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

Roles

Executive Summary

Issues found

Severity	Number of issues found
High	5
Medium	1
Low	2
Info	4
Total	12

Findings

High

[H-1] Incorrect fees calculation in `TSwapPool::getInputAmountBasedOnOutput` causing 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 take more fees than expected from users.

Recommended Mitigation:

```
1           return
2 -             ((inputReserves * outputAmount) * 10000) /
3 +             ((inputReserves * outputAmount) * 1000) /
4               ((outputReserves - outputAmount) * 997);
5       }
```

[H-3] Lack oof slippage protection in `TSwapPool::swapExactOutput` causes users to potentially receive 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 specified a `minOutputAmount`, the `swapExactOutput`function should specify a `maxInputAmount`

Impact: If market conditions change before the transaction processes, the user can get a muche worse swap.

Proof of Concept: 1. The price of 1 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 allow a maxInput amount 4. As the transcation is pending in the mempool, the market changes ! 5. And the price moves HUGEEEE -> 1 WETH is now 10 000 USDC. 6. 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 receive

```
1   function swapExactOutput(
```

```
        IERC20 inputToken,
+       uint256 maxInputAmount
        IERC20 outputToken,
        uint256 outputAmount,
        uint64 deadline
    )
.
.
.
  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 are willing to sell in the `poolTokenAmount` parameter. However, the function currently miscalcules the swapped amount.

This is due to the fact that the `swapExactOutput` function is called, whereas the `swapExactInput` function is 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 wich is the severe disruption of the protocol functionnality.

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));
+       swapExactInput(i_poolToken, poolTokenAmount, i_wethToken,
    minWethToReceive, uint64(block.timestamp));
    }
```

[H-5] In `TSwapPool::_swapthe extra tokens given to users after every swapCount break 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 contant product of the two balances

This means that whenever the balance change in the protocol, the ratio between the two amounts should remain constant, hence `k`. However, this is broken due to the extra incentive in the `_swap` function. Meaning that over time the protocol funds wil be drained.

```
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 malicious user could drain the protocol of unds by doing a lot of swaps and collecting the extra incentive given out by the protocol. The protocol core invarian is broken.

Proof of Concept: 1. A user swap 10 tumes, and collects the extra incentive of 1`_000_000_000_000_000_000` 2. The same user continues to swap all the protocol funds are drained

Poc

Place the following into `TSwapPool.t.sol`

```
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 = 10e18;
10         int256 startingY = int256(weth.balanceOf(address(pool)));
11         int256 expectedDeltaY = int256(-1) * int256(outputWeth);
12
13         vm.startPrank(user);
14         poolToken.approve(address(pool), type(uint256).max);
15         poolToken.mint(user, 100e18);
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          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
24          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
25          vm.stopPrank();
26
27          uint256 endingY = weth.balanceOf(address(pool));
28          int256 actualDeltaY = int256(endingY) - int256(startingY);
29
30          assertEq(actualDeltaY, expectedDeltaY);
31      }
```

Recommended Mitigation: Remove the extra incentive or take account of the change

```
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  -         }
```

Medium

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

Description: The `deposit` functions accepts a deadline parameter, which according to the documentation is “The deadline for the transaction to be completed by”. However this parameters is not 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: Transacations 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 following change to the function:

```
1    function deposit(
```

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

Low

emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);

[L-1] `TSwapPool::LiquidityAddedevent` has parameters out of order, causing event to emit incorrect information

Description: When the `liquidityAddedevents` is emitted in the function `TSwapPool::_addLiquidityMintAndTransfer` it logs values in an incorrect order. The `poolTokensToDeposit` value should be in the third parameter position, whereas `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 declare the named return value `output` it is never assigned a value, nor used an explict return statement.

Impact: The return value will be 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 (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);
}
```

Informationals

[I-1] `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 used `.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` Swap should have Indexed parameters

```
event Swap(
        address indexed swapper,
-        IERC20 tokenIn,
```

```
-         uint256 amountTokenIn,
-         IERC20 tokenOut,
-         uint256 amountTokenOut
+         IERC20 indexed tokenIn,
+         uint256 indexed amountTokenIn,
+         IERC20 indexed tokenOut,
+         uint256 indexed amountTokenOut
      );
```