Protocol Audit Report

Version 1.0

Cyfrin.io

September 24, 2024

Protocol Audit Report

Rosen Yordanov

September 23, 2024

Prepared by: Rosen Yordanov

Table of Contents

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The Rosen Yordanov 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.

Scope

- 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

Severtity	Number of issues found
High	4
Medium	0
Low	2
Info	6
Total	12

Findings

High

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

Description The `deposit` function accepts a deadline parameter, which acoarding 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 time, in market condition where the deposit rate is unfavorable,

Impact Transactions could be send when market conditions are unfavorable to deposit, even when adding a deadline parameter

Proof of Concepts 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
        revertIfZero(wethToDeposit)
+       revertIfDeadlinePassed(deadline)
        returns (uint256 liquidityTokensToMint)
    {}
```

[H-2] Incorrect fee calculations in `TSwapPool::getInputAmountBasedOnOutput` function causes protocol to take too many tokens from users, resulting in lost fees.

Description The `TSwapPool::getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens a user should deposit, given an amount 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 the 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);
+            ((inputReserves * outputAmount) * 1000) / ((outputReserves
     - outputAmount) * 997);
    }
```

[H-3] Lack of slippage protection in `TSwapPool::swapExactOutput` causes a 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 Concepts 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 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.

```
    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.

Proof of Concepts

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] 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 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.

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 `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));
        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);
-         }
```

Low

[L-1] `TSwapPool::LiquidityAdded` event has parameters out of order causing event to emmit incorrect information

Description When liquidity is added event is emitted in `TSwapPool::_addLiquidityMintAndTransfer` function, logs values in incorrect order.The `poolTokensToDeposit` paramter should go on the third position.

Impact Event emission is incorrect, leading to off-chain functions, potentially malfunctioninig.

[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);
```

Recommended mitigation

```
- emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
+ emit LiquidityAdded(msg.sender, wethToDeposit, poolTokensToDeposit);
```

Informationals

[I-1] `PoolFactory_PoolDoesNotExist` is not used and should be removed

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

[I-2] Lacking address(0) check in `PoolFactory` constructor

```
  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::Swap` event should have 3 indexed parameters

```
      event Swap(
          address indexed swapper,
+          IERC20 indexed tokenIn,
          uint256 amountTokenIn,
+          IERC20 indexed tokenOut,
          uint256 amountTokenOut
      );
```

[I-5] `poolTokenReserves` variable in `TSwapPool::deposit` function is never used.

```
- uint256 poolTokenReserves = i_poolToken.balanceOf(address(this));
```

[I-6] Magic numbers are not recommended. Create constants instead. For example `TSwapPool::getOutputAmountBasedOnInput` function uses magic numbers

```
-          uint256 inputAmountMinusFee = inputAmount * 997;
+          uint256 inputAmountMinusFee = inputAmount * someConstant;
           uint256 numerator = inputAmountMinusFee * outputReserves;
-          uint256 denominator = (inputReserves * 1000) +
   inputAmountMinusFee;
+          uint256 denominator = (inputReserves * someConstant) +
   inputAmountMinusFee;
```