TSwap-Pool Audit Report

Sodiq Adewole

October 22, 2024

Protocol Audit Report

Version 1.0

Sodiq Adewole

October 22, 2024

TSwap-Pool Audit Report

Sodiq Adewole

October 22, 2024

Prepared by: Sodiq Adewole Lead Auditors: - Sodiq Adewole

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
- High
- Medium
- Low
- Informational
- Gas

Protocol Summary

Protocol does X, Y, Z

Disclaimer

Sodiq Adewole 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

-Commit Hash: 466d7636abc6c0183a2242323a34b809dd16421d

Scope

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

Roles

Liquidity Provider: The provider of liquidity to the pool User/swapper: The user that swaps and use the protocol.

Executive Summary

Issues found

Severity	Number Of Issues Found
High	4
Medium	1
Low	2
Gas	2
Info	5
Total	14

Findings

Informationals

[I-1] `PoolFactory_PoolDoesNotExist` is not used

Description: The error code used `PoolFactory__PoolDoesNotExist(address tokenAddress)` in `PoolFactory` contract is not used in anywhere in the code 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;
    }
```

```
 constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
+        if (i_wethToken = address(0)) {
+            revert();
+        }
        i_wethToken = IERC20(wethToken);
+        if (i_poolToken = address(0)){
+            revert();
+        }
        i_poolToken = IERC20(poolToken);
    }
```

[I-2] Token symbol function call

Description: The function call for the the liquidity token symbol is wrong in the `PoolFactory`

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

[I-3]: 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

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

- Found in src/TSwapPool.sol Line: 52

```
event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
event Swap(
```

[I-4] Not following CEI in `TSwapPool::deposit` function

Description: The following changes should be made to follow CEI

```
else {

+         liquidityTokensToMint = wethToDeposit;
            _addLiquidityMintAndTransfer(
                wethToDeposit,
                maximumPoolTokensToDeposit,
                wethToDeposit
            );
-           liquidityTokensToMint = wethToDeposit;
        }
```

[I-5]: `public` functions not used internally could be marked `external`

Instead of marking a function as `public`, consider marking it as `external` if it is not used internally.

1 Found Instances

- Found in src/TSwapPool.sol Line: 298

```
function swapExactInput(
```

Medium

[M-1] `TSwapPool::deposit` is missing deadline check causing the transactions to completer after the deadline check

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

Impact: Transactions could be sent when the market conditions is unfavourable to deposit, even when adding a deadline parameter.

Proof of Concept: The `deadline` parameter is unused.

Recommended Mitigation: Consider making the following the change to the function

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

Description:

Impact:

Proof of Concept:

Recommended Mitigation:

Gas

[G-1] Using the constant `MINIMUM_WETH_LIQUIDITY` as an emitted event is a waste of gas

Description:

```
if (wethToDeposit < MINIMUM_WETH_LIQUIDITY) {
            revert TSwapPool__WethDepositAmountTooLow(
-                MINIMUM_WETH_LIQUIDITY,
                wethToDeposit
            );
        }
```

[G-2] `TSwapPool::poolTokenReserves` is not used anywhere in the contract, resulting to waste of gas

Description:

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

Low

[L-1] `TSwapPool:LiquidityAdded` event has parameter out of order, causing the event to emit incorrect information.

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 be second.

Impact: Event emission is incorrect, leading to off-chains potentially malfunctioning.

Recommended Mitigation: The following change should be made

```
    function _addLiquidityMintAndTransfer(
        uint256 wethToDeposit,
        uint256 poolTokensToDeposit,
        uint256 liquidityTokensToMint
    ) private {
        _mint(msg.sender, liquidityTokensToMint);
-       emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
+         emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
```

[L-2] Default value returned by `TSwap::swapExactInput` results in incorrect return value given.

Description: The `swapExactInput` to return the actual amount of tokens bought by the caller. However, 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 zero(0), giving incorrect information to the caller.

Recommended Mitigation: The following the changes should be made

```
 {
        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 (outputAmount < minOutputAmount) {
            revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
        }
-        _swap(inputToken, inputAmount, outputToken, outputAmount);
+       _swap(inputToken, inputAmount, outputToken, output);
    }
```

High

[H-1] Incorrect fee in calculation in `getInputAmountBasedOutput`, causing the protocol to take many amount tokens from users, resulting to lost fees

Description: The `getInputAmountBasedOutput` is intended to calculate the amount of tokens a user deposit given an amount of tokens of output tokens. However, the function currently miscalculates the resulting amount. When the calculating the fee, it scales the amount by 10,000 instead of 1000.

Impact: Protocol takes more fees than expected from users.

Proof of Concept:

Poc

This function shows the test how a liquidity provider provides liquidity to a pool at a starting of 1:1 and a swapper with 11 pool tokens swapped a pool token a weth token. The swapper later lost some pool token fees after the swap.

```
    function testFlawedSwapExactOutput() public {
        uint256 initialLiquidity = 100e18;
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), initialLiquidity);
        poolToken.approve(address(pool), initialLiquidity);

        pool.deposit(100e18, 0, 100e18, uint64(block.timestamp));

        vm.stopPrank();
```

```
        //User has 11 tokens

        address someUser = makeAddr("someUser");
        uint256 initialPoolTokenBalance = 11e18;
        poolToken.mint(someUser, initialPoolTokenBalance);
        vm.startPrank(someUser);

        // User buys 1 Weth from the pool, paying with pool tokens
        poolToken.approve(address(pool), type(uint256).max);
        pool.swapExactOutput(poolToken, weth, 1e18, uint64(block.timestamp));

        //Initial liquidity was 1:1 , so user should have paid one pool token
        // However, it spent much more, the user has 11 token before and now has less than
        vm.stopPrank();
        assertLt(poolToken.balanceOf(someUser), 1 ether);
    }
```

Recommended Mitigation: The following changes should be made

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
-       return ((inputReserves * outputAmount) * 10_000) / ((outputReserves - outputAmount)
+       return ((inputReserves * outputAmount) * 1_000) / ((outputReserves - outputAmount)
    }
```

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

Description: The `swapExactOutput` function does not include any sort slippage protection. The function is similar to what is done `TSwapPool::swapExactInput`, where the function specifies the `minInputAmount`, the `swapExactOutput` should also specifies the `maxInputAmount`.

Impact: if market conditions changes before the transaction processes, the user could get a much 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 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, outputReserve
+       if(inputAmount > maxInputAmount){
+           revert();
+       }
        _swap(inputToken, inputAmount, outputToken, outputAmount);
```

[H-3] `TSwapPool::sellPoolTokens` mismatches input and output tokens, causing the user to receive the amount of tokens.

Description: The `sellPoolTokens` function is intended to allow the users to easily sell pool tokens and receive WETH in exchange. Users indicate how many pool tokens they are willing to sell `poolTokenAmount` parameter. However, the function currently miscalculates the swapped amount.

This is due to the fact `swapExactOutput` function is called, instead of the function `swapExactInput` that should be called. Because users specify the amount of input tokens, not output.

Impact: Users will swap the wrong amount of tokens, which is a severe disruption of protocol functionality.

Proof of Concept:

If we we have a liquity provider that provides liquidity on the pool and also a user that an input of a 1e18 as the swap amount The following will be the balances of the swapper.

poolToken balance —> 19086859568604804 3120 // 190.865 wethbalance —-> 1000000000000000000 // 1

Poc

```
function testSellPoolTokenFunctionFailed() public {

     weth.mint(liquidityProvider, 200e18);
     poolToken.mint(liquidityProvider, 200e18);

        address swapper = makeAddr("swapper");
        poolToken.mint(swapper, 200e18);
        uint256 swapAmount = 1e18;
        uint256 poolTokenBalanceOfSwapperBefore = poolToken.balanceOf(swapper);
        uint256 wethBalanceOfSwapperBefore = weth.balanceOf(swapper);

        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        vm.startPrank(swapper);
        poolToken.approve(address(pool), 200e18);
        pool.sellPoolTokens(swapAmount);
        vm.stopPrank();

        uint256 poolTokenBalanceOfSwapperAfter = poolToken.balanceOf(swapper);
        uint256 wethBalanceOfSwappeAfter = weth.balanceOf(swapper);
        assertEq(poolTokenBalanceOfSwapperBefore, poolTokenBalanceOfSwapperAfter + swapAmoun
    }
```

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.time
+        return swapExactInput(i_poolToken, poolTokenAmount, i_wethToken, minWethToReceive,
    }
```

Additionaly, it might be wise to add a deadline to the function, as there is not currently no deadline.

[H-4] In `TSwapPool::_swap`, the extra tokens given to users after every `swapCount` breaks 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 token
- `k` The constant of the two balances.

This means that whenever the balances change in the protocol, the ratio between the two amounts should remain the same, hence the `k`. However, this is broken due to the extra incentive in the `_swap` function. Meaning that overtime, the protocol funds will be drained.

Impact: A user can maliciously drain the protocol of funds by doing a lot of swaps and collecting the extra incentive given by the protocol.

Most simply, the protocol core invariant is broken.

The following block of code is responsible for issue.

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

Proof of Concept:

1. A user swaps 10 times, and collect the extra incentive `1_000_000_000_000_000_000` tokens
2. The user continues the swap until the protocol funds are drained

Poc

```
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();

        address someUser = makeAddr("someUser");

        uint256 outputWeth = 1e17;
        int256 startingY = int256(weth.balanceOf(address(pool)));
        int256 expectedDeltaY = int256(-1) * int256(outputWeth);

        vm.startPrank(someUser);
```

```
        poolToken.mint(someUser, 100e18);
        weth.mint(someUser, 100e18);
        poolToken.approve(address(pool), type(uint64).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));


        vm.stopPrank();

        uint256 endingY = weth.balanceOf(address(pool));
        int256 actualDeltaY = int256(endingY) - startingY;
        assertEq(actualDeltaY, expectedDeltaY);
    }
```

Recommended Mitigation: Remove the extra incentive mechanism. if you want keep this in, we should account for the change in x * y = k protocol invariant. Or we should we set aside tokens the same 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);
-        }
```