TSwapPool Audit Report

February 9, 2025

Protocol Audit Report

Claudia Romila

9 February 2025

Prepared by: Claudia Romila

Table of Contents

Protocol Summary

The PoolFactory, serves as a factory contract for creating and managing liquidity pools (TSwapPool) associated with different ERC-20 tokens.

The TSwapPool contract is a decentralized liquidity pool that enables users to deposit and withdraw two types of tokens: WETH (Wrapped Ether) and Pool Tokens. It also facilitates token swaps, where users can exchange one token for another based on the current liquidity reserves. The contract ensures that liquidity ratios are maintained and provides an incentive mechanism to the users.

Disclaimer

Claudia 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

Audit Details

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda
- In Scope:

Scope

```
./src/
|__ PoolFactory.sol
|__ TSwapPool.sol
```

Roles

TSwapPool

Liquidity Providers: Deposit WETH and pool tokens to add liquidity, withdraw liquidity when needed.

Swappers: Swap between WETH and pool tokens with defined pricing rules.

Contract: Manages liquidity, enforces trading rules, calculates pricing, handles swaps, and distributes incentives.

PoolFactory

Owner: Deploys the contract.

Users: Create liquidity pools for ERC-20 tokens.

Executive Summary

Auditing PoolFactory and TSwapPool was rewarding, uncovering areas to improve security and efficiency.

Issues found

Severity	Number of issues found
High	5
Medium	0
Low	2
Info	4
Gas	0
Total	11

Findings

High

[H-1] `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 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 conditions where 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 `dealdline` parameter is unused.

Recommandation Mitigation: Consider making the following change to the function.

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

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

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 miscalculate the resulting amount. When calculating the fee, it scale the amount by 10_000 instead of 1_000.

Impact: Protocol takes more fees than expected from users.

Recommandation Mitigation:

```
    function getInputAmountBasedOnOutput(
         uint256 outputAmount,
         uint256 inputReserves,
         uint256 outputReserves
     )
         public
         pure
         revertIfZero(outputAmount)
         revertIfZero(outputReserves)
         returns (uint256 inputAmount)
     {
-         return((inputReserves * outputAmount) * 10_000) /
         ((outputReserves - outputAmount) * 997);

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

[H-3] Lack of slippage protection in `PoolFactory:: swapExactOutput` causes users to potentially receive way fewer tokens

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

Impact: If market conditions change 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. outToken = WETH 3. outPutAmount = 1 4. deadline = whatever 3. The function doesn't offer a maxInput amount 4. As the transaction is pending in the mempool, the market changes! The price moves is 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.

Recommandation Mitigation: We should include a `maxInputAmount` so the user only have 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();
+         }
```

[H-4] `TSwapPool:: sellPoolTokens` mismatches input and output tokens causing users to receive incorrect amount of tokens

Description: The `sellPoolTokens` function is intended to allow users to sell pool tokens and receive WETH in exchange. Users indicate how many pool tokens they're willing to sell in the `poolAmount` parameter. However, the function currently miscalculates the swapped amount. This is due to the fact that the `swapExactOutput` function is called, whereas the `swapExactInput` is the one that should be called. Because users specify the exact amount of input tokens.

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

Recommandation Mitigation: Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require changing the `sellPoolTokens` function to accept 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)
        );
    }
```

Additionally, it might be wise to add a deadline to the function, as there is currently no deadline.

[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 follow a strict invariant of `x * y = k`. Where: - `x`: The balance of the pool tokens - `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 overtime the protocol funds will be drained.

The follow block of code is responsible for the issue. `javascript 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: The 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)
        );
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );

        int256 startingY = int256(weth.balanceOf(address(this)));
        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);
    }
```

Recommandation Mitigation: Remove the extra incentive mechanism. If you want to keep this in, then we should account for the changes in 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 emit incorrect information

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

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool:: _addLiquidityMintAndTra` function, it logs values in an incorrect order. The `poolTokensToDeposit` value should go in the third parameter possition, whereas the `wethToDeposit` value should go second.

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

Recommandation 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 `TSwapPool:: swapExactInput` function is expected to return the actual amount of tokens bought by the caller. However, while is declares the named return value `output` is never assigned a value, nor uses an explecit return statment.

Impact: The return value will always be 0, given incorrect information to the caller.

Recommandation Mitigation:

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

-    uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount,
    inputReserves, outputReserves);
+    uint256 output = getOutputAmountBasedOnInput(inputAmount,
    inputReserves, outputReserves);

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

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

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

```
16 +    _swap(inputToken, inputAmount, outputToken, output);
17
18 }
```

Informationals

[I-1] `PoolFactory:: PoolFactory__PoolDoesNotExist` is not used and should be removed

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

[I-2] Lacking zero address check

```
1     constructor(address wethToken) {
2 +   if ( i_wethToken == address(0)) {
3 +       revert();
4      }
5         i_wethToken = wethToken;
6      }
```

[I-3] `PoolFactory:: liquidityTokenSymbol` should use `.symbol()` instead of `.name()`

```
1 -  string memory liquidityTokenSymbol = string.concat("ts", IERC20    (
      tokenAddress).name());
2
3 +  string memory liquidityTokenSymbol = string.concat("ts", IERC20    (
      tokenAddress).symbol());
```

[I-4] `PoolFactory:: Swap event` each event should use three indexed fields if there are three or more fields

```
1     event Swap(
2 -       address indexed swapper,
3 -       IERC20 tokenIn,
4 -       uint256 amountTokenIn,
5 -       IERC20 tokenOut,
6 -       uint256 amountTokenOut
7
8 +       address indexed swapper,
9 +       IERC20 tokenIn,
```

```
10 +       uint256 indexed amountTokenIn,
11 +       IERC20 tokenOut,
12 +       uint256 indexed amountTokenOut
13     );
```