T-SWAP Audit Report

Version 1.0

Ramprasad

April 26, 2025

T-SWAP Audit Report

Ramprasad

April 26, 2025

Prepared by: Ramprasad

Table of Contents

    * [M-1] `TSwapPool::deposit` is missing deadline check causing transactions to complete even after the deadline
- LOW
    * [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order
    * [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
- INFORMATIONAL
    * [I-1] `PoolFactory::PoolFactory_PoolDoesNotExist` is not used and should be removed
    * [I-2] Lacking zero address check
    * [I-3] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`
- [I-4]: Event is missing `indexed` fields

Disclaimer

This audit was independently conducted by Ramprasad. Every effort was made to identify as many vulnerabilities as possible within the given time frame. However, no guarantees are provided regarding the discovery of all existing issues.

This report focuses solely on the security aspects of the Solidity implementation at the time of the audit. It does not constitute an endorsement of the underlying business model or project viability. Readers should perform their own independent assessments.

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

The findings described in this document corresponded the following commit hash:**

```
e643a8d4c2c802490976b538dd009b351b1c8dda
```

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.

Executive Summary

Issues found

Severity	Number of issues found
HIGH	3
MEDIUM	1
LOW	2
INFORMATIONAL	4
Total	10

Findings

HIGH

[H-1] Incorrect Fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes 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 output tokens. However, the function is currently miscalculating the resulting amount. When calculating the fee, it scales the amount by 10_000 instead of 1_000.

Impact: The protocol takes more fee than expected from users.

Recommended 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-2] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially receive significantly fewer tokens

Description: The `swapExactOutput` function does not include any slippage protection. This function is similar to `TSwapPool::swapExactInput`, where the function specifies a `minOutputAmount`. The `swapExactOutput` function should specify a `maxInputAmount` to protect users from excessive input amounts due to market changes.

Impact: If market conditions change before the transaction is processed, the user could receive a worse swap than expected.

Proof of Concept: 1. The price of 1 WETH is currently 1,000 USDC. 2. A user calls `swapExactOutput` to receive exactly 1 WETH. - inputToken = USDC - outputToken = WETH - outputAmount = 1 - deadline = any valid timestamp 3. The function does not enforce a maximum input amount. 4. While the transaction is pending in the mempool, the market price changes drastically: 1 WETH now costs 10,000 USDC (10x increase). 5. The transaction completes, but the user ends up spending 10,000 USDC instead of the expected 1,000 USDC.

Recommended Mitigation: Include a `maxInputAmount` parameter so the user only spends up to a specified maximum, protecting against unexpected price slippage.

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

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

[H-3] `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 sell pool tokens and receive WETH in exchange. Users specify how many pool tokens they want to sell via the `poolTokenAmount` parameter. However, the function currently calls `swapExactOutput` instead of `swapExactInput`. Since users specify the exact amount of input tokens, `swapExactInput` should be used.

Impact: Users will swap the wrong amount of tokens, severely disrupting protocol functionality.

Recommended Mitigation: Change the implementation to use `swapExactInput` instead of `swapExactOutput`. This will also require adding a new parameter (e.g., `minWethToReceive`) to `sellPoolTokens` to specify the minimum acceptable WETH amount.

```
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 is recommended to add a deadline parameter to this function, as there is currently no deadline enforcement.

MEDIUM

[M-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 the documentation is "The deadline by which the transaction must be completed". However, this parameter is never used. As a consequence, liquidity additions might be executed at unfavorable times or market conditions.

Impact: Transactions may be executed when market conditions are unfavorable, despite specifying a deadline.

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

Recommended Mitigation: Add a deadline check modifier to the function.

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        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::_addLiquidityMintAndTransfer` function, it logs values in an incorrect order. The `poolTokensToDeposit` value should be the third parameter, whereas the `wethToDeposit` value should be the second.

Impact: Event emission is incorrect, potentially causing off-chain tools to malfunction.

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, the named return value `output` is never assigned a value nor explicitly returned.

Impact: The return value is always zero, providing incorrect information to the caller.

Recommended Mitigation:

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

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

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

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

INFORMATIONAL

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

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

[I-2] Lacking zero address check

```
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]: Event is missing `indexed` fields

Indexed event fields make them more quickly accessible to off-chain tools that parse events. However, each indexed field costs extra gas during emission, so it's not always best to index the maximum allowed per event (three fields). Events with three or more fields should use three indexed fields if gas usage is not a major concern. Events with fewer than three fields should have all fields indexed.

- 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(
```