TSwap Protocol Audit Report

October 19, 2025

Protocol Audit Report

Enchanted17

October 19, 2025

Prepared by: Enchanted17

Email: luo_dz@163.com

Table of Contents

    * [M-1] `TSwapPool::deposit` is missing deadline check causing transaction to complete even after the deadline.
  - Low
    * [L-1] `TSwapPool::LiquidityAdded` event has parameter out of order.
    * [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given.
  - Information
    * [I-1] Unused `PoolFactory::_PoolDoesNotExist` error in `PoolFactory` contract and should be remove.
    * [I-2] Lacking zero-check in `PoolFactory::constructor`
    * [I-3] Wrong methods called
    * [I-4] Event is missing `indexed` fields
    * [I-5] Lacking zero-check in `TSwapPool::constructor`
    * [I-6] Unused variable in `TSwapPool::deposit`
    * [I-7] Use of Hard-Coded Magic Numbers in TSwapPool, reducing Readability and Maintenance Difficulty.
    * [I-8] `SWapPool::swapExactInput`: `public` functions not used internally could be marked `external`
    * [I-9] Missing deadline param in natspec of `TSwapPool::swapExactOutput`
    * [I-10] `SWapPool::totalLiquidityTokenSupply`: `public` functions not used internally could be marked `external`

Protocol Summary

Protocol does X, Y, Z

Disclaimer

Our 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

Likelihood\Impact	High	Medium	Low
High	H	H/M	M
Medium	H/M	M	M/L
Low	M	M/L	L

Audit Details

Scope

- PoolFactory.sol
- TSwapPool.sol

Tools

- Slither
- Aderyn

Executive Summary

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Info	10
Total	17

Findings

High

[H-1] Incorrect fee calculation 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 the 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 users.

Recommended Mitigation:

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
        return
-           ((inputReserves * outputAmount) * 10000) /
+           ((inputReserves * outputAmount) * 1000) /
            ((outputReserves - outputAmount) * 997);
    }
```

[H-2] Lack of slippage protection 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 specifies a `minOutputAmount`, the `swapExactOutput` function should specify a `maxInputAmount`.

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 SDC 2. User inputs a `swapExactOutput` looking for i 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-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 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.

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-4] The extra tokens given to users after every swapCount breaks the protocol invariant 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.

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. That user continues to swap untill all the protocol funds are drained

Proof of Code: Place the following into TSwapPool.t.sol.

```
    function testBreakInvariant() 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.mint(user, 100e18);
        poolToken.approve(address(pool), type(uint256).max);
        for (uint8 i = 0; i<9; i++){
            pool.swapExactOutput({inputToken: poolToken, outputToken: weth,
↪  outputAmount: outputWeth, deadline: uint64(block.timestamp)});
        }

        pool.swapExactOutput({inputToken: poolToken, outputToken: weth,
↪  outputAmount: outputWeth, deadline: uint64(block.timestamp)});

        int256 startingY = int256(poolToken.balanceOf(address(pool)));
        int256 expectedDeltaY = int256(-1) * int256(outputWeth);
        vm.stopPrank();

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

A user swaps 10 times, and collects the extra incentive of 1_000_000_000_000_000_000 tokens That user continues to swap untill all the protocol funds are drained **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++;
-         // Fee-on-transfer
-         if (swap_count >= SWAP_COUNT_MAX) {
-             swap_count = 0;
-             outputToken.safeTransfer(msg.sender,
↪  1_000_000_000_000_000_000);
-         }
```

Medium

[M-1] `TSwapPool::deposit` is missing deadline check causing transaction to conplete even after the deadline.

Description: The `deposit` function accept 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 rate is unfavorable.

Impact: Transactons could be sent when market conditions are unfavorable to deposit, even when adding a deadline parameter.

Proof of Concept: `deadline` is never used.

```
Warning (5667): Unused function parameter. Remove or comment out the
 ↪  variable name to silence this warning.
   --> src/TSwapPool.sol:120:9:
    |
120 |         uint64 deadline
    |         ^^^^^^^^^^^^^^^
```

Recommended Mitigation: Consider making the following change to this 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 parameter out of order.

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

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

Proof of Concept:

```
    event LiquidityAdded(
        address indexed liquidityProvider,
        uint256 wethDeposited,
        uint256 poolTokensDeposited
    );
...
```

```
@>      emit LiquidityAdded(msg.sender, poolTokensToDeposit,
 ↪  wethToDeposit);
```

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 declares the named return value `output` 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.

Proof of Concept(Proof of code): Add an `assert` in `TSwapPoolTest::testCollectFees`, it passed.

```
    function testCollectFees() public {
        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(user);
        uint256 expected = 9e18;
        poolToken.approve(address(pool), 10e18);
-       pool.swapExactInput(poolToken, 10e18, weth, expected,
 ↪  uint64(block.timestamp));
+       uint256 return_value = pool.swapExactInput(poolToken, 10e18, weth,
 ↪  expected, uint64(block.timestamp));
+       assert(return_value == 0);
        vm.stopPrank();

        vm.startPrank(liquidityProvider);
        pool.approve(address(pool), 100e18);
        pool.withdraw(100e18, 90e18, 100e18, uint64(block.timestamp));
```

```
        assertEq(pool.totalSupply(), 0);
        assert(weth.balanceOf(liquidityProvider) +
↪  poolToken.balanceOf(liquidityProvider) > 400e18);
    }
```

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);
    }
```

Information

[I-1] Unused `PoolFactory::_PoolDoesNotExist` error in `PoolFactory` contract and should be remove.

Recommended Mitigation:

```
...
contract PoolFactory {
    error PoolFactory__PoolAlreadyExists(address tokenAddress);
-   error PoolFactory__PoolDoesNotExist(address tokenAddress);
...
```

[I-2] Lacking zero-check in `PoolFactory::constructor`

Recommended Mitigation:

```
    constructor(address wethToken) {
+       if(wethToken == address(0)) {
+           revert();
+       }
        i_wethToken = wethToken;
    }
```

[I-3] Wrong methods called

Description: `PoolFactory::createPool` should use `.symbol()` instead of `.name()`

```
    function createPool(address tokenAddress) external returns (address) {
        if (s_pools[tokenAddress] != address(0)) {
            revert PoolFactory__PoolAlreadyExists(tokenAddress);
        }
        string memory liquidityTokenName = string.concat("T-Swap ",
 ↪  IERC20(tokenAddress).name());
@>      string memory liquidityTokenSymbol = string.concat("ts",
 ↪  IERC20(tokenAddress).name());
        TSwapPool tPool = new TSwapPool(tokenAddress, i_wethToken,
 ↪  liquidityTokenName, liquidityTokenSymbol);
        s_pools[tokenAddress] = address(tPool);
        s_tokens[address(tPool)] = tokenAddress;
        emit PoolCreated(tokenAddress, address(tPool));
        return address(tPool);
    }
```

Recommended Mitigation:

```
        string memory liquidityTokenName = string.concat("T-Swap ",
 ↪  IERC20(tokenAddress).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

Description: 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.

- Found in src/PoolFactory.sol Line: 35
- Found in src/TSwapPool.sol Line: 52
- Found in src/TSwapPool.sol Line: 57
- Found in src/TSwapPool.sol Line: 62

[I-5] Lacking zero-check in `TSwapPool::constructor`

Recommended Mitigation:

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

[I-6] Unused variable in `TSwapPool::deposit`

Description: Unused local variable `poolTokenReserves`.

Impact: Wasting gas.

Proof of Concept:

```
Warning (2072): Unused local variable.
   --> src/TSwapPool.sol:135:13:
    |
```

```
135 |             uint256 poolTokenReserves =
 ↪  i_poolToken.balanceOf(address(this));
    |             ^^^^^^^^^^^^^^^^^^^^^^^^
```

Recommended Mitigation: Removing it.

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

[I-7] Use of Hard-Coded Magic Numbers in TSwapPool, reducing Readability and Maintenance Difficulty.

Description: The `TSwapPool` contract uses hard-coded "magic numbers" without defining them as named constants. This reduces code readability and makes maintenance error-prone, as the purpose of these values (e.g., 10 as max swaps, 1e18 as 1 WETH) is unclear without context.

Proof of Concept:

```
...
@>      uint256 inputAmountMinusFee = inputAmount * 997;
        uint256 numerator = inputAmountMinusFee * outputReserves;
@>      uint256 denominator = (inputReserves * 1000) + inputAmountMinusFee;
        return numerator / denominator;
    }

    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
        return
@>          ((inputReserves * outputAmount) * 10000) /
@>          ((outputReserves - outputAmount) * 997);
    }
...
```

Recommended Mitigation: Define named constants.

[I-8] `SWapPool::swapExactInput: public` functions not used internally could be marked `external`

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

Recommended Mitigation:

```
    function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
-       public
+       internal
```

[I-9] Missing deadline param in natspec of `TSwapPool::swapExactOutput`

Recommended Mitigation:

```
    /*
     * @notice figures out how much you need to input based on how much
     * output you want to receive.
     *
     * Example: You say "I want 10 output WETH, and my input is DAI"
     * The function will figure out how much DAI you need to input to get
 ↪  10 WETH
     * And then execute the swap
     * @param inputToken ERC20 token to pull from caller
     * @param outputToken ERC20 token to send to caller
     * @param outputAmount The exact amount of tokens to send to caller
+    * @param deadline The deadline for the transaction to be completed by
     */
    function swapExactOutput(
```

[I-10] `SWapPool::totalLiquidityTokenSupply`: `public` functions not used internally could be marked `external`

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

Recommended Mitigation:

```
-   function totalLiquidityTokenSupply() public view returns (uint256) {
+   function totalLiquidityTokenSupply() external view returns (uint256) {
        return totalSupply();
    }
```