AUDIT

T-Swap Protocol

Security Audit Report

Version 1.0

PREPARED BY

Cyfrin.io

LEAD AUDITOR

Yusuf

DATE

February 11, 2026

Table of Contents

1. Protocol Summary

TSwap is a decentralized exchange (DEX) protocol implementing an automated market maker (AMM) model. Users can create liquidity pools pairing any ERC20 token with WETH, provide liquidity to earn fees, and swap tokens using a constant product formula (x * y = k).

2. Disclaimer

The Yusuf 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.

3. Risk Classification

We use the CodeHawks severity matrix to determine severity.

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

4. Audit Details

• Commit Hash: TBD

• Audit Date: February 11, 2026

4.1 Scope

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

4.2 Roles

• **Owner:** Deployer of PoolFactory, responsible for deploying new token/WETH pools.

• **Liquidity Provider:** Deposits WETH and pool tokens to earn liquidity tokens and trading fees.

• **Swapper:** Trades between WETH and pool tokens using swapExactInput or swapExactOutput.

5. Executive Summary

5.1 Issues Found

Severity	Count
High	3
Medium	0
Low	3
Informational	5
Gas	2
Total	13

6. Findings

6.1 High Severity

[H-1] TSwapPool::deposit ignores the deadline parameter, allowing transactions to execute after the deadline

Description: The deposit function accepts a deadline parameter and even applies the revertIfDeadlinePassed modifier to other functions, yet it never applies it to deposit. This means a user's deposit transaction can sit in the mempool indefinitely and execute at any future time, regardless of market conditions.

```
function deposit(
    uint256 wethToDeposit,
    uint256 minimumLiquidityTokensToMint,
    uint256 maximumPoolTokensToDeposit,
    uint64 deadline // @audit never enforced
)
    external
    revertIfZero(wethToDeposit)
    // missing: revertIfDeadlinePassed(deadline)
    returns (uint256 liquidityTokensToMint)
```

Impact: A liquidity provider who sets a deadline expecting their transaction to expire may still have their deposit executed at an unfavorable time – for example after a significant price movement – resulting in unexpected losses.

Proof of Concept:

1. User calls deposit with a deadline of block.timestamp + 1 hour.
2. Transaction gets stuck in the mempool due to low gas.
3. 2 hours pass and the pool price has shifted significantly.
4. Transaction is included. Deadline is never checked. Deposit executes at a bad price.

Recommended Mitigation: Apply the existing revertIfDeadlinePassed modifier to the deposit function.

```
function deposit(...)
    external
    revertIfZero(wethToDeposit)
+   revertIfDeadlinePassed(deadline)
    returns (uint256 liquidityTokensToMint)
```

[H-2] TSwapPool::getInputAmountBasedOnOutput uses wrong fee constant, charging ~91% instead of 0.3%

Description: The getInputAmountBasedOnOutput function uses 10000 as the fee multiplier instead of the correct 1000, resulting in a fee that is 10x too large. This causes users calling swapExactOutput to be charged approximately 90.3% in fees rather than the intended 0.3%.

```
// In getOutputAmountBasedOnInput (correct):
uint256 inputAmountMinusFee = inputAmount * 997;
uint256 denominator = (inputReserves * 1000) + inputAmountMinusFee;

// In getInputAmountBasedOnOutput (WRONG):
return
    ((inputReserves * outputAmount) * 10000) / // @audit should be 1000
    ((outputReserves - outputAmount) * 997);
```

Impact: Any user calling swapExactOutput is massively overcharged. This effectively drains user funds on every output-based swap, making the function financially harmful to use.

Proof of Concept:

1. User calls swapExactOutput wanting 10 WETH output.
2. getInputAmountBasedOnOutput is used to calculate required input.
3. Due to the 10000 multiplier, the required input is ~10x larger than it should be.
4. User is charged ~91% of their input as fees instead of 0.3%.

Recommended Mitigation:

```
return
-   ((inputReserves * outputAmount) * 10000) /
+   ((inputReserves * outputAmount) * 1000) /
    ((outputReserves - outputAmount) * 997);
```

[H-3] TSwapPool::_swap breaks the x*y=k invariant by sending bonus tokens every 10 swaps

Description: The _swap function includes an undocumented incentive mechanism that sends 1e18 extra output tokens to the caller on every 10th swap. This extra transfer is not accounted for in the AMM pricing formula, directly violating the constant product invariant x * y = k.

```
swap_count++;
if (swap_count >= SWAP_COUNT_MAX) {
    swap_count = 0;
    // @audit 1e18 tokens sent outside the invariant calculation
    outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000);
}
```

Impact: Every 10 swaps, the pool loses 1e18 tokens not priced into the swap. Over time this:

• Drains funds from liquidity providers.

• Breaks the pool price ratio, enabling manipulation.

• Eventually causes pool insolvency.

Recommended Mitigation: Remove the bonus mechanism entirely.

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

6.2 Low Severity

[L-1] TSwapPool::_addLiquidityMintAndTransfer emits LiquidityAdded with swapped parameters

Description: The LiquidityAdded event is emitted with poolTokensToDeposit and wethToDeposit in the wrong order. The event signature expects wethDeposited first, then poolTokensDeposited.

```
// Event definition:
event LiquidityAdded(
    address indexed liquidityProvider,
    uint256 wethDeposited,
    uint256 poolTokensDeposited
);

// Actual emission (backwards):
emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
```

Impact: Off-chain tools, indexers, and frontends reading this event will record incorrect deposit amounts.

Recommended Mitigation: Swap the parameter order in the emit statement.

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

[L-2] TSwapPool::swapExactInput never assigns the named return variable, always returning 0

Description: The function declares a named return variable output but never assigns it before the function ends. The actual computed outputAmount is passed to _swap and transferred correctly, but the return value is silently discarded.

```
function swapExactInput(...)
    returns (uint256 output) // declared but never assigned
{
    uint256 outputAmount = getOutputAmountBasedOnInput(...);

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

    _swap(inputToken, inputAmount, outputToken, outputAmount);
    // @audit output is never set -- caller always receives 0
}
```

Impact: Any contract or user relying on the return value will always get 0. The actual token transfer is unaffected, but downstream logic depending on this value will silently malfunction.

Recommended Mitigation: Assign the output variable.

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

[L-3] TSwapPool::sellPoolTokens calls swapExactOutput instead of swapExactInput, mismatching user intent

Description: sellPoolTokens is meant to sell a specific poolTokenAmount. Selling an exact input amount should use swapExactInput. Instead it calls swapExactOutput, treating poolTokenAmount as the desired WETH output amount.

```
function sellPoolTokens(uint256 poolTokenAmount)
    external returns (uint256 wethAmount)
{
    return swapExactOutput(
        i_poolToken,
        i_wethToken,
        poolTokenAmount, // @audit treated as outputAmount, not inputAmount
        uint64(block.timestamp)
    );
}
```

Impact: Users expecting to sell a fixed number of pool tokens will instead have an unpredictable amount of pool tokens pulled from their wallet.

Recommended Mitigation: Change the function to call swapExactInput instead.

```
- function sellPoolTokens(uint256 poolTokenAmount)
-     external returns (uint256 wethAmount)
- {
-     return swapExactOutput(
-         i_poolToken, i_wethToken, poolTokenAmount, uint64(block.timestamp)
-     );
- }
+ function sellPoolTokens(
+     uint256 poolTokenAmount,
+     uint256 minWethToReceive
+ ) external returns (uint256 wethAmount) {
+     return swapExactInput(
+         i_poolToken, poolTokenAmount, i_wethToken,
+         minWethToReceive, uint64(block.timestamp)
+     );
+ }
```

6.3 Informational

[I-1] PoolFactory::PoolFactory__PoolDoesNotExist error is defined but never used

The custom error is declared but never referenced. Dead code increases bytecode size.

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

[I-2] Missing zero address checks in constructors

Both PoolFactory and TSwapPool constructors accept address parameters without validating against address(0). Deploying with a zero address would permanently brick the contract.

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

[I-3] PoolFactory::createPool uses .name() instead of .symbol() for liquidity token symbol

This produces verbose symbols like 'tsDAI Stablecoin' instead of 'tsDAI'.

```
// Current (wrong -- produces "tsDAI Stablecoin"):
- string memory liquidityTokenSymbol =
-     string.concat("ts", IERC20(tokenAddress).name());

// Should be (produces "tsDAI"):
+ string memory liquidityTokenSymbol =
+     string.concat("ts", IERC20(tokenAddress).symbol());
```

[I-4] Events are missing indexed fields

Index event fields make them more quickly accessible to off-chain tools. Each event should use three indexed fields if there are three or more fields.

[I-5] TSwapPool::swapExactInput should be external not public

The function is not called internally and should be marked external to save gas.

6.4 Gas Optimizations

[G-1] TSwapPool::deposit declares an unused poolTokenReserves variable

This storage read wastes gas on every deposit. The line can be safely removed.

```
- uint256 poolTokenReserves = i_poolToken.balanceOf(address(this));
  // ^ never read again
```

[G-2] TSwapPool::totalLiquidityTokenSupply should be external not public

Functions only called externally should be declared external to save gas on calldata copying.

Instances: totalLiquidityTokenSupply, swapExactInput