TSwap Smart Contract Security Audit Report

Philani A Dlamini

August 2025

Contents

1 TSwap Smart Contract Security Audit **1**
1.1 Vulnerability Summary 1
1.2 Table of Contents 1
1.3 High Severity Findings 2
1.4 Low Severity Findings 3
1.5 Informationals 4

1 TSwap Smart Contract Security Audit

1.1 Vulnerability Summary

Severity Level	Count
High	3
Low	2
Informational	6

1.2 Table of Contents

- TSwap Smart Contract Security Audit
  - Vulnerability Summary
  - Table of Contents
  - High Severity Findings
    - [H-1] Reentrancy Risk in `PoolFactory::createPool`
    - [H-2] Missing Deadline Check in deposit `TSwapPool::deposit`
    - [H-3] Incorrect fee calculation in `TSwap::getInputAmountBasedOnOutput`
  - Low Severity Findings
    - [L-1] `TSwap::_addLiquidityMintAndTransfer` function emits `TSwap::LiquidityAdded` with parameters out of order

TSwap Protocol Architecture

Figure 1: TSwap Protocol Architecture

    * [L-2] Default value returned by `TSwapFactory::swapExactInput` function is not updated resulting in an incorrect return value
- Informationals
    * [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist`is not used and should be removed
    * [I-2] Lacking 0 address checks
    * [I-3] `PoolFactory::createPool` should use `symbol()` instead of `name()`
    * [I-4] `PoolFactory::PoolCreated` events is missing an indexed field
    * [I-5] `TSwapFactory::swapExactInput` function is marked as public and not used internally, it should be updated to external
    * [I-6] `TSwapFactory::revertIfZero` modifier has a strict zero check

1.3 High Severity Findings

1.3.1 [H-1] Reentrancy Risk in `PoolFactory::createPool`

Description:
The `createPool` function updates contract state **after** making external calls to retrieve `name()` and `symbol()` from the token contract. This opens the contract to potential **reentrancy attacks**.

Impact:

- **Loss of funds:** A malicious token could exploit this to drain the pool.

Proof of Concept:

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

    emit PoolCreated(tokenAddress, address(tPool));
    return address(tPool);
}
```

Recommended Mitigation:

- **Adopt the Checks-Effects-Interactions (CEI) pattern:** update state before making external calls.

1.3.2 [H-2] Missing Deadline Check in deposit `TSwapPool::deposit`

Description: The `deposit` function takes a `deadline` parameter but never checks it. This allows liquidity deposits to execute after the intended time.

Impact:

- **Unexpected execution:** Deposits could be executed under unfavorable market conditions.

Recommended Mitigation:

Add a revert condition if the deadline has passed.

```
function deposit(uint256 wethToDeposit, uint256 minimumLiquidityTokensToMint,
     uint256 maximumPoolTokensToDeposit, uint64 deadline
) external revertIfZero(wethToDeposit) returns (uint256 liquidityTokensToMint) {
+   if (uint64(block.timestamp) > deadline) revert();
    ...
}
```

1.3.3 [H-3] Incorrect fee calculation in `TSwap::getInputAmountBasedOnOutput`

Description: The fee calculation uses 10000 instead of 1000, leading to overcharging users.

Impact:

- **Overcharging users** on swaps.

Recommended Mitigation:

Replace magic numbers with constants and use the correct scale.

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

1.4 Low Severity Findings

1.4.1 [L-1] `TSwap::_addLiquidityMintAndTransfer` function emits `TSwap::LiquidityAdded` with parameters out of order

Description: When `LiquidityAdded` is emitted in the `_addLiquidityMintAndTransfer` logs value in an incorrect order. The `poolTokensToDeposit` which is the second parameter should be on the third parameter postion and `wethToDeposit` on the second position.

Impact: Event emmision will be incorrect, leading to off-chain functions to potentially malfunction.

Recommended Mitigation:

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

1.4.2 [L-2] Default value returned by `TSwapFactory::swapExactInput` function is not updated resulting in an incorrect return value

Description: The `swapExactInput` function is expected to return the actuall amount of tokens bought by the caller. However, while it declared the named return value `output`, it is never assigned a value nor uses an explicit return statement.

Impact:

- The return value will always be 0, giving incorrect information to the user.

Recommended Mitigation:

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

-        uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount, inputReserves, output
+        output = getOutputAmountBasedOnInput(inputAmount, inputReserves, outputReserves);
        if (outputAmount < minOutputAmount) {
            revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
        }

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

1.5 Informationals

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

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

1.5.2 [I-2] Lacking 0 address checks

`PoolFactory::contructor`

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

`TSwapPool::constructor`

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

1.5.3 [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());
```

1.5.4 [I-4] `PoolFactory::PoolCreated` events is missing an indexed field

```
- event PoolCreated(address tokenAddress, address poolAddress);
+ event PoolCreated(address indexed tokenAddress, address poolAddress);
```

1.5.5 [I-5] `TSwapFactory::swapExactInput` function is marked as public and not used internally, it should be updated to external

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

1.5.6 [I-6] `TSwapFactory::revertIfZero` modifier has a strict zero check

```
 modifier revertIfZero(uint256 amount) {
-       if (amount == 0) {
+       if (amount <=  0) {
            revert TSwapPool__MustBeMoreThanZero();
        }
        _;
    }
```