T-Swap Audit Report

Author

Rubén Cruz

Date

May 9, 2025

Protocol Description

The T-Swap protocol is a decentralized exchange (DEX) system that uses Automated Market Makers (AMMs) to facilitate token swaps on the blockchain. Its main function is to allow users to trade tokens efficiently without relying on a centralized intermediary, using liquidity pools.

Key Features of T-Swap:

1.Liquidity Pools:

-Liquidity providers (LPs) deposit token pairs into pools, enabling others to swap tokens.

-In return, they earn trading fees and native token rewards.

2.AMM Model:

-Uses a mathematical formula (e.g., x * y = k) to determine prices and execute swaps automatically.

3.Fees & Rewards:

-Charges a small fee per swap (e.g., 0.3%), distributed to LPs.

-May include additional incentives like governance tokens or staking rewards.

4.Slippage Reduction:

-Optimizes transactions to minimize price impact for large trades.

5.Decentralized Governance:

-In some cases, native token holders can vote on protocol changes (fees, supported pools, etc.).

Primary Goal: To provide an efficient, secure, and decentralized exchange for token trading while incentivizing liquidity providers and traders.

Issues found

Severity	Number of issues found
High	4
Medium	1

Severity	Number of issues found
Low	2
Info	3
Total	10

Highs

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many token form users, resulting in lost fees.

Description:

The `TSwapPool::getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens a user should send to the pool given an amount of output tokens. However, the current functions misscalculates the resulting amount. When calculating the fee, it scales the amount by 10 000, instead of 1 000

Impact:

The impact of this vulnerability is that the protocol will take much more fees than expected from users.

Proof of Concept:

Recommendations

```
    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 in `TSwapPool::swapExactOutput`, causing users to potentially receive way fewer tokens

Description:

The `TSwapPool::swapExactOutput` function does not inlcude any slippage protection. This function is similar to `TSwapPool::swapExactInput`, where the function specifies a `minOutputAmount`. The `TSwapPool::swapExactOutput` function should specify a `maxInputAmount`.

Impact:

If market conditions change before the transaction happens, the user can receive way fewer tokens than expected.

Proof of Concept:

1. WETH price is 1,000 USDC.
2. User calls `TSwapPool::swapExactOutput` looking for 1 WETH.
3. The function does not offer a maxInputAmount.
4. As the transaction is pending on the mempool, the markets changes and the price moves to 1 WETH = 1,100 USDC.
5. The user receives 1 WETH, but they paid 1,100 USDC.

** Recomendations**

```
    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 `TSwapPool::sellPoolTokens` function is intended to sell pool tokens for WETH. Users indicate how many pool tokens they are willing to sell in the `poolTokenAmount` parameter. However, the function currently miscalculates the swapped amount.

This is due to the fact that the `swapExactOutput` function is called, whereas the function called should be `swapExactInput`.

Impact:

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

Proof of Concept:

Recommended Mitigation:

Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note: this would also require to change `sellPoolTokens` 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)
-            );
+            swapExactInput(
+                i_poolToken,
+                poolTokenAmount,
+                i_wethToken,
+                minWethToReceive,
+                uint64(block.timestamp)
+            );
    }
```

[H-4] In `TSwapPool::_swap`, the extra tokens given to users after every `swapCount` breaks the protocol invariant of `x * y = 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`: constant product of two balances.

This means that every time the balances change in the protocol, the ratio between the two amounts remains the constant. However, this is broken due to the extra incentive in the swap function, meaning that overtime protocol funds will be drained.

The following block of code is the responsible:

```
     swap_count++;
        if (swap_count >= SWAP_COUNT_MAX) {
            swap_count = 0;
```

```
            outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000);
        }
```

Impact:

Protocol invariant is broken.

Proof of Concept:

1. A user swaps 10 times, collecting the extra incentive.
2. That user continues to swap until the protocol funds are drained.

▶ Details

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));
        vm.stopPrank();

        int256 startingY = int256(weth.balanceOf(address(pool)));
        int256 expectedDeltaY = int256(-1) * int256(outputWeth);
```

```
        uint256 endingY = weth.balanceOf(address(pool));
        int256 actualDeltaY = int256(endingY) - int256(startingY);

        assertEq(actualDeltaY, expectedDeltaY);
    }
```

Recommende Mitigation:

Remove the extra incentive. If the desire is to keep it, then accounting to the change in the x * y = K invariant is mandatory.

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

Mediums

[M-1] `TSwapPool::deposit` function is missing deadline checks, allowing after-deadline deposits

Description:

The `TSwapPool::deposit` function accepts a `deadline` parameter, which according to the documentation should be "The deadline for the transaction to be executed". However, this parameter is never use, which can provoke the adding of liquidity at non-desired times.

Impact:

The impact of this vulnerability is that an attacker can add liquidity at non-desired times, when market conditions are not optimal to deposit.

Proof of Concept:

The `deadline` parameter is never used in the function.

Recommended Mitigation:

Consider making the following change to the function:

```
    function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(uint64 deadline)
```

```
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

Lows

[L-1] `TSwapPool::LiquidityAdded` event has parameters out of order

Description:

When `TSwapPool::LiquidityAdded` event is emitted after `TSwapPool::_addLiquidityMintAndTransfer`, the parameters given are out of order, giving an incorrect log. The parameters should be `wethDeposited` should go in the second parameter position, whereas `poolTokensDeposited` should go third.

Impact:

Incorrect event emission, which can lead to off-chain tools to show incorrect information or to malfunction.

Proof of Concept:

The `deadline` parameter is never used in the function.

Recommended Mitigation:

```
-    event LiquidityAdded(
-        msg.sender,
-        poolTokensDeposited,
-        wethDeposited,
-    );
+    event LiquidityAdded(
+        msg.sender,
+        wethDeposited,
+        poolTokensDeposited
+    );
```

[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 it declares the named return value `output` it is never assigned a value, nor uses an explicit return statement.

Impact:

The returned value will always be 0, giving incorrect information to the caller.

Proof of Concept:

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

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

Informationals

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

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

[I-2] Lacking zero address checks

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

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

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