T-Swap Audit Report

Version 1.0

June 5, 2024

T-Swap Audit Report

Nanachiki

Jun 5, 2024

Auditor: - Nanachiki

Table of Contents

Protocol Summary

T-Swap is a decentralized exchange(DEX) designed to allow users to swap assets permissionlessly at fair prices. As an Automated Market Maker(AMM), T-Swap does not use a traditional order book. Instead, it relies on "Pools" of assets to facilitate trades. This model is similar to that of Uniswap.

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 correspond to the following commit hash:

```
e643a8d4c2c802490976b538dd009b351b1c8dda
```

Scope

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

- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum
- Tokens:
    - Any ERC20 token

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	2
Medium	2
Low	2
Info	9
Total	15

Findings

High

[H-1] Missing a deadline check in the `TSwapPool::deposit` function causes the transaction to be executed even after the deadline

Description: Although the `deadline` parameter is present in the `TSwapPool::deposit` function, it's not used anywhere in the code. If a user selects a low transaction fee that miners are not incentivized to include the user's transaction in a block, it could remain in the mempool and be executed much later than the user expects, causing the transaction to complete at an unfavorable market price.

Impact: The transaction could not be executed even after the deadline has passed until the gas price becomes reasonable for miners.

Proof of Concept:

1. A user deposits `1 ETH` and `100 DAI` to add liquidity to a pool.
2. The transaction is submitted and remains in the mempool for a long time because it doesn't revert.
3. A few hours later, the gas price finally drops enough to process it; however, it is executed at a sub-optimal rate, such as `1 ETH` being worth `10 DAI`.

Recommended Mitigation: Consider adding the deadline check in the same way the protocol does in other functions.

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint) {}
```

[H-2] The `TSwapPool::getInputAmountBasedOnOutput` function is miscalculating fee amount, resulting in users being charged too many fees

Description: The `TSwapPool::swapExactOutput` function is one of the main functions using the `TSwapPool::getInputAmountBasedOnOutput` function to determine the input amount for a specified output amount in exchange. However, due to the incorrect fee calculation, every time users swap tokens using `swapExactOutput`, the protocol will have to charge too many fees from them.

```
@>    return ((inputReserves * outputAmount) * 10000) / ((
    outputReserves - outputAmount) * 997);
```

Impact: Trading based on `swapExactOutput`, which utilizes `getInputAmountBasedOnOutput`, charges about 10 times larger fees, breaking the protocol's functionality.

Proof of Concept:

1. A liquidity provider supplies `100e18 WETH` and `100e18 pool tokens` as liquidity.
2. A swapper sends `10e18 pool tokens` as the output token in exchange for a WETH amount as the input token using the `swapExactOutput` function.
3. As a result, the swapper has paid a 10 times higher price with excessive fees

Proof of Code:

Place the following into `TSwapPool.t.sol`.

Code

```

function test_usersHaveToPayExtraFeesToSwapWithSwapExactOutput() public
     {
        // Swapper setup
        address swapper = makeAddr("swapper");
        weth.mint(swapper, 200e18);
        poolToken.mint(swapper, 200e18);

        // Add liquidity
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), wethToDeposit);
        poolToken.approve(address(pool), poolTokensToDeposit);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        uint256 inputTokenAmountStart = poolToken.balanceOf(swapper);
        uint256 inputReserves = poolToken.balanceOf(address(pool));
        uint256 outputReserves = weth.balanceOf(address(pool));
        uint256 outputAmount = 10e18;

        // Swap
        vm.startPrank(swapper);
        poolToken.approve(address(pool), type(uint256).max);
        uint256 expectedAmount = ((inputReserves * outputAmount) *
            1000) / ((outputReserves - outputAmount) * 997);

        // Because there is no maximum limit for the input amount in
            the function, it does not revert and the swapper
        // loses too many funds
        pool.swapExactOutput(poolToken, weth, outputAmount, uint64(
            block.timestamp));
        vm.stopPrank();

        uint256 inputTokenAmountAfter = poolToken.balanceOf(swapper);
        uint256 actualAmount = inputTokenAmountStart -
            inputTokenAmountAfter;

        console.log("The expected amount: ", expectedAmount); //
            11,144,544,745,347,152,568
        console.log("The actual amount  : ", actualAmount);   //
            111,445,447,453,471,525,688

        assert(actualAmount > expectedAmount);
    }
```

Recommended Mitigation:

Consider revising the following line in the `getInputAmountBasedOnOutput` function to correct fee calculation.

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

Medium

[M-1] `TSwapPool::swapExactOutput` is used for a return value of the `TSwapPool::sellPoolTokens` function, resulting in the swap being executed with incorrect values

Description: `sellPoolTokens` is intended to be used for trading pool tokens for WETH and returning the WETH amount. However, due to the use of `swapExactOutput`, which is designed to return an input value, `sellPoolTokens` returns an amount of pool tokens sold instead of an amount of WETH received, which is the correct value the function should return.

```
                                                  // Wrong argument
return swapExactOutput(i_poolToken, i_wethToken, poolTokenAmount,
    uint64(block.timestamp));
```

Additionally, `swapExactOutput` is called with the wrong argument `poolTokenAmount`, meaning the `poolTokenAmount` will be incorrectly specified as the amount of WETH that users end up receiving.

Impact: The `sellPoolTokens` completes a swap and returns pool token amounts based on the input of pool tokens, instead of the expected WETH amount, thus breaking the intended functionality.

Recommended Mitigation:

Consider using the `TSwapPool::swapExactInput` function for the swap execution and the `TSwapPool::getOutputAmountBasedOnInput` for the `wethAmount` to return a correct value.

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

+          wethAmount = getOutputAmountBasedOnInput(poolTokenAmount,
    i_poolToken.balanceOf(this), i_wethToken.balanceOf(this));
+         swapExactInput(i_poolToken, poolTokenAmount, i_wethToken,
     minWethAmountToReceive, uint64(block.timestamp));
+         return wethAmount;
      }
```

[M-2] Rebase, fee-on-transfer, and ERC777 break protocol invariant

Description: The protocol allows users to receive the incentive tokens after every 10 times token swaps, as described in the documentation for `TSwap::_swap`: `Every 10 swaps, we give the caller an extra token as an extra incentive to keep trading on T-Swap`. However, this disrupts the x * y = k protocol invariant.

Impact: The incentive feature could prompt malicious users to engage in excessive trading solely to earn the incentive tokens over time until the pool is drained.

Proof of Concept: 1. A user wants to trade 1e18 pool tokens using `swapExactInput` 10 times. 2. According to the `getOutputAmountBasedOnInput` function, an output amount of WETH per trade should be `8.317e17`. 3. To maintain the protocol invariant, the output amount must remain constant regardless of the number of swaps. 4. However, after 10 trades, the pool sends out the extra `1_000_000_000_000_000_000` WETH as the incentive, breaking the `x * y = k` formula and violating the protocol invariant.

Proof Of Code

```
function test_tokenSwapsBreakProtocolInvariant() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), type(uint256).max);
        poolToken.approve(address(pool), type(uint256).max);
        pool.deposit(100e18, 100e18, 100e18, 0);

        uint256 inputPoolTokens = 1e18;
        vm.startPrank(user);
        poolToken.mint(user, 100e18);
        poolToken.approve(address(pool), type(uint256).max);

        // swapExactInput() * 9
        for (uint256 i = 0; i < 9; i++) {
            pool.swapExactInput(poolToken, inputPoolTokens, weth, 8e17,
                type(uint64).max);
        }

        int256 expectedOutputAmount = (-1)
            * int256(
                pool.getOutputAmountBasedOnInput(
                    inputPoolTokens, poolToken.balanceOf(address(pool))
                        , weth.balanceOf(address(pool))
```

```
                  )
              );
          int256 startPoolTokenBalance = int256(weth.balanceOf(address(
             pool)));
          pool.swapExactInput(poolToken, inputPoolTokens, weth, 8e17,
             type(uint64).max);
          vm.stopPrank();

          int256 endPoolTokenBalance = int256(weth.balanceOf(address(pool
             )));
          int256 actualOutputAmount = endPoolTokenBalance -
             startPoolTokenBalance;

          console.logInt(expectedOutputAmount); //
             -831,762,937,399,464,797
          console.logInt(actualOutputAmount); //
             -1,831,762,937,399,464,797
          assert(expectedOutputAmount == actualOutputAmount);
      }
```

Recommended Mitigation:

Consider removing the incentive feature to maintain the protocol invariant, or set aside tokens for incentives in a similar way that the protocol implemented for fees.

```
-    uint256 private swap_count = 0;
-    uint256 private constant SWAP_COUNT_MAX = 10;
.
.
.
-        swap_count++;
-        if (swap_count >= SWAP_COUNT_MAX) {
-            swap_count = 0;
-            outputToken.safeTransfer(msg.sender, 1
   _000_000_000_000_000_000);
-        }
```

Low

[L-1] The `TSwapPool::LiquidityAdded` event for `TSwapPool::_addLiquidityMintAndTransfer` is emitted with the values in the wrong order

Description: Whenever we try to add liquidity to a pool, the `TSwapPool::_addLiquidityMintAndTransfer` function is called and the `LiquidityAdded` event is emitted. However, the second parameter, `poolTokensToDeposit`, and the third parameter, `wethToDeposit`, are in reverse order.

Impact: The event logs incorrect values.

Recommended Mitigation: Make sure to emit the event with the parameters in the correct order.

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

[L-2] Output parameter from the `TSwapPool::swapExactInput` function is unused and returns 0

Description: `swapExactInput` is designed to return an output amount. However, the return parameter `output` in `swapExactInput` is never used and the function just returns zero value as the default value of uint256.

Impact: The `swapExactInput` gives users incorrect information

Recommended Mitigation:

Example 1: Consider using the `TSwapPool::getOutputAmountBasedOnInput` function to assign an output Amount parameter.

```diff
function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (
-            uint256 output
+            uint256 outputAmount
        )
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

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

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

        _swap(inputToken, inputAmount, outputToken, outputAmount);

+       return outputAmount;
    }
```

Example 2: Add a return statement.

```
function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (
            uint256 output
        )
    {
        .
        .
        uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount,
            inputReserves, outputReserves);
+       return outputAmount;
    }
```

Informational

[I-1] Event is missing `indexed` fields

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: 43
- Found in src/TSwapPool.sol Line: 44
- Found in src/TSwapPool.sol Line: 45

[I-2] Unused error message should be removed

The `PoolFactory::PoolFactory__PoolDoesNotExist` error message is never emitted and should be removed.

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

[I-3] Missing zero address checks

Assigning values to state variables without checking for address(0) in the TSwapPool and PoolFactory constructors.

- Found in src/PoolFactory.sol Line: 43
- Found in src/TSwapPool.sol Line: 77

[I-4] The use of the name function for `PoolFactory::liquidityTokenSymbol` in `PoolFactory::createPool` should be replaced with the `symbol` function.

It seems to be better to use the `symbol` function to get the correct liquidity token symbol for a pool.

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

[I-5] The `TSwapPool::MINIMUM_WETH_LIQUIDITY` doesn't need to be emitted because it's a constant value

The `TSwapPool::MINIMUM_WETH_LIQUIDITY` value is a constant value that anyone can look up from the `TSwapPool::getMinimumWethDepositAmount` function and therefore doesn't need to be emitted.

```
-         if (wethToDeposit < MINIMUM_WETH_LIQUIDITY) {
-                     revert TSwapPool__WethDepositAmountTooLow(
    MINIMUM_WETH_LIQUIDITY, wethToDeposit);
-                 }
```

[I-6] Avoid using magic numbers

Magic numbers can be difficult to understand and maintain. Please consider using constants instead to improve code redability and maintainability.

Examples:

```
uint256 inputAmountMinusFee = inputAmount * 997;
uint256 denominator = (inputReserves * 1000) + inputAmountMinusFee;
return ((inputReserves * outputAmount) * 10000) / ((outputReserves -
    outputAmount) * 997);
outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000);
return getOutputAmountBasedOnInput(1e18, i_wethToken.balanceOf(address(
    this)), i_poolToken.balanceOf(address(this)));
return getOutputAmountBasedOnInput(1e18, i_poolToken.balanceOf(address(
    this)), i_wethToken.balanceOf(address(this)));
```

It is recommended to assign constant variables such as:

```
uint256 constant FEE_AMOUNT_ONE = 997;
uint256 constant FEE_AMOUNT_TWO = 1000;
uint256 constant INCENTIVE_AOUTPUT_TOKEN_AMOUNT =  1
    _000_000_000_000_000_000;
uint256 constant ONE_INPUT_TOKEN_AMOUNT = 1e18;
```

[I-7] Missing comments for function details

There are no comments explaining the function details for the `TswapPool::swapExactInput` function. Additionally, the comments for the`TswapPool::swapExactOutput` function are missing an explanation for the deadline parameter.

Consider adding corresponding comments for each function.

Comments about function details:

```
// Missing comments here
function swapExactInput(
        IERC20 inputToken, uint256 inputAmount, IERC20 outputToken,
            uint256 minOutputAmount, uint64 deadline
    )
```

Deadline parameter:

```
    /*
     * @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
@>    * @audit missing deadline parameter
      */

     function swapExactOutput(
         IERC20 inputToken,
         IERC20 outputToken,
         uint256 outputAmount,
         uint64 deadline
     )
```

Gas

[G-1] Unused local variable should be removed

The local variable `poolTokenReserves` in the `TSwapPool::deposit` function is not used in the code and should be removed for gas efficiency.

```
function deposit(
         uint256 wethToDeposit,
         uint256 minimumLiquidityTokensToMint,
         uint256 maximumPoolTokensToDeposit,
         uint64 deadline
     )
         external
         revertIfZero(wethToDeposit)
         returns (uint256 liquidityTokensToMint)
     {
         if (wethToDeposit < MINIMUM_WETH_LIQUIDITY) {
             revert TSwapPool__WethDepositAmountTooLow(
                 MINIMUM_WETH_LIQUIDITY, wethToDeposit);
         }

         if (totalLiquidityTokenSupply() > 0) {
             uint256 wethReserves = i_wethToken.balanceOf(address(this))
                 ;

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

[G-2] The `TSwapPool::swapExactInput` and `TSwapPool::totalLiquidityTokenSupply` functions can be external functions

The `TSwapPool::swapExactInput` and `TSwapPool::totalLiquidityTokenSupply` functions are not used internally in the codebase, so both of them can be made as external functions to reduce gas costs.

```
function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
@>      public // -> external
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 output){}
```

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