Protocol Audit Report

Prepared by: Bizarro

Table of Contents

Protocol Summary

This project is meant to be a permissionless way for users to swap assets between each other at a fair price. You can think of T-Swap as a decentralized asset/token exchange (DEX). T-Swap is known as an Automated Market Maker (AMM) because it doesn't use a normal "order book" style exchange, instead it uses "Pools" of an asset. It is similar to Uniswap. To understand Uniswap, please watch this video: Uniswap Explained

Disclaimer

Bizarro found 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

		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 the following commit hash:

```
1ec3c30253423eb4199827f59cf564cc575b46db
```

Scope

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

Roles

Executive Summary

Issues found

Severity	**Number of issues found**
High	5

Severity	Number of issues found
Medium	0
Low	3
Info	5
Gas	
Total	13

Findings

High

[H-1] `TSwapPool::deposit` is missing a deadline check, causing the user to add liquidity even after the deadline.

Description: The `deposit` function has a deadline, but it doesn't tell you if the deadline was met. This could be a problem because if you add liquidity after the deadline, it might cost more or less than you expected, which isn't fair for everyone.

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

Proof of Concept: the `deadline` parameter is unused

Recommended Mitigation:

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

[H-2] `TSwapPool::getInputAmountBasedOnOutput` is returning wrong valur of inputAmount beacuse of incorrect fee calculation.

Description: The `getInputAmountbasedOnOutput` function calculates the input amount required to obtain a desired output amount, deducting a 0.3% fee for liquidity providers. However, the current implementation contains an error where the value 10_000 should be replaced with 1_000 as specified in the

documentation. This discrepancy could lead to inaccurate calculations and potential financial losses for users.

Impact: The Protocol takes more fees than usual from users.

Recommended Mitigation:

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
-         return ((inputReserves * outputAmount) * 10_000) /
((outputReserves - outputAmount) * 997);
+         return ((inputReserves * outputAmount) * 1_000) /
((outputReserves - outputAmount) * 997);

    }
```

[H-3] Lack of slippage protection from `TSwapPool::SwapExactOutput` causes user to receive way fewer tokens

Description: The `SwapExactOutput` function does not include any sort of slippage protection. This function is similar to what is done in `TSwapPool::SwapExactOutput`, where the function specifies a `minOutputAmount`, the SwapExactOutput function should specify a maxInputAmount.

Impact: If market conditions change before the transaciton processes, the user could get a much worse swap.

Proof of Concept:

1. The price of 1 weth right now is 1,000 USDC
2. User inputs a swapExactOuput looking for 1 weth inputToken = USDC outputToken = WETH outputAmount = 1 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-4] `TSwapPool::seelPoolTokens` is calling the `swapExactOuptut` with wrong parameters causing users to receive incorrect amount of tokens.

Description: the `sellPoolTokens` function is intended 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 miscalculates the swapped amout.

This is due to the fact that the `swapExactOutpt` function is called, whereas the swapExactInput function is the one that should be called, because users sepcify the exact amount of input tokens, not output.

Impact: users will sell wrong amount of tokens, which is severe disruption of protocol functionality.

Proof of Concept:

Recommended Mitigation:

```
    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-5] `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 pool token y: the balance of weth k: 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 the 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

```
    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);
        for(uint256 i=0; i<9; i++){
                    pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        }
        int256 startingY = int256(weth.balanceOf(address(pool)));
        int256 expectedDeltaY = int256(-1) * int256(outputWeth);

        pool.swapExactOutput(poolToken, weth, outputWeth,
uint64(block.timestamp));
        vm.stopPrank();

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

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 so the 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);
-        }
```

Low

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

Description: When `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTransfer` function the order of parameter in the LiquidityAdded event is incorrect, the 2nd parameter and the 3rd parameter should be swapped.

Impact: Event emission is incorrect, leading to offchain functions to potentailly malfunciton.

Recommended Mitigation:

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

[L-2] The Return value in `TSwapPool::swapExactInput` is incorrect

Description: The `TSwapPool::swapExactInput` function is expected to return a uint256 representing the `output` amount, but it currently does not have a return statement. This discrepancy could lead to unexpected behavior or potential vulnerabilities.

Impact: The return value will always be 0, giving incorrect information to the caller.

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);
```

Informational

[I-1] Lacking of zero address check in constructor

Description: In the `PoolFactory::constructor` the `i_wethToken` is directly getting selected as the parameter, causing the i_wethToken to be zero.

Recommended Mitigation:

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

[I-2] `PoolFactory::createPool` Should Use `.symbol()` rather than `.name()`

Description: In `PoolFactory::createPool` the function is creating a liquidity token symbol using the `.name()` function of IERC20 token contract. rather than using the .name funciton the protocol should use .symbol function.

Recommended Mitigation:

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

[I-3] In `TSwapPool::deposit` function poolTokenReserves variable is not getting used.

Recommended Mitigation:

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

[I-4] In `TSwapPool::deposit` the CEI is not getting followed

Description: liquidityTokensToMint is getting updated after the external contract calls.

Recommended Mitigation:

```
+       liquidityTokensToMint = wethToDeposit;
        _addLiquidityMintAndTransfer(wethToDeposit,
maximumPoolTokensToDeposit, wethToDeposit);
-       liquidityTokensToMint = wethToDeposit;
```

[I-5] the `TSwapPool::swapExactInput`, `TSwapPool::swapExactOutput` and `SwapPool::totalLiquidityTokenSupply` function should be external rather than public.