TSwap Protocol Audit Report

Version 1.0

Mohan

January 7, 2024

TSwap Protocol Audit Report

Mohan

January 07, 2024

Prepared by: Mohan Lead Auditors: - Mohan

Table of Contents

    * [H-5] In `TSwapPool::_swap` the extra tokens given to users after every swapCount breaks the protocol invariant of x * y = k

- Low
    * [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order
    * [L-2] `TSwapPool::swapExactInput` results in incorrect return value given
- informational
    * [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed
    * [I-2] Lacking zero address checks
    * [I-3] `PoolFactory.createPool` should use `.symbol()` instead of `.name()`
    * [I-4] Event is missing `indexed` fields
    * [I-5] MINIMUM_WETH_LIQUIDITY is a constant and therefore not required to be emitted
    * [I-6] In `TSwapPool::deposit` function It would be better if we update `liquidityTokensToMint` before the '_addLiquidityAndTransfer call to follow CEI
    * [I-7] In `TSwapPool::PoolgetOutputAmountBasedOnInput` use magic numbers instead of constants
    * [I-8] Add the natspec to `TSwapPool::swapExactInput` function
    * [I-9] The function in `TSwapPool::swapExactInput` should be marked as external instead of public as this functions are not using interally to reduce gas
    * [I-10] make the visibility of the function `totalLiquidityTokenSupply` external external instead of public as it is not being used internally to reduce gas
    * [I-11] missing deadline param in natspec for the function `swapExactOutput`
- Gas
    * [G-1] In `TSwapPool::deposit` function `poolTokenReserves` is not being used so remove this line and save gas

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

I make all efforts 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.

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

Scope

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

Roles

Liquidity Providers: In order for the system to work, users have to provide liquidity, aka, “add tokens into the pool”.

Users : Users who want to swap tokens

Executive Summary

Issues found

Severity	Number of issues found
High	5
Medium	0
Low	2
Info	11
Gas	1
Total	19

Findings

High

[H-1] `TSwapPool::deposit` is missing deadline check causing transactions to complete even after the deadline

Description: The `deposit` function accepts deadline parameter which according to the documentation is "The deadline for the transaction to be completed by". However, this parameter is never used.As a consequence, operations that add liquidity to the pool might be unexpected times, in market conditions where the deposit rate is unfavorable.

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: Consider making the following change to the function.

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

```
8           revertIfZero(wethToDeposit)
9           returns (uint256 liquidityTokensToMint)
10      {
```

[H-2] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users,resulting in lost fees

Description: The `getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens a user should deposit given an amount of tokens of output tokens of output tokens. However, the function currently miscalculates the resulting amount. When calculating fee, it scales the amount by 10_000 instead of 1_000

Impact: Protocol takes more fees than expected from users.

Proof of Concept: Add this test suite to the `TSwapPool.t.sol`

```
1  function testFlawedSwapExactOutput() public{
2          uint256 initialLiquidity = 100e18;
3          vm.startPrank(liquidityProvider);
4          weth.approve(address(pool),initialLiquidity);
5          poolToken.approve(address(pool),initialLiquidity);
6
7          pool.deposit({
8              wethToDeposit: initialLiquidity,
9              minimumLiquidityTokensToMint: 0,
10             maximumPoolTokensToDeposit: initialLiquidity,
11             deadline: uint64(block.timestamp)
12         });
13         vm.stopPrank();
14
15         // User has 11 pool tokens
16         address someUser = makeAddr("someUser");
17         uint256 userInitialPoolTokenBalance = 11e18;
18         poolToken.mint(someUser,userInitialPoolTokenBalance);
19         vm.startPrank(someUser);
20
21         // Users buys i WETH from the pool, paying the pool tokens
22         poolToken.approve(address(pool),type(uint256).max);
23         pool.swapExactOutput(poolToken,weth,1 ether, uint64(block.
               timestamp));
24
25         // Initial liquidity was 1:1, so user should have paid ~1 pool
               token
26         // However, it spent much more than that. The user started with
                11 tokens, and now only has less than 10 tokens
27         assertLt(poolToken.balanceOf(someUser),1 ether);
28         vm.stopPrank();
29
```

```
30          // The liquidity provider can rug all funds from the pool now,
31          // inlcuding those deposited by user.
32          vm.startPrank(liquidityProvider);
33          pool.withdraw(
34              pool.balanceOf(liquidityProvider),
35              1, // minWethToWithdraw
36              1, // minPoolTokensToWithdraw
37              uint64(block.timestamp)
38          );
39          vm.stopPrank();
40
41          assertEq(weth.balanceOf(address(pool)),0);
42          assertEq(poolToken.balanceOf(address(pool)),0);
43      }
```

Recommended Mitigation:

```
1  function getInputAmountBasedOnOutput(
2          uint256 outputAmount,
3          uint256 inputReserves,
4          uint256 outputReserves
5      )
6          public
7          pure
8          revertIfZero(outputAmount)
9          revertIfZero(outputReserves)
10         returns (uint256 inputAmount)
11     {
12 -         return ((inputReserves * outputAmount) * 10000) /((
       outputReserves - outputAmount) * 997);
13
14 +         return ((inputReserves * outputAmount) * 1000) /((
       outputReserves - outputAmount) * 997);
15     }
```

[H-3] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially veceive way fewer tokens

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

Impact: If market conditions change before the transaction 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 `swapExactOutput` looking for I WETH
    1. inputToken = USDC
    2. outputToken = WETH
    3. outputAmount = 1
    4. deadline = whatever
3. The function does not offer a maxInput amount
4. As the transaction is pending ni 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::sellPoolTokens` mismatches input and output tokens causing users to receive the incorrect amount of tokens

Description: The `sellPoolTokens` function is intended to allow users 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 amount.

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

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

Recommended Mitigation:

Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require changing the `sellPoolTokens` function to accept a new parameter (ie `minWethToReceive` to be passed to `swapExactInput`)

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

Additionally, it might be wise to add a deadline to the function, as there is currently no deadline.

[H-5] 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 script invariant of x * y = k. Where: - `x` : The balance of the pool token - `y` : The balance of WETH - `k` : The constant product of the two balances

This means, that whenever the balances changes 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 following block of 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: A user could maliciouly 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 Code

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));

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

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

        uint 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 do with 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 parameters out of order

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTrans` function, it logs values in an incorrect order. The `poolTokensToDeposit` value should go in the third parameter position, whereas the `wethToDeposit` value should go second

Impact: Event emission is incorrect, leading to off-chain functions potentially malfunctioning

Recommended Mitigation:

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

[L-2] `TSwapPool::swapExactInput` results in incorrect return value given

Description: The `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 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(outputAmount, minOutputAmount);
         }

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

informational

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

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

[I-2] Lacking zero address checks

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

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

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

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

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

[I-4] 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

```
1     event PoolCreated(address tokenAddress, address poolAddress);
```

- Found in src/TSwapPool.sol Line: 52

```
1     event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
1     event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
1     event Swap(
```

[I-5] MINIMUM_WETH_LIQUIDITY is a constant and therefore not required to be emitted

```
1 - error TSwapPool__MinLiquidityTokensToMintTooLow(uint256
    minimumLiquidityTokensToMint,uint256 liquidityTokensToMint);
2 + error TSwapPool__MinLiquidityTokensToMintTooLow(uint256
    minimumLiquidityTokensToMint);
```

```
1 - revert TSwapPool__WethDepositAmountTooLow(MINIMUM_WETH_LIQUIDITY,
    wethToDeposit);
2 + revert TSwapPool__WethDepositAmountTooLow(wethToDeposit);
```

[I-6] In `TSwapPool::deposit` function It would be better if we update `liquidityTokensToMint` before the '_addLiquidityAndTransfer call to follow CEI

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

- liquidityTokensToMint = wethToDeposit;
```

[I-7] In `TSwapPool::PoolgetOutputAmountBasedOnInput` use magic numbers instead of constants

create the below constant variables in the contract and use them instead of constants

also use magic numbers in `TSwapPool::getInputAmountBasedOnOutput`

```
+ uint256 private constant FEE_PRECISION = 997;
+ uint256 private constant TOTAL_PRECISION = 1000;
```

```
- uint256 inputAmountMinusFee = inputAmount * 997;
+ uint256 inputAmountMinusFee = inputAmount * FEE_PRECISION;
uint256 numerator = inputAmountMinusFee * outputReserves;
- uint256 denominator = (inputReserves * 1000) + inputAmountMinusFee;
+ uint256 denominator = (inputReserves * TOTAL_PRECISION) +
    inputAmountMinusFee;
return numerator / denominator;
```

[I-8] Add the natspec to `TSwapPool::swapExactInput` function

[I-9] The function in `TSwapPool::swapExactInput` should be marked as external instead of public as this functions are not using interally to reduce gas

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

[I-10] make the visibility of the function `totalLiquidityTokenSupply` external external instead of public as it is not being used internally to reduce gas

[I-11] missing deadline param in natspec for the function `swapExactOutput`

Gas

[G-1] In `TSwapPool::deposit` function `poolTokenReserves` is not being used so remove this line and save gas

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