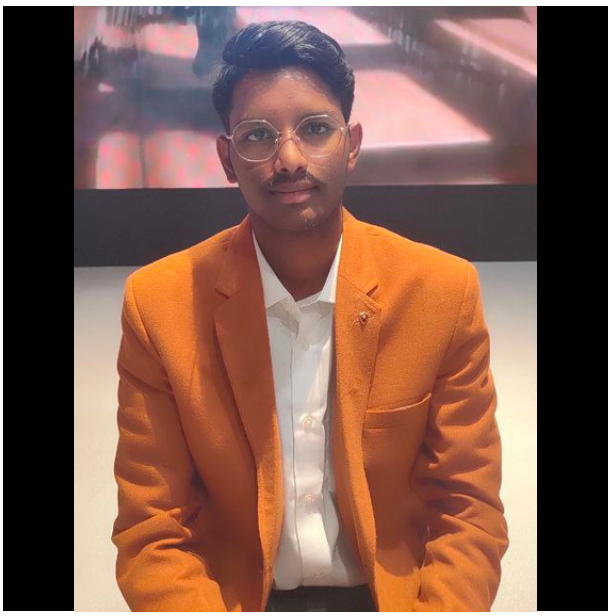

TSwap Protocol Audit Report

Version 1.0

December 31, 2023

TSwap Protocol Audit Report

AKHIL MANGA

December 30, 2023

Prepared by: AKHIL MANGA Lead Auditors: - AKHIL MANGA

Table of Contents

- Medium
  - [M-1] The `TSwapPool::deposit` is missing deadline checks, so that transactions can be done after deadline also
- Low
  - [L-1] The `TSwapPool::LiquidityAdded` event has parameters in wrong order
  - [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value
- Informational
  - [I-1] The `poolFactory::PoolFactory_PoolDoesNotExist` is not used and it should be removed
  - [I-2] Lacking the zero address checks
  - [I-3] The `PoolFactory::createPool` should use `.symbol()` instead of `.name()`
  - [I-4]: Event is missing `indexed` fields
  - [I-5] `MINIMUM_ETH_LIQUIDITY` is a constant and therfore not required to be emitted
  - [I-6] It will be great to update `liquidityTokensToMint` before a external call [Follow CEI]
  - [I-7] Magic numbers
  - [I-8] Add documentaion to the `TSwapPool::swapExactInput`
  - [I-9] The `swapExactInput` function should be external
  - [I-10] The `totalLiquidityTokenSupply` function should be external
- Gas
  - [G-1] `PoolTokenReserves` variable is not used anywhere, remove the `PoolTokenReserves` variable

Protocol Summary

This protocol 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.

Disclaimer

The AKHIL MANGA team makes all effort 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
Likelihood	High	H	H/M	M
	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

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda
- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum
- Tokens: Any ERC20 token

Scope

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

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

By auditing this codebase, i have learned about DEX, AMM, stateless and stateful(Invariant) fuzzing, weird ERC20's, slippage protection.

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Info/Gas	11
Total	18

Findings

High

[H-1] Wrong calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users, so that there will be lost in fees

Description: The `getInputAmountBasedOnOutput` function is to calculate the tokens a user should deposit given amount of tokens of output tokens. This function miscalculates the resulting amount. When calculating the fee, it is taken 10_000 instead of 1000.

Impact: Protocol charges more fees than expected from users.

Proof of Concept: Test suite for this finding can be found in `TSwapPool.t.sol`

```
function testWrongSwapExactOutput() public {
        uint256 initialLiquidity = 100e18;
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), initialLiquidity);
        poolToken.approve(address(pool), initialLiquidity);

        pool.deposit({
            wethDeposit: initialLiquidity,
            minimumLiquidityTokensToMint: 0,
            maximumPoolTokensToDeposit: initialLiquidity,
            deadline: uint64(block.timestamp)
        });
        vm.stopPrank();

        // User has 10 pool Tokens
        address someUser = makeAddr("someUser");
        uint256 userInitialPoolTokenBalance = 10e18;
        poolToken.mint(someUser, userInitialPoolTokenBalance);
        vm.startPrank(someUser);

        // someUser buys 1 weth from the pool, paying with poolTokens
        poolToken.approve(address(pool), type(uint256).max);
        pool.swapExactOutput(poolToken, weth, 1 ether, uint64(block.
            timestamp));

        assertEq(poolToken.balanceOf(someUser), 1 ether);
        vm.stopPrank();

        // The liquidity provider can rug all funds from the pool now,
            including uesr's deposited amount
        vm.startPrank(liquidityProvider);
        pool.withdraw(pool.balanceOf(liquidityProvider), 1, 1, uint64(
            block.timestamp));
        vm.stopPrank();

        assertEq(weth.balanceOf(address(pool)), 0);
        assertEq(poolToken.balanceOf(address(pool)), 0);
    }
```

Recommended Mitigation:

```
function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
```

```
10          returns (uint256 inputAmount)
11      {
12  -         return ((inputReserves * outputAmount) * 10000) / ((
        outputReserves - outputAmount) * 997);
13  +         return ((inputReserves * outputAmount) * 1000) / ((
        outputReserves - outputAmount) * 997);
14      }
```

[H-2] There is no slippage protection in `TSwapPool::SwapExactOutput` causes users to receive way fewer tokens

Description: The `swapExactOutput` function does not contain any slippage protection. This function is similar to `TSwapPool::swapExactInput`. Where the `swapExactInput` function specifies `minOutputAmount` and `swapExactOutput` should specify `maxInputAmount`.

Impact: If market conditions are changed before the transaction, the user will get a worst swap.

Proof of Concept: 1. The price of 1 WETH right now is 1,000 USDC 2. User inputs a `swapExactOutput` looking for 1 WETH => inputToken = USDC => outputToken = WETH => outputAmount = 1 => deadline = anything 3. The function does not offer a maxInput amount 4. As the transaction is pending in the mempool, the market changes. And the price changes huge that is 1 WETH is now equal to 10,000 USDC. 10x more than user expected. 5. The transaction completes, but the user sent the protocol 10,000 USDC instead of 1,000 USDC

Recommended Mitigation: we should add `maxInputAmount` so the user cannot spend more than specific amount.

```
1
2   function swapExactOutput(
3           IERC20 inputToken, // pooltoken
4           IERC20 outputToken, // wethtoken
5           uint256 outputAmount, // wethtokenamount
6   +       uint256 maxInputAmount, // pooltokenamount
7           uint64 deadline
8       )
9
10  inputAmount = getInputAmountBasedOnOutput(outputAmount, inputReserves,
        outputReserves);
11
12  + if(inputAmount > maxInputAmount) {
13  +     revert();
14  + }
15
16          _swap(inputToken, inputAmount, outputToken, outputAmount);
17      }
```

[H-3] The `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to receive the wrong amount of tokens

Description: The `sellPoolTokens` function will allow users to easily sell pool tokens and receive WETH in exchange. Users will sell pool tokens that is represented in `poolTokenAmount` parameter. The function currently miscalculates the swapped amount.

This is due to the `swapExactOutput` function is called, where the `swapExactInput` function should be called. Because users specify the exact amount of input tokens.

Impact: Users will swap the wrong amount of tokens.

Recommended Mitigation: Use `swapExactInput` instead of `swapExactOutput`. And change the `sellPoolTokens` function, to accept a new parameter called `minWethToReceive`

```
1  function sellPoolTokens(uint256 poolTokenAmount,
2  + uint256 minWethToReceive
3  ) external returns (uint256 wethAmount) {
4  -        return swapExactOutput(i_poolToken, i_wethToken,
       poolTokenAmount, uint64(block.timestamp));
5  +        return swapExactInput(i_poolToken, poolTokenAmount,
       i_wethToken, minWethToReceive, uint64(block.timestamp));
6       }
```

[H-4] In `TSwapPool::_swap` the extra tokens will be given to users after every `swapCount` breaks the protcol invariant that is `x * y = k`

Description: The protocool follows a invariant that is `x * y = k`. Where - `x`: The balance of the pool - `y`: The balance of WETH - `k`: The constant product of the two balances

This means, whenever the balances change in the protocol, the ratio between the 2 amounts should be constant. This is broken due to the extra incentive in the `_swap` function. Overtime, the funds will be drained.

The following code is responsible for the issue

```
1  - swap_count++;
2  -         if (swap_count >= SWAP_COUNT_MAX) {
3  -            swap_count = 0;
4  -            outputToken.safeTransfer(msg.sender, 1
       _000_000_000_000_000_000);
5  -         }
```

Impact: A user could maliciously drain the protocol of funds by a doing more number of swaps and collecting the incentives given by the protocol.

=> Protocol's core invariant is broken

Proof of Concept: 1. A user swaps 10 times and collects the incentive of 1`_000_000_000_000_000_000` tokens 2. That user continous to swap untill the funds drain.

Proof of Code

Place the below test suite in the `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();
        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));
        vm.stopPrank();

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

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

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

        uint256 endingY = weth.balanceOf(address(pool));
```

```
32
33          int256 actualDeltaY = int256(endingY) - int256(startingY);
34          assertEq(actualDeltaY, expectedDeltaY);
35      }
```

Recommended Mitigation: Remove the extra incentive mechanism.

Medium

[M-1] The `TSwapPool::deposit` is missing deadline checks, so that transactions can be done after deadline also

Description: The `deposit` function has a `deadline` parameter, but the parameter is never used.

This causes MEV attacks too

Impact: Transactions can be sent when the market conditions are unfavourable to deposit, event if we set the deadline parameter.

Proof of Concept: The `deadline` parameter is not used

Recommended Mitigation: Change the following in the function

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

Low

[L-1] The `TSwapPool::LiquidityAdded` event has parameters in wrong order

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTrans` function, it logs values in wrong order. The `poolTokensToDeposit` value should be in the 3rd position and `wethToDeposit` value should be in the 2nd position

Impact: Event emission isn incorrect, leading to the off-chain functions potentially malfunctioning.

Recommended Mitigation:

```
- event LiquidityAdded(address indexed liquidityProvider, uint256 wethDeposited, uint256 poolTokensDeposited);
+ event LiquidityAdded(address indexed liquidityProvider, uint256 poolTokensDeposited, uint256 wethDeposited);
```

[L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value

Description: The `swapExactInput` function is expected to return the actual amount of tokens bought by the caller. It is named return value `output`, but it does not assign to any value.

Impact: The return value will always be 0. Giving wrong 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] The `poolFactory::PoolFactory_PoolDoesNotExist` is not used and it should be removed

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

[I-2] Lacking the zero address checks

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

[I-3] The `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());
```

[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: 36
- Found in src/TSwapPool.sol: Line: 43
- Found in src/TSwapPool.sol: Line: 44
- Found in src/TSwapPool.sol: Line: 45

[I-5] `MINIMUM_ETH_LIQUIDITY` is a constant and therfore not required to be emitted

```
- error TSwapPool__WethDepositAmountTooLow(uint256 minimumWethDeposit, uint256 wethToDeposit);
+ error TSwapPool__WethDepositAmountTooLow(uint256 wethToDeposit);
```

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

[I-6] It will be great to update `liquidityTokensToMint` before a external call [Follow CEI]

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

Line no: 150

[I-7] Magic numbers

=> create 2 constant variables in the contract as below

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

```
function getOutputAmountBasedOnInput(
        uint256 inputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(inputAmount)
        revertIfZero(outputReserves)
        returns (uint256 outputAmount)
    {
-        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;
    }
```

=> The same magic numbers issue in the `getInputAmountBasedOnOutput` function

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

=> But here the total precision is 10_000, fees will be high. To prevent this, we are creating constant varaibles for numbers

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

// @notice figures out decide how much input you want to swap and this function gives max output you want to receive // @param inputToken input token to swap / sell that is DAI // @param inputAmount amount of input token to swap / sell that is DAI // @param outputToken output token to buy that is WETH // @param miinOutputAmount minimum output amount expected to receive // @param deadline deadline for when the transaction should expire

[I-9] The `swapExactInput` function should be external

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

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

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

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

[I-10] The `totalLiquidityTokenSupply` function should be external

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

Gas

[G-1] `PoolTokenReserves` variable is not used anywhere, remove the `PoolTokenReserves` variable

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

=> Gas cost will be reduced