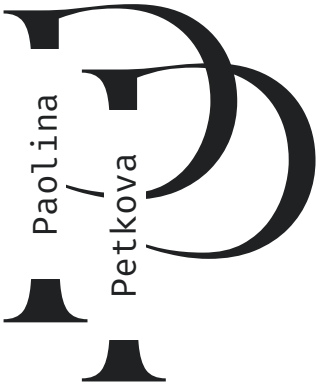

TSwap Protocol Audit Report

Version 1.0

Paolina Petkova

February 10, 2024

Protocol Audit Report

Paolina Petkova

Feb 10, 2024

Prepared by: Paolina Petkova

- Lead Auditors: Paolina Petkova

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.

Disclaimer

Paolina 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 her 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
```

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

The audit went great, I spent a couple of days on reading, finding and documenting all the findings. I learned a lot of new things while auditing this protocol.

Issues found

Severity	Number of issues found
High	4
Medium	2
Low	2
Gas	1
Info	10
Total	19

Findings

High

[H-1] In `TSwapPool::_swap` function the extra tokens given to users after every `swap_count` breaks the protocol invariant `x * y = k`

Description:

The protocol follows a strict invarian 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 change in the protocol, the ration 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:

The user could maliciously drain the protocol of funds by doing a lot of swaps and collecting the extra incentive given out by the protocol.

More simply put, the protocol's core invariant is broken.

Proof of Concept:

1. The user swaps 10 times and collects extra incentive of `1_000_000_000_000_000_000` tokens.
2. The user continues to swap until all the protocol funds are drained.

PoC

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

```
    function testBrokenInvariant() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();
```

```
7
8           int256 startingY;
9           int256 expectedDeltaY;
10
11          uint256 outputWeth = 1e17;
12
13          vm.startPrank(user);
14          poolToken.approve(address(pool), type(uint256).max);
15          poolToken.mint(user, 100e18);
16          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
17          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
18          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
19          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
20          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
21          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
22          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
23          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
24          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
25          vm.stopPrank();
26
27          startingY = int256(weth.balanceOf(address(pool)));
28          expectedDeltaY = int256(-1) * int256(outputWeth);
29
30          uint256 endingY = weth.balanceOf(address(pool));
31          int256 actualDeltaY = int256(endingY) - int256(startingY);
32
33          assertEq(actualDeltaY, expectedDeltaY);
34      }
```

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.

```
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 -         }
```

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

Description:

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

Impact:

Protocol takes more fees than expected from users.

Proof of Concept:

The following test should be added in `TSwapPool.t.sol`. It proves that after `swapExactOutput` the `poolToken` balance of the user is less than 1 ether which means the protocol changed the user much more money than expected:

PoC

```
    function testgetInputAmountBasedOnOutput() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        console.log(weth.balanceOf(user));
        console.log(poolToken.balanceOf(user));

        vm.startPrank(user);
        poolToken.approve(address(pool), 11e18);
        pool.swapExactOutput(poolToken, weth, 1e18, uint64(block.
            timestamp));
        vm.stopPrank();

        console.log(weth.balanceOf(user));
        console.log(poolToken.balanceOf(user));

        assertLt(poolToken.balanceOf(user), 1 ether);
        vm.stopPrank();
    }
```

Recommended Mitigation:

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
```

```
4           uint256 outputReserves
5       )
6           public
7           pure
8           revertIfZero(outputAmount)
9           revertIfZero(outputReserves)
10          returns (uint256 inputAmount)
11      {
12  -        return
13  -            ((inputReserves * outputAmount) * 10_000) /
14  -            ((outputReserves - outputAmount) * 997);
15  +        return
16  +            ((inputReserves * outputAmount) * 1_000) /
17  +            ((outputReserves - outputAmount) * 997);
18      }
```

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

Description:

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

Impact:

if the market conditions change before the transaction process, the user can 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 1 WETH.
    1. inputToken = USDC
    2. outputToken = WETH
    3. outputAmount = 1
    4. deadline = whatever
3. The function does not offer `maxInputAmount`.
4. As the transaction is pending in the mempool, the market changes. And the price moves HUGE -> 1 WETh is now 10,000 USDC. 10 times 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.

The function should be added in `TSwapPool.t.sol` file. It proves that the second swap the price move is higher than the first swap and this was not expected by the user. PoC

```
    function testswapExactOutputSlippageProtectionMissing() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 150e18);
        poolToken.approve(address(pool), 150e18);
        pool.deposit(150e18, 150e18, 150e18, uint64(block.timestamp));
        vm.stopPrank();

        uint256 userInitialPoolTokenBalance = poolToken.balanceOf(user)
            ;


        vm.startPrank(user);
        poolToken.approve(address(pool), 11e18);
        pool.swapExactOutput(poolToken, weth, 1e18, uint64(block.
            timestamp));
        vm.stopPrank();

        uint256 userAfterFirstTransactionPoolTokenBalance = poolToken.
            balanceOf(user);

        weth.mint(user, 10e18);
        poolToken.mint(user, 11e18);

        uint256 userAfterMintMoreMoneyPoolTokenBalance = poolToken.
            balanceOf(user);

        vm.startPrank(user);
        poolToken.approve(address(pool), 11e18);
        pool.swapExactOutput(poolToken, weth, 1e18, uint64(block.
            timestamp));
        vm.stopPrank();

        uint256 userAfterSecondTransactionPoolTokenBalance = poolToken.
            balanceOf(user);

        assert(
            userInitialPoolTokenBalance -
                userAfterFirstTransactionPoolTokenBalance
                < userAfterMintMoreMoneyPoolTokenBalance -
                    userAfterSecondTransactionPoolTokenBalance
        );
    }
```

Recommended Mitigation:

We should include `maxInputAmount` so the user only has to spend a specific amount, and can predict how much they can spend on the protocol.

```
    function swapExactOutput(
        IERC20 inputToken,
        IERC20 outputToken,
        uint256 outputAmount,
+       uint256 maxInputAmount,
        uint64 deadline
    )
        public
        revertIfZero(outputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 inputAmount)
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

        inputAmount = getInputAmountBasedOnOutput(
            outputAmount,
            inputReserves,
            outputReserves
        );

+       if(inputAmount > maxInputAmount){
+           revert();
+       }

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

[H-4] `TSwapPool::sellPoolTokens` mismatches input and output tokens, causing the users to receive the incorrect amount of tokens

Description:

The `sellPoolTokens` function is intended to allow users to easily sell Pool Tokens and receve WETH in exchange. Users indicate how many pool tokens the 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 `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.

Proof of Concept:

The following test should be added to `TSwapPool.t.sol`. The test proves that when selling only 1 from the 11 eth of Pool Token the user have, after the transaction user's balance of Pool Token is less than 9 eth which is lower than expected. PoC

```
    function testSellPoolTokensIsIncorect() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 150e18);
        poolToken.approve(address(pool), 150e18);
        pool.deposit(150e18, 150e18, 150e18, uint64(block.timestamp));
        vm.stopPrank();

        uint256 userInitialWethBalance = weth.balanceOf(user);
        uint256 userInitialPoolTokenBalance = poolToken.balanceOf(user)
            ;
        console.log("userInitialWethBalance", userInitialWethBalance);
        console.log("userInitialPoolTokenBalance",
            userInitialPoolTokenBalance);

        vm.startPrank(user);
        poolToken.approve(address(pool), 11e18);
        pool.sellPoolTokens(1e18);
        vm.stopPrank();

        uint256 userAfterFirstTransactionWethBalance = weth.balanceOf(
            user);
        uint256 userAfterFirstTransactionPoolTokenBalance = poolToken.
            balanceOf(user);
        console.log("userAfterFirstTransactionWethBalance",
            userAfterFirstTransactionWethBalance);
        console.log("userAfterFirstTransactionPoolTokenBalance",
            userAfterFirstTransactionPoolTokenBalance);

        assert(userAfterFirstTransactionPoolTokenBalance > 9e18);
```

Recommended Mitigation:

Consider changing the implementation to use `swapExactInput`, instead of `swapExactOutput` function. Note that this will require changing the `sellPoolTokens` to accept new parameter `minWethToReceive` to be passed to `swapExactOutput`.

```
    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)
+            );
     }
```

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

Medium

[M-1] `TSwapPool::deposit` function has no deadline checks causing transactions to complite 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 consequece, operators that add liquidity to the pool might be executed at unexpected times, in market conditions where the deposit rate is unfavorable.

Impact:

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

Proof of Concept:

The `deadline` parameter is passed but never used.

Recommended Mitigation:

Consider making the following change to the function.

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

[M-2] Weird ERC-20 tokens - Rebase, fee-on-transfer, ERC-777 and centralized tokens break protocol invariant

Description: Some tokens may make arbitrary balance modifications outside of transfers (rebase). Some protocols charge a fee when calling `transfer` or `transferFrom` functions - `fee-on-transfer`. It will be okay if the fee is taken from the tranfered amount, but some protocols take extra money for the fee. All these tokens will break the `TSwapPool` invariant `x * y = k`.

Impact:

A malicious user could drain all teh funds from the protol using weird ERC-20 tokens. They all break the core invariant

Low

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

Description:

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

Impact:

Event emission is incorrect, leading to off-chanin functions potentiolly malfunctioning.

Recommended Mitigation:

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

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

Description:

The `swapExactInput` function is expect to return the exact amount of token bought by the caller. However 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.

Proof of Concept:

The following test proves that the return value of `swapExactInput` is always equals to 0. The test should be added in `TSwapPool.t.sol`:

PoC

```
    function testswapExactInputReturnValue() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        vm.startPrank(user);
        uint256 expected = 9e18;
        poolToken.approve(address(pool), 10e18);
        uint256 returnValue = pool.swapExactInput(poolToken, 10e18,
            weth, expected, uint64(block.timestamp));
        vm.stopPrank();

        assertEq(returnValue, 0);
    }
```

Recommended Mitigation:

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

+       output = getOutputAmountBasedOnInput(
+         inputAmount,
```

```
24 +           inputReserves,
25 +           outputReserves
26 +       );
27
28 -       if (outputAmount < minOutputAmount) {
29 -           revert TSwapPool__OutputTooLow(outputAmount,
       minOutputAmount);
30 -       }
31 +       if (output < minOutputAmount) {
32 +           revert TSwapPool__OutputTooLow(output, minOutputAmount);
33 +       }
34
35 -       _swap(inputToken, inputAmount, outputToken, outputAmount);
36 +       _swap(inputToken, inputAmount, outputToken, output);
37     }
```

Informationals

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

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

[I-2] 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-3] Missing `address(0)` checks

Missing `address(0)` check can cause `i_wethToken` to be set to zero address.

Recommended Mitigation:

```
1      constructor(address wethToken) {
2  +        if (wethToken == address(0)) {
3  +            revert();
4  +        }
5          i_wethToken = wethToken;
6      }
```

[I-4] In `PoolFactory::createPool` should use `.symbol()` instead of `.name()` for `PoolFactory::liquidityTokenSymbol`

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

[I-5] `TSwapPool::MINIMUM_WETH_LIQUIDITY` is a constant and there is no need to be emitted

There is no need a constant to be emitted because it is never gonna change. Also emitting `TSwapPool::MINIMUM_WETH_LIQUIDITY` is a waste of gas.

```
1      error TSwapPool__WethDepositAmountTooLow(
2  -       uint256 minimumWethDeposit,
3          uint256 wethToDeposit
4      );
5
6      ...
7
8      revert TSwapPool__WethDepositAmountTooLow(
9  -           MINIMUM_WETH_LIQUIDITY,
10             wethToDeposit
11         );
```

[I-6] Better initializing `liquidityTokensToMint` before the actual transfer of money in `TSwapPool::deposit` function, CEI is not followed strictly

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

[I-7] Usage of magic numbers, better define them with proper names for better readability and less mistakes

```
+   uint256 private constant FEE_MULTIPLIER = 997;
+   uint256 private constant TOTAL_MULTIPLIER = 1000;
.......
-       uint256 inputAmountMinusFee = inputAmount * 997;
+       uint256 inputAmountMinusFee = inputAmount * FEE_MULTIPLIER;
        uint256 numerator = inputAmountMinusFee * outputReserves;
-       uint256 denominator = (inputReserves * 1000) +
    inputAmountMinusFee;
+       uint256 denominator = (inputReserves * TOTAL_MULTIPLIER) +
    inputAmountMinusFee;
        return numerator / denominator;
```

[I-8] Missing natspec for `TSwapPool::swapExactInput` function

```
+   // inputToken: input token to swap/sell DAI
+   // inputAmount: amount of input token to sell
+   // outputToken: autput token to buy WETH
+   // minOutputAmount: minimum output amount expected to receive
+   // deadline: deadline for when the transaction should expire
    function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
```

[I-9] Defining `TSwapPool::swapExactInput` function as `public` is waste of gas because it is not used in the contract, better define it as `external`

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

[I-10] Missing `deadline` parameter explanation in the natspec

```
    /*
     * @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
+    * @param deadline The deadline for when the transaction should
    expire
     */
```

Gas

[G-1] The variable `poolTokenReserves` in `TSwapPool::deposit` function is never used, and definition and initialization of it is a waste of gas

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