TSwap Protocol Audit Report

Version 1.0

Andrii Antonenko

April 20, 2024

Protocol Audit Report

Andrii Antonenko

April 19, 2024

Prepared by: Andrii Antonenko Lead Auditors:

- Andrii Antonenko

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

The Andrii Antonenko 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
```

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

I loved auditing this codebase. Great for beginners and contains a lot of classic issues.

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Info	9
Total	16

Findings

High

H-1 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 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 fee than expected from users.

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 Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially 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::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: Add the following test suite to the `test/TSwapPool.t.sol` file:

```
1       function testLackOfSlippage() public {
2           vm.startPrank(liquidityProvider);
3           weth.approve(address(pool), 100e18);
4           poolToken.approve(address(pool), 100e18);
5           pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
6           vm.stopPrank();
7
8           uint256 wethOutputAmount = 1e16;
9           uint256 inputReserves = poolToken.balanceOf(address(pool));
10          uint256 outputReserves = weth.balanceOf(address(pool));
11
12          // user going to swap X pool tokens for 0.01 WETH
13          uint256 expectedInputAmount = pool.getInputAmountBasedOnOutput(
               wethOutputAmount, inputReserves, outputReserves);
14          console.log("expectedInputAmount", expectedInputAmount);
15
16          uint256 poolTokenBalanceBefore = poolToken.balanceOf(user);
17          uint256 wethBalanceBefore = weth.balanceOf(user);
18
19          // user expects that he has to pay expectedInputAmount
20          // but other user is going to swap some tokens before him
21          // he is gonna make WETH more expensive, so expectedInputAmount
                should be higher
22          // so he is gonna make a swap of X pool tokens, to receive 1
               WETH, so WETH liquidity will be lower
23          address otherUser = makeAddr("otherUser");
24          vm.startPrank(otherUser);
25          poolToken.mint(otherUser, 100e18);
```

```
26          poolToken.approve(address(pool), 100e18);
27          pool.swapExactOutput(poolToken, weth, 1e18, uint64(block.
                timestamp));
28          vm.stopPrank();
29
30          // user is going to make a swap with expectedInputAmount
31          // he doesn't know that WETH is more expensive now
32          // and he is going to pay more than expected
33          vm.startPrank(user);
34          poolToken.approve(address(pool), type(uint256).max);
35          pool.swapExactOutput(poolToken, weth, wethOutputAmount, uint64(
                block.timestamp));
36
37          uint256 poolTokenBalanceAfter = poolToken.balanceOf(user);
38          uint256 wethBalanceAfter = weth.balanceOf(user);
39          vm.stopPrank();
40
41          console.log("poolTokenBalanceBefore", poolTokenBalanceBefore);
42          console.log("poolTokenBalanceAfter", poolTokenBalanceAfter);
43          console.log("poolTokenBalanceBefore - poolTokenBalanceAfter",
                poolTokenBalanceBefore - poolTokenBalanceAfter);
44          console.log("expectedInputAmount", expectedInputAmount);
45          assert(poolTokenBalanceBefore - poolTokenBalanceAfter <=
                expectedInputAmount);
46      }
47  }
```

A test suite has been devised to demonstrate this vulnerability. By simulating a scenario where market conditions change before a swap transaction is executed, the test illustrates how users could end up paying more than expected due to the absence of slippage protection.

Recommended Mitigation:

```
1       function swapExactOutput(
2           IERC20 inputToken,
3           IERC20 outputToken,
4           uint256 outputAmount,
5   +       uint256 maxInputAmount,
6           uint64 deadline
7       )
8           public
9           revertIfZero(outputAmount)
10          revertIfDeadlinePassed(deadline)
11          returns (uint256 inputAmount)
12      {
13          uint256 inputReserves = inputToken.balanceOf(address(this));
14          uint256 outputReserves = outputToken.balanceOf(address(this));
15
16          inputAmount = getInputAmountBasedOnOutput(outputAmount,
                inputReserves, outputReserves);
17  +       if (inputAmount > maxInputAmount) {
```

```
18 +           revert();
19 +       }
20         _swap(inputToken, inputAmount, outputToken, outputAmount);
21     }
```

H-3 `TSwapPool::sellPollTokens` 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 `pollTokenAmount` 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)

```
1     function sellPoolTokens(
2         uint256 poolTokenAmount,
3 +       uint256 minWethToReceive,
4         ) external returns (uint256 wethAmount) {
5 -        return swapExactOutput(i_poolToken, i_wethToken,
      poolTokenAmount, uint64(block.timestamp));
6 +        return swapExactInput(i_poolToken, poolTokenAmount,
      i_wethToken, minWethToReceive, uint64(block.timestamp));
7     }
```

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

H-4 In `TSwapPool::_swap` the extra tokens given to users after every swap breaks the protocol invariant of `x * y = k`

Description: The protocol follows a strict 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 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 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

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));
```

```
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
23          int256 startingY = int256(weth.balanceOf(address(pool)));
24          int256 expectedDeltaY = int256(-1) * int256(outputWeth);
25
26          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
27          vm.stopPrank();
28
29          uint256 endingY = weth.balanceOf(address(pool));
30          int256 actualDeltaY = int256(endingY) - int256(startingY);
31          assertEq(actualDeltaY, expectedDeltaY);
32      }
```

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  -        // Fee-on-transfer
3  -        if (swap_count >= SWAP_COUNT_MAX) {
4  -            swap_count = 0;
5  -            outputToken.safeTransfer(msg.sender, 1
       _000_000_000_000_000_000);
6  -        }
```

Medium

M-1 `TSwapPool::deposit` is missing deadline check causing transactions to complete event after the deadline

Description: The `deposit` function accepts a `deadline` parameter, which according to the documentation "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 executed at unexpected times, in market conditions where the deposit rate is unfavorable.

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

Proof of Concept: After running `forge compile`, in the terminal we can see warning, that tells us about unused `deadline` parameter:

```
[] Compiling...
[] Compiling 7 files with 0.8.20
[] Solc 0.8.20 finished in 2.21s
Compiler run successful with warnings:
Warning (5667): Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> src/TSwapPool.sol:98:9:
   |
98 |         uint64 deadline
   |         ^^^^^^^^^^^^^^^
```

Recommended Mitigation: Consider making the following change to the function:

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

Low

L-1 `TSwapPool::LiquidityAdded` event has parameters out of order

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTran` 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 function potentially malfunctioning.

Recommended Mitigation:

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

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

Description: The `swapExactInput` function is expected to return 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) {
+       if (output < minOutputAmount) {
-           revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
+           revert TSwapPool__OutputTooLow(output, minOutputAmount);
        }

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

Informational

I-1 `PoolFactory::PoolFactory__PoolDoesNotExist` error is unused and should be removed

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

I-2 Lacking zero-address check in the `PoolFactory::constructor`

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

```
6       }
```

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

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

I-4 Lacking zero address check in the `TSwapPool::constructor`

```
1       constructor(
2           address poolToken,
3           address wethToken,
4           string memory liquidityTokenName,
5           string memory liquidityTokenSymbol
6       )
7           ERC20(liquidityTokenName, liquidityTokenSymbol)
8       {
9   +       if (wethToken == address(0) || poolToken == address(0)) {
10  +           revert();
11  +       }
12          i_wethToken = IERC20(wethToken);
13          i_poolToken = IERC20(poolToken);
14      }
```

I-5 Variable `poolTokenReserves` in the function `TSwapPool::deposit` is unused

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

I-6 The event `TSwapPool::TSwapPool__WethDepositAmountTooLow` shouldn't include `minimumWethDeposit` in parameters, cause it's a constant value, and never will change

Description: In the smart contract `TSwapPool` minimum weth deposit - is a constant: `MINIMUM_WETH_LIQUIDITY`. It usually passed as argument to the event `TSwapPool::TSwapPool__WethDepositAmountTooLow`, but this value is never change, so it's not necessary to pass it ass argument, to avoid gas waste.

Recommended Mitigation: Consider making the following change:

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

And in the function `TSwapPool::deposit` make the following change too:

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

I-7 Enhancement of fee calculation clarity in `TSwapPool::getOutputAmountBasedOnInput`

Description: Within the smart contract TSwapPool, the utilization of the hardcoded values 997 and 1000 in the getOutputAmountBasedOnInput function presents a readability concern. Such magic numbers obscure the code's intent and may impede comprehension for developers and stakeholders.

Recommended Mitigation: To enhance code transparency and maintainability, it is advised to introduce two constants for clarity:

```
+ uint256 public constant FEE_PRECISION = 1_000;
+ uint256 public constant FEE_COEFFICIENT = 997; // fee equal to 0.03 %
```

Additionally, the following modifications are proposed for the getOutputAmountBasedOnInput function:

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

Implementing these adjustments will not only improve code clarity but also facilitate future maintenance and comprehension.

I-8 Missing docs for the public function `TSwapPool::swapExactInput`

Description: The public function swapExactInput within the TSwapPool smart contract lacks comprehensive documentation. The absence of sufficient documentation poses a challenge for developers

aiming to understand the function's purpose, inputs, outputs, and potential side effects. This deficiency hampers the seamless integration of the function into broader systems and may lead to misunderstandings or errors in its utilization.

I-9 The function `TSwapPool::swapExactInput` could be external

Description: Currently, the swapExactInput function within the TSwapPool smart contract is designated as public, despite not being invoked by any internal methods within TSwapPool. Given this context, optimizing the function's visibility by marking it as external can enhance gas efficiency and streamline execution.

Impact: The function's public visibility triggers additional checks mandated for all public functions by Solidity, resulting in unnecessary gas consumption. By converting the function to external, these redundant checks can be bypassed, leading to potential gas savings and improved contract efficiency.

Recommended Mitigation:

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