TSwap Protocol Audit Report

Version 1.0

SageIAm

May 27, 2025

Protocol Audit Report

SageIAm

March 7, 2023

Prepared by: SageIAm Lead Auditors:

- SageIAm

Table of Contents

Protocol Summary

TSwap

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

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

Scope

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda
- In Scope:

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

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

Core Invariant

Our system works because the ratio of Token A & WETH will always stay the same. Well, for the most part. Since we add fees, our invariant technially increases.

```
x * y = k
```

- x = Token Balance X
- y = Token Balance Y

- k = The constant ratio between X & Y

Our protocol should always follow this invariant in order to keep swapping correctly!

Actors / 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

Severity	Number of issues found
High	5
Medium	0
Low	2
Gas	0
Info	10
Total	17

Issues found

Findings

High

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

Description: The `deposit` function accepts a 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 executed at unexpected times, in market conditions where the deposit rate is unfavorable.

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

Proof of Concept: The `deadline` parameter is unused.

Warning (5667): Unused function parameter. Remove or comment out the variable name to silence this warning. –> src/TSwapPool.sol:96:9: | 96 | uint64 deadline | ^^^^^^^^^^^^^^^^

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)
            {...}
```

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

Description: The `getInputAmountBasedOnOutput` function is intended to calculate input amount after fees , according to ouput amount. however the function currently miscalculates the fees , When calculating fees it is scaling to `10_000` instead of `1_000`.

Impact: Protocols takes more fees from user than expected.

Recommended Mitigation: Correct the values.

```
  function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
        // @audit-high - Erroneous fee calculation resulting in 90.03%
           fees
        // Impact - High - Users are charged >90% of tokens transferred
            in fees
```

```
14          // Likelihood - High - swapExactOutput, which calls this
                function is a main swapping function
15
16          return
17  -           ((inputReserves * outputAmount) * 10000) /((outputReserves
        - outputAmount) * 997);
18  +           ((inputReserves * outputAmount) * 1000) /((outputReserves -
         outputAmount) * 997);
19      }
```

[H-3] 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:

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 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.

```
1      function swapExactOutput(
2          IERC20 inputToken,
3  +       uint256 maxInputAmount,
4  .
```

```
5  .
6  .
7          inputAmount = getInputAmountBasedOnOutput(outputAmount,
               inputReserves, outputReserves);
8  +       if(inputAmount > maxInputAmount){
9  +           revert();
10 +       }
11         _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`)

```
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. (MEV later)

[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 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 until all the protocol funds are drained

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);
```

```
12          poolToken.approve(address(pool), type(uint256).max);
13          poolToken.mint(user, 100e18);
14          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
15          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
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
24          int256 startingY = int256(weth.balanceOf(address(pool)));
25          int256 expectedDeltaY = int256(-1) * int256(outputWeth);
26
27          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
               timestamp));
28          vm.stopPrank();
29
30          uint256 endingY = weth.balanceOf(address(pool));
31          int256 actualDeltaY = int256(endingY) - int256(startingY);
32          assertEq(actualDeltaY, expectedDeltaY);
33      }
```

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 -         }
```

Low

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

Description: What 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. When it comes to auditing smart contracts, there are a lot of nitty-gritty details that one needs to pay attention to in order to prevent possible vulnerabilities.

Recommended Mitigation:

```
- 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 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 (output < minOutputAmount) {
-           revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
+       if (output < minOutputAmount) {
+           revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
    }

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

```
15 +         _swap(inputToken, inputAmount, outputToken, output);
16 }
17 }
```

Informationals

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not being used , so it must be removed

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

[I-2] Lacking Zero Address checks in constructors

PoolFactory.sol

```
1     constructor(address wethToken) {
2 +       if(wethToken ==  address(0x0)){
3 +       revert;
4 +       }
5         i_wethToken = wethToken;
6     }
```

TSwapPool.sol

```
1     constructor(
2         address poolToken,
3         address wethToken,
4         string memory liquidityTokenName,
5         string memory liquidityTokenSymbol
6     ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
7         //@audit-info should be checked if wethToken and poolToken are
              valid ERC20 tokens/ are non zero address
8 +       if(wethToken ==  address(0x0) || poolToken == address(0x0)){
9 +       revert;
10 +      }
11        i_wethToken = IERC20(wethToken);
12        i_poolToken = IERC20(poolToken);
13    }
```

[I-3] In `PoolFactory::createPool .name()` is used instead of `.symbol()` for making pool symbol

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

[I-4] Literal Instead of Constant

Define and use `constant` variables instead of using literals. If the same constant literal value is used multiple times, create a constant state variable and reference it throughout the contract.

4 Found Instances

- Found in src/TSwapPool.sol Line: 274

```
        uint256 inputAmountMinusFee = inputAmount * 997;
```

- Found in src/TSwapPool.sol Line: 293

```
            ((outputReserves - outputAmount) * 997);
```

- Found in src/TSwapPool.sol Line: 444

```
                1e18,
```

- Found in src/TSwapPool.sol Line: 453

```
                1e18,
```

[I-5] Public Function Not Used Internally

If a function is marked public but is not used internally, consider marking it as `external`.

1 Found Instances

- Found in src/TSwapPool.sol Line: 296

```
    function swapExactInput(
```

[I-6] PUSH0 Opcode

Solc compiler version 0.8.20 switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain other than mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.

2 Found Instances

- Found in src/PoolFactory.sol Line: 15

```
1 pragma solidity 0.8.20;
```

- Found in src/TSwapPool.sol Line: 15

```
1 pragma solidity 0.8.20;
```

[I-7] Events having more than 3 parameters should have 3 indexed parameters

TSwapPool.sol

```
1 event Swap(
2         address indexed swapper,
3         IERC20 tokenIn,
4         uint256 amountTokenIn,
5         IERC20 tokenOut,
6         uint256 amountTokenOut
7     );
```

[I-8] Variables should be updated before calling external calls so that it follow CEI (Check Effect Interaction)

```
1 +               liquidityTokensToMint = wethToDeposit;
2     _addLiquidityMintAndTransfer(
3                     wethToDeposit,
4                     maximumPoolTokensToDeposit,
5                     wethToDeposit
6                 );
7                 // @audit-info variable liquidityTokensToMint shoul be
                     updated before `_addLiquidityMintAndTransfer` so
                     that it follows CEI
8 -               liquidityTokensToMint = wethToDeposit;
```

[I-9] In `TSwapPool::_addLiquidityMintAndTransfer`, `_safe_mint` can be used instead of `_mint`.

```
1 - _mint(msg.sender, liquidityTokensToMint);
2 + _safe_mint(msg.sender, liquidityTokensToMint);
```

[I-10] In `TSwapPool::sellPoolTokens`, deadline for `swapExactOuput` can be set manually so that transaction does not hang indefinetly

```
1   swapExactOutput(
2                   i_poolToken,
3                   i_wethToken,
4                   poolTokenAmount,
5   @>              uint64(block.timestamp)
6               );
```