TSwap Protocol Audit Report

Version 1.0

Geekybot

April 18, 2025

TSwap Protocol Audit Report

geekybot

April 18, 2025

Prepared by: Geekybot

Lead Auditors:

- Utpal Pal

Table of Contents

Protocol Summary

TSwap Protocol

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

TSwap Pools

The protocol starts as simply a `PoolFactory` contract. This contract is used to create new "pools" of tokens. It helps make sure every pool token uses the correct logic. But all the magic is in each `TSwapPool` contract.

You can think of each `TSwapPool` contract as it's own exchange between exactly 2 assets. Any ERC20 and the WETH token. These pools allow users to permissionlessly swap between an ERC20 that has a pool and WETH. Once enough pools are created, users can easily "hop" between supported ERC20s.

For example: 1. User A has 10 USDC 2. They want to use it to buy DAI 3. They `swap` their 10 USDC -> WETH in the USDC/WETH pool 4. Then they `swap` their WETH -> DAI in the DAI/WETH pool

Every pool is a pair of `TOKEN X` & `WETH`.

There are 2 functions users can call to swap tokens in the pool. - `swapExactInput` - `swapExactOutput`

We will talk about what those do in a little.

Liquidity Providers

In order for the system to work, users have to provide liquidity, aka, "add tokens into the pool".

Why would I want to add tokens to the pool?

The TSwap protocol accrues fees from users who make swaps. Every swap has a `0.3` fee, represented in `getInputAmountBasedOnOutput` and `getOutputAmountBasedOnInput`. Each applies a `997` out of `1000` multiplier. That fee stays in the protocol.

When you deposit tokens into the protocol, you are rewarded with an LP token. You'll notice `TSwapPool` inherits the `ERC20` contract. This is because the `TSwapPool` gives out an ERC20 when Liquidity Providers (LP)s deposit tokens. This represents their share of the pool, how much they put in. When users swap funds, 0.03% of the swap stays in the pool, netting LPs a small profit.

LP Example

1. LP A adds 1,000 WETH & 1,000 USDC to the USDC/WETH pool
    1. They gain 1,000 LP tokens
2. LP B adds 500 WETH & 500 USDC to the USDC/WETH pool
    1. They gain 500 LP tokens
3. There are now 1,500 WETH & 1,500 USDC in the pool
4. User A swaps 100 USDC -> 100 WETH.
    1. The pool takes 0.3%, aka 0.3 USDC.
    2. The pool balance is now 1,400.3 WETH & 1,600 USDC
    3. aka: They send the pool 100 USDC, and the pool sends them 99.7 WETH

Note, in practice, the pool would have slightly different values than 1,400.3 WETH & 1,600 USDC due to the math below.

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!

Make a swap

After a pool has liquidity, there are 2 functions users can call to swap tokens in the pool. - `swapExactInput` - `swapExactOutput`

A user can either choose exactly how much to input (ie: I want to use 10 USDC to get however much WETH the market says it is), or they can choose exactly how much they want to get out (ie: I want to get 10 WETH from however much USDC the market says it is.

Disclaimer

The Geekybot 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
	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

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda

Scope

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

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

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

Auditing `TSwap Protocol` was exceptionally fun and challenging at the same time, the clear scope of audit and the documentations helped a lot to process all the informations required to perform the security review, we found several high impacting issues to gas to informational reports. We hope the protocol fixes the issues before going live with the recommendations provided in each of the reportings.

Issues found

Severity	No of Issues Found
HIGH	4
MEDIUM	1
LOW	2
Gas	1
INFO	12
TOTAL	20

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` function, causing exorbitantly higher fees collected

Description: In function `TSwapPool::getInputAmountBasedOnOutput` the calculation of `inputAmount` is very high as mathematical error. According to the protocol documentation the swap fee should be `0.3`, which implies the amount after fee `997/1000` but fees taken here is `997/10000` which makes the fee collected `0.9003`. Severly disrupting the swap function

Impact: Swapper loses ~`90%` as fee as compared to `0.3` fee

Proof of Concept: Paste the below code in `TswapPool.t.sol` and test

```
    function test_IncorrectFeeCollection() public {
        // deposit to the pool
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();
        address swapper = makeAddr("swapper");
        poolToken.mint(swapper, 11e18);
        vm.startPrank(swapper);
        // initial liquidity provided as 1:1, so the swap of poolToken
           to weth
        // should cost the user ~1 poolToken to buy 1 weth
        poolToken.approve(address(pool), 11e18);
        pool.swapExactOutput(poolToken, weth, 1e18, uint64(block.
           timestamp));
        // user starts with 10 poolTokens, to buy 1 weth, plus fee 0.3,
            user should not
        // spend more than 5 poolTokens
        uint256 ptBalanceAfterSwap = poolToken.balanceOf(swapper);
        assert(ptBalanceAfterSwap < 5e18);
        console.log("pool token balance after buying 1 weth: ",
           ptBalanceAfterSwap);
        // 868595686048043120 or 0.868595686048O432 poolToken

        // swapper paid very high fee to buy 1 weth, and liquidity
           provider can
        // withdraw all this extra fee from the pool making a rugpull
    }
```

Recommended Mitigation:

```
function getInputAmountBasedOnOutput(
            uint256 outputAmount,
            uint256 inputReserves,
            uint256 outputReserves
    )
            public
            pure
            revertIfZero(outputAmount)
            revertIfZero(outputReserves)
            returns (uint256 inputAmount)
    {
            return
-             ((inputReserves * outputAmount) * 10000) /
+             ((inputReserves * outputAmount) * 1000) /
              ((outputReserves - outputAmount) * 997);
    }
```

[H-2] In function `TSwapPool::swapExactOutput`, `inputToken` has no slippage protection, which may cause the user get bad swap rates

Description: Missing slippage protection leads the user to spent more input tokens to buy output token, which may lead an attack, where attacker watches mempool in `TSwapPool::swapExactOutput` and executes his trade faster by providing more gas for the transaction, which will lead the user's transaction be executed at higher value according to bonding curve formula. Then after the user's transaction is confirmed the attacker will sell his tokens at higher price leading the user at huge loss, This attack vector is known as frontrunning. Also it leads to MEV, Gas war, Sandwich Attack.

Even when market condition drastically change the swapper will get way worse swap rate.

Impact: User is impacted by huge financial loss due to sandwich, frontrunning attack as he pays more for a desired output token, making severe impact to protocol functionality.

Proof of Concept: 1. Current market price of 1 WETH is 1000 USDC 2. User initiates his transaction, expecting to buy 1 WETH for 1000 USDC 1. inputToken = USDC 2. outputToken = WETH 3. outputAmount = 1 4. deadline = some future time 3. Now user's transaction is in mempool, and market condition changed drastically, now 1 WETH = 2000 USDC 4. Without slippage protection user pays 2000 USDC for 1 WETH, which was not desired.

Recommended Mitigation: Add a `uint256 maxInputAmount` to the `TSwapPool::swapExactOutput` function

```
 function swapExactOutput(
        IERC20 inputToken,
+       uint256 maxInputAmount,
.
.
.
        inputAmount = getInputAmountBasedOnOutput(
            outputAmount,
            inputReserves,
            outputReserves
        );
+       if(inputAmount > maxInputAmount){
+           revert();
+       }
```

[H-3] In function `TSwapPool::sellPoolTokens` the input to `TSwapPool::swapExactOutput` is mismatched, causing users to receive incorrect amount of tokens

Description: The `sellPoolTokens` function is intended to allow users to sell `poolTokens` and receive `WETH`, however inside the function `swapExactOutput` function is called, which is wrong, because the user provides the input as `poolTokenAmount`, which should be used in function `swapExactInput`.

Impact: Users will swap incorrect amount of tokens, which will severly disrupt the protocol functionalitty.

Proof of Concept: `sellPoolTokens` function only takes input of `poolTokenAmount` as user intends to sell.

Add the following to the `TSwapPool.t.sol` and test

```
    function test_incorrectSellPoolTokens() public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();
        address swapper = makeAddr("swapper");
        uint256 intialPoolTokenBalance = 15e18;
        poolToken.mint(swapper, intialPoolTokenBalance);
        // initial liquidity provided as 1:1,
        // user is selling 1 poolToken, with added fees, user should
            not pay more than
        // 1.1 (~2) poolTokens
        vm.startPrank(swapper);
        poolToken.approve(address(pool), intialPoolTokenBalance);
        uint256 onePoolToken = 1e18;
        pool.sellPoolTokens(onePoolToken);
        uint256 finalTokenBalance = poolToken.balanceOf(swapper);
        assertTrue(finalTokenBalance - finalTokenBalance < 2e18);  //
            spent more than 2 pool tokens for the swap
        vm.stopPrank();
    }
```

Recommended Mitigation: Consider changing the implementation to `swapExactInput` from `swapExactOutput`, note this change will also require passing additional parameter `minOutputWETH` to be passed to `swapExactInput` function.

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

[H-4] In `TSwapPool::_swap` function the extra token given to user after every `swap_count` breaks the protocol invariant of `x * y= k`

Description: The protocol follows a strict invariant `x * y= k` where - `x`: The balance of the poolToken - `y`: The balance of WETH - `k`: The constant product of `x` and `y`

This means that whenever balance changes through swaps, the protocol should maintain a constant ratio of `x` and `y` such that `x * y` is always equal to `k`.

However in `_swap` function extra incentive is given after `swap_count` reaches `SWAP_COUNT_MAX` which will lead to break the core invariant and the protocol will be drained by malicious users making lot of swaps.

```
    if (swap_count >= SWAP_COUNT_MAX) {
        swap_count = 0;
        outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000)
            ;
    }
```

Impact: A malicious user can drain the funds by doing a lot of swaps, and receive the extra incentives over and over again resulting the funds to be drained.

Proof of Concept: 1. A user swaps 10 times to collect the extra incentive of `1_000_000_000_000_000_000` tokens 2. User continues to do so until all the funds are drained from the protocol

Add this test in `TSwapPool.t.sol` and test

Proof Of Code

```
function test_InvariantBroken() 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);
        poolToken.mint(user, 100e18);
        poolToken.approve(address(pool), type(uint256).max);
        uint256 outputWeth = 1e17;

        // making 9 swaps
        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 startingWethPool = int256(weth.balanceOf(address(pool)));
        int256 expectedDeltaWethPool = int256(-1) * int256(outputWeth);
        // breaks invariant at 10th swap
        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
        int256 endingWethPool = int256(weth.balanceOf(address(pool)));
        int256 actualDeltaWethPool = endingWethPool - startingWethPool;
        assertEq(actualDeltaWethPool, expectedDeltaWethPool);
}
```

Recommended Mitigation: We highly recommend to remove this extra incentive payout to keep the protocol invariant intact.

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

Medium

[M-1] `TSwapPool::deposit` function has a `deadline` parameter which is never used, causing the deposit function to execute even after the deadline

Description: According to the natspec “@param deadline The deadline for the transaction to be completed by”. However, `deadline` param is not checked `TSwapPool::deposit` function, which may cause a deposit function to be excuted in an unfavorable time for the depositor even after providing `deadline`

Impact: This will impact severley for the depositor as the transaction gets executed even when the the market is unfavorable, making financial loss to the depositor

Proof of Concept: Compiler errror `Unused function parameter. Remove or comment out the variable name to silence this warning.`

Recommended Mitigation:

Check the `deadline` parameter.

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

Low

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

Description:

In `TSwapPool::_addLiquidityMintAndTransfer` function `LiquidityAdded` event should have `poolTokensToDeposit` in third position, whereas `wethToDeposit` should be in second position.

Impact: This incorrect positioning might disrupt off-chain functionality that relies on this event.

Recommended Mitigation:

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

[L-2] In `TSwapPool::swapExactInput` returns default value, causing the caller receiving incorrect information

Description: `swapExactInput` function was expected to return an actual amount of token bought by user but has a return value `output` which was never assigned nor a value was explicitly returned.

Impact: Causing the caller receive default value of `0`, which is incorrect information

Recommended Mitigation:

```
{
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

-       uint256 outputAmount = getOutputAmountBasedOnInput(
+       uint256 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);
    }
```

Gas

[G-1] Unused local variable, wasting gas

Description: In function `TSwapPool::deposit` local variable `poolTokenReserves` reads the balance of `poolToken` wasting gas, as it's never used in the function.

Impact: Wastes gas by calling an external contract to fetch balance.

Recommended Mitigation:

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

Information

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is never used

Description:

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

[I-2] Lacking zero address check

Description:

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

[I-3] `PoolFactory::liquidityTokenSymbol` should `.symbol()` instead `.name()`

Description:

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

[I-4] `TSwapPool::Swap` event should have index params

Description:

It's a good practice to index 3 params when there are more than 3 params in an event

```
        event Swap(
            address indexed swapper,
-           IERC20 tokenIn,
+        IERC20 indexed tokenIn,
         uint256 amountTokenIn,
-        IERC20 tokenOut,
+        IERC20 indexed tokenOut,
         uint256 amountTokenOut
     );
```

[I-5] Zero address check in `TSwapPool::constructor`

Description:

```
+    if(i_wethToken == address(0)){
+        revert();
+    }
+    if(i_poolToken == address(0)){
+        revert();
+    }
     i_wethToken = IERC20(wethToken);
     i_poolToken = IERC20(poolToken);
```

[I-6] Constant is not required to pass to an error

Description:

In `TSwapPool::TSwapPool__WethDepositAmountTooLow`, `MINIMUM_WETH_LIQUIDITY` is passed, which is redundant.

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

[I-7] Variable update should follow Check Effect Interaction (CEI)

Description: Though `liquidityTokensToMint` is not a state variable still, following CEI is recommended.

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

[I-8] Missing Natspec in `TSwapPool::getOutputAmountBasedOnInput`

Description: Having natspec for functions are recommended `TSwapPool::getOutputAmountBasedOnInput` function does not have natspec

[I-9] Using magic numbers are discouraged in functions

Description:

```
+   uint256 constant FEE_NUMERATOR = 997;
+   uint256 constant FEE_DENOMINATOR = 1000;
-   uint256 inputAmountMinusFee = inputAmount * 997;
+   uint256 inputAmountMinusFee = inputAmount * FEE_NUMERATOR;
    uint256 numerator = inputAmountMinusFee * outputReserves;
-   uint256 denominator = (inputReserves * 1000) + inputAmountMinusFee;
+   uint256 denominator = (inputReserves * FEE_DENOMINATOR) +
   inputAmountMinusFee;
```

[I-10] Using magic numbers are discouraged in functions

Description:

```
+   uint256 constant FEE_NUMERATOR = 997;
+   uint256 constant FEE_DENOMINATOR = 10000;
    return
-       ((inputReserves * outputAmount) * 10000) /
-       ((outputReserves - outputAmount) * 997);
+       ((inputReserves * outputAmount) * FEE_DENOMINATOR) /
+       ((outputReserves - outputAmount) * FEE_NUMERATOR);
```

[I-11] `TSwapPool::swapExactInput` is marked public, but is never used internally

Description: Function `swapExactInput` was never used internally, it's recommended to markt 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-12] Missing Natspec in function `TSwapPool::swapExactInput`

Description: Having natspec for functions are recommended `TSwapPool::swapExactInput` function does not have natspec.