---

title: Protocol Audit Report author: Mauro Júnior date: February 11, 2024 header-includes:

- \usepackage
- \usepackage

\begin \centering \begin[h] \centering \includegraphics[width=0.5\textwidth] \end \vspace*{2cm} {\Huge\bfseries Protocol Audit Report\par} \vspace{1cm} {\Large Version 1.0\par} \vspace{2cm} {\Large\itshape Mauro\par} \vfill {\large \today\par}v \end

\maketitle

Prepared by: [Mauro Júnior] Lead Auditors:

- Mauro

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) (https://chain.link/education-hub/what-is-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 (https://www.youtube.com/watch?v=DLu35sIqVTM)

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 (https://etherscan.io/token/0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2) 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

```
y = Token Balance Y
x = Token Balance X
x * y = k
x * y = (x + Δx) * (y - Δy)
Δx = Change of token balance X
Δy = Change of token balance Y
β = (Δy / y)
α = (Δx / x)

Final invariant equation without fees:
Δx = (β/(1-β)) * x
Δy = (α/(1+α)) * y

Invariant with fees
ρ = fee (between 0 & 1, aka a percentage)
γ = (1 - p) (pronounced gamma)
Δx = (β/(1-β)) * (1/γ) * x
Δy = (αγ/1+αγ) * 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.

This codebase is based loosely on Uniswap v1 (https://github.com/Uniswap/v1-contracts/tree/master)

Getting Started

Requirements

- git (https://git-scm.com/book/en/v2/Getting-Started-Installing-Git)
  - You'll know you did it right if you can run `git --version` and you see a response like `git version x.x.x`

- foundry (https://getfoundry.sh/)
    - You'll know you did it right if you can run `forge --version` and you see a response like `forge 0.2.0 (816e00b 2023-03-16T00:05:26.396218Z)`

Quickstart

```
git clone https://github.com/Cyfrin/5-t-swap-audit
cd 5-t-swap-audit
make
```

Usage

Testing

```
forge test
```

Test Coverage

```
forge coverage
```

and for coverage based testing:

```
forge coverage --report debug
```

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

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.

Known Issues

- None

Disclaimer

The Mauro 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 (https://docs.codehawks.com/hawks-auditors/how-to-evaluate-a-finding-severity) severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

- In Scope:

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

Executive Summary

I liked alot auditing this swap codebase.

Issues found

Findings

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

Highs

[H-1] Incorrect fee calculation in `TSwapPool:getInputAmountBasedOnOutput` causing protocol take many tokens from the user, resulting in lost fees and giving the Liquidity Provider too much tokens from the fees.

Description The function `getInputAmountBasedOnOutput` function is supposed to calculate the amount of tokens a user should be depositing given an amount of output token. But, the function is miscalculating the resulting amount. Because it's calculating the fee by 10_000 instead of 1_000.

Impact Protocol taking more fees than expected from the users that are going to deposit

Recomended Mitigations

```
    function getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
-       return ((inputReserves * outputAmount) * 10_000) / ((outputReserves - outputAmount) * 997);
+       return ((inputReserves * outputAmount) * 1_000) / ((outputReserves - outputAmount) * 997);
    }
```

You could also add some constant in the state variables that calculates the fees.

[H-2] No slippage protection in `TSwapPool::swapExactOutput` can cause users to receive fewer tokens than wished for.

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`should specify a `maxInputAmount`.

Impact If market conditions change before a transaction process, the user could get a much worse swap than expected.

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 alot -> 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 They are gettinh charged way more money(user who swaps).

Recommended Mitigations Include an `maxInputAmount` so each user only has to spend to a specific amount and the user himself can predict how much money they will spend on this swap protocol.

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

[H-3] In `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to receive the incorrect amount of tokens they wanted to.

Description: The `sellPoolTokens` function is supposed to allow users to 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, creating a disruption of the protocol functionality.

Recommended Mitigations 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 (the `minWethToReceive` to be passed to `swapExactInput`)

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

You can also add a deadline to the function as there isn't a deadline.

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

Description: The protocol follows a 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 malicious user could drain the protocol by doing swaps over and over again and collecting the extra incentive given by the protocol The protocol 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

▶ Code

Recommended Mitigations You can just remove the extra incentive, or if you want to keep this functionality in, you should change the protocol invariant `x*y=z`. Or we should set aside tokens in the same way you can do with the fees.

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

Medium

[M-1] In `TSwapPool::deposit` is missing deadline check and this causes transactions to complete even after deadline since it isn't checked

Description: The `deposit` function accepts a deadline parameter, which according to the documentation and the natspec is "The deadline for the transaction to be completed by". But, this parameter is never used. Therefore, operations that add liquidity to the pool might be executed at unexpected times, in market conditions where the deposit rate is not favorable.

Impact: Transactions could be sent when market conditions are not favorable in a certain moment, even when adding a deadline parameter.

Proof of Concept: The `deadline` parameter is unused. This is the solidity compiler telling us this warning:

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)
    {...}
```

Now you are doing a check and using the deadline parameter!

Low

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

Description When the `TSwapPool::LiquidityAdded` event is emitted at the `TSwapPool::_addLiquidityMintAndTransfer` function, it puts parameter in the incorrect order, the `poolTokensToDeposit` value should go in third parameter position and `wethToDeposit` value should be the second.

Impact Event emission is incorrect, this could lead to off-chain malfunctioning in some functions.

Recomended Mitigations

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

[L-2] Default value returned by `TSwapPool::swapExactInput` will result in incorrect value given.

Description The `swapExactInput` function is supposed to return the actual amount of tokens bought by the caller of the msg. However, while it declares the named return value `output` it is never assigned a value, neither uses an explicit return statement.

Impact The return value will always be 0, giving incorrect information to the caller.

Recommended Mitigations

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

Informational

[I-1] `PoolFactory:PoolFactory__PoolDoesNotExist` error is not used anywhere in the code.

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

Should be removed!

[I-2] Lacking address(0) checks

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

[I-3] `PoolFactory:createPool` function should use the `.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/TSwapPool.sol: Line: 44
- Found in src/PoolFactory.sol: Line: 37
- Found in src/TSwapPool.sol: Line: 46
- Found in src/TSwapPool.sol: Line: 43