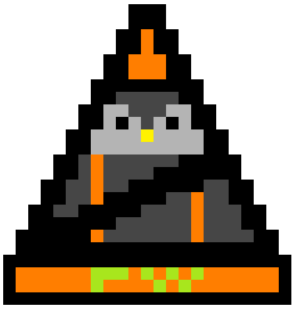

# Protocol Audit Report

Prepared by: Rong Wei Zhang

# 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. It is similar to Uniswap. To understand Uniswap, please watch this video: Uniswap Explained

# Disclaimer

Rong Wei Zhang 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: 1ec3c30253423eb4199827f59cf564cc575b46db

## 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

Followed Cyfrin Updraft's smart contract security course to identify vulnerabilities and possible improvement.

## Issues found

| Severity | Number of issues found |
|---|---|
| High | 4 |
| Medium | 2 |
| Low | 2 |
| info | 5 |
| Total | 13 |

# Findings

# High

[H-1] Give 1 token every 10 rounds breaks the invariant $X * Y = K$ and make the logic complex.

IMPACT: HIGH

LIKELIHOOD: HIGH

**Description:** Every 10 swaps, one extra token is gifted. This results in (for X = weth, Y = poolToken):

- `Y - 1` A `poolToken` is gifted.
- `X * (Y - 1) = XY - X = K - X != K` Invariant is broken.

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

**Impact:** This breaks the invariant and makes the program vulnerable. A malicious attacker could perform small trades on large value tokens to steal tokens from the pool.

**Proof of Concept:** One extra token is given every 10 swaps and breaks the invariant

```
emit log_named_int(key: "      Left", val: -1615624291816074340
[-1.615e18])
    ├─ emit log_named_int(key: "     Right", val: -615624291816074340
[-6.156e17])
```

An attacker can do this infinitely until there is not liquidity left in the pool.

**Recommended Mitigation:** Remove this or move the logic to take the gift token from somewhere that will not affect the invariant.

### [H-2] Incorrect calculations of the input token in `TSwapPool::getInputAmountBasedOnOutput()`.

IMPACT: HIGH

LIKELIHOOD: ALWAYS/HIGH

**Description:** Typo in hardcoded magic numbers resulting in a 91.3% fee.

```
(inputReserves * outputAmount * 10000) /
  ((outputReserves - outputAmount) * 997);
```

**Impact:** An unintended high ratio of fees

**Recommended Mitigation:** Use constant variables instead of hardcoded magic numbers as shown in [I-4].

```
+       uint256 constant FEE_PRECISION = 1000;
+       uint256 constant RATIO_AFTER_FEE = 997;
```

```
-       (inputReserves * outputAmount * 10000) /
-       ((outputReserves - outputAmount) * 997);
+       (inputReserves * outputAmount * FEE_PRECISION) /
+       ((outputReserves - outputAmount) * RATIO_AFTER_FEE);
```

### [H-3] No slippage protection in `TSwapPool::swapExactOutput()`

IMPACT: HIGH

LIKELIHOOD: MEDIUM

**Description:** There is no max input parameter that protects the user from slippage.

```
function swapExactOutput(
        IERC20 inputToken,
        IERC20 outputToken,
        uint256 outputAmount,
        uint64 deadline
    )
```

**Impact:** The user may pay significantly more when buying large amounts relative to the pool liquidity.

**Proof of Concept:**

1. There are 100 DAI and 10 Weth in the pool
2. The user wants to buy 90 DAI
3. It will cost him 90 Weth
4. Since there is no slippage protection and no maxInputAmount check, the user buys 90 DAI with 90 WETH which is 1 DAI/WETH
5. Meanwhile, the market price is still around 10 DAI/WETH
6. The user swapped DAI for WETH at 10x the market price.

**Recommended Mitigation:**

```
function swapExactOutput(
        uint256 maxInputAmount,
        IERC20 inputToken,
        IERC20 outputToken,
        uint256 outputAmount,
        uint64 deadline
    )
        public
        //audit-info: this function should be external
        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
TSwapPool__ExcessiveInputAmount(inputAmount, maxInputAmount);

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

### [H-4] `TSwapPool::sellPoolTokens()` uses the wrong function to sell tokens.

IMPACT: HIGH

LIKELIHOOD: ALWAYS/HIGH

**Description:** `TSwapPool::sellPoolTokens()` calls `TSwapPool::swapExactOutput()` to sell the token instead of `TSwapPool::swapExactInput()`. The `outputToken` for `TSwapPool::swapExactOutput()` is also wrong. The outputToken should be weth not the poolTokens and the output amount should at least be calculated using `TSwapPool::getOutputAmountBasedOnInput()`.

```
function sellPoolTokens(
        uint256 poolTokenAmount
    ) external returns (uint256 wethAmount) {
        return
@>          swapExactOutput(
                i_poolToken,
                i_wethToken,
@>              poolTokenAmount,
                uint64(block.timestamp)
            );
    }
```

**Impact:** It could cause loss and failure of selling. The function will not function as intended.

**Recommended Mitigation:** Use `TSwapPool::swapExactInput()` to sell.

## Medium

### [M-1] Unused parameter `uint64 deadline` in `TSwapPool::deposit` leads to missing deadline check, allowing swaps to be performed even after deadline.

IMPACT: HIGH

LIKELIHOOD: MEDIUM/LOW

**Description:** The `deadline` parameter is not used in `TSwapPool::deposit` to check if the deadline for the transaction has passed.

```
function deposit(
uint256 wethToDeposit,
uint256 minimumLiquidityTokensToMint,
uint256 maximumPoolTokensToDeposit,
//@audit unused parameter
uint64 deadline
)
```

**Impact:** Deposit could be sent when market conditions are unfavorable to deposit.

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

**Recommended Mitigation:** Add a check in the function `TSwapPool::deposit`

▶ Code

```
 function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        //@audit unused parameter
        uint64 deadline
    )
        external
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
+       if (block.timestamp > deadline) revert
TSwapPool__DeadlineHasPassed(deadline);
        .
        .
        .

    }
```

### [M-2] Weird ERC20s (Rebase, Fee on Transfer, and ERC777 tokens can break the invariants)

IMPACT: HIGH

LIKELIHOOD: MEDIUM

**Description:** ERC20 tokens that have weird transfer functions or logic can break the invariants.

**Impact:** It can steal the balance from the pool and lead to malfunctioning.

**Proof of Concept:**

1. A pool is created for a token that takes fee on transfer.
2. Someone deposits some amount of liquidity `K = X * Y`.
3. Someone swaps WETH for token.
4. The contract transfers token to the swapper.
5. A fee is deducted from the liquidity pool reserve, making `Y' = Y - fee`.
6. Now this invariant `K` becomes `K' = X * Y' = X * (Y - fee) = K - X * fee != K` for X and fee != 0.

**Recommended Mitigation:** Have a list of accepted tokens and know which tokens you are going to work with.

# Low

### [L-1] Event field emitted out of order in `TSwapPool::LiquidityAdded`

IMPACT: LOW

LIKELIHOOD: HIGH/ALWAYS

**Description:** `TSwapPool::_addLiquidityMintAndTransfer` emit the event with parameters out of order.

```
event LiquidityAdded(
        address indexed liquidityProvider,
        uint256 wethDeposited,
        uint256 poolTokensDeposited
    );

    .
    .
    .


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

**Impact:** Off-chain features may malfunction

**Recommended Mitigation:**

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

### [L-2] Output never updated in `TSwapPool::swapExactInput()`

IMPACT: LOW

LIKELIHOOD: HIGH/ALWAYS

**Description:** The output is never given a value in the function.

```
function swapExactInput(
    IERC20 inputToken, //user sell token
    uint256 inputAmount, //user sell amount
    IERC20 outputToken, //user buy token
    uint256 minOutputAmount, //use minimum buy amount
    uint64 deadline
)
    public
    //audit-info: this function should be external
    revertIfZero(inputAmount)
    revertIfDeadlinePassed(deadline)
returns (
//@audit output never updated
    uint256 output
)
```

**Impact:** The output will always be zero, misleading the caller.

**Recommended Mitigation:**

```
    function swapExactInput(
        IERC20 inputToken, //user sell token
        uint256 inputAmount, //user sell amount
        IERC20 outputToken, //user buy token
        uint256 minOutputAmount, //use minimum buy amount
        uint64 deadline
    )
        public
        //audit-info: this function should be external
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (
            //@audit output never updated
            uint256 output
        )
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

-         uint256 outputAmount = getOutputAmountBasedOnInput(
+         output = getOutputAmountBasedOnInput(
            inputAmount,
            inputReserves,
            outputReserves
        );

        if (output < minOutputAmount) {
```

```
            revert TSwapPool__OutputTooLow(output, minOutputAmount);
        }

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

# Informational

[I-1] `PoolFactory` constructor lacks zero address check

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

[I-2] The `liquidityTokenSymbol` should use ``.symbol()of the IERC20 token instead of the.name()inPoolFactory::createPool`

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

[I-3] Event fields are not all indexed

For events with more than 3 fields, 3 fields should be indexed, and for those with less than 3 fields, all the fields should be indexed for easier off-chain accessibility.

[I-4] Do not use magic numbers for code clarity

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

[I-5] No need to emit a constant

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