Protocol Audit Report

Version 1.0

MrSaade

Protocol Audit Report

MrSaade

October 2, 2024

Prepared by: [MrSaade] Lead Auditor/Auditors:

- MrSaade

Table of Contents

    * [H-4] `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to receive the incorrect amount of tokens
    * [H-5] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` breaks the protocol invariant of `x * y = k`
  - Low
    * [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order
    * [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
  - Informationals
    * [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed
    * [I-2] Lacking zero address checks
    * [I-3] `PoolFacotry::createPool` should use `.symbol()` instead of `.name()`
    * [I-4] Event is missing `indexed` fields

Protocol Summary

T-Swap is a decentralized Automated Market Maker (AMM) protocol designed for permissionless token swaps between ERC20 tokens and WETH. The protocol uses liquidity pools to facilitate swaps, where liquidity providers deposit tokens into pools and receive LP tokens representing their share. T-Swap features two primary swap functions: swapExactInput and swapExactOutput, allowing users to specify the amount of tokens they wish to exchange or receive. The protocol accrues a 0.3% fee on every swap, incentivizing liquidity providers and ensuring continuous liquidity for the platform.

Disclaimer

This audit report is intended to identify potential vulnerabilities and issues in the T-Swap smart contract. The analysis is based on the provided source code and aims to highlight security risks, bugs, and inefficiencies. While every effort has been made to identify all potential issues, no audit can guarantee the complete security or functionality of the protocol. The responsibility for addressing and mitigating any identified risks lies solely with the project team. The audit should not be considered a warranty of the protocol's security, and the auditor assumes no liability for any consequences arising from the use or interpretation of this report.

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

Scope

- Commit Hash: 1ec3c30253423eb4199827f59cf564cc575b46db
- In Scope:

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

- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum

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

The T-Swap protocol was reviewed for security, functionality, and alignment with its intended design. The audit identified several critical and low-severity issues, including a significant miscalculation in the fee logic that overcharges users during swaps. Additionally, issues related to event handling, token management, and unused parameters were found. Correcting these vulnerabilities is crucial to maintaining the integrity of the platform and ensuring users are not unfairly penalized during token swaps. Despite these findings, the protocol demonstrates a solid foundation with well-established AMM mechanics, providing a robust framework for decentralized asset exchange once the issues are addressed.

Issues found

Severtity	Number of issues found
High	5
Medium	0
Low	2
Info	4
Total	11

Findings

High

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

Description: The `deposit` function in the TSwapPool contract includes a `deadline` parameter, but it is not used in the function logic. Typically, a deadline parameter is used to impose a time constraint on when a transaction can be executed. If this constraint is missing, it could expose the protocol to front-running, delayed transactions, or malicious exploits where users manipulate transaction timing for their advantage.

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

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

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

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint, // LP tokens -> if empty,
            we can pick 100% (100% == 17 tokens)
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
+       revertIfDeadlinePassed(deadline)
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
```

[H-2] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocll to take too many tokens from users, resulting in lost fees

Description: The `getInputAmountBasedOnOutput` function is designed to calculate how much input a user must provide to receive a specified amount of output tokens, taking into account the pool's reserves and the protocol fee. However, the function incorrectly scales the input by 10_000 when calculating the fee instead of the expected 1_000. This error results in the protocol collecting more fees than intended, which negatively impacts users.

Impact: Protocol takes more fees than expected from users.

Recommended Mitigation:

```
    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);
    }
```

[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 transaciton 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 = …
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.

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

Proof of Concept:

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`)

```
    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));
    }
```

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

[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 untill 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);
        poolToken.approve(address(pool), type(uint256).max);
        poolToken.mint(user, 100e18);

        // Perform 9 swaps
        for (uint256 i = 0; i < 9; i++) {
            pool.swapExactOutput(
                poolToken,
                weth,
                outputWeth,
                uint64(block.timestamp)
            );
        }

        // Capture starting balance
        int256 startingY = int256(weth.balanceOf(address(pool)));

        // Perform the 10th swap
        pool.swapExactOutput(
            poolToken,
            weth,
            outputWeth,
            uint64(block.timestamp)
        );
        vm.stopPrank();

        // The expected delta should now account for the 1 WETH reward
        int256 expectedDeltaY = int256(-1) * int256(outputWeth) - int256(1e18); // deduct 1e18 for the reward

        // Check ending balance
        uint256 endingY = weth.balanceOf(address(pool));
        int256 actualDeltaY = int256(endingY) - int256(startingY);

        // Assert that the actual delta matches the expected delta
        assertEq(actualDeltaY, expectedDeltaY);
    }
```

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.

```
-        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);
-        }
```

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 functions potentially malfunctioning.

Recommended Mitigation:

```diff
- 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 `ouput` it is never assigned a value, nor uses an explict return statement.

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

Recommended Mitigation:

```diff
    {
        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);
    }
```

Informationals

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed

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

[I-2] Lacking zero address checks

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

[I-3] `PoolFacotry::createPool` should use `.symbol()` instead of `.name()`

```diff
-        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