T-Swap Protocol Audit Report

Version 1.0

Katlego Molokoane

April 25, 2025

Protocol Audit Report

Katlego Molokoane

Prepared by:

- Katlego Molokoane

Assisting Auditors:

- None

Table of Contents

    * [H-1] Missing Deadline Check in `TSwapPool::deposit` causes transactions to complete even after the deadline.
    * [H-2] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users, resulting in lost fees
    * [H-3] The absence of slippage protection in `TSwapPool::swapExactOutput` exposes users to the risk of receiving significantly fewer tokens than expected.
    * [H-4] The `TSwapPool::sellPoolTokens` function incorrectly matches input and output tokens, resulting in users receiving an inaccurate amount of tokens.
    * [H-5] In `TSwapPool::_swap`, the additional tokens distributed to users after every `swapCount` violate the protocol's invariant of `x * y = k`, disrupting the balance between token reserves.
  - Low
    * [L-1] The `TSwapPool::LiquidityAdded` event logs its parameters in an incorrect order, resulting in potential inconsistencies with off-chain data processing.
    * [L-2] The `TSwapPool::swapExactInput` function returns a default value, resulting in an incorrect return value being provided to the caller.
  - Informational
    * [I-1] The `PoolFactory::PoolFactory__PoolDoesNotExist` error is unused within the contract and should be removed to improve code clarity and maintainability.
    * [I-2] The constructor is lacking zero address checks
    * [I-3] `PoolFacotry::createPool` should use `.symbol()` instead of `.name()`
    * [I-4] Event is missing `indexed` fields

Protocol Summary

The T-Swap Protocol is a decentralized exchange (DEX) enabling permissionless token swaps using Automated Market Maker (AMM) pools. It pairs ERC20 tokens with WETH, maintaining the invariant `x * y = k` for fair pricing. Liquidity providers earn fees by depositing tokens into pools, while users can swap tokens efficiently without intermediaries. The protocol supports two swap methods: `swapExactInput` and `swapExactOutput`, ensuring flexibility and transparency for users in a decentralized trading environment.

Disclaimer

Disclaimer

This security audit was conducted with the utmost diligence to identify vulnerabilities within the provided codebase during the allocated time frame. However, the findings in this document are not exhaustive, and no guarantees are made regarding the absence of additional vulnerabilities. This audit does not constitute an endorsement of the underlying business model, product, or team. The review was strictly limited to the security aspects of the Solidity implementation of the smart contracts. The auditor assumes no liability for any issues arising from the use of the audited code.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

The CodeHawks severity matrix is utilized to determine severity. See the documentation for more details.

Audit Details

Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda

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

The T-Swap Protocol is a decentralized exchange (DEX) enabling permissionless token swaps using Automated Market Maker (AMM) pools. It ensures fair pricing through the invariant `x * y = k`. Liquidity providers earn fees, while users benefit from efficient, transparent, and decentralized trading. The audit identified 11 issues, including 5 high-severity findings, requiring immediate attention to enhance security.

Issues found

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

Findings

High

[H-1] Missing Deadline Check in `TSwapPool::deposit` causes transactions to complete even after the deadline.

Description:
The `deposit` function includes a `deadline` parameter, which is intended to specify the latest time by which the transaction should be completed. However, this parameter is not utilized within the

function. As a result, transactions can be executed even after the specified deadline, potentially under unfavorable market conditions.

Impact:
This issue allows transactions to proceed beyond the intended deadline, exposing users to unfavorable market conditions and potential financial losses.

Proof of Concept:
The `deadline` parameter is declared but never used in the function implementation.

Recommended Mitigation:
Introduce a deadline validation mechanism to ensure the transaction is reverted if the deadline has passed. For example:

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint, // LP tokens -> if empty,
            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 protocol to take too many tokens from users, resulting in lost fees

Description: The `getInputAmountBasedOnOutput` function is designed to calculate the amount of input tokens required to obtain a specified amount of output tokens. However, the function currently miscalculates the fee by scaling the amount by `10_000` instead of the correct value of `1_000`. This results in users being charged higher fees than intended.

Impact: The protocol collects more fees than expected, leading to user dissatisfaction and potential loss of trust in the system.

Proof of Concept:
The following line in the function demonstrates the incorrect scaling factor:

```
return ((inputReserves * outputAmount) * 10_000) / ((outputReserves -
    outputAmount) * 997);
```

Recommended Mitigation:

```
1      function getInputAmountBasedOnOutput(
2          uint256 outputAmount,
3          uint256 inputReserves,
4          uint256 outputReserves
5      )
6          public
7          pure
8          revertIfZero(outputAmount)
9          revertIfZero(outputReserves)
10         returns (uint256 inputAmount)
11     {
12 -         return ((inputReserves * outputAmount) * 10_000) / ((
       outputReserves - outputAmount) * 997);
13 +         return ((inputReserves * outputAmount) * 1_000) / ((
       outputReserves - outputAmount) * 997);
14     }
```

The adjustment ensures that the protocol xharges the correct fees, maintaining user trust and protocol integrity.

[H-3] The absence of slippage protection in `TSwapPool::swapExactOutput` exposes users to the risk of receiving significantly fewer tokens than expected.

Description: The `swapExactOutput` function lacks slippage protection. Similar to the `TSwapPool::swapExactInput` function, which enforces a `minOutputAmount` to safeguard users, the `swapExactOutput` function should implement a `maxInputAmount` to limit the maximum tokens a user is willing to spend.

Impact: If market conditions fluctuate before the transaction is processed, users may receive a significantly less favorable swap outcome.

Proof of Concept:

1. The price of 1 WETH is initially 1,000 USDC.
2. A user initiates a `swapExactOutput` transaction to receive 1 WETH with the following parameters:
    - `inputToken = USDC`
    - `outputToken = WETH`
    - `outputAmount = 1`

- `deadline = any value`

3. The function does not enforce a `maxInputAmount`.

4. While the transaction is pending in the mempool, the market price of WETH increases significantly to 10,000 USDC (a 10x increase).

5. The transaction completes, and the user ends up sending 10,000 USDC to the protocol instead of the expected 1,000 USDC.

Recommended Mitigation: Protocol 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] The `TSwapPool::sellPoolTokens` function incorrectly matches input and output tokens, resulting in users receiving an inaccurate amount of tokens.

Description: The `sellPoolTokens` function is designed to allow users to seamlessly sell pool tokens in exchange for WETH. Users specify the exact number of pool tokens they wish to sell through the `poolTokenAmount` parameter. However, the function currently miscalculates the swapped amount due to the use of the `swapExactOutput` function instead of the appropriate `swapExactInput` function, as users are specifying the exact input tokens rather than the output.

Impact: Users may exchange an incorrect amount of tokens, leading to a significant disruption in the protocol's 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`)

```
    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 wise to add a deadline to the function, as there is currently no deadline.

[H-5] In `TSwapPool::_swap`, the additional tokens distributed to users after every `swapCount` violate the protocol's invariant of `x * y = k`, disrupting the balance between token reserves.

Description: The protocol adheres to a strict invariant defined as `x * y = k`, where:
- `x` represents the balance of the pool token,
- `y` represents the balance of WETH, and
- `k` is the constant product of these two balances.

This implies that any changes to the balances within the protocol must preserve the ratio between the two amounts, ensuring the constant `k` remains unchanged. However, this invariant is violated due to the additional incentive mechanism in the `_swap` function, which ultimately leads to the gradual depletion of the protocol's funds over time.

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 exploit the protocol by performing numerous swaps to repeatedly collect the additional incentive distributed by the protocol, ultimately draining its funds.

Proof of Concept:

1. A user performs 10 swaps and collects the additional incentive of `1_000_000_000_000_000_000` tokens.

2. The user continues executing swaps repeatedly, ultimately depleting all the protocol's funds.

Proof Of Code

Place the following into `TSwapPool.t.sol`.

```
    function testProtocolInvariantViolation() 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 inputPoolTokens = 1e18;

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

    // Perform multiple swaps to exploit the incentive mechanism
    for (uint256 i = 0; i < 10; i++) {
        pool.swapExactInput(poolToken, inputPoolTokens, weth, 1, uint64
            (block.timestamp));
    }

    int256 startingX = int256(poolToken.balanceOf(address(pool)));
    int256 expectedDeltaX = int256(inputPoolTokens * 10);

    pool.swapExactInput(poolToken, inputPoolTokens, weth, 1, uint64(
        block.timestamp));
    vm.stopPrank();

    uint256 endingX = poolToken.balanceOf(address(pool));
    int256 actualDeltaX = int256(endingX) - int256(startingX);

    // Assert that the protocol's invariant is violated
    assertEq(actualDeltaX, expectedDeltaX);
    }
```

Recommended Mitigation: Remove the additional incentive mechanism. If retaining this feature is necessary, adjustments should be made to account for the impact on the `x * y = k` protocol invariant. Alternatively, tokens should be set aside in a manner similar to how fees are handled.

```
-        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] The `TSwapPool::LiquidityAdded` event logs its parameters in an incorrect order, resulting in potential inconsistencies with off-chain data processing.

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTrans` function, the logged values are in an incorrect order. The `poolTokensToDeposit` value should be logged as the third parameter, while the `wethToDeposit` value should be logged as the second parameter.

Impact: The incorrect emission of the event may result in potential malfunctions or inconsistencies in off-chain processes that rely on accurate event data.

Recommended Mitigation:

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

[L-2] The `TSwapPool::swapExactInput` function returns a default value, resulting in an incorrect return value being provided to the caller.

Description: The `swapExactInput` function is intended to return the actual amount of tokens purchased by the caller. However, while it declares the named return value `output`, it is neither assigned a value nor utilizes 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);
        }

```

```
14 -         _swap(inputToken, inputAmount, outputToken, outputAmount);
15 +         _swap(inputToken, inputAmount, outputToken, output);
16       }
```

Informational

[I-1] The `PoolFactory::PoolFactory__PoolDoesNotExist` error is unused within the contract and should be removed to improve code clarity and maintainability.

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

[I-2] The constructor is lacking zero address checks

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

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

```
1 -         string memory liquidityTokenSymbol = string.concat("ts",
      IERC20(tokenAddress).name());
2 +         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.

4 Found Instances

- Found in src/PoolFactory.sol Line: 35

```
1       event PoolCreated(address tokenAddress, address poolAddress);
```

- Found in src/TSwapPool.sol Line: 52

```
1       event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
1       event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
1       event Swap(
```