LOKAPAL

BLOCKCHAIN SECURITY AUDITS

TSwap Audit Report

Version 1.0

Lokapal

February 1, 2025

TSwap Audit Report

Lokapal

February 1, 2025

Prepared by:

- Lokapal

Lead Auditor:

- Ricardo Pintos

Table of Contents

    * [H-2] Lack of slippage protection in `TSwapPool::swapExactOutput`, which allows users to potentially receive fewer tokens
    * [H-3] `TSwapPool::sellPoolTokens` mismatches input and output tokens causing users to receive the incorrect amount of tokens
    * [H-4] In `TSwapPool::_swap`, the extra tokens given to users after every `swapCount` breaks the protocol invariant of `x * y = k`
  - Medium
    * [M-1] `TSwapPool::deposit` is missing deadline check causing transactions to complete even after the user's deadline
  - Low
    * [L-1] `TSwapPool::LiquidityAdded` event parameters out of order
    * [L-2] `TSwapPool::swapExactInput` does not have a natspec
    * [L-3] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
  - Informational
    * [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed
    * [I-2] Missing zero address check in the `PoolFactory::constructor`
    * [I-3] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`
    * [I-4] `PoolFactory::PoolCreated` event does not have indexed parameters
    * [I-5] Missing zero address check in the `TSwapPool::constructor`
    * [I-6] `TSwapPool__WethDepositAmountTooLow` is emitting the constant `MINIMUM_WETH_LIQUIDITY`
    * [I-7] The `TSwapPool::poolTokenReserves` variable is not used:
    * [I-8] Use of magic numbers
    * [I-9] `TSwapPool::swapExactInput` function should be marked as `external`
    * [I-10] Some `public` functions are in the same section as the getter functions
    * [I-11] Missing `deadline` parameter in `swapExactOutput` natspec

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.

Disclaimer

The LOKAPAL 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

This security review was conducted as part of Cyfrin Updraft's `Smart Contract Security` course. We added 2 findings to the final list of the original project.

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	3
Info	11
Total	19

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users, resulting in lost fees

Description: The `getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens a user should deposit given an amount of output tokens. However, the function currently miscalculates the resulting amount. When calculating the fee, it scales the amount by 10_000 instead of 1_000.

Impact: Protocol takes more fees than expected from users.

Proof of Concept: Run this test on your `TSwapPool.t.sol` unit suite:

```
function testHigherFees() public view {
        uint256 outputAmount = 10e18;
        uint256 inputReserves = 100e18;
        uint256 outputReserves = 100e18;
        uint256 correctFeesPrecision = 1000;
        uint256 expectedFees = ((inputReserves * outputAmount) *
            correctFeesPrecision) /
            ((outputReserves - outputAmount) * 997);
        uint256 actualFees = pool.getInputAmountBasedOnOutput(
            outputAmount,
            inputReserves,
            outputReserves);
        assert(actualFees > expectedFees);
    }
```

Recommended Mitigation: Consider fixing the precision value:

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

[H-2] Lack of slippage protection in `TSwapPool::swapExactOutput`, which allows users to potentially receive fewer tokens

Description: The `swapExactOutput` function does not include any slippage protection. The `TSwapPool::swapExactInput` function specifies a `minOutputAmount`. So, the `swapExactOutput` should specify a `maxInputAmount`.

```
// IN THE `swapExactInput` FUNCTION
```

```
if (outputAmount < minOutputAmount) {
    revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
}
```

Impact: If market conditions change before the transaction 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 = 1 minute 3. The function does not offer a maximum input amount, 4. As the transaction is pending in the mempool, the market conditions change, 5. The price moves from 1,000 USDC per WETH to 10,000. 10x more than the user expected, 6. The transaction completes, but the user sent the protocol 10,000 USDC instead of the expected 1,000.

Recommended Mitigation: Consider adding the proper slippage protection and the corresponding custom error:

```
    // ERRORS
+   error TSwapPool__InputTooHigh(uint256 actual, uint256 max);

    function swapExactOutput(
        IERC20 inputToken,
        IERC20 outputToken,
        uint256 outputAmount,
+       uint256 maxInputAmount,
        uint64 deadline
    )
        public
        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__InputTooHigh(inputAmount, maxInputAmount);
+   }

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

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

Description: The `seelPoolTokens` 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. But because users specify the exact amount of input tokens, the `swapExactInput` function is the one that should be called.

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

Impact: Users will swap the wrong amount of tokens, which is a severe disruption of protocol functionality.

Proof of Concept: Run this test on your `TSwapPool.t.sol` unit suite:

```
function testSellPoolTokensReturnsInput() public {
    vm.startPrank(liquidityProvider);
    weth.approve(address(pool), type(uint256).max);
    poolToken.approve(address(pool), type(uint256).max);
    pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));

    uint256 amountToSell = 1e18;

    uint256 inputReserves = poolToken.balanceOf(address(pool));
    uint256 outputReserves = weth.balanceOf(address(pool));
    uint256 amountToSellWithFees = pool.getInputAmountBasedOnOutput(
        amountToSell, inputReserves, outputReserves);

    uint256 ammounToReceiveWithFees = pool.sellPoolTokens(amountToSell);
    vm.stopPrank();
    assert(ammounToReceiveWithFees == amountToSellWithFees);
}
```

Recommended Mitigation: Consider using the `swapExactInput` function instead of `swapExactOutput` in the `seelPoolTokens` function. Also, the format of the previous parameters should be modified to match the `swapExactInput` function:

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

[H-4] 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` is the balance of the pool token, - `y` is the balance of WETH, - `k` is 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 overtime the protocol funds will be drained.

```
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 multiples swaps and collecting the extra incentive given out by the protocol. This breaks the invariant `k`.

Proof of Concept: Run this test on your `TSwapPool.t.sol` unit suite:

```
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.mint(user, 100e18);
    poolToken.approve(address(pool), type(uint256).max);
    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 startingY = int256(weth.balanceOf(address(pool)));
    int256 expectedDeltaY = int256(-1) * int256(outputWeth);
    // The 10th swap gives the incentive, breaking the invariant
    pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.timestamp));
    vm.stopPrank();

    uint256 endingY = weth.balanceOf(address(pool));
    int256 actualDeltaY = int256(endingY) - int256(startingY);
    // This assertion will fail
    assertEq(actualDeltaY, expectedDeltaY);
}
```

Recommended Mitigation: Consider removing the extra incentive. Otherwise, the protocol should account for the change in the invariant after each 10th swap.

```diff
-   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` is missing deadline check causing transactions to complete even after the user's deadline

Description: The `deposit` function accepts a deadline parameter, which according to the documentation is "The deadline for the transaction to be completed by". However, this parameter is never used. As a consequence, operations that add liquidity to the pool might be executed at unexpected times, at market conditions where the deposit rate is unfavorable.

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

Proof of Concept: This is the terminal message when running `forge build` on this project:

```
Compiler run successful with warnings:
Warning (5667): Unused function parameter. Remove or comment out the variable
    name to silence this warning.
   --> src/TSwapPool.sol:117:9:
    |
117 |         uint64 deadline
    |         ^^^^^^^^^^^^^^^
```

Recommended Mitigation: Consider adding the `revertIfDeadlinePassed` modifier:

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

Low

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

Description: There is a mismatch between the parameters in the event `LiquidityAdded`:

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

And the emitted `LiquidityAdded` in the `_addLiquidityMintAndTransfer` function:

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

Impact: When querying the event logs, users will have the information mixed up.

Proof of Concept:

1. User adds liquidity,
2. User queries the amount of `weth` deposited,
3. User actually gets the amount of `poolTokens` deposited.

Recommended Mitigation: Consider fixing the `emit` to match the event parameters order:

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

[L-2] `TSwapPool::swapExactInput` does not have a natspec

Description: The `swapExactInput` function does not have a natspec. This can leave users without proper information about the function that they are calling. Additionally, other important functions have their respective natspec.

Recommended Mitigation: Consider adding the natspec for the `swapExactInput` function. This is a suggestion:

```
/*
* @notice swapExactInput will swap the input token for the output token
* @param inputToken the token the user is swapping
* @param inputAmount the amount the user is swapping
* @param outputToken the token the user is receiving
* @param minOutputAmount the minimum amount the user wants to receive
* @param deadline the deadline for the transaction to be completed by
*/
```

[L-3] 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 `output` it is never assigned a value, nor uses an explicit return statement.

Impact: The returned value will always be zero, giving incorrect information to the caller.

Proof of Concept: Run this test on your `TSwapPool.t.sol` unit suite:

```solidity
function testSwapExactInputReturnsZero() 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.approve(address(pool), 10e18);
        uint256 expected = 9e18;

        uint256 shouldNotBeZero = pool.swapExactInput(poolToken, 10e18, weth,
            expected, uint64(block.timestamp));
        vm.stopPrank();
        assert(shouldNotBeZero == 0);
    }
```

Recommended Mitigation: Consider replacing `outputAmount` for `output` on the `swapExactInput` function:

```diff
{
    uint256 inputReserves = inputToken.balanceOf(address(this));
    uint256 outputReserves = outputToken.balanceOf(address(this));
+   uint256 output = getOutputAmountBasedOnInput
-   uint256 outputAmount = getOutputAmountBasedOnInput
    (
        inputAmount,
        inputReserves,
        outputReserves
    );

+   if (output < minOutputAmount){revert TSwapPool__OutputTooLow(output,
   minOutputAmount);}
-   if (outputAmount < minOutputAmount){revert TSwapPool__OutputTooLow(
   outputAmount, minOutputAmount);}

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

Informational

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

Description: The `PoolFactory::PoolFactory__PoolDoesNotExist` error is not used in the contract:

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

Recommended Mitigation: Consider removing the error or implementing it in the corresponding section:

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

[I-2] Missing zero address check in the `PoolFactory::constructor`

Description: The `constructor` does not have a zero address check on the address argument:

```
    constructor(address wethToken) {
        i_wethToken = wethToken;
    }
```

Recommended Mitigation: Consider adding a zero address check and the corresponding custom error:

```
    // ERRORS
+   error PoolFactory__NotZeroAddress();

    // FUNCTIONS
    constructor(address wethToken) {
+   if (address(wethToken) != address(0)) {
+       i_wethToken = wethToken;
+   } else {
+       revert PoolFactory__NotZeroAddress();
+   }
-       i_wethToken = wethToken;
    }
```

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

Description: The `PoolFactory::createPool` function uses the `.name()` to complete the LT symbol, but there is already the `.symbol()` function, which returns a more appropriate reference and is a shorter string than the token name:

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

Recommended Mitigation: Consider replacing `.name()` for `.symbol()`:

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

[I-4] `PoolFactory::PoolCreated` event does not have indexed parameters

Description: The `PoolFactory::PoolCreated` event has two address parameters, but none are flagged as `indexed`:

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

Recommended Mitigation: Consider adding the `indexed` flag to both parameters, which will make them easier to query by off-chain resources:

```
+   event PoolCreated(address indexed tokenAddress, address indexed poolAddress
    );
-   event PoolCreated(address tokenAddress, address poolAddress);
```

[I-5] Missing zero address check in the `TSwapPool::constructor`

Description: The `constructor` does not have a zero address check on the address argument:

```
    constructor(
            address poolToken,
            address wethToken,
            string memory liquidityTokenName,
            string memory liquidityTokenSymbol
        ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
            i_wethToken = IERC20(wethToken);
            i_poolToken = IERC20(poolToken);
        }
```

Recommended Mitigation: Consider adding a zero address check and the corresponding custom error:

```
    // ERRORS
```

```diff
+   error TSwapPool__NotZeroAddress();

    // FUNCTIONS
    constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
+       if (address(wethToken) != address(0) && address(poolToken) != address
    (0)) {
+           i_wethToken = IERC20(wethToken);
+           i_poolToken = IERC20(poolToken);
+       } else {
+           revert PoolFactory__NotZeroAddress();
+       }
-       i_wethToken = IERC20(wethToken);
-       i_poolToken = IERC20(poolToken);
    }
```

[I-6] `TSwapPool__WethDepositAmountTooLow` is emitting the constant `MINIMUM_WETH_LIQUIDITY`

Description: The `TSwapPool__WethDepositAmountTooLow` error has the parameter `minimumWethDeposit` :

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

This error is used only in the `deposit` function, taking the constant `MINIMUM_WETH_LIQUIDITY` as an argument.

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

That constant can already be accessed with the `getMinimumWethDepositAmount` getter function, in case the user needs that information. This makes the revert less gas efficient.

Recommended Mitigation: Consider removing the `minimumWethDeposit` as a parameter from the `TSwapPool__WethDepositAmountTooLow` error, therefore removing the need to add the `MINIMUM_WETH_LIQUIDITY` constant:

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

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

[I-7] The `TSwapPool::poolTokenReserves` variable is not used:

Description: This line in the `deposit` function has a variable that is not used:

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

This issue even appears when building the project:

```
Warning (2072): Unused local variable.
   --> src/TSwapPool.sol:131:13:
    |
131 |              uint256 poolTokenReserves = i_poolToken.balanceOf(address(
    this));
    |
```

Recommended Mitigation: Consider removing it from the function:

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

[I-8] Use of magic numbers

Description:

getOutputAmountBasedOnInput

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

getInputAmountBasedOnOutput return ((inputReserves * outputAmount) * 10000) / ((outputReserves - outputAmount) * 997);

Recommended Mitigation:

[I-9] `TSwapPool::swapExactInput` function should be marked as `external`

Description: The `swapExactInput` function is marked `public` but is not used internally. This increase gas cost in the deployment.

Recommended Mitigation: Consider using the `external` flag for the `swapExactInput` function.

[I-10] Some `public` functions are in the same section as the getter functions

Description: The `getPoolTokensToDepositBasedOnWeth` and `totalLiquidityTokenSupply` functions are grouped with the getter functions.

Recommended Mitigation: Consider adding a // PUBLIC VIEW // section for those functions.

[I-11] Missing `deadline` parameter in `swapExactOutput` natspec

Description: The `swapExactOutput` function does not have the corresponding natspec information for the `deadline` parameter.

Recommended Mitigation: Consider adding the same natspec line of the `swapExactInput` for the `deadline` parameter:

```
    /* @param deadline the deadline for the transaction to be completed by
```