TSwap Audit Report

Prepared by: HEXXA Protocol

TSwap Audit Report

- Title: TSwap Audit Report
- author: Gaurang Bharadava
- date: December 31, 2024

Prepared by: HEXXA Protocol

- Lead Auditors: Gaurang Bharadava

Assisting Auditors:

- None

Table of contents

▶ Details
See table

About Gaurang Bharadava

Gaurang Bharadava is experienced smart contract engineer and security researcher. Building HEXXA Protocol, will provide Smart contract development and security audit.

Disclaimer

The HEXXA Protocol 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

Audit Details

The findings described in this document correspond the following commit hash:

```
e643a8d4c2c802490976b538dd009b351b1c8dda
```

Scope

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

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

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

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Info	3
Gas	0
Total	10

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` function causes protocol to take to many tokens from users

Description: The `getInputAmountBasedOnOutput` function intented to calculate the amount of token a user should deposit given amount of tokens of output tokens. However thr function currently miscalculate the resulting amount. when calculating the fees, it scales the amont by `10_000` instaed of `1_000`.

Impact: Protocol takes more fee then 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) * 10000) /
            ((outputReserves - outputAmount) * 997);
        return
+           ((inputReserves * outputAmount) * 1000) /
            ((outputReserves - outputAmount) * 997);
    }
```

[H-2] Lack of slippage protection in `TSwapPool::swapExectOutput` function cause user to potentially recieve way fewer tokens

Description: The `swapExectOutput` function does not include any sort of slippage protection. This function is same to what is done in `TSwapPool::swapExectInput` function, where the function specifies a `minOutputAmount`, The `swapExectOutput` function should specify `maxInputAmount`.

Impact: If market conditions change before the transaction processes, the upper could get a much worse swap.

Recommended Mitigation:

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

[H-3] `TSwapPool::sellPoolToken` mismatches input and output tokens causing user to recieve incorrect amount of token

Description: The `sellPoolToken` function is intended to allow user to easily sell pool tokens and recieve WETH in exchange. User indecates that how many pool tokens they are willing to sell in the `poolTokenAmount` parameter. Howwver thr function currently miscalculate the swapped amount.

This is due to the fact that the `swapExectOutput` fuction is called, whereas the `swapExectInput` function is the one that should be called. Because user specify the exect amount of input tokens, not output tokens.

Impact: User will swap wrong amount of token, which is severe disruption of protocol functionalities.

Recommended Mitigation: Consider changing the implementation to use `swapExectInput` instead of `swapExectOutput` function. Note that this whould also require changing the `sellPoolTokens` function to accept a new parameter (ie `minWethToRecieve` To be passed to `swapExectInput`).

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

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

Description: The protocol follows a strict invariant of `x* y = k`. Where:

- `x`: The balance of pool token
- `y`: The balance of WETH
- `k`: The constant product of balances

This means, that when the balances change in the protocol, the ratio between the two amounts should remain constant. However this is broken every `10` swap. Meaning that over time the protocol funds will be drain.

The following block of code is responcible 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);
        }
        emit Swap(
            msg.sender,
            inputToken,
            inputAmount,
            outputToken,
            outputAmount
        );
```

Impact: A user could maliciously drain the protocol of the funds by doing lots of swaps and collecting the extra incentive given out by the protocol.

Proof of Concept:

```
        function testInveriantBroken() 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;
        int256 startingY = int256(poolToken.balanceOf(address(pool)));
        int256 expectedDeltaY = int256(-1) * int256(outputWeth);

        vm.startPrank(user);
        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));
        vm.stopPrank();

        uint256 endingY = weth.balanceOf(address(pool));
        int256 actualDeltaY = int256(endingY) - int256(startingY);
```

```
        assertEq(actualDeltaY, expectedDeltaY);
    }
```

Recommended Mitigation: Remove the extra incentive.

```
-       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` function is missing deadline check causing transaction to complete even after deadline

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

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

Recommended Mitigation: Consider making following changes to the function

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

Low

[L-1] `TSwapPool::LiquidityAdded` event has parameter out of order causing event to emit incorrect information

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTransfer` function, it logs value in an incorrect order.

Impact: event emission is incorrect, leading to off-chain functions potentially malfunctioning.

Recommended Mitigation:

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

[L-2] default value returned by `TSwapPool::swapExectInput` function results in incorrect return value given

Description: The `swapExectInput` function is expected to return the actual amount of token bought by a caller. Hovever, while it decalres the named return value `output` it is never assigned a value, nor used an explicit return statement.

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

Recommended Mitigation:

```
    function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 output)
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

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

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

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

Informational

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

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

[I-2] Laking zero address checks

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

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

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

Gas