TSwapPool Audit Report

Version 1.0

Luka Nikolic

February 17, 2024

TSwapPool Audit Report

Luka Nikolic

Feb 11, 2024

Prepared by: Luka Nikolic

Lead Security Researcher:

- Luka Nikolic

Table of Contents

  - [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 transaction to complete even after the deadline
- 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
- Gas
  - [G-1] `TSwapPool::deposit` don't need this line
- Informational
  - [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed
  - [I-2] Lacking zero address checks
  - [I-3] `PoolFactory;;liquidityTokenSymbol` should use `.symbol()` instead of `.name()`
  - [I-4]: Event is missing `indexed` fields
  - [I-5] `TSwapPool::deposit` `MINIMUM_WETH_LIQUIDITY` is a constant and not required to be emitted
  - [I-6] `TSwapPool::deposit` Don't follow CEI
  - [I-7] Magic numbers founded
  - [I-8] Functions not used internally could be marked external
  - [I-9] Missing `deadline` param in natspec in `TSwapPool::swapExactOutput`

Protocol Summary

This project is to enter a raffle to win a cute dog NFT. The protocol should do the following:

1. Call the `enterRaffle` function with the following parameters:
   1. `address[] participants`: A list of addresses that enter. You can use this to enter yourself multiple times, or yourself and a group of your friends.
2. Duplicate addresses are not allowed

3. Users are allowed to get a refund of their ticket & `value` if they call the `refund` function
4. Every X seconds, the raffle will be able to draw a winner and be minted a random puppy
5. The owner of the protocol will set a feeAddress to take a cut of the `value`, and the rest of the funds will be sent to the winner of the puppy. .

Disclaimer

The Luka 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
Likelihood	High	H	H/M	M
	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

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

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda
- In Scope:

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

It went good.

Spent 2 days on auditing this

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Gas	1
Info	9
Total	17

Findings

High

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

Description: The `getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens a user should deposit given an amount of tokens of output tokens. However the function currently miscalculates the resulting amount. When calculating fee, it scales the amount by 10000 instead of 1000.

Impact: Protocol takes more fees than expected from users.

Proof of Code: As a result, users swapping tokens via the swapExactOutput function will pay far more tokens than expected for their trades. This becomes particularly risky for users that provide infinite allowance to the TSwapPool contract. Moreover, note that the issue is worsened by the fact that the swapExactOutput function does not allow users to specify a maximum of input tokens, as is described in another issue in this report.

It's worth noting that the tokens paid by users are not lost, but rather can be swiftly taken by liquidity providers. Therefore, this contract could be used to trick users, have them swap their funds at unfavorable rates and finally rug pull all liquidity from the pool.

To test this, include the following code in the TSwapPool.t.sol file:

PoC

```
function testFlawedSwapExactOutput() public {
    uint256 initialLiquidity = 100e18;
    vm.startPrank(liquidityProvider);
    weth.approve(address(pool), initialLiquidity);
    poolToken.approve(address(pool), initialLiquidity);

    pool.deposit({
        wethToDeposit: initialLiquidity,
        minimumLiquidityTokensToMint: 0,
        maximumPoolTokensToDeposit: initialLiquidity,
        deadline: uint64(block.timestamp)
    });
    vm.stopPrank();

    // User has 11 pool tokens
    address someUser = makeAddr("someUser");
    uint256 userInitialPoolTokenBalance = 11e18;
    poolToken.mint(someUser, userInitialPoolTokenBalance);
    vm.startPrank(someUser);

    // Users buys 1 WETH from the pool, paying with pool tokens
    poolToken.approve(address(pool), type(uint256).max);
    pool.swapExactOutput(
        poolToken,
        weth,
        1 ether,
        uint64(block.timestamp)
    );

    // Initial liquidity was 1:1, so user should have paid ~1 pool token
    // However, it spent much more than that. The user started with 11 tokens, and now only has less than 1.
    assertLt(poolToken.balanceOf(someUser), 1 ether);
    vm.stopPrank();

    // The liquidity provider can rug all funds from the pool now,
    // including those deposited by user.
    vm.startPrank(liquidityProvider);
    pool.withdraw(
        pool.balanceOf(liquidityProvider),
        1, // minWethToWithdraw
        1, // minPoolTokensToWithdraw
        uint64(block.timestamp)
    );

    assertEq(weth.balanceOf(address(pool)), 0);
    assertEq(poolToken.balanceOf(address(pool)), 0);
}
```

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::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 transaction process, the user could get a much worse swap.

Proof of Concept: 1. The price of 1 WETH right now is 2782 USDC 2. User inputs a `swapExactOutput` looking for 1 WETH 1. inputToken = USDC 2. outputToken = WETH 3. outputAmount = 1 4. deadline = some deadline 3. The function does not offer a maxInput amount 4. The transaction is pending in the mempool, the market changes and the price moves HUGE -> 1 WETH is now 4564 USDC 5. The transaction completes, but the user sent the protocol 2x USDC instead of expected 2782

Recommended Mitigation: We should include a `maxInputAmount` so the user only has to spend up to a specified amount that they can predict.

```
    function swapExactOutput(
        IERC20 inputToken,
+        uint256 maxInputAmount,
.
.
.
        inputAmount = getInputAmountBasedOnOutput(outputAmount, inputReserves, outputReserves);

+        if (inputAmount > maxInputAmount) {
+            revert();
+        }

        _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 `sellPoolTokens` function is intended to allow users to easily sell pool tokens and receive WETH in exchange. Users indicate how many pool tokens they are willing to sell in the `poolTokenAmount` parameter. However, the function currently miscalculates the swapped aount of input tokens, not output.

Impact: Users will swap the wrong amount of tokens, which is a severe disruption of protocol 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 (`minWethToReceive` to be passed to `swapExactInput`)

```
-   function sellPoolTokens(uint256 poolTokenAmount) external returns (uint256 wethAmount) {
+   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 is wise to add a deadline to the function, as there is currently no deadline.

[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`: The balance of the pool token - `y`: The balance of WETH - `k`: The constant product of the two balances

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

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.

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. The protocol 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 until all the protocol funds are drained

PoC

Place the following in `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);
        // Approve tokens so they can be pulled by the pool during the swap
        poolToken.approve(address(pool), type(uint256).max);
        poolToken.mint(user, 100e18);

        // Execute swap, giving pool tokens, receiving WETH
        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);
        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. 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++;
-        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 transaction to complete even after the 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, in market conditions where the deposit rate is unfavourable.

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

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 and `wethToDeposit` value should go second.

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

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

    }
```

Gas

[G-1] `TSwapPool::deposit` don't need this line

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

Informational

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

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

[I-2] Lacking zero address checks

- Found in src/PoolFactory.sol [Line: 42]

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

- Found in src/TSwapPool.sol [Line: 76]

```
  constructor(
        address poolToken,
        address wethToken,
        string memory liquidityTokenName,
        string memory liquidityTokenSymbol
    )
        ERC20(liquidityTokenName, liquidityTokenSymbol)
    {
        // audit-info this needs zero address check
        i_wethToken = IERC20(wethToken);
        i_poolToken = IERC20(poolToken);
    }
```

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

```
-        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/PoolFactory.sol Line: 35

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

- Found in src/TSwapPool.sol Line: 52

  ```
      event LiquidityAdded(
  ```

- Found in src/TSwapPool.sol Line: 57

  ```
      event LiquidityRemoved(
  ```

- Found in src/TSwapPool.sol Line: 62

  ```
      event Swap(
  ```

[I-5] `TSwapPool::deposit` `MINIMUM_WETH_LIQUIDITY` is a constant and not required to be emitted

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

[I-6] `TSwapPool::deposit` Don't follow CEI

```
        } else {
            // This will be the "initial" funding of the protocol. We are starting from blank here!
            // We just have them send the tokens in, and we mint liquidity tokens based on the weth
+            liquidityTokensToMint = wethToDeposit;
            _addLiquidityMintAndTransfer(wethToDeposit, maximumPoolTokensToDeposit, wethToDeposit);
-            liquidityTokensToMint = wethToDeposit;
        }
```

[I-7] Magic numbers founded

Description: We don't recommend using numbers in your code, constant should be defined and used instead of literals.

- Found in src/TSwapPool.sol [Line: 275]

```
        uint256 inputAmountMinusFee = inputAmount * 997;
```

- Found in src/TSwapPool.sol [Line: 277]

```
        uint256 denominator = (inputReserves * 1000) + inputAmountMinusFee;
```

- Found in src/TSwapPool.sol [Line: 293]

```
            ((inputReserves * outputAmount) * 10000) /
```

- Found in src/TSwapPool.sol [Line: 294]

```
            ((outputReserves - outputAmount) * 997);
```

- Found in src/TSwapPool.sol [Line: 402]

```
            outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000);
```

- Found in src/TSwapPool.sol [Line: 454]

```
                1e18,
```

- Found in src/TSwapPool.sol [Line: 463]

```
                1e18,
```

[I-8] Functions not used internally could be marked external

- Found in src/TSwapPool.sol

```
      function swapExactInput(
    IERC20 inputToken,
    uint256 inputAmount,
    IERC20 outputToken,
    uint256 minOutputAmount,
    uint64 deadline
)
    public
```

```
    function totalLiquidityTokenSupply() public view returns (uint256) {
```

[I-9] Missing `deadline` param in natspec in `TSwapPool::swapExactOutput`

```
    /*
     * @notice figures out how much you need to input based on how much
     * output you want to receive.
     *
     * Example: You say "I want 10 output WETH, and my input is DAI"
     * The function will figure out how much DAI you need to input to get 10 WETH
     * And then execute the swap
     * @param inputToken ERC20 token to pull from caller
     * @param outputToken ERC20 token to send to caller
     * @param outputAmount The exact amount of tokens to send to caller
     */
    function swapExactOutput(
        IERC20 inputToken,
        IERC20 outputToken,
        uint256 outputAmount,
        uint64 deadline
    )
```