TSwap Audit Report

Version 1.0

Audit

January 17, 2024

TSwap Audit report

Badal Sharma

january 17, 2024

Prepared by: Badal Sharma Lead Auditors: - xxxxxxx

Table of Contents

- Medium
  * **[M-1]** TSwapPool::deposit is missing deadline check causing transactions to complete even after the deadline
- Low
  * **[L-1]** TSwapPool::LiquidityAdded event has parameters out of order
  * **[L-2]** IDefault value returned by TSwapPool::swapExactInput results in incorrect return value given
- Gas
  * **[G-1]** Unused local variable in `TSwapPool::deposit`
- Informational
  * **[I-1]** Missing check for `address(0)` when assining the values to address state variables.
  * **[I-2]** In `PoolFActory.sol::createPool` should be used `.symbol()` insted of `.name()`
  * **[I-3]** Even is missing, indexed field
  * **[I-4]** In `TSwapPool::deposit` does not follow CEI, It is not best practice.
  * **[I-5]** Use of `magic number` is discouraged.
  * **[I-6]** Should be provide netspec
  * **[I-7]** It Should be `external` insted of **`public`**
  * **[I-8]** Poor test coverage

Protocol Summary

The protocol starts as simply a `PoolFactory` contract. This contract is used to create new "pools" of tokens. It helps make sure every pool token uses the correct logic. But all the magic is in each `TSwapPool` contract.

You can think of each `TSwapPool` contract as it's own exchange between exactly 2 assets. Any ERC20 and the WETH token. These pools allow users to permissionlessly swap between an ERC20 that has a pool and WETH. Once enough pools are created, users can easily "hop" between supported ERC20s.

For example: 1. User A has 10 USDC 2. They want to use it to buy DAI 3. They `swap` their 10 USDC -> WETH in the USDC/WETH pool 4. Then they `swap` their WETH -> DAI in the DAI/WETH pool

Every pool is a pair of `TOKEN X` & `WETH`.

There are 2 functions users can call to swap tokens in the pool. - `swapExactInput` - `swapExactOutput`

We will talk about what those do in a little.

Disclaimer

The Badal Sharma 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 code is audit by Badal Sharma…..

Issues found

Severty	No of issue found
High	4
Medium	1
Low	2
Info	8
Gas	1
Total	16

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocll 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 tokens 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: To test this, include the following code in the `TSwapPool.t.sol` file:

POC

```
function testFlawedSwapExactOutput() 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);
        uint256 expected = 9e18;
        poolToken.approve(address(pool), 10e18);
        pool.swapExactInput(poolToken, 10e18, weth, expected, uint64(block.timestamp));
        vm.stopPrank();

        vm.startPrank(liquidityProvider);
        pool.approve(address(pool), 100e18);
        pool.withdraw(100e18, 90e18, 100e18, uint64(block.timestamp));
        assertEq(pool.totalSupply(), 0);
        assert(weth.balanceOf(liquidityProvider) + poolToken.balanceOf(liquidityProvider) > 400e18);
    }

    function testFlawedSwapExactOutput() public {
        uint256 initialLiquidity = 100e18;
        vm.startPrank(liquidityProvider);

        console.log("liquidityProvider weth balance before deposit:", weth.balanceOf(address(liquidityProvider)));
        console.log(
            "liquidityProvider poolToken balance after deposit:", poolToken.balanceOf(address(liquidityProvider))
        );

        weth.approve(address(pool), initialLiquidity);
        poolToken.approve(address(pool), initialLiquidity);
        pool.deposit({
            wethToDeposit: initialLiquidity,
            minimumLiquidityTokensToMint: 0,
            maximumPoolTokensToDeposit: initialLiquidity,
            deadline: uint64(block.timestamp)
        });
        console.log("liquidityProvider weth balance after deposit:", weth.balanceOf(address(liquidityProvider)));
        console.log("liquidityProvider poolToken balance after deposit:", weth.balanceOf(address(liquidityProvider)));

        vm.stopPrank();

        // User has 11 pool tokens
        address someUser = makeAddr("someUser");
        uint256 userInitialPoolTokenBalance = 11e18;
        poolToken.mint(someUser, userInitialPoolTokenBalance);
        console.log("someUser poolToken balance before withdrew:",
            poolToken.balanceOf(address(someUser)));
        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();

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

        console.log("liquidityProvider weth balance after withdrew:",
            weth.balanceOf(address(liquidityProvider)));
        console.log(
            "liquidityProvider poolToken balance after withdrew:",
                poolToken.balanceOf(address(liquidityProvider))
        );

        assertEq(weth.balanceOf(address(pool)), 0);
        assertEq(poolToken.balanceOf(address(pool)), 0);
        console.log("someUser poolToken balance after withdrew:",
            poolToken.balanceOf(address(someUser)));
}
```

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-2] `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 miscalculaes 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.

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-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

(i) inputToken = USDC
(ii) outputToken = WETH
(iii) outputAmount = 1
(iv) deadline = whatever

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

POC

```

    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);
        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: 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] 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.

POC

```
function testFundDrained() 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);
        console.log("Starting balance of weth:", weth.balanceOf(address
            (pool)));
        poolToken.approve(address(pool), type(uint256).max);
        poolToken.mint(user, 100e18);
        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));

        uint256 startingY = weth.balanceOf(address(pool));
        console.log("Balance of weth after 9 swap:", weth.balanceOf(
            address(pool)));

        pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
            timestamp));
        console.log("Balance of weth after 10 swap:", weth.balanceOf(
            address(pool)));
        vm.stopPrank();

        uint256 endingY = weth.balanceOf(address(pool));
        uint256 total = startingY - endingY;
        console.log("Extra token given to user:", total);
    }
```

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);
-         }
```

Medium

[M-1] TSwapPool::deposit is missing deadline check causing transactions 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, operationrs that add liquidity to the pool might be executed at unexpected times, in 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: 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)
    {
```

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:

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

[L-2] IDefault 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, In `TSawpPool::swapExactInput` the `output` variable not update so that's why the `output` return always 0, and give the caller wrong information.

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

Recommended Mitigation: Add this line in `TSawpPool::swapExactInput` function:

```
returns (
+            uint256 outputAmount
-            uint256 output
        )
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

        uint256 outputAmount = getOutputAmountBasedOnInput(inputAmount,
            inputReserves, outputReserves);

        if (outputAmount < minOutputAmount) {
            revert TSwapPool__OutputTooLow(outputAmount,
               minOutputAmount);
        }
        _swap(inputToken, inputAmount, outputToken, outputAmount);
+       return outputAmount;
```

Gas

[G-1] Unused local variable in `TSwapPool::deposit`

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

Informational

[I-1] Missing check for `address(0)` when assining the values to address state variables.

Assining values to address state variables without checking `address(0)`.

1. `PoolFActory.sol`

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

1. `TswapPool.sol`

```
 constructor(
         address poolToken,
         address wethToken,
         string memory liquidityTokenName,
         string memory liquidityTokenSymbol
     )
         ERC20(liquidityTokenName, liquidityTokenSymbol)
     {
+        if(poolToken == address(0)) && (wethToken == address(0)) {
+        revert error_addressIsZero();
+       }
         i_wethToken = IERC20(wethToken);
         i_poolToken = IERC20(poolToken);
     }
```

[I-2] In `PoolFActory.sol::createPool` should be used `.symbol()` insted of `.name()`

```
function createPool(address tokenAddress) external returns (address) {
        if (s_pools[tokenAddress] != address(0)) {
            revert PoolFactory__PoolAlreadyExists(tokenAddress);
        }
        string memory liquidityTokenName = string.concat("T-Swap ",
           IERC20(tokenAddress).name());

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

        TSwapPool tPool = new TSwapPool(tokenAddress, i_wethToken,
           liquidityTokenName,
         liquidityTokenSymbol);
        s_pools[tokenAddress] = address(tPool);
        s_tokens[address(tPool)] = tokenAddress;
        emit PoolCreated(tokenAddress, address(tPool));
        return address(tPool);
    }
```

[I-3] Even is missing, indexed field

- 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.

1. Found in src/TSwapPool.sol: Line: 44
2. Found in src/PoolFactory.sol: Line: 37
3. Found in src/TSwapPool.sol: Line: 46
4. Found in src/TSwapPool.sol: Line: 43

[I-4] In `TSwapPool::deposit` does not follow CEI, It is not best practice.

- It's the best to keep the function ckean and follow CEI, (checks, effects, interactions).
- It should be better if it is used before the "_addLiquidityMintAndTransfer" function

```
+           liquidityTokensToMint = wethToDeposit;
            liquidityTokensToMint = (wethToDeposit *
               totalLiquidityTokenSupply()) / wethReserves;
            if (liquidityTokensToMint < minimumLiquidityTokensToMint) {
                revert TSwapPool__MinLiquidityTokensToMintTooLow(
                   minimumLiquidityTokensToMint, liquidityTokensToMint)
                   ;
            }
            _addLiquidityMintAndTransfer(wethToDeposit,
               poolTokensToDeposit, liquidityTokensToMint);
        } else {
            _addLiquidityMintAndTransfer(wethToDeposit,
               maximumPoolTokensToDeposit, wethToDeposit);
-           liquidityTokensToMint = wethToDeposit;
        }
```

[I-5] Use of `magic number` is discouraged.

It can be confusing to see literals in a codebase, and it's much more readable if the number are given a name.

Examples:

1. In `TSwapPool::getOutputAmountBasedOnInput`

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

Insted, you could use:

```
   uint256 private constant FIRST_AMOUNT = 997;
   uint256 private constant SECOND_AMOUNT = 1000;

   uint256 inputAmountMinusFee = inputAmount * FIRST_AMOUNT;
```

```
    uint256 numerator = inputAmountMinusFee * outputReserves;
    uint256 denominator = (inputReserves * SECOND_AMOUNT) +
        inputAmountMinusFee;
    return numerator / denominator;
```

2. In `TSwapPool::getInputAmountBasedOnOutput`

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

Insted, you could use:

```
     uint256 private constant FIRST_AMOUNT = 997;
     uint256 private constant SECOND_AMOUNT = 1000;

     return ((inputReserves * outputAmount) * SECOND_AMOUNT) / ((
         outputReserves - outputAmount)*
     FIRST_AMOUNT);
```

[I-6] Should be provide netspec

- In `TSwapPool::swapExactInput`

[I-7] It Should be `external` insted of `public`

- In `TSwapPool::swapExactInput`
- In `TSwapPool::totalLiquidityTokenSupply`

[I-8] Poor test coverage

Running tests… | File | % Lines | % Statements | % Branches | % Funcs | | ————— | ———— | ———— | ———- | ———- | | src/PoolFactory.sol | 100.00% (11/11) | 100.00% (16/16) | 100.00% (2/2) | 100.00% (3/3) | | src/TSwapPool.sol | 54.84% (34/62) | 59.14% (55/93) | 33.33% (6/18) | 37.50% (6/16) |

Recommended Mitigation: Aim to get test coverage up to over 90% for all files.