TSwap Audit Report

Version 1.0

Nihavent

January 28, 2024

TSwap Audit Report

Nihavent

Jan 22, 2024

Prepared by: [Nihavent] Lead Auditors: - xxxxxxx

Table of Contents

    * [H-5] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` breaks the protocol invarian of `x * y = k`
  - Medium
    * [M-1] In `TSwapPool::deposit` is missing deadline check causing transactions to complete after the deadline
    * [M-2] Rebase, fee-on-transfer, and ERC777 tokens break protocol invariant
  - Low
    * [L-1] The `TSwapPool::LiquidityAdded` event parameters are given out of order in `TSwapPool::deposit` causing event to emit incorrect information
    * [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
  - Informationals
    * [I-1] The `PoolFactory::PoolFactory__PoolDoesNotExist` error is defined but not used, it should be removed
    * [I-2] Lacking zero address checks
    * [I-3] `PoolFactory::createPool` should use `.symbol()` insteald of `.name()`
    * [I-4]: Event is missing `indexed` fields
    * [I-5] The `TSwapPool::deposit::poolTokenReserves` variable is unused, consider removing for gas benefit
    * [I-6] Use of "magic numbers" leads to unclear code
    * [I-7] Natspec missing for param `deadline` in `TSwapPool::swapExactOutput`
    * [I-8] `TSwapPool::swapExactInput` function is not called internally so it should be marked external

Protocol Summary

Disclaimer

The YOUR_NAME_HERE 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

The findings in this document correspond to the follwoing Commit Hash:

```
xxx
```

Scope

- Commit Hash: e643a8d4c2c802490976b538dd009b351b1c8dda
- In 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

Issues found

Severity	Number of issues found
High	5
Medium	1
Low	2
Info	8
Total	16

Findings

High

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

Description The `getInputBasedOnOutput` function is intended to calculate the amount of tokens a user should deposit (input) based on an amount of expected output tokens. The function is also takes a small fee (1- (997/1000)) ~ 0.03% to increase the value of the pool over time. During the fee calculation the "1,000" value is "10,000", causing the fees.

Impact Protocol takes more fees than expected from users.

Proof of Concept

Run the following test in the `TSwapPool.t.sol` contract:

```

    function testIncorrectFeesCalculatedIn_getInputAmountBasedOnOutput
        () public {
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
        vm.stopPrank();

        console2.log("pool.getPriceOfOneWethInPoolTokens(): ", pool.
            getPriceOfOneWethInPoolTokens());
        //0.987158034397061298, ie. the user should pay ~0.9871
            poolTokens for 1 WETH
        console2.log("pool.getPriceOfOnePoolTokenInWeth(): ", pool.
            getPriceOfOnePoolTokenInWeth());


        uint256 startingPoolWethBalance = weth.balanceOf(address(pool))
            ;
        uint256 startingPoolPoolTokenBalance = poolToken.balanceOf(
            address(pool));
        uint256 startingUserWethBalance = weth.balanceOf(address(user))
            ;
        uint256 startingUserPoolTokenBalance = poolToken.balanceOf(
            address(user));

        console2.log("Starting weth pool balance: ",
            startingPoolWethBalance);
        console2.log("Starting poolToken pool balance: ",
            startingPoolPoolTokenBalance);
        console2.log("Starting weth user balance: ",
            startingUserWethBalance);
        console2.log("Starting poolToken user balance: ",
            startingUserPoolTokenBalance);


        //Example: User says "I want 10 output WETH, and my input is
            poolToken"

        vm.startPrank(user);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);

        uint256 expectedOutput = 5e17;

        // After we swap, there will be ~110 tokenA, and ~91 WETH
        // 100 * 100 = 10,000
        // 110 * ~91 = 10,000
        //uint256 expected = 9e18;

        pool.swapExactOutput(poolToken, weth, expectedOutput, uint64(
            block.timestamp));

        //Check balances
        uint256 endingPoolWethBalance = weth.balanceOf(address(pool));
        uint256 endingPoolPoolTokenBalance = poolToken.balanceOf(
            address(pool));
        uint256 endingUserWethBalance = weth.balanceOf(address(user));
        uint256 endingUserPoolTokenBalance = poolToken.balanceOf(
            address(user));

        console2.log("Ending weth pool balance: ",
            endingPoolWethBalance);
        console2.log("Ending poolToken pool balance: ",
            endingPoolPoolTokenBalance);
        console2.log("Ending weth user balance: ",
            endingUserWethBalance);
        console2.log("Ending poolToken user balance: ",
            endingUserPoolTokenBalance);

        //Check deltas as a result of swap
        int256 deltaPoolWethBalance = int256(endingPoolWethBalance) -
            int256(startingPoolWethBalance);
        int256 deltaPoolPoolTokenBalance = int256(
            endingPoolPoolTokenBalance) - int256(
            startingPoolPoolTokenBalance);
        int256 deltaUserWethBalance = int256(endingUserWethBalance) -
            int256(startingUserWethBalance);
        int256 deltaUserPoolTokenBalance = int256(
            endingUserPoolTokenBalance) - int256(
            startingUserPoolTokenBalance);

        console2.log("deltaPoolWethBalance: ", deltaPoolWethBalance);
        console2.log("deltaPoolPoolTokenBalance: ",
            deltaPoolPoolTokenBalance);
        console2.log("deltaUserWethBalance: ", deltaUserWethBalance);
        console2.log("deltaUserPoolTokenBalance: ",
            deltaUserPoolTokenBalance);


        //User has effectively swapped 5.04 poolTokens for 0.5 WETH at
            a ratio of ~10.08, ie. the user ended up paying ~10.08
            poolTokens per WETH
        //-.500000000000000000
        // .500000000000000000
        // 5.040246367242430810
        //-5.040246367242430810

        // If we correct the magic numbers in fees... User swaps 0.5025
             poolTokens for 0.5 WETH at a ratio of 1.005
        // -.500000000000000000
        //  .500000000000000000
        //  .502512562814070351
        // -.502512562814070351
    }
```

Recommended Mitigation

```diff
        return
-           ((inputReserves * outputAmount) * 10000) /
+           ((inputReserves * outputAmount) * 1000) /
            ((outputReserves - outputAmount) * 997);
```

[H-2] No slippage protection in `TSwapPool::swapExactOutput` may result in significant variance between expected swap and actual swap.

Description The `TSwapPool::swapExactOutput` function does not include any sort of slippage protection. This function is similar to `TSwapPool::swapExactInput` where the function specifies `minOutputAmount` and performs a check that the actual amount of tokens about to be swapped exceeds the minimum the user expects. The `TSwapPool::swapExactOutput` function should specificy and check for a `maxInputAmount`.

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 is 1,000 USDC 2. A user inputs a `swapExactOutput` looking to receive 1 WETH for their USDC:" 1. inputToken: USDC 2. outputToken: WETH 3. outputAnount: 1 4. deadline: whenever 3. The function does not offer a maxInput amount 4. As the transaction is pending in the mempool, the market changes. The price moves and now 1 WETH is worth 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 - todo

Recommended Mitigation We should include a `maxInputAmount` paramater so the user knows the maximum amount of tokens they might spend.

```
    function swapExactOutput(
        IERC20 inputToken,
+       uint256 maxInputAmount
        IERC20 outputToken,
        uint256 outputAmount,
        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(); // optional: add custom error>
+           }

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

[H-3] `TSwapPool::sellPoolTokens` calls `TSwapPool::swapExactOutput` when it should call `TSwapPool::swapExactInput`. This causes the user to make an unintended swap.

Description The `TSwapPool::sellPoolTokens` function allows users to sell a given amount of poolTokens. However the function misuses the input calling `TSwapPool::swapExactOutput` with the user's expect sell amount of poolTokens in the `outputAmount` argument. This calls a swap that the user did not intend.

Impact A swap is called with amounts which may vary significantly from the requested sell (dependant on price).

Proof of Concept

1. User wishes to sell 10 USDC (pool token)
2. User makes call:

```
    pool.sellPoolTokens(10);
```

3. This function then calls:

```
    swapExactOutput(i_poolToken,
                    i_wethToken,
                    10, //outputAmount argument
                    uint64(block.timestamp));
```

4. This function calls _swap:

```
    _swap(inputToken,   // i_poolToken
          inputAmount,  // Calculated amount of input tokens, not the
             10 USDC the user wanted to input
          outputToken,  // i_wethToken
          10);          // Weth user receives
```

If the price of 1 WETH is 1000 USDC, the user is required to send ~10,000 USDC instead of the 10 they requested to sell.

Recommended Mitigation

```
    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));
     }
```

[H-5] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` breaks the protocol invarian 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, and `k` is the constant product of the two balances.

This means whenever the balances change in the protocol, the ratio between the two amounts should remain coknstant, hence `k`. However, this is broken due to the extra incentive in the `_swap` function. Meaning that over time 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 The protocol's core invariant is broken.

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

Proof of Concept

```
    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);
        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 added incentive in the `TSwapPools::_swap` function.

```
-        if (swap_count >= SWAP_COUNT_MAX) {
-            swap_count = 0;
-            //@report-written magic numbers
-            outputToken.safeTransfer(msg.sender, 1
    _000_000_000_000_000_000);
-        }
```

Medium

[M-1] In TSwapPool::deposit is missing deadline check causing transactions to complete after the deadline

Description The `TSwapPool::deposit` function accepts a deadline parameter, which according to the documentation is "@param deadline The deadline for the transaction to be completed by". However, this parameter is never used. Consequently, operations that add liquidity to the pool may be executed at unexpected times.

(also MEV attacks)

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

Proof of Concept The `deadline` parameter is unused.

Recommended Mitigation Consider using the modifier `TSwapPool::revertIfDeadlinePassed` similar to other functions in the `TSwapPools` contract.

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

[M-2] Rebase, fee-on-transfer, and ERC777 tokens break protocol invariant

Todo

Low

[L-1] The `TSwapPool::LiquidityAdded` event parameters are given out of order in `TSwapPool::deposit` causing event to emit incorrect information

Description The `TSwapPool::LiquidityAdded` event is defined as follows:

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

The above event is called in `TSwapPool::deposit` as follows:

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

The second and third parameter are swapped in the call.

Impact The event emits the wrong information in two fields.

Proof of Concept See above

Recommended Mitigation Swap the parameters when calling `TSwapPool::LiquidityAdded`:

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

[L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given

Description The `TSwapPool::swapExactInput` function is expected to return the actual amount of tokens bought by the caller. However, while it declares the named value `output`, this variable is never assigned a value, nor is uses an explicity return statement.

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

Proof of Concept

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

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

    }
```

Informationals

[I-1] The `PoolFactory::PoolFactory__PoolDoesNotExist` error is defined but not used, it should be removed

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

[I-2] Lacking zero address checks

Recommended to implement these fixes as modifiers

In `PoolFactory::constructor`:

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

Issue also found in `PoolFactory::createPool`

[I-3] `PoolFactory::createPool` should use `.symbol()` insteald 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] The `TSwapPool::deposit::poolTokenReserves` variable is unused, consider removing for gas benefit

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

[I-6] Use of "magic numbers" leads to unclear code

Magic numbers can be replaced with cosntant variables with a name and natspec describing their purpose. This improves code readability and reduces the chance of errors. Some examples:

- `TSwapPool::getOutputAmountBasedOnInput`

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

- `TSwapPool::getInputAmountBasedOnOutput`

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

- TSwapPool::_swap

```

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

[I-7] Natspec missing for param `deadline` 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
    )
```

[I-8] `TSwapPool::swapExactInput` function is not called internally so it should be marked external

- Found in src/TSwapPool.sol Line: 300

```
        function swapExactInput(
```