Protocol Audit Report

Diasfusuy

February 28, 2025

Prepared by: Cyfrin Lead Auditors: - Yusuf Sahin

Table of Contents

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 Yusuf Sahin 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 Scope Details

- 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

Actors / 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.

Known Issues

- None

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Info	5
Total	12

Findings

High

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

Description: The `getInputAmountBasedOnOutput` function is intended to calculate the amount of tokens user should deposit given an amount of token 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.

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 recieve way fewer tokens

Description: The `swapExactOutput` function does not include any sort of slippage protection. This function is similar wo what is done in `TSwapPool::swapExactOutput`, 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 1,000 USDC 2. User inputs a `swapExactOutput` looking for 1 WETH 1. inputToken = USDC 2. outputToken = WETH 3. outputAmount = 1 4. 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 isntead of 1,000 USDC

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,
+       uin256 maxInputAmount,
.
.
.
        inputAmount = getInputAmountBasedOnOutput(
            outputAmount,
            inputReserves,
            outputReserves
        );
+       if(inputAmount > maxInputAmount) {
+           revert();
+       }

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

[H-3] `TSwapPool::seelPoolTokens` mismatch input and output tokens causing users to recieve the incorrect amount of tokens

Description: The `sellPoolTokens` function is intended to allow users to easily sell pool tokens and recieve WETH in exchange. Users indicate how many pool tokens whey're willing to sell in the `poolTokenAmount` parameeter. However, the function currently miscalculates 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 wrong amount of tokens, which is a severe disruption of protocol functionality.

Proof of Concept: 1. Seed pool with equal reserves: `100e18` WETH and `100e18` pool tokens. 2. Have a user call `sellPoolTokens(1e18)`. 3. Expected behavior: user sells exactly `1e18` pool tokens and receives corresponding WETH output. 4. Actual behavior: function routes through `swapExactOutput(..., outputAmount = 1e18)`, so user receives exactly `1e18` WETH while the protocol pulls the computed input amount of pool tokens (greater than `1e18`).

Proof Of Code

Place the following into `TSwapPool.t.sol`

```
function testSellPoolTokensSwapsWrongAmount() 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 poolTokenAmountToSell = 1e18;
    poolToken.mint(user, 100e18);

    uint256 expectedPoolTokenInput = pool.getInputAmountBasedOnOutput(
        poolTokenAmountToSell,
        poolToken.balanceOf(address(pool)),
        weth.balanceOf(address(pool))
    );

    uint256 userPoolTokenBalanceBefore = poolToken.balanceOf(user);
    uint256 userWethBalanceBefore = weth.balanceOf(user);

    vm.startPrank(user);
    poolToken.approve(address(pool), type(uint256).max);
    uint256 returnedAmount = pool.sellPoolTokens(poolTokenAmountToSell);
    vm.stopPrank();
```

```
    uint256 userPoolTokenBalanceAfter = poolToken.balanceOf(user);
    uint256 userWethBalanceAfter = weth.balanceOf(user);
    uint256 actualPoolTokenSold = userPoolTokenBalanceBefore - userPoolTokenBalanceAfter;
    uint256 actualWethReceived = userWethBalanceAfter - userWethBalanceBefore;

    assertEq(actualWethReceived, poolTokenAmountToSell);
    assertEq(actualPoolTokenSold, expectedPoolTokenInput);
    assertGt(actualPoolTokenSold, poolTokenAmountToSell);
    assertEq(returnedAmount, actualPoolTokenSold);
}
```

Recommended Mitigation:

Consider changin implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would also require changing the `sellPoolTokens` function to accept a new paramete (ie `minWethToRecieve` to be passed to `swapExactInput`)

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

Additionally, it might be wse to add deadline to the function, as there is currently no deadline.

[H-4] `TSwapPool::_swap` the extra tokens given to users after every `swapCount` breas 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 measn, 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 follwoing bock 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 wsap until all the protocol funds are drained

Proof Of Code

Place the following into `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);
        poolToken.approve(address(pool), type(uint256).max);
        poolToken.mint(user, 10e18);
        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();

        int256 startingY= int256(weth.balanceOf(address(pool)));
        int256 expectedDeltaY = int256(-1) * int256(outputWeth);

        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 invaariant. Or, we shuld 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 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 unfavorable.

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

Proof of Concept: The `deadline` parameter is unused.

Recommended Mitigation: Consider making 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::LiquiditAdded` event has parameters are out of order

Description: When the `LiquiditAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTransfer` fnction. it logs value 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 malfunnctioning.

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 wull always be 0, giving incorrect information to the caaller.

Proof of Concept:

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);
}
```

Informationals

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

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

[I-2] `PoolFactory::constructor` Lacking zero address check

```
    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());
```

[I-4] `TSwapPool::Swap` should index key parameters for better log filtering

```
    event Swap(
        address indexed swapper,
-       IERC20 tokenIn,
+       IERC20 indexed tokenIn,
        uint256 amountTokenIn,
-       IERC20 tokenOut,
+       IERC20 indexed tokenOut,
        uint256 amountTokenOut
    );
```

[I-5] `TSwapPool::constructor` Lacking zero address check

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

```
+       }
        i_wethToken = IERC20(wethToken);
        i_poolToken = IERC20(poolToken);
    }
```