Protocol Audit Report

Version 1.0

Cyfrin.io

February 27, 2025

Protocol Audit Report

Firefly

March 1, 2025

Prepared by: Cyfrin Lead Auditors: - Firefly

Table of Contents

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The Firefly 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

	Impact			
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

Roles

Executive Summary

Issues found

Severity	Number of issues found
High	4
Medium	2
Low	2
Info	9
Gas Optimizations	0
Total	17

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol 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 10000 instead of 1000.

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 receive less tokens than expected

Description: The `swapExactOutput` function does not include any sort of slippage protection. This function is similar to what is done is `TSwapPool::swapExactInput`,where the function specifies a `minOutputAmount`, and `swapExactOutput` function should specify 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 right now is 1000 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 10000 USDC. 10x more than the user expected. 5. The transaction completes, but the user sent the protocol 10000 USDC instead of expected 1000 USDC. 6. **Recommended Mitigation:** We should include a `maxInputAmount` to the user only has to spend up to a specified 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-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're willing to sell in the `poolTokenAmount` parameter. However, the function currently miscalculate 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 sawp 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 (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. MEV attack.

[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` is the amount of pool tokens in the pool - `y` is the amount of WETH in the pool - `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 from the pool.

The follow block of code is responsible for the broken.

```
        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 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, 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)
        );
```

```
55          pool.swapExactOutput(
56              poolToken,
57              weth,
58              outputWeth,
59              uint64(block.timestamp)
60          );
61          pool.swapExactOutput(
62              poolToken,
63              weth,
64              outputWeth,
65              uint64(block.timestamp)
66          );
67
68          int256 startingY = int256(weth.balanceOf(address(pool)));
69          int256 expectedDeltaY = int256(-1) * int256(outputWeth);
70
71          pool.swapExactOutput(
72              poolToken,
73              weth,
74              outputWeth,
75              uint64(block.timestamp)
76          );
77          vm.stopPrank();
78
79          uint256 endingY = weth.balanceOf(address(pool));
80          int256 actualDeltaY = int256(endingY) - int256(startingY);
81          assertEq(actualDeltaY, expectedDeltaY);
82      }
```

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.

```
1 -         swap_count++;
2 -          if (swap_count >= SWAP_COUNT_MAX) {
3 -              swap_count = 0;
4 -              outputToken.safeTransfer(msg.sender, 1
      _000_000_000_000_000_000);
5 -          }
```

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

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

Low

[L-1] `TSwapPool::LiquidityAddedevent` has paramters out of order

Description: When the `LiquidityAdded` event is emitted in the `TSwapPool::_addLiquidityMintAndTrans` function, it logs values in an incorrect order. The `poolTokensToDeposit` value should go in the third paramter position, where the `wethToDeposit` value should go in the second parameter position.

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 by the caller. However, while it declares the named return value `output` it is never assigned a value, nor uses an explict return statement.

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

Recommended Mitigation:

```
        returns (
-           uint256 output
+           uint256 outputAmount
        )
```

Informationals

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

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

[I-2] lacks a zero address check

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

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

[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());
```