Protocol Audit Report

0xGh0stc3

August 2, 2024

TSwap Protocol Audit Report

Version 1.0

0xgh0stc3

August 2, 2024

Protocol Audit Report

0xGh0stc3

August 2, 2024

Prepared by: [0xGh0stc3] Lead Auditors: 0xGh0stcybers3c - xxxxxxx

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
- High
- Medium
- Low
- Informational
- Gas

Protocol Summary

Protocol does X, Y, Z

Disclaimer

The 0xGh0stcyubers3c 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
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

Scope

Roles

Executive Summary

Issues found

Category	No. of Issues
Critical	0
High	4
Medium	1
Low	2
Infomational	4
Totals	10

Findings

High

[H-1] Incorrect Fee Calculation in `TSwapPool::getInputAmountBasedOnOutput`causing the protocol to take too many tokens from users, resulting in loss of fee.

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

Impact: The protocol takes more fees than expected from users.

Proof of Code:

```
    function testDepositSwap() 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);
        poolToken.approve(address(pool), 10e18);
        // After we swap, there will be ~110 tokenA, and ~91 WETH
        // 100 * 100 = 10,000
        // 110 * ~91 = 10,000
        uint256 expected = 9e18;

        pool.swapExactInput(poolToken, 10e18, weth, expected, uint64(block.timestamp));
        assert(weth.balanceOf(user) >= expected);
    }
```

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] No 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 funnction is similar to to what is done in

`TswapPool::swapExactInput` wher the function specifies `minOutputAmount`, the `swapExactOutput` function should specify a `maxInputAmount`

Impact: If the market condition changed before the transaction process, the user would get a much worse swap.

Proof of Code: 1. The price of Weth right now is 1000USDC 2. The 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 function is pending in the mempool the market changes and the price moves HUGE –> 1 weth is now 10000USDC, 10x more than the user expected. 5. The transaction completes but the user sent the protocol 10000USDSC instead of the expected 1000USDC.

Recommended Mitigation: We should include `maxInputAmount` so tht the user is able to predict how much they will spend on the protocol.

```
  function swapExactInput(
        IERC20 inputToken,
+        uint256 maxInputAmount,
.
.
.
    inputAmount = getInputAmountBasedOnOutput(
                outputAmount,
                inputReserves,
                outputReserves
            );
+            if(inputAmount > maxInoutAmount) {
+                revert();
+            }
```

[H-3] `TSwapPool::sellPoolTokens` mismatches input and output tokens causing the 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 amount. This is resulting from the fact that the `swapExactOutput` function is called, while `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 the protocol functionality.

Recommended Mitigation: Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note that this would 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)
-            );
+      external returns (uint256 wethAmount) {
+        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-4] The `TSwapPool::_swap` the extra tokens given to the 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 pool token, - `y`: The balance of WETH, - `k`: The constant product of the two balances.

This means, that whenever the balances changes in the protocol, the ratio between the 2 amounts remain constant, hence the `k`. However, this gets broken due to the extra incentives in `_swap` function. Meaning that over time the funds in the protocol gets drained.

This 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 funds in the protocol by doing so many swaps to collect the extra incentives given by the protocol.

Proof of Code: 1. A user swaps 10 times and collects the extra incentives of `1_000_000_000_000_000_000` tokens 2. User continues to swap until all the funds are drained.

Proof Of Code

place trhe following into the `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;

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

            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)
    );
    pool.swapExactOutput(
        poolToken,
        weth,
        outputWeth,
        uint64(block.timestamp)
    );
    pool.swapExactOutput(
        poolToken,
        weth,
        outputWeth,
        uint64(block.timestamp)
    );
    vm.stopPrank();

    uint256 endingY = weth.balanceOf(address(pool));
    int256 actualDeltaY = int256(endingY) - int(startingY);
    assertEq(actualDeltaY, expectedDeltaY);
```

```
        }
```

Recommended Mitigation: Remove the extra incentives. If you ant to keep this, we should account for the change in the x * y = k protocol invariant, or else 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 the transaction to complete even after 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 the market conditions where deposit rate is unfavorable.

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

Proof of Code: The `dedline` parameter is not used

Recommended Mitigation: Consider making these changes to the deposit function;

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

Low

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

Description: When the `LiquidityAdded` event is emmitted in the `TSwapPool::_addLiquidityMintAndTransfer` function it logs the value in an incorrect order. The `wethToDeposit` value should come second as the `poolTokensToDeposit` value comes in the third parameter position.

Impact: Event emission is incorrect, leading to off-chain functions potentially malfunctions.

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 te caller.

Informational

[I-1] `PoolFactory::PoolFactory__PoolDoesNotExistis` not used and should be Removed

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

[I-2] Lacking zero Address check

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

             i_wethToken = wethToken;
    }
```

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

```
-               string memory liquidityTokenSymbol = string.concat("ts", IERC20 (tokenAddres
+           string memory liquidityTokenSymbol = string.concat("ts", IERC20(tokenAddress).sy
```

[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.

4 Found Instances

- 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(
  ```