TSwap Protocol (UniswapV1 Fork)

xKeywordx

January 19, 2024

TSwap (Uniswap V1 Fork)

xKeywordx

January 19, 2024

TSwap Audit Report

Prepared by: xKeywordx

Table of contents

See table

About xKeywordx

I am a security researcher aiming to improve the Web3 space by making smart contracts more reliable and secure.

Audit Details

Scope

```
src/TSwapPool.sol
src/PoolFactory.sol
```

Protocol Summary

Uniswap V1 Fork

Executive Summary

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Info	3
Total	10

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` function 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 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) /
+           ((inputReserves * outputAmount) * 1000) /
            ((outputReserves - outputAmount) * 997);
    }
```

[H-2] 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 `minOutputAmount`, the `swapExactoutput` function should specicy `maxInputAmount`.

Impact: If market conditions change before the transaction processes, the user could get a much worse swap rate.

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 sees a significant movement to 1 WETH == 10,000 USDC, which is 10 times more than what the user expected.
5. The transaction completes, but the user sent the protocol 10,000 USDC for 1 WETH, instead of the expected 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,
+   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 miscalculates the swapped amount.

This is due to the fact that `swapExactOutput` function is called whereas the `swapExactInput` function is the one that should be called. Because users sepcify the exact amount of input tokens, not output.

Impact: Users will swap 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` istead 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,
+       uint 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-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 balance of pool token, Y is the balance of WETH, and K is 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. However, this is broken due to the incentive in the `_swap` function. Meaning that over time the protocol funds will be drained.

The following code produces 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.

Simply put, the protocol's main 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.

Code

Place the following into `TSwapPool.t.sol`

```
 function testDepositBroken() 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)));
        int256 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));
        vm.stopPrank();

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

Recommended Mitigation: Remove the extra incentive.

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, operations 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)
```

Low

[L-1] `TswapPool::LiquidityAdded` event has parameters out of order

Description: When the `LiquidityAdded` event is emitted it logs values in an incorrect order. The `poolTokensDeposit` parameter 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: Reorder the parameters

```
-   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 bougth 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 the caller.

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);
    }
```

Informational

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

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

[I-2] Lacking zero address checks

```
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());
```