TSwap Audit Report

0xVadar

June 9, 2024

Prepared by: Cyfrin Lead Auditors: - 0xVadar

Table of Contents

Protocol Summary

Protocol does Swaps between tokens, it is similar to uniswap

Disclaimer

The 0xVadar 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

Scope

Roles

Executive Summary

Issues found

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 10_000 instead of 1_000

Impact: Protocol takes more fees than expected from users

Recommended Mitigation:

```
    function (

    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
```

```
        returns(uint256 inputAmount)
    {
-         return ((inputReserves * outputAmount) * 10_000) / ((outputReserves - outputAmount)
+         return ((inputReserves * outputAmount) * 1_000) / ((outputReserves - outputAmount)
    }
```

[H-2] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to eceive potentially 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 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 is 1,000 USDC 2. User inputs a `swapExactOutput` looking for 1 WETH a. inputToken = USDC b. outputToken = WETH c. outputAmount = 1 d. 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 sends the protocol 10,000 USDC instead of the expected 1,000 USDC

Recommended Mitigation:

[H-4] `TSwapPool::sellPoolToken` mismatches input and output tokens causing users to receive the incorrect ammount 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 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 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 (i.e `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

[H-5] In `TSwapPool::_swap` the extra tokens given to users after every swap count breaks 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 means, that the balances change in 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 following block of code is responsible for the issue;

```
        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);
        }
```

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 just continues to swap until all the protocol funds are drained

Proof Of Code

Place the following into `TSwapPool.t.sol`

Recommended Mitigation: Remove the extra incentive. If you ant to keep this in, we should account for the change in the x * y = k protocol invariant. Or, we should set aside tokens in thesame way we do with fees # Medium

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

Description: the `deposit` function accepts a deadlone parameter, which according to the documentation is "The deadline for the transaction to be completed by". However, this parameter is not 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 unfavourable.

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 making the following change to the function

```
function deposit(
    uint256 wethDeposit,
    uint256 minimumLiquidityTokensToMint, // LP tokens -> if empty,
    we can pick 100% (100% == 17 tokens)
    uint256 maximumPoolTokensToDeposit,
    uint64 deadline
)
    external
+   revertIfDeadlinePassed(deadline)
    revertIfZero(wethToDeposit)
    returns (uint256 liquidityTokensToMint)
    {

    }
```

Low

[L-1] `TSwapPool::LiquidityAdded` event has parameters out of order causing event to emit incorrect information

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

Impact: Event emission is incorrect leading to offchain 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 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 the caller

Informational

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

Description:

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

[I-2] Lacking zero address checks

Description:

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

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

Description:

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

Gas