TSwap Protocol Audit Report

Version 1.0

Mangekyou.network

June 20, 2024

TSwap Protocol Audit Report

Mangekyou.network

June 20, 2024

Prepared by: Mangekyou

Lead Auditors: - zerefwth

Table of Contents

Protocol Summary

TSWAP is an constant-product AMM that allows users permissionlessly trade WETH and any other ERC20 token set during deployment. Users can trade without restrictions, just paying a tiny fee in each swapping operation. Fees are earned by liquidity providers, who can deposit and withdraw liquidity at any time.

Disclaimer

The Mangekyou 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

Severity	Number of issues found
High	4
Medium	2
Low	2
Info	8
Total	16

Findings

High

[H-1] Incorrect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to mtake 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 a 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.

Proof of Concepts

Recommended mitigation

```
1     function getInputAmountBasedOnOutput(
2         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) * 1_000) /
+           ((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 a `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 Concepts 1. The price of 1 WETH right now 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 maxInputAmount 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 then the user expected 5. The transaction completes, but the user sent the protocol 10,000 USDC 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,
```

```
10              outputReserves
11          );
12  +       if(inputAmount > maxInputAmount) {
13  +           revert();
14          }
15          _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 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 Concepts

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`)

```
1      function sellPoolTokens(
2          uint256 poolTokenAmount,
3  +       uint256 minWethToReceive,
4      ) external returns (uint256 wethAmount) {
5          return
6  -           swapExactOutput(i_poolToken, i_wethToken, poolTokenAmount,
       uint64(block.timestamp));
7  +           swapExactInput(i_poolToken, poolTokenAmount, i_wethToken,
       minWethToReceive, uint64(block.timestamp));
8      }
```

Additionally, it might be wise to add a deadline to the function, as there is currently no deadline. MEV later.

[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`: The balance of the pool token - `y`: The balance of WETH - `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.

The following block 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 simly put, the protocol's core invariant is broken.

Proof of Concepts 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));
```

```
16          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
17          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
18          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
19          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
20          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
21          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
22
23          int256 startingY = int256(weth.balanceOf(address(pool)));
24          int256 expectedDeltaY = int256(-1) * int256(outputWeth);
25
26          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
27          vm.stopPrank();
28
29          uint256 endingY = weth.balanceOf(address(pool));
30
31          int256 actualDeltaY = int256(endingY) - int256(startingY);
32          assertEq(actualDeltaY, expectedDeltaY);
33      }
```

Recommended mitigation Remove the extra incentive mechanism. 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 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 param 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 Concepts: 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 invariant

/// findings…

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 `TSwapPool::_addLiquidityMintAndTrans` function, it logs values in an incorrect order. The `poolTokensToDeposit` value should go in the third parameter posistion, whereas the `wethToDeposit` value should go second.

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

Proof of Concepts

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.

Recommended mitigation

```
    function swapExactInput(
        IERC20 inputToken, // e input token to swap / sell ie: DAI
        uint256 inputAmount, // e amount of input token to sell ie: DAI
        IERC20 outputToken, // e output token to buy / buy ie: WETH
        // e 7 DAI -> 1 WETH
        uint256 minOutputAmount, // e minimum output amount expected to receive
        uint64 deadline // e deadline for when the transaction should expire
    )
    //  -info this should be external
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (
            // @audit-low
            // IMPACT: SUPER LOW - protocol is giving the wrong return
            // LIKELIHOOD: HIGH - always the case
            uint256 output
        )
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

-       uint256 outputAmount = getOutputAmountBasedOnInput(
-           inputAmount,
-           inputReserves,
-           outputReserves
-       );
+       uint256 output = getOutputAmountBasedOnInput(
+           inputAmount,
+           inputReserves,
+           outputReserves
+       );


-       if (outputAmount < minOutputAmount) {
-           revert TSwapPool__OutputTooLow(outputAmount, minOutputAmount);
```

```
37 +        if (output < minOutputAmount) {
38 +            revert TSwapPool__OutputTooLow(outputAmount,
   minOutputAmount);
39          }
40
41 -        _swap(inputToken, inputAmount, outputToken, outputAmount);
42 +        _swap(inputToken, inputAmount, outputToken, output);
43      }
```

Informationals

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

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

[I-2] Lacking zero address checks

```
1      constructor(address wethToken) {
2 +        if(wethToken == address(0)) {
3 +            revert();
4 +        }
5          i_wethToken = wethToken;
6      }
```

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

```
1 -    string memory liquidityTokenSymbol = string.concat("ts", IERC20(
   tokenAddress).name());
2 +    string memory liquidityTokenSymbol = string.concat("ts", IERC20(
   tokenAddress).symbol());
```

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

```
1       event PoolCreated(address tokenAddress, address poolAddress);
```

- Found in src/TSwapPool.sol Line: 52

```
1       event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
1       event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
1       event Swap(
```