TSwap Protocol Audit Report

Version 1.0

Cyfrin.io

February 12, 2024

TSwap Protocol Audit Report

Shurjeel Khan

1, 2, 2024

Prepared by: Cyfrin Lead Auditors:

- Shurjeel Khan

Table of Contents

Protocol Summary

The protocol starts as simply a `PoolFactory` contract. This contract is used to create new "pools" of tokens. It helps make sure every pool token uses the correct logic. But all the magic is in each `TSwapPool` contract.

You can think of each `TSwapPool` contract as it's own exchange between exactly 2 assets. Any ERC20 and the WETH token. These pools allow users to permissionlessly swap between an ERC20 that has a pool and WETH. Once enough pools are created, users can easily "hop" between supported ERC20s.

For example:

1. User A has 10 USDC
2. They want to use it to buy DAI
3. They `swap` their 10 USDC -> WETH in the USDC/WETH pool
4. Then they `swap` their WETH -> DAI in the DAI/WETH pool

Every pool is a pair of `TOKEN X` & `WETH`.

There are 2 functions users can call to swap tokens in the pool.

- swapExactInput
- swapExactOutput

We will talk about what those do in a little.

Liquidity Providers

In order for the system to work, users have to provide liquidity, aka, "add tokens into the pool".

Why would I want to add tokens to the pool?

The TSwap protocol accrues fees from users who make swaps. Every swap has a `0.3` fee, represented in `getInputAmountBasedOnOutput` and `getOutputAmountBasedOnInput`. Each applies a `997` out of `1000` multiplier. That fee stays in the protocol.

When you deposit tokens into the protocol, you are rewarded with an LP token. You'll notice `TSwapPool` inherits the `ERC20` contract. This is because the `TSwapPool` gives out an ERC20 when Liquidity Providers (LP)s deposit tokens. This represents their share of the pool, how much they put in. When users swap funds, 0.03% of the swap stays in the pool, netting LPs a small profit.

LP Example

1. LP A adds 1,000 WETH & 1,000 USDC to the USDC/WETH pool

    i. They gain 1,000 LP tokens

2. LP B adds 500 WETH & 500 USDC to the USDC/WETH pool

    i. They gain 500 LP tokens

3. There are now 1,500 WETH & 1,500 USDC in the pool
4. User A swaps 100 USDC -> 100 WETH.

    i. The pool takes 0.3%, aka 0.3 USDC.
    ii. The pool balance is now 1,400.3 WETH & 1,600 USDC
    iii. aka: They send the pool 100 USDC, and the pool sends them 99.7 WETH

Note, in practice, the pool would have slightly different values than 1,400.3 WETH & 1,600 USDC due to the math below.

Disclaimer

The Shurjeel 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

The findings described in this document correspond the following commit hash:

```
e643a8d4c2c802490976b538dd009b351b1c8dda
```

Scope

```
src/
#-- TSwapPool.sol
#-- PoolFactory.sol
```

Roles

- Liquidity Providers: Users who have liquidity deposited into the pools. Their shares are represented by the LP ERC20 tokens. They gain a 0.3% fee every time a swap is made.
- Users: Users who want to swap tokens.

Executive Summary

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	2
Info	9
Gas Optimizations	0
Total	16

Findings

High

[H-1] `TSwapPool::getInputAmountBasedOnOutput` The calculation is incorrect causing user to charge to much

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 getInputAmountBasedOnOutput(
        uint256 outputAmount,
        uint256 inputReserves,
        uint256 outputReserves
    )
        public
        pure
        revertIfZero(outputAmount)
        revertIfZero(outputReserves)
        returns (uint256 inputAmount)
    {
-        return ((inputReserves * outputAmount) * 10_000) / ((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 Concept:

1. The price of 1 WETH right now is 1,000 USDC
2. User inputs a `swapExactOutput` looking for 1 WETH
    i. inputToken = USDC
    ii. outputToken = WETH
    iii. outputAmount = 1
    iv. 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 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, 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 miscalculaes 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 protcol functionality.

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));
     }
```

[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 follow 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 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 untill all the protocol funds are drained

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

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

        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 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.

```
-         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);
-         }
```

Medium

[M-1] `TSwapPool::deposit` is missing deadline checks causing transaction to be complete even after the deadline

Description: The `deposit` function accepts deadline parameter, which according to documentations "The deadline for the transaction to be completed by". However, this parameter is never used

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 change to the `deposit` function

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

Low

[L-1] `TSwapPool::_addLiquidityMintAndTransfer` event emits incorrect, is backward, Should be

```
-         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 `ouput` 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:

```
    {
        uint256 inputReserves = inputToken.balanceOf(address(this));
        uint256 outputReserves = outputToken.balanceOf(address(this));

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

-        if (output < 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, should removed

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

[I-2] `PoolFactory` lacking zero address check

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

```
    }
```

[I-3] `PoolFactory::createPool` should use `.symbol()` not `.name()`

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

- Found in src/PoolFactory.sol Line: 35

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

- Found in src/TSwapPool.sol Line: 43

```
    event LiquidityAdded(address indexed liquidityProvider, uint256 wethDeposited, uint256 poolTokensDeposited);
```

- Found in src/TSwapPool.sol Line: 44

```
    event LiquidityRemoved(address indexed liquidityProvider, uint256 wethWithdrawn, uint256 poolTokensWithdrawn);
```

- Found in src/TSwapPool.sol Line: 45

```
    event Swap(address indexed swapper, IERC20 tokenIn, uint256 amountTokenIn, IERC20 tokenOut, uint256 amountTokenOut);
```

[I-5] Constants should be defined and used instead of literals

- Found in src/TSwapPool.sol Line: 229

```
        uint256 inputAmountMinusFee = inputAmount * 997;
```

- Found in src/TSwapPool.sol Line: 231

```
        uint256 denominator = (inputReserves * 1000) +
            inputAmountMinusFee;
```

- Found in src/TSwapPool.sol Line: 246

```
        return ((inputReserves * outputAmount) * 10000) / ((
            outputReserves - outputAmount) * 997);
```

- Found in src/TSwapPool.sol Line: 332

```
            outputToken.safeTransfer(msg.sender, 1
                _000_000_000_000_000_000);
```

- Found in src/TSwapPool.sol Line: 375

```
            1e18, i_wethToken.balanceOf(address(this)),
                i_poolToken.balanceOf(address(this))
```

- Found in src/TSwapPool.sol Line: 381

```
            1e18, i_poolToken.balanceOf(address(this)),
                i_wethToken.balanceOf(address(this))
```

[I-6] TSwapPool lacking zero address check

```
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);
    }
```