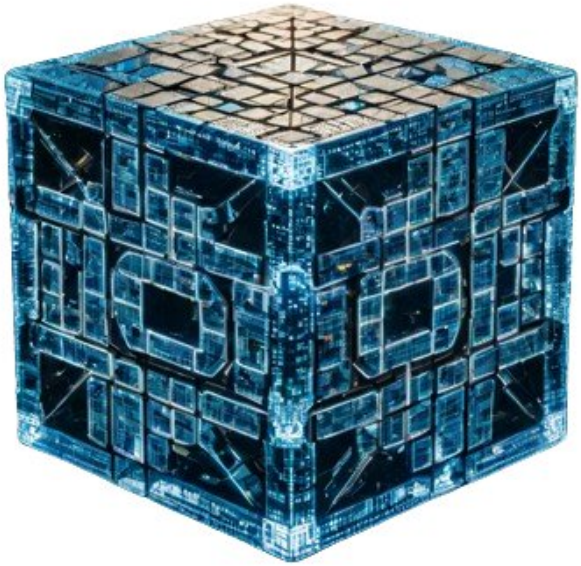

TSwap Protocol Audit Report

Version 1.0

PsychoPunkSage

January 21, 2024

Protocol Audit Report

PsychoPunkSage

Jan 21, 2024

Prepared by: PsychoPunkSage

Table of Contents

        * Description:
        * Impact:
        * Proof of Concept:
        * Recommended Mitigation:
    - [H-3] `TSwapPool::sellPoolToken` mismatches input and output tokens causing users to recieve the incorrect amount of tokens
        * Description:
        * Impact:
        * Recommended Mitigation:
    - [H-4] In `TSwapPool::_swap` the extra tokens given to user after every `swapCount` breaks the protocol invariant of `x * y = k`
        * Description:
        * Impact:
        * Proof of Concept:
        * Recommended Mitigation:

- Medium
    - [M-1] `TSwapPool::deposit` is missing deadline check, can cause transaction to complete even after the deadline has been reached
        * Description:
        * Impact:
        * Proof of Concept:
        * Recommended Mitigation:

- Low
    - [L-1] `TSwapPool::LiquidityAdded` has paramaters out of order, events will emit wrong information
        * Description:
        * Impact:
        * Recommended Mitigation:
    - [L-2] PUSH0 is not supported by all chains
        * Description:
    - [L-3] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given.
        * Description:
        * Impact:
        * Recommended Mitigation:

- Informational
  - [I-1] `PoolFactory__PoolDoesNotExist` is not used anywhere, it should be removed.
    * Description:
    * Recommended Mitigation
  - [I-2] Lacking zero address checks
    * Description:
    * Recommended Mitigation:
  - [I-3] `PoolFactory__createPool` should use `.symbol()` instead of `.name()`
    * Description:
    * Recommended Mitigation:
  - [I-4] If an event has more than parameters, 3 must be indexed
    * Description:
    * Recommended Mitigation:
  - [I-5] It is always a good practice to follow `CEI` (Check, Execute, Interact)
    * Description:
    * Recommended Mitigation:
  - [I-6] Use of “Magic numbers” are discouraged, it can be confusing to see random numbers pop out
    * Description:
    * Recommended Mitigation:
  - [I-7] Each and every functions should have its own `Natspec`
    * Description:
    * Recommended Mitigation:
  - [I-8] Functions not used internally could be marked external
    * Description:
    * Recommended Mitigation:
- Gas
  - [G-1] `TSwapPool:deposit:poolTokenReserves` is never used, so it should be removed from the code.
    * Description:
    * Recommended Mitigation:

Protocol Summary

This project is meant to be a permissionless way for users to swap assets between each other at a fair price. You can think of T-Swap as a decentralized asset/token exchange (DEX). T-Swap is known as an Automated Market Maker (AMM) because it doesn't use a normal "order book" style exchange, instead it uses "Pools" of an asset. It is similar to Uniswap.

Disclaimer

I make 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 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

The findings described below in this doc is base on following commit hash:

`e643a8d4c2c802490976b538dd009b351b1c8dda`

Scope

```
./src/
#-- PoolFactory.sol
#-- TSwapPool.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	3
Info	8
Gas	1
Total	17

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 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 user to recieve way fewer tokens.

Description:

> The `swapExactOutput` function doesn' provides any kind of slipage protection. This function is similar to what is done in `TSwapPool::swapExactInput`, where the function specify the `minOutputAmount`, the `swapExactOutput` function should specify a `maxInputAmount`.

Impact:

> If the market conditions changes before the transaction processes, the user could get a much worse swap.

Proof of Concept:

1. The price of WETH is 1000 USDC.
2. User inputs a `swapExactOutput` function looking for 1 WETH.
    1. inputToken = 1000 USDC
    2. outputToken = WETH
    3. outputAmount = 1
    4. deadline = whatever
3. The function didn't offer the `maxInput` amount.
4. As the transaction is pending in the mempool, the market changes!! Now 1 WETH = 10000 USDC (i.e. 10X more than what user expected)
5. The transaction completes but the user sent the Pool 10000 USDC instead of 1000 USDC.

Recommended Mitigation:

> We should include a `maxInputAmount` ao that the user only have to send a specific amount, and can predict how much the have to spend in the pool.

```
function swapExactOutput(
        IERC20 inputToken,
        IERC20 outputToken,
        uint256 outputAmount,
+       uint256 maxInputAmount
        uint64 deadline
    )
        .....................
    {
        .....................
+       if (inputAmount > maxInputAmount) {
+           revert();
+       }
        _swap(inputToken, inputAmount, outputToken, outputAmount);
    }
```

[H-3] `TSwapPool::sellPoolToken` mismatches input and output tokens causing users to recieve the incorrect amount of tokens

Description:

> The `sellPoolTokens` function is intended to allow users to easily sell poolTokens and receive WETH in exchange. Users indicate how many pool tokens they're willing to sell in the `poolTokenAmount` parameter. However, the function currently mismatches the swapped amount. This is due to the fact that `swapExactOutput` function is called, whereas `swapExactInput` function is the one to be called. Because user specify the exact amount of token, not amount.

Impact:

> Uwer will swap wrong amount of pool tokens, which is severe disruption of protocol functionality.

Recommended Mitigation:

> Consider changing the implementation to use `swapExactInput` instead of `swapExactOutput`. Note, ths would also require changing the `selllPoolTokens` 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 would 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 user after every `swapCount` breaks the protocol invariant of `x * y = k`

Description:

> Protocol follws the strict invariant of `x * y = k`. where: - `x`: balance of pool token. - `y`: balance of WETH token - `k`: constant product of two balances.

> This mean that whenever the balances change in the protocol, the ratio of the amount of tokens should remain constant i.e. `k`. However, this is broken due to the extra incentive in the `TSwapPool::_swap` function. Meaning, overtime protocol funds will be drained.

Impact:

> A user could maliciously drain the protocol funds by doing a lot of swaps and collecting th extra incentive given out by the protocol.

> Most simply put, the protocol's core invariant is broken.

> Following block of code is resposible for issue:

```
swap_count++;
if (swap_count >= SWAP_COUNT_MAX) {
    swap_count = 0;
    outputToken.safeTransfer(msg.sender, 1_000_000_000_000_000_000);
}
```

Proof of Concept:

1. User swaps 10 times, and the collects extra incentive of `1_000_000_000_000_000_000` tokens.
2. The user continues to swap ubtil all the protocol funds are drained.

PoC

```
function testInvariantBroken() public {
    vm.startPrank(liquidityProvider);
    weth.approve(address(pool), 100e18);
    poolToken.approve(address(pool), 100e18);
    pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
```

```
6       vm.stopPrank();
7
8       uint256 outputWeth = 1e17;
9       int256 startingY = int256(weth.balanceOf(address(pool)));
10      int256 expectedDeltaY = int256(-10) * int256(outputWeth);
11
12      // Swap
13      vm.startPrank(user);
14      poolToken.approve(address(pool), type(uint64).max);
15      for (uint i = 0; i < 10; i++) {
16          pool.swapExactOutput(
17              poolToken,
18              weth,
19              outputWeth,
20              uint64(block.timestamp)
21          );
22      }
23      vm.stopPrank();
24
25      int256 endingY = int256(weth.balanceOf(address(pool)));
26      int256 actualDeltaY = int256(endingY) - int256(startingY);
27      assertEq(actualDeltaY, expectedDeltaY);
28  }
```

Recommended Mitigation:

> Remove the extra incentive. If you want to keep it, 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++;
- 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, can cause transaction to complete even after the deadline has been reached

Description:

> The `Deposit` function accepts a **deadline** parameter, according to natspec it is `The deadline for the transaction to be completed by`. But unfortunatel it is never used inside `TSwapPool::deposit`

> As a consequesnce, operations that add liquidity to the Pool might get executed at unexpected times, in market conditions where deposit rate is unfavourable.

Impact:

> Transaction could be sent even though the market conditon is unfavourable, even adding a deadline parameter.

Proof of Concept:

> The `deadline` parameter is unused.

Recommended Mitigation:

Consider making the following changes:

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

Low

[L-1] `TSwapPool::LiquidityAdded` has paramaters out of order, events will emit wrong information

Description:

> When `LiquidityAdded` event is emitted in `TSwapPool::_addLiquidityMintAndTransfer` function, it logs value in incorrect order. The `poolTokensToDeposit` value should go in the 3rd parameter position whereas the `wethToDeposit` value should go to second parameter position.

Impact:

> Event emission is incorrect, this will lead to malfunction in **off-chain functions**.

Recommended Mitigation:

```
-emit LiquidityAdded(msg.sender, poolTokensToDeposit, wethToDeposit);
+emit LiquidityAdded(msg.sender, wethToDeposit, poolTokensToDeposit);
```

[L-2] PUSH0 is not supported by all chains

Description:

> Solc compiler version `0.8.20` switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain other than mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.

[L-3] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given.

Description:

> The `swapExactInput` fucntion 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 an explicit return statement is used.

Impact:

> The return value will always be `0`, giving the caller wrong information.

Recommended Mitigation:

```
function swapExactInput(
        .........
    )
        public
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        returns (uint256 output)
    {
        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__PoolDoesNotExist` is not used anywhere, it should be removed.

Description:

> Since **PoolFactory__PoolDoesNotExist** is not used anywhere, it should be removed as it may create confusion in future.

Recommended Mitigation

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

[I-2] Lacking zero address checks

Description:

> Zero address checks are important to avoid any unintentional erroneous addresses.

Recommended Mitigation:

```
# src/PoolFactory.sol
constructor(address wethToken) {
+        if (wethToken == address(0)){
+            revert();
+        }
         i_wethToken = wethToken;
     }

# src/TSwapPool.sol
constructor(
         address poolToken,
         address wethToken,
         string memory liquidityTokenName,
         string memory liquidityTokenSymbol
     ) ERC20(liquidityTokenName, liquidityTokenSymbol) {
+        if (wethToken == address(0)){
+            revert();
+        }
+        if (poolToken == address(0)){
+            revert();
+        }
         i_wethToken = IERC20(wethToken);
         i_poolToken = IERC20(poolToken);
     }
```

[I-3] `PoolFactory__createPool` should use `.symbol()` instead of `.name()`

Description:

Contract intends to concat `symbol` not name of token.

Recommended Mitigation:

```
string memory liquidityTokenName = string.concat(
            "T-Swap ",
-            IERC20(tokenAddress).name()
+            IERC20(tokenAddress).symbol()
        );
        string memory liquidityTokenSymbol = string.concat(
            "ts",
-            IERC20(tokenAddress).name()
+            IERC20(tokenAddress).symbol()
        );
```

[I-4] If an event has more than parameters, 3 must be indexed

Description:

It is a good practice to index some parameters of the event, tis will also help some external services to used those indexed parameters.

Recommended Mitigation:

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

[I-5] It is always a good practice to follow `CEI` (Check, Execute, Interact)

Description:

> CEI convention should be followed to avoid any kind of misbehaviour in smart-contract.

Recommended Mitigation:

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline // @Done-Audit-H: not being used...
    )
        external
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
        ......
        if (totalLiquidityTokenSupply() > 0) {
            ......
        } else {
            // This will be the "initial" funding of the protocol. We
               are starting from blank here!
            // We just have them send the tokens in, and we mint
               liquidity tokens based on the weth
+           liquidityTokensToMint = wethToDeposit;
            _addLiquidityMintAndTransfer(
                wethToDeposit,
                maximumPoolTokensToDeposit,
                wethToDeposit
            );
-           liquidityTokensToMint = wethToDeposit;
        }
    }
```

[I-6] Use of “Magic numbers” are discouraged, it can be confusing to see random numbers pop out

Description:

> It is the best practice to avoid using magic numbers as it often confuses people, it is much more readable if the numbers are given names.

Recommended Mitigation:

```
+uint256 public constant PRIZE_POOL_PERCENTAGE = 997;
+uint256 public constant POOL_PRECISSION = 1000;
```

[I-7] Each and every functions should have its own `Natspec`

Description:

> All functions used in smart contract should have a Natspec, these are important to get an insight of the paramaters being used by the function and the functionality of the function provides.

Recommended Mitigation:

> `TSwapPool::swapExactInput` does not have a Natspec.

[I-8] Functions not used internally could be marked external

Description:

> Function `TSwapPool::swapExactInput` is never used inside the contract an hence should be marked `external`.

Recommended Mitigation:

```
function swapExactInput(
        IERC20 inputToken,
        uint256 inputAmount,
        IERC20 outputToken,
        uint256 minOutputAmount,
        uint64 deadline
    )
-       public
+       external
        revertIfZero(inputAmount)
        revertIfDeadlinePassed(deadline)
        // @Audit-L: output not used anywhere...
        returns (uint256 output)
```

Gas

[G-1] `TSwapPool:deposit:poolTokenReserves` is never used, so it should be removed from the code.

Description:

> Unused/unnecessary variables should be removed form the codebase

Recommended Mitigation:

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline
    )
        external
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
        if (wethToDeposit < MINIMUM_WETH_LIQUIDITY) {
            revert TSwapPool__WethDepositAmountTooLow(
                MINIMUM_WETH_LIQUIDITY,
                wethToDeposit
            );
        }
        if (totalLiquidityTokenSupply() > 0) {
            uint256 wethReserves = i_wethToken.balanceOf(address(this))
                ;

-           uint256 poolTokenReserves = i_poolToken.balanceOf(address(
    this));
```