Audit Report

Version 1.0

elpabl0.eth

January 14, 2024

Audit Report

elpabl0.eth

January 10, 2024

Prepared by: elpabl0.eth Lead Auditors:

- elpabl0.eth

Table of Contents

    * [H-5] In `TSwapPool::_swap` the extra tokens given to users after every `swapCount` breaks the protocol invariant of `x * y = k`
  - Low
    * [L-1] `TSwapPool::LiquidityAdded` event has parameters out of order
    * [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
  - Informational
    * [I-1] Event is missing `indexed` fields
    * [I-2] `TSwapPool::MINIMUM_WETH_LIQUIDITY` is constant and therefore not required to be emitted
    * [I-3] Unused local variables
    * [I-4] External call in `TSwapPool::deposit` should be placed before the `_addLiquidityMintAndTransfer` call to follow CEI
    * [I-5] Magic numbers
    * [I-6] `TSwapPool::swapExactInput` and `TSwapPool::totalLiquidityTokenSupply` should use `external` instead of **`public`** since it's not being called internally
    * [I-7] `TSwapPool::swapExactInput` missing `natspec`
    * [I-8] `TSwapPool::swapExactOutput` missing param for `deadline` in `natspec`
    * [I-8] `TSwapPool::PoolFactory__PoolDoesNotExist` is not being used anywhere.
    * [I-9] Lacking zero address checks
    * [I-10] `PoolFacotry::createPool` should use `.symbol()` instead of `.name()`

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

I 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:

```
1ec3c30253423eb4199827f59cf564cc575b46db
```

Scope

```
-- src
|---- PoolFactory.sol
|---- TSwapPool.sol
```

Roles

- Liquidity Provider: An account who deposits assets into the pool to earn trading fees.
- User: An account who swaps tokens.

Issues found

Severity	Number of issues found
High	5

Severity	Number of issues found
Medium	0
Low	2
Info	10
Gas	0
Total	17

Findings

High

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

[H-2] 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-3] 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 transaciton 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 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 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-4] `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.

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 (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));
```

```
7       }
```

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

[H-5] 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.

```
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           }
```

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

Proof Of Code

Place the following into `TSwapPool.t.sol`.

```
1
2       function testInvariantBroken() public {
3           vm.startPrank(liquidityProvider);
4           weth.approve(address(pool), 100e18);
5           poolToken.approve(address(pool), 100e18);
6           pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp));
7           vm.stopPrank();
8
9           uint256 outputWeth = 1e17;
10
11          vm.startPrank(user);
12          poolToken.approve(address(pool), type(uint256).max);
```

```
13          poolToken.mint(user, 100e18);
14          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
15          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          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
23
24          int256 startingY = int256(weth.balanceOf(address(pool)));
25          int256 expectedDeltaY = int256(-1) * int256(outputWeth);
26
27          pool.swapExactOutput(poolToken, weth, outputWeth, uint64(block.
                timestamp));
28          vm.stopPrank();
29
30          uint256 endingY = weth.balanceOf(address(pool));
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 -         // Fee-on-transfer
3 -         if (swap_count >= SWAP_COUNT_MAX) {
4 -             swap_count = 0;
5 -             outputToken.safeTransfer(msg.sender, 1
      _000_000_000_000_000_000);
6 -         }
```

Low

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

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 position, whereas the `wethToDeposit` value should go second.

Impact: Event emission is incorrect, leading to off-chain 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 `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] Event is missing `indexed` fields

Description: Index event fields make the field more quickly accessible to off-chain tools that parse events.

- Found in src/PoolFactory.sol Line: 35

```
1       event PoolCreated(address tokenAddress, address poolAddress);
            //ii @audit-i Event is missing indexed fields
```

- Found in src/TSwapPool.sol Line: 44

```
1       event LiquidityAdded(address indexed liquidityProvider,
            uint256 wethDeposited, uint256 poolTokensDeposited); //ii
            @audit-i Event is missing indexed fields
```

- Found in src/TSwapPool.sol Line: 45

```
1       event LiquidityRemoved(address indexed liquidityProvider,
            uint256 wethWithdrawn, uint256 poolTokensWithdrawn); //ii
            @audit-i Event is missing indexed fields
```

- Found in src/TSwapPool.sol Line: 46

```
1       event Swap(address indexed swapper, IERC20 tokenIn, uint256
            amountTokenIn, IERC20 tokenOut, uint256 amountTokenOut); //
            ii @audit-i Event is missing indexed fields
```

[I-2] `TSwapPool::MINIMUM_WETH_LIQUIDITY` is constant and therefore not required to be emitted

[I-3] Unused local variables

Description: There are some variables that are not being used anywhere, consider removing it for more code clarity and gas efficiency.

Recommended Mitigation:

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

[I-4] External call in `TSwapPool::deposit` should be placed before the `_addLiquidityMintAndTransfer` call to follow CEI

Description: Consider using the CEI (Check-Effect-Interactions) design pattern to minimize code vulnerabilities.

Recommended Mitigation:

```
function deposit(
        uint256 wethToDeposit,
        uint256 minimumLiquidityTokensToMint,
        uint256 maximumPoolTokensToDeposit,
        uint64 deadline //ii @audit-i Unused local variable
        //!!! @audit-h deadline is not being used anywhere, may disrupt protocol functionality
    )
        external
        revertIfZero(wethToDeposit)
        returns (uint256 liquidityTokensToMint)
    {
        if (wethToDeposit < MINIMUM_WETH_LIQUIDITY) {
            //* ii @audit-i MINIMUM_WETH_LIQUIDITY is constant and therefore not required to be emitted
            revert TSwapPool__WethDepositAmountTooLow(
                MINIMUM_WETH_LIQUIDITY, wethToDeposit);
        }
        if (totalLiquidityTokenSupply() > 0) {
            uint256 wethReserves = i_wethToken.balanceOf(address(this));
            uint256 poolTokenReserves = i_poolToken.balanceOf(address(this));
            uint256 poolTokensToDeposit = getPoolTokensToDepositBasedOnWeth(wethToDeposit);
            if (maximumPoolTokensToDeposit < poolTokensToDeposit) {
                revert TSwapPool__MaxPoolTokenDepositTooHigh(
                    maximumPoolTokensToDeposit, poolTokensToDeposit);
            }
            liquidityTokensToMint = (wethToDeposit * totalLiquidityTokenSupply()) / wethReserves;
            if (liquidityTokensToMint < minimumLiquidityTokensToMint) {
                revert TSwapPool__MinLiquidityTokensToMintTooLow(
                    minimumLiquidityTokensToMint, liquidityTokensToMint);
            }
            _addLiquidityMintAndTransfer(wethToDeposit, poolTokensToDeposit, liquidityTokensToMint);
        } else {
-           _addLiquidityMintAndTransfer(wethToDeposit, maximumPoolTokensToDeposit, wethToDeposit);
-           liquidityTokensToMint = wethToDeposit;
```

```
31 +            liquidityTokensToMint = wethToDeposit;
32 +            _addLiquidityMintAndTransfer(wethToDeposit,
       maximumPoolTokensToDeposit, wethToDeposit);
33          }
34      }
```

[I-5] Magic numbers

Description: All number literals should be replaced with constant. This makes the code more readable and easier to maintain. Number without context are called “Magic Numbers”.

- Found in src/TSwapPool.sol Line: 233

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

- Found in src/TSwapPool.sol Line: 236

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

- Found in src/TSwapPool.sol Line: 254

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

- Found in src/TSwapPool.sol Line: 343

```
1              outputToken.safeTransfer(msg.sender, 1
                   _000_000_000_000_000_000); //ii @audit-u magic
                   numbers
```

- Found in src/TSwapPool.sol Line: 387

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

- Found in src/TSwapPool.sol Line: 394

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

Recommended Mitigation: Replace all “Magic Numbers” with constant.

[I-6] `TSwapPool::swapExactInput` and `TSwapPool::totalLiquidityTokenSupply` should use `external` instead of `public` since it's not being called internally

Description: Function that not being called internally could be marked as `external` instead.

[I-7] `TSwapPool::swapExactInput` missing `natspec`

Description: Consider adding `natspec` comments to provide more context about the purpose and behavior of crucial functions.

[I-8] `TSwapPool::swapExactOutput` missing param for `deadline` in `natspec`

[I-8] `TSwapPool::PoolFactory__PoolDoesNotExist` is not being used anywhere.

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

[I-9] Lacking zero address checks

- Found in src/PoolFactory.sol Line: 42

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

- Found in src/TSwapPool.sol Line: 77

```
1      constructor(
2          address poolToken,
3          address wethToken,
4          string memory liquidityTokenName,
5          string memory liquidityTokenSymbol
6      )
7          ERC20(liquidityTokenName, liquidityTokenSymbol)
8      {
9  +       if(poolToken == address(0) || wethToken == address(0)) {
10 +           revert();
11 +        }
12         i_wethToken = IERC20(wethToken);
13         i_poolToken = IERC20(poolToken);
14     }
```

[I-10] `PoolFacotry::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());
```