Prepared by: Yavor

Lead Auditors: yavor.eth

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
  - [H-1] `TSwapPool::deposit` is missing deadline check causing transactions to complete even after deadline
  - [H-2] Inccorect fee calculation in `TSwapPool::getInputAmountBasedOnOutput` causes protocol to take too many tokens from users, resulting in lost fees.
    - [H-3] Lack of slippage protection in `TSwapPool::swapExactOutput` causes users to potentially receive way fewer tokens.
    - [H-4] TSwapPool::sellPoolTokens mismatches input and output tokens causing users to receive the incorrect amount of tokens
    - [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::LiqudidityAdded` event has parametars out of order
  - [L-2] Default value returned by `TSwapPool::swapExactInput` results in incorrect return value given
  - Informational
    - [I-1] `PoolFactory::PoolFactory__PoolDoesNotExist` is not used and should be removed.
    - [I-2] Lacking zero address checkes
    - [I-3] `PoolFactory::createPool` should use `.symbol()` instead of `.name()`
  - [I-4]: Event is missing `indexed` fields

Protocol Summary

TSwapPool is a decentralized liquidity pool protocol that facilitates token swaps between two ERC20 tokens. Users can deposit tokens into the pool to provide liquidity and receive pool tokens in return. The protocol allows users to swap tokens by specifying either the exact amount of input or output tokens. Liquidity providers earn fees from swap transactions, which can be collected and withdrawn. The protocol ensures proper token management and error handling for transactions, with liquidity governed by the amounts deposited and withdrawn.

Disclaimer

The yavor.eth 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

Severity	**Number of issues found**
High	5
Medium	0
Low	2
Info	4
Gas	0
Total	11

Findings

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

Description: The `deposit` function accepts a deadline parametar, which according to the documentation is "The deadline for the transaction to be completed by". However, this parametar is never used. As a consuquence, 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 parametar.

Proof of Concept: The `deadline` parametar is unusable.

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)
        {
```

[H-2] Inccorect 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, this 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 Concept: As a result, users swapping tokens via the swapExactOutput function will pay far more tokens than expected for their trades. This becomes particularly risky for users that provide infinite allowance to the TSwapPool contract. Moreover, note that the issue is worsened by the fact that the swapExactOutput function does not allow users to specify a maximum of input tokens, as is described in another issue in this report. It's worth noting that the tokens paid by users are not lost, but rather can be swiftly taken by liquidity providers. Therefore, this contract could be used to trick users, have them swap their funds at unfavorable rates and finally rug pull all liquidity from the pool. To test this, include the following code in the TSwapPool.t.sol file:

```
 function testFlawedSwapExactOutput() public {
        uint256 initialLiquidity = 100e18;
        vm.startPrank(liquidityProvider);
        weth.approve(address(pool), 100e18);
        poolToken.approve(address(pool), 100e18);
        pool.deposit(initialLiquidity, 0, initialLiquidity,
uint64(block.timestamp));
        vm.stopPrank();

        address someUser = makeAddr("someUser");
        uint256 UserInitialPoolTokenBalace = 11e18;
        poolToken.mint(someUser, UserInitialPoolTokenBalace);
        vm.startPrank(someUser);
        poolToken.approve(address(pool), type(uint256).max);
        pool.swapExactOutput(poolToken, weth, 1 ether,
uint64(block.timestamp));

        assertLt(poolToken.balanceOf(someUser), 1 ether);
        vm.stopPrank();

        vm.startPrank(liquidityProvider);
        pool.withdraw(pool.balanceOf(liquidityProvider), 1, 1,
uint64(block.timestamp));

        assertEq(weth.balanceOf(address(pool)), 0);
        assertEq(poolToken.balanceOf(address(pool)), 0);
         }
```

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 similiar 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 would get a much worse swap.

Proof of Concept:

The price of 1 WETH right now is 1,000 USDC User inputs a swapExactOutput looking for 1 WETH inputToken = USDC outputToken = WETH outputAmount = 1 deadline = whatever The function does not offer a maxInput amount 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 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:

▶ Proof of Code

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

        vm.startPrank(user);
        poolToken.approve(address(pool), 9e18);
        poolToken.mint(user, 9e18);
        uint256 expectedWeth = pool.getOutputAmountBasedOnInput(9e18,
poolToken.balanceOf(address(pool)), weth.balanceOf(address(pool)));

        vm.expectRevert();
        pool.sellPoolTokens(9e18);
    }
```

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));
    }
```

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.

```
        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 continues to swap until all the protocol funds are drained

▶ Proof of Code

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

```
function testNottBroken() 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(uint64).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);
-        }
```

Low

[L-1] `TSwapPool::LiqudidityAdded` event has parametars out of order

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

Impact: Event emission is incorrect, leading to off-chain functions potentionally 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 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);
+            uint256 output =
getOutputAmountBasedOnInput(inputAmount,inputReserves,outputReserves);

-           if (outputAmount < minOutputAmount) {
-               revert TSwapPool__OutputTooLow(outputAmount,
minOutputAmount);
            }
+           if (output < minOutputAmount) {
+               revert TSwapPool__OutputTooLow(output, minOutputAmount);
            }

-           _swap(inputToken, inputAmount, outputToken, outputAmount);
+           _swap(inputToken, inputAmount, outputToken, output);
        }
```

Informational

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

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

[I-2] Lacking zero address checkes

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

[I-3] `PoolFactory::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());
```

[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

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

- Found in src/TSwapPool.sol Line: 52

```
        event LiquidityAdded(
```

- Found in src/TSwapPool.sol Line: 57

```
        event LiquidityRemoved(
```

- Found in src/TSwapPool.sol Line: 62

```
        event Swap(
```