Audit Findings Report

Contest / Project Name: [T-Swap]]

Auditor: [Harisuthan]

Bug-Report

1. Missing Deadline Check in deposit

Description

The deposit function allows users to provide a deadline parameter but does not validate it. As a result, a user can accidentally pass an incorrect (past) deadline. If this happens, their funds may be locked, and they won't be able to withdraw.

Impact

User funds may get stuck in the contract.

The function continues execution even with an invalid deadline, causing logical errors.

Proof of Code

```
function testdepositnotcheckingdeadline() external {
    vm.warp(block.timestamp + 1 days);
    vm.startPrank(liquidityProvider);
    weth.approve(address(pool), 100e18);
    poolToken.approve(address(pool), 100e18);

    // Passing a deadline older than the current time
    pool.deposit(100e18, 100e18, 100e18, uint64(block.timestamp - 1 days));
    vm.stopPrank();

    assertEq(pool.balanceOf(liquidityProvider), 100e18);
    assertEq(weth.balanceOf(liquidityProvider), 100e18);
    assertEq(poolToken.balanceOf(liquidityProvider), 100e18);

    assertEq(weth.balanceOf(address(pool)), 100e18);
    assertEq(poolToken.balanceOf(address(pool)), 100e18);
}
```

Here, the deposit proceeds even though the deadline is already expired.

Run with:

```
solidity forge test --mt testdepositnotcheckingdeadline
```

Recommendation

Add a deadline validation check:

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

2. Zero Value in deposit Breaks Ratio Logic

Description

The deposit function requires users to maintain the correct ratio of WETH and pool tokens. However, if the first depositor provides 0 pool tokens, the contract logic breaks. Subsequent deposits ignore pool token amounts and only consider WETH, making the pool unbalanced.

Impact

Contract logic is broken.

Users cannot deposit pool tokens anymore.

Proof of Code

function testdepositzeroamountandmakeitstopothersdeposit() external { vm.startPrank(user); weth.approve(address(pool), 100e18); poolToken.approve(address(pool), 100e18); pool.deposit(10e18, 10e18, 0, uint64(block.timestamp)); vm.stopPrank();

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

assertEq(weth.balanceOf(address(pool)), 110e18);
assertEq(poolToken.balanceOf(address(pool)), 0);
```

}

Here, the pool token balance remains 0 even after deposits.

Run with:

forge test --mt testdepositzeroamountandmakeitstopothersdeposit

Recommendation

Add a zero check for pool tokens:

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

3. Incorrect Calculation in getInputAmountBasedOnOutput

Description

The getInputAmountBasedOnOutput function incorrectly uses 10000 instead of 1000 in its formula, causing inflated input requirements. For example, a user who should only pay 100 tokens may instead be required to pay 1000.

Impact

Users overpay for swaps.

Incorrect calculations discourage participation.

Proof of Code

function getInputAmountBasedOnOutput(uint256 outputAmount, uint256 inputReserves, uint256 outputReserves) public pure revertIfZero(outputAmount) revertIfZero(outputReserves) returns (uint256 inputAmount) { return ((inputReserves * outputAmount) * 10000) / ((outputReserves - outputAmount) * 997); }

Here, the multiplier should be 1000 instead of 10000.

Recommendation

Correct the calculation:

return ((inputReserves * outputAmount) * 1000) / ((outputReserves - outputAmount) * 997);

4. Missing Minimum Input Check in swapExactOutput

Description

The swapExactOutput function does not take a minimum input amount from the user. This means users cannot specify the maximum tokens they are willing to spend. The function may take more tokens than expected, frustrating users.

Impact

Users may lose more funds than intended.

Poor user experience and trust issues.

Proof of Code

function swapExactOutput(IERC20 inputToken, IERC20 outputToken, uint256 outputAmount, uint64 deadline) public revertIfZero(outputAmount) revertIfDeadlinePassed(deadline) returns (uint256 inputAmount) { uint256 inputReserves = inputToken.balanceOf(address(this)); uint256 outputReserves = outputToken.balanceOf(address(this));

```
 inputAmount = getInputAmountBasedOnOutput(
    outputAmount,
    inputReserves,
    outputReserves
 );

 _swap(inputToken, inputAmount, outputToken, outputAmount);
```

}

Recommendation

Require a user-provided minInputToken parameter and validate against it:

function swapExactOutput(IERC20 inputToken, IERC20 outputToken, uint256 outputAmount, uint256 minInputToken, uint64 deadline) public revertIfZero(outputAmount) revertIfDeadlinePassed(deadline) returns (uint256 inputAmount) { uint256 inputReserves = inputToken.balanceOf(address(this)); uint256 outputReserves = outputToken.balanceOf(address(this));

```
inputAmount = getInputAmountBasedOnOutput(
    outputAmount,
    inputReserves,
    outputReserves
);

if (inputAmount > minInputToken) {
    revert("Input required exceeds user-specified minimum");
}

_swap(inputToken, inputAmount, outputToken, outputAmount);
```

}

5. Incorrect Function Used in sellPoolTokens

Description

The sellPoolTokens function currently calls swapExactOutput. However, the intended behavior is for users to input a fixed amount of pool tokens and receive the corresponding output. For this logic, swapExactInput should be used instead.

Impact

Wrong function behavior.

Contract logic is broken, leading to unexpected results.

Proof of Code

function sellPoolTokens(uint256 poolTokenAmount) external returns (uint256 wethAmount) { return swapExactOutput(i_poolToken, i_wethToken, poolTokenAmount, uint64(block.timestamp) // should be user-provided); }

Recommendation

Replace swapExactOutput with swapExactInput:

function sellPoolTokens(uint256 poolTokenAmount) external returns (uint256 wethAmount) { return swapExactInput(i_poolToken, poolTokenAmount, i_wethToken, deadline); }