Here's the updated `report.md` file with the categorization of findings by their impact:

```
# Smart Contract Audit Report

## Overview
This document outlines the findings from the audit of the `TSwapPool` smart
contract. The audit covers several aspects including access control, integer
overflows and underflows, and deadline handling.

## Summary of Findings
- **High Impact:** 3
- **Medium Impact:** 1
- **Low Impact:** 2

## Findings

### 1. Unused Function Parameter
**Location:** `src/TSwapPool.sol:117:9` and `src/TSwapPool.sol:308:18`

**Impact:** Low
Unused function parameters can lead to confusion and potential misunderstandings
about the function's intent.

**Proof of Concept:**
```solidity
// Unused parameter 'deadline'
function someFunction(uint64 deadline) public {
    // Function body
}
```

Mitigation: Remove or comment out the unused parameter:

```
function someFunction(uint64 /*deadline*/) public {
    // Function body
}
```

2. Unused Local Variable

Location: `src/TSwapPool.sol:131:13`

Impact: Low Unused local variables can lead to unnecessary gas costs and reduce the readability of the code.

Proof of Concept:

```
// Unused local variable 'poolTokenReserves'
uint256 poolTokenReserves = i_poolToken.balanceOf(address(this));
```
```

Mitigation: Remove or comment out the unused variable:

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

3. Access Control Violation on Withdraw

Location: `test/unit/swaptest.t.sol:TSwapPoolTest`, `src/TSwapPool.sol:280:9`

Impact: High A user without the required permissions can attempt to withdraw tokens, leading to potential loss of funds.

Proof of Concept:

```
function testAccessControlOnWithdraw() public {
    // Attempting to withdraw without permissions
    try swapPool.withdraw(100 ether, 50 ether, 50 ether, uint64(block.timestamp +
1)) {
        fail();
    } catch Error(string memory reason) {
        assertEq(reason, "Access denied");
    }
}
```

Mitigation: Ensure that only authorized users can perform withdrawals by implementing proper access control checks.

4. Integer Overflow on Deposit

Location: `test/unit/swaptest.t.sol:TSwapPoolTest`,`src/TSwapPool.sol:177:9`

Impact: High Depositing a large amount can cause an integer overflow, resulting in incorrect balances and potential loss of funds.

Proof of Concept:

```
function testIntegerOverflowOnDeposit() public {
    uint256 maxUint256 = type(uint256).max;

    // Approve maximum amount of tokens
    weth.approve(address(swapPool), maxUint256);
    poolToken.approve(address(swapPool), maxUint256);

    // Try to add liquidity with max amounts (this should fail due to overflow)
    try swapPool.deposit(maxUint256, 1 ether, maxUint256, uint64(block.timestamp +
1)) {
        fail();
    } catch Error(string memory reason) {
        assertEq(reason, "SafeMath: addition overflow");
```

```
    }
}
```

Mitigation: Use SafeMath functions to handle arithmetic operations and prevent overflows.

5. Integer Underflow on Withdraw

Location: `test/unit/swaptest.t.sol:TSwapPoolTest`, `src/TSwapPool.sol:280:9`

Impact: High Withdrawing more tokens than the user's balance can cause an integer underflow, leading to incorrect balances and potential loss of funds.

Proof of Concept:

```
function testIntegerUnderflowOnWithdraw() public {
    uint256 wethAmount = 100 ether;
    uint256 poolTokenAmount = 100 ether;
    uint256 liquidityTokens;

    // Approve tokens
    weth.approve(address(swapPool), wethAmount);
    poolToken.approve(address(swapPool), poolTokenAmount);

    // Add liquidity
    liquidityTokens = swapPool.deposit(wethAmount, 50 ether, poolTokenAmount,
uint64(block.timestamp + 1));

    // Try to withdraw more liquidity than available (this should fail due to
underflow)
    try swapPool.withdraw(liquidityTokens + 1, 50 ether, 50 ether,
uint64(block.timestamp + 1)) {
        fail();
    } catch Error(string memory reason) {
        assertEq(reason, "SafeMath: subtraction overflow");
    }
}
```

Mitigation: Use SafeMath functions to handle arithmetic operations and prevent underflows.

6. Deadline Handling

Location: `test/unit/swaptest.t.sol:TSwapPoolTest`, `src/TSwapPool.sol:339:9`

Impact: Medium Incorrect handling of deadlines can lead to failed transactions if the deadline has already passed.

Proof of Concept:

```
function testRemoveLiquidityWithLowDeadline() public {
    uint256 wethAmount = 100 ether;
    uint256 poolTokenAmount = 100 ether;
    uint256 liquidityTokens;

    // Approve tokens
    weth.approve(address(swapPool), wethAmount);
    poolToken.approve(address(swapPool), poolTokenAmount);

    // Add liquidity
    liquidityTokens = swapPool.deposit(wethAmount, 50 ether, poolTokenAmount,
uint64(block.timestamp + 1));

    // Try to remove liquidity with an expired deadline
    try swapPool.withdraw(liquidityTokens, 50 ether, 50 ether,
uint64(block.timestamp - 1)) {
        fail();
    } catch Error(string memory reason) {
        assertEq(reason, "Deadline expired");
    }
}
```

Mitigation: Ensure that deadlines are set appropriately to a future timestamp during the test.

Conclusion

The audit identified several issues with varying degrees of impact. It is recommended to address all findings, particularly those related to access control and integer overflows/underflows, to ensure the security and robustness of the smart contract.

By implementing the suggested mitigations, the `TSwapPool` contract can be significantly improved, making it safer and more reliable for users.

```
This updated report includes the categorization of findings based on their impact
levels and detailed descriptions, proof of concept code, and mitigation steps for
each identified issue.
```