1. Vulnerability Analysis

The key vulnerabilities in the vulnerable ERC777 implementation are:

1. **Hook Execution Order**: Hooks are called before state updates
2. **No Reentrancy Protection**: Missing nonReentrant modifier
3. **Multiple External Calls**: Both sender and recipient hooks create attack vectors

2. Attack Flow

1. **Initial Setup**:
    - Attacker deploys `MaliciousRecipient` contract
    - Registers it with ERC1820 as an ERC777 recipient
    - Sets target number of reentrant calls
2. **Attack Impact**:
    - Each reentrant call occurs before balance updates
    - Multiple transfers executed with same balance
    - Final state becomes inconsistent

3. Example Attack Scenario

```
// Initial state
User balance: 1000 tokens
Attacker balance: 0 tokens

// Attack sequence
1. User sends 100 tokens to Attacker
2. Before balance update, Attacker reenters and withdraws 100 tokens again
3. Process repeats until targetAttackCount reached
4. All balance updates process at the end

// Final state (if targetAttackCount = 3)
User balance: 700 tokens (lost 300 instead of 100)
Attacker balance: 300 tokens (gained 300 from 100)
```

4. Prevention Measures

1. **Use Reentrancy Guard**:

```
modifier nonReentrant() {
    require(!_locked, "ReentrancyGuard: reentrant call");
    _locked = true;
    _;
    _locked = false;
}
```

2. **Correct State Update Order**:

```
function _send(...) internal virtual {
    // 1. State updates first
    _balances[from] = _balances[from].sub(amount);
    _balances[to] = _balances[to].add(amount);

    // 2. External calls after
    _callTokensToSend(...);
    _callTokensReceived(...);
}
```

3. **Transaction Tracking**:

```
mapping(bytes32 => bool) private _executedTransactions;

function _send(...) internal virtual {
    bytes32 txId = keccak256(abi.encodePacked(
        block.number,
        from,
        to,
        amount
    ));
    require(!_executedTransactions[txId], "Transaction already executed");
    _executedTransactions[txId] = true;
    // ... rest of the function
}
```

5. Protection Testing

Always test your ERC777 implementation against reentrancy:

```
contract ERC777Test {
    function testReentrancy() public {
        // Setup
        VulnerableERC777 token = new VulnerableERC777("Test", "TST");
        MaliciousRecipient attacker = new
MaliciousRecipient(address(token));

        // Initial balances
        assertEq(token.balanceOf(user), 1000);
        assertEq(token.balanceOf(address(attacker)), 0);

        // Attempt attack
        token.send(address(attacker), 100, "");

        // Verify balances
        assertEq(token.balanceOf(user), 900);  // Should fail in vulnerable
contract
        assertEq(token.balanceOf(address(attacker)), 100);  // Should fail
in vulnerable contract
    }
}
```