TermMaxOrder Contract Vulnerability Report

Malicious Maker Can Break Order Functionality Through Empty Curve Cuts

MEDIUM SEVERITY

Summary

A vulnerability in the TermMaxOrder contract enables a maker to modify an order with empty curve cuts, leading to critical functions failing because of arithmetic underflow. This results in a denial of service situation where users are unable to interact with the order, effectively locking up liquidity and disrupting integrations that rely on the order's functionality.

Finding Description

The updateOrder function in TermMaxOrder.sol allows a maker to update an order's configuration, including its curve cuts. While the _updateCurve function performs some validation on curve cuts, it doesn't prevent them from being empty arrays. However, core functions like apr() and swap functions assume these arrays have elements and attempt to perform calculations with them. When these arrays are empty, it causes arithmetic underflow when trying to access array indices.

> **Attack Scenario:**
>
> 1. Maker creates an order with valid curve cuts and attracts liquidity
> 2. Users begin trading against the order, building up positions
> 3. Maker maliciously updates the order with empty curve cuts arrays
> 4. All subsequent trading operations fail due to arithmetic underflow

> 5. User funds become effectively trapped in the order

Proof of Concept

A coded function in the OrderTest demonstrating the attack:

```
function testEmptyCurveCutsAttack() public {
    vm.startPrank(maker);

    // 1. First setup order with valid curve cuts and get some trading volume
    uint128 tokenAmtIn = 100e8;
    res.debt.mint(maker, tokenAmtIn);
    res.debt.approve(address(res.order), tokenAmtIn);
    res.order.swapExactTokenToToken(res.debt, res.ft, maker, tokenAmtIn, 0);

    // 2. Maker maliciously updates to empty curve cuts
    OrderConfig memory emptyConfig = orderConfig;
    emptyConfig.curveCuts.lendCurveCuts = new CurveCut[](0);
    emptyConfig.curveCuts.borrowCurveCuts = new CurveCut[](0);
    res.order.updateOrder(emptyConfig, 0, 0);

    // 3. Now all trading operations will fail
    vm.stopPrank();

    // Innocent user tries to trade
    vm.startPrank(address(0x123));
    res.debt.mint(address(0x123), tokenAmtIn);
    res.debt.approve(address(res.order), tokenAmtIn);

    vm.expectRevert();
    res.order.swapExactTokenToToken(res.debt, res.ft, address(0x123), tokenAmtI
    vm.stopPrank();
}
```

Impact Explanation

> - **Denial of Service:** Users cannot execute trades or interact with the order once it's in this state

- **Integration Failures:** Protocols integrating with these orders may make incorrect decisions based on failed APR calculations
- **Liquidity Lock:** User funds could effectively become trapped in the order as normal trading operations would fail
- **Market Manipulation:** Malicious makers can selectively disable orders during favorable market conditions

Recommendation

Solution 1: Add Validation in _updateCurve

Prevent empty curve cuts from being set in the first place:

```
function _updateCurve(CurveCuts memory newCurveCuts) internal {
    if (newCurveCuts.lendCurveCuts.length == 0 || newCurveCuts.borrowCurve
        revert InvalidCurveCuts();
    }
    // existing validation...
}
```

Solution 2: Graceful Handling in Core Functions

Alternatively, modify functions like apr() to handle empty curve cuts gracefully:

```
function apr() external view returns (uint256 lendApr_, uint256 borrowApr_
    if (curveCuts.lendCurveCuts.length == 0 || curveCuts.borrowCurveCuts.l
        return (0, 0);
    }
    // existing logic...
}
```

Recommended Approach: Implement Solution 1 as the primary defense to prevent the issue at its source. This ensures data integrity and prevents the

> contract from entering an invalid state. Solution 2 can be added as a defense-in-depth measure.

Additional Security Considerations

Consider implementing additional safeguards such as:

- Minimum curve cut requirements to ensure meaningful price curves
- Time locks or multi-signature requirements for critical order updates
- Emergency pause functionality for affected orders
- Events emitted when orders are updated to enable monitoring