Contract Name: NonVulnerable_Cross-functionReentrancy_1
Label: Non-Vulnerable
Category: Cross-function Reentrancy
Explanation: Uses CEI even across functions. Safe.

```solidity
pragma solidity ^0.8.0;
contract CrossSafe {
    mapping(address => uint) public balances;
    function trigger() public {
        internalWithdraw();
    }
    function internalWithdraw() internal {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: Vulnerable_MutexLock_2
Label: Vulnerable
Category: Mutex Lock
Explanation: Has no mutex or state protection against reentrancy.

```solidity
pragma solidity ^0.8.0;
contract NoMutex {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: NonVulnerable_MutexLock_3
Label: Non-Vulnerable
Category: Mutex Lock
Explanation: Uses a boolean lock to prevent reentrancy.

```solidity
pragma solidity ^0.8.0;
contract MutexProtected {
    mapping(address => uint) public balances;
    bool private locked;
    modifier noReentrancy() {
        require(!locked);
        locked = true;
```

```
        _;
        locked = false;
    }
    function withdraw() public noReentrancy {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: Vulnerable_CEIPattern_4
Label: Vulnerable
Category: CEI Pattern
Explanation: Violates CEI pattern. External call is made before state update.

```solidity
pragma solidity ^0.8.0;
contract UnsafeCEI {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: NonVulnerable_MutexLock_5
Label: Non-Vulnerable
Category: Mutex Lock
Explanation: Uses a boolean lock to prevent reentrancy.

```solidity
pragma solidity ^0.8.0;
contract MutexProtected {
    mapping(address => uint) public balances;
    bool private locked;
    modifier noReentrancy() {
        require(!locked);
        locked = true;
        _;
        locked = false;
    }
    function withdraw() public noReentrancy {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
```

}
```

# Contract Name: NonVulnerable_Cross-functionReentrancy_6
# Label: Non-Vulnerable
# Category: Cross-function Reentrancy
# Explanation: Uses CEI even across functions. Safe.

```solidity
pragma solidity ^0.8.0;
contract CrossSafe {
    mapping(address => uint) public balances;
    function trigger() public {
        internalWithdraw();
    }
    function internalWithdraw() internal {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: NonVulnerable_Cross-functionReentrancy_7
Label: Non-Vulnerable
Category: Cross-function Reentrancy
Explanation: Uses CEI even across functions. Safe.

```solidity
pragma solidity ^0.8.0;
contract CrossSafe {
    mapping(address => uint) public balances;
    function trigger() public {
        internalWithdraw();
    }
    function internalWithdraw() internal {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: Vulnerable_callvssendvstransfer_8
Label: Vulnerable
Category: call vs send vs transfer
Explanation: Uses .call without CEI. Reentrancy possible.

```solidity
pragma solidity ^0.8.0;
```
```

```
contract UnsafeCall {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = payable(msg.sender).call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: NonVulnerable_OpenZeppelinnonReentrant_9
Label: Non-Vulnerable
Category: OpenZeppelin nonReentrant
Explanation: Uses OpenZeppelin's nonReentrant modifier to block reentrancy.

```solidity
// import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
pragma solidity ^0.8.0;
contract OZProtected {
    mapping(address => uint) public balances;
    bool private locked;
    modifier nonReentrant() {
        require(!locked, "Reentrant call");
        locked = true;
        _;
        locked = false;
    }
    function withdraw() public nonReentrant {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: Vulnerable_Cross-functionReentrancy_10
Label: Vulnerable
Category: Cross-function Reentrancy
Explanation: Withdraw function can be re-entered from another function.

```solidity
pragma solidity ^0.8.0;
contract CrossReentrant {
    mapping(address => uint) public balances;
    function trigger() public {
        withdraw();
    }
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
```

```
        (bool sent, ) = payable(msg.sender).call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: NonVulnerable_OpenZeppelinnonReentrant_11
Label: Non-Vulnerable
Category: OpenZeppelin nonReentrant
Explanation: Uses OpenZeppelin's nonReentrant modifier to block reentrancy.

```solidity
// import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
pragma solidity ^0.8.0;
contract OZProtected {
    mapping(address => uint) public balances;
    bool private locked;
    modifier nonReentrant() {
        require(!locked, "Reentrant call");
        locked = true;
        _;
        locked = false;
    }
    function withdraw() public nonReentrant {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: Vulnerable_MutexLock_12
Label: Vulnerable
Category: Mutex Lock
Explanation: Has no mutex or state protection against reentrancy.

```solidity
pragma solidity ^0.8.0;
contract NoMutex {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: NonVulnerable_OpenZeppelinnonReentrant_13
Label: Non-Vulnerable
Category: OpenZeppelin nonReentrant
Explanation: Uses OpenZeppelin's nonReentrant modifier to block reentrancy.

```solidity
// import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
pragma solidity ^0.8.0;
contract OZProtected {
    mapping(address => uint) public balances;
    bool private locked;
    modifier nonReentrant() {
        require(!locked, "Reentrant call");
        locked = true;
        _;
        locked = false;
    }
    function withdraw() public nonReentrant {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: NonVulnerable_callvssendvstransfer_14
Label: Non-Vulnerable
Category: call vs send vs transfer
Explanation: Uses .send with return check. Follows CEI.

```solidity
pragma solidity ^0.8.0;
contract SafeSend {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        bool success = payable(msg.sender).send(amount);
        require(success);
    }
}
```

Contract Name: NonVulnerable_MutexLock_15
Label: Non-Vulnerable
Category: Mutex Lock
Explanation: Uses a boolean lock to prevent reentrancy.

```solidity
pragma solidity ^0.8.0;
contract MutexProtected {
```

```
    mapping(address => uint) public balances;
    bool private locked;
    modifier noReentrancy() {
        require(!locked);
        locked = true;
        _;
        locked = false;
    }
    function withdraw() public noReentrancy {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: Vulnerable_CEIPattern_16
Label: Vulnerable
Category: CEI Pattern
Explanation: Violates CEI pattern. External call is made before state update.

```solidity
pragma solidity ^0.8.0;
contract UnsafeCEI {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: Vulnerable_MutexLock_17
Label: Vulnerable
Category: Mutex Lock
Explanation: Has no mutex or state protection against reentrancy.

```solidity
pragma solidity ^0.8.0;
contract NoMutex {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

```

# Contract Name: NonVulnerable_callvssendvstransfer_18
# Label: Non-Vulnerable
# Category: call vs send vs transfer
# Explanation: Uses .send with return check. Follows CEI.

```solidity
pragma solidity ^0.8.0;
contract SafeSend {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        bool success = payable(msg.sender).send(amount);
        require(success);
    }
}
```

Contract Name: NonVulnerable_callvssendvstransfer_19
Label: Non-Vulnerable
Category: call vs send vs transfer
Explanation: Uses .send with return check. Follows CEI.

```solidity
pragma solidity ^0.8.0;
contract SafeSend {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        bool success = payable(msg.sender).send(amount);
        require(success);
    }
}
```

Contract Name: NonVulnerable_CEIPattern_20
Label: Non-Vulnerable
Category: CEI Pattern
Explanation: Follows CEI pattern. State is updated before the external call.

```solidity
pragma solidity ^0.8.0;
contract SafeCEI {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
```
```

```
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: NonVulnerable_callvssendvstransfer_21
Label: Non-Vulnerable
Category: call vs send vs transfer
Explanation: Uses .send with return check. Follows CEI.

```solidity
pragma solidity ^0.8.0;
contract SafeSend {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        bool success = payable(msg.sender).send(amount);
        require(success);
    }
}
```

Contract Name: Vulnerable_callvssendvstransfer_22
Label: Vulnerable
Category: call vs send vs transfer
Explanation: Uses .call without CEI. Reentrancy possible.

```solidity
pragma solidity ^0.8.0;
contract UnsafeCall {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = payable(msg.sender).call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: Vulnerable_CEIPattern_23
Label: Vulnerable
Category: CEI Pattern
Explanation: Violates CEI pattern. External call is made before state update.

```solidity
pragma solidity ^0.8.0;
contract UnsafeCEI {
```

```
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: Vulnerable_CEIPattern_24
Label: Vulnerable
Category: CEI Pattern
Explanation: Violates CEI pattern. External call is made before state update.

```solidity
pragma solidity ^0.8.0;
contract UnsafeCEI {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: Vulnerable_CEIPattern_25
Label: Vulnerable
Category: CEI Pattern
Explanation: Violates CEI pattern. External call is made before state update.

```solidity
pragma solidity ^0.8.0;
contract UnsafeCEI {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: Vulnerable_CEIPattern_26
Label: Vulnerable
Category: CEI Pattern

Explanation: Violates CEI pattern. External call is made before state update.

```solidity
pragma solidity ^0.8.0;
contract UnsafeCEI {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: Vulnerable_Cross-functionReentrancy_27
Label: Vulnerable
Category: Cross-function Reentrancy
Explanation: Withdraw function can be re-entered from another function.

```solidity
pragma solidity ^0.8.0;
contract CrossReentrant {
    mapping(address => uint) public balances;
    function trigger() public {
        withdraw();
    }
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = payable(msg.sender).call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: Vulnerable_Cross-functionReentrancy_28
Label: Vulnerable
Category: Cross-function Reentrancy
Explanation: Withdraw function can be re-entered from another function.

```solidity
pragma solidity ^0.8.0;
contract CrossReentrant {
    mapping(address => uint) public balances;
    function trigger() public {
        withdraw();
    }
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
```

```
        (bool sent, ) = payable(msg.sender).call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: NonVulnerable_CEIPattern_29
Label: Non-Vulnerable
Category: CEI Pattern
Explanation: Follows CEI pattern. State is updated before the external call.

```solidity
pragma solidity ^0.8.0;
contract SafeCEI {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: NonVulnerable_CEIPattern_30
Label: Non-Vulnerable
Category: CEI Pattern
Explanation: Follows CEI pattern. State is updated before the external call.

```solidity
pragma solidity ^0.8.0;
contract SafeCEI {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: Vulnerable_Cross-functionReentrancy_31
Label: Vulnerable
Category: Cross-function Reentrancy
Explanation: Withdraw function can be re-entered from another function.

```solidity
pragma solidity ^0.8.0;
contract CrossReentrant {
    mapping(address => uint) public balances;
```

```
    function trigger() public {
        withdraw();
    }
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = payable(msg.sender).call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: Vulnerable_CEIPattern_32
Label: Vulnerable
Category: CEI Pattern
Explanation: Violates CEI pattern. External call is made before state update.

```solidity
pragma solidity ^0.8.0;
contract UnsafeCEI {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: NonVulnerable_Cross-functionReentrancy_33
Label: Non-Vulnerable
Category: Cross-function Reentrancy
Explanation: Uses CEI even across functions. Safe.

```solidity
pragma solidity ^0.8.0;
contract CrossSafe {
    mapping(address => uint) public balances;
    function trigger() public {
        internalWithdraw();
    }
    function internalWithdraw() internal {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: NonVulnerable_CEIPattern_34
Label: Non-Vulnerable
Category: CEI Pattern
Explanation: Follows CEI pattern. State is updated before the external call.

```solidity
pragma solidity ^0.8.0;
contract SafeCEI {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: Vulnerable_Cross-functionReentrancy_35
Label: Vulnerable
Category: Cross-function Reentrancy
Explanation: Withdraw function can be re-entered from another function.

```solidity
pragma solidity ^0.8.0;
contract CrossReentrant {
    mapping(address => uint) public balances;
    function trigger() public {
        withdraw();
    }
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = payable(msg.sender).call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: NonVulnerable_Cross-functionReentrancy_36
Label: Non-Vulnerable
Category: Cross-function Reentrancy
Explanation: Uses CEI even across functions. Safe.

```solidity
pragma solidity ^0.8.0;
contract CrossSafe {
    mapping(address => uint) public balances;
    function trigger() public {
        internalWithdraw();
    }
    function internalWithdraw() internal {
```

```
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: Vulnerable_callvssendvstransfer_37
Label: Vulnerable
Category: call vs send vs transfer
Explanation: Uses .call without CEI. Reentrancy possible.

```solidity
pragma solidity ^0.8.0;
contract UnsafeCall {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = payable(msg.sender).call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: Vulnerable_Cross-functionReentrancy_38
Label: Vulnerable
Category: Cross-function Reentrancy
Explanation: Withdraw function can be re-entered from another function.

```solidity
pragma solidity ^0.8.0;
contract CrossReentrant {
    mapping(address => uint) public balances;
    function trigger() public {
        withdraw();
    }
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = payable(msg.sender).call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: NonVulnerable_MutexLock_39
Label: Non-Vulnerable
Category: Mutex Lock

Explanation: Uses a boolean lock to prevent reentrancy.

```solidity
pragma solidity ^0.8.0;
contract MutexProtected {
    mapping(address => uint) public balances;
    bool private locked;
    modifier noReentrancy() {
        require(!locked);
        locked = true;
        _;
        locked = false;
    }
    function withdraw() public noReentrancy {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: NonVulnerable_Cross-functionReentrancy_40
Label: Non-Vulnerable
Category: Cross-function Reentrancy
Explanation: Uses CEI even across functions. Safe.

```solidity
pragma solidity ^0.8.0;
contract CrossSafe {
    mapping(address => uint) public balances;
    function trigger() public {
        internalWithdraw();
    }
    function internalWithdraw() internal {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: NonVulnerable_OpenZeppelinnonReentrant_41
Label: Non-Vulnerable
Category: OpenZeppelin nonReentrant
Explanation: Uses OpenZeppelin's nonReentrant modifier to block reentrancy.

```solidity
// import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
pragma solidity ^0.8.0;
contract OZProtected {
    mapping(address => uint) public balances;
```

```
    bool private locked;
    modifier nonReentrant() {
        require(!locked, "Reentrant call");
        locked = true;
        _;
        locked = false;
    }
    function withdraw() public nonReentrant {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: NonVulnerable_CEIPattern_42
Label: Non-Vulnerable
Category: CEI Pattern
Explanation: Follows CEI pattern. State is updated before the external call.

```solidity
pragma solidity ^0.8.0;
contract SafeCEI {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: NonVulnerable_MutexLock_43
Label: Non-Vulnerable
Category: Mutex Lock
Explanation: Uses a boolean lock to prevent reentrancy.

```solidity
pragma solidity ^0.8.0;
contract MutexProtected {
    mapping(address => uint) public balances;
    bool private locked;
    modifier noReentrancy() {
        require(!locked);
        locked = true;
        _;
        locked = false;
    }
    function withdraw() public noReentrancy {
        uint amount = balances[msg.sender];
        require(amount > 0);
```

```
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: NonVulnerable_OpenZeppelinnonReentrant_44
Label: Non-Vulnerable
Category: OpenZeppelin nonReentrant
Explanation: Uses OpenZeppelin's nonReentrant modifier to block reentrancy.

```solidity
// import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
pragma solidity ^0.8.0;
contract OZProtected {
    mapping(address => uint) public balances;
    bool private locked;
    modifier nonReentrant() {
        require(!locked, "Reentrant call");
        locked = true;
        _;
        locked = false;
    }
    function withdraw() public nonReentrant {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: NonVulnerable_callvssendvstransfer_45
Label: Non-Vulnerable
Category: call vs send vs transfer
Explanation: Uses .send with return check. Follows CEI.

```solidity
pragma solidity ^0.8.0;
contract SafeSend {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        bool success = payable(msg.sender).send(amount);
        require(success);
    }
}
```

Contract Name: Vulnerable_callvssendvstransfer_46

Label: Vulnerable
Category: call vs send vs transfer
Explanation: Uses .call without CEI. Reentrancy possible.

```solidity
pragma solidity ^0.8.0;
contract UnsafeCall {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = payable(msg.sender).call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: NonVulnerable_Cross-functionReentrancy_47
Label: Non-Vulnerable
Category: Cross-function Reentrancy
Explanation: Uses CEI even across functions. Safe.

```solidity
pragma solidity ^0.8.0;
contract CrossSafe {
    mapping(address => uint) public balances;
    function trigger() public {
        internalWithdraw();
    }
    function internalWithdraw() internal {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Contract Name: Vulnerable_callvssendvstransfer_48
Label: Vulnerable
Category: call vs send vs transfer
Explanation: Uses .call without CEI. Reentrancy possible.

```solidity
pragma solidity ^0.8.0;
contract UnsafeCall {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = payable(msg.sender).call{value: amount}("");
        require(sent);
```

```
        balances[msg.sender] = 0;
    }
}
```

Contract Name: Vulnerable_MutexLock_49
Label: Vulnerable
Category: Mutex Lock
Explanation: Has no mutex or state protection against reentrancy.

```solidity
pragma solidity ^0.8.0;
contract NoMutex {
    mapping(address => uint) public balances;
    function withdraw() public {
        uint amount = balances[msg.sender];
        require(amount > 0);
        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent);
        balances[msg.sender] = 0;
    }
}
```

Contract Name: NonVulnerable_Cross-functionReentrancy_50
Label: Non-Vulnerable
Category: Cross-function Reentrancy
Explanation: Uses CEI even across functions. Safe.

```solidity
pragma solidity ^0.8.0;
contract CrossSafe {
    mapping(address => uint) public balances;
    function trigger() public {
        internalWithdraw();
    }
    function internalWithdraw() internal {
        uint amount = balances[msg.sender];
        require(amount > 0);
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

What is Reentrancy?

A reentrancy attack occurs when a malicious contract recursively calls back into a vulnerable function before the original invocation completes, draining funds or manipulating state.

Historical Incident: The DAO Hack (2016)

Exploited reentrancy in withdraw() functions.
Resulted in a $60M loss and Ethereum hard fork.