Problem 1

i) Vulnerabilities within the contract:

1. Re-entrancy-eth:
    a. The withdraw() function is vulnerable to re-entrancy-eth attack. An attacker can exploit this vulnerability to drain all user funds from the contract.
2. Unchecked ownership transfer
    a. The transferOwnership() function allows null address or address(0) to be set as owner of the contract. If null address is set as the owner of the contract , it can neither send nor receive transactions.
3. Missing Events
    a. For deposit(), withdraw() and transferOwnership() function , whenever the contract state is updated , no event is emitted. Without these logs, users or platform cannot monitor the state updation.

ii) Updated Contract

Solutions

Solution 1 : https://github.com/ArpanManna/Lightcurve/blob/main/coding1/answer1.sol

Solution 2 : https://github.com/ArpanManna/Lightcurve/blob/main/coding1/answer2.sol
Solution 3: https://github.com/ArpanManna/Lightcurve/blob/main/coding1/answer3.sol

1. Re-entrancy-eth:
    a. Update contract state before ether transfer

```
balances[msg.sender] - = amount;
(bool success, ) = msg.sender.call{value: amount}("");
```

    b. Make a modifier nonReentrant()

```
modifier nonReentrant{
    require(!locked, "No reentrancy");
    locked = true;
    _;
    locked = false;
}
```

    Use as - function withdraw(uint256 amount) public **nonReentrant**

    c. Use Openzeppelin Reentrancy Guard modifier for access control to withdraw() function

        i. https://docs.openzeppelin.com/contracts/4.x/api/security#ReentrancyGuard
    d. We can use **transfer()** instead of low level **call()**, but there are few drawbacks
        i. Receiving contract should have a fallback() function
        ii. It has an gas limit of 2300 gas

```
function withdraw(uint256 amount) public nonReentrant{
    require(balances[msg.sender] >= amount, "Insufficient balance");
    require(amount > 0, "Transfer amount is too low!");
    balances[msg.sender] -= amount;
    payable(msg.sender).transfer(amount);
    emit Withdraw(msg.sender, amount);
}
```

2. Unchecked ownership transfer
    a. Add a check in transferOwnership() to validate the new owner is not Null address.

```
function transferOwnership(address newOwner) public  onlyOwner{
    require(newOwner != address(0), "Null address cannot be owner");
    owner = newOwner;
    emit OwnerUpdated(owner, newOwner);
}
```

3. Missing Events
    a. Add events for Deposit, Withdraw and OwnerUpdated, when state updation is occurred

```
event Deposit(address indexed depositor, uint amount);
event Wihdraw(address indexed addr, uint amount);
event OwnerUpdated(address indexed oldOwner, address indexed newOwner);
```