Lottery Contract

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.6.0;
// WARNING THIS CODE IS AWFUL, NEVER DO ANYTHING LIKE THIS
contract Oracle{
        uint8 private seed; // Hide seed value!!
        constructor (uint8 _seed) public {
                seed = _seed;
        }

        function getRandomNumber() external returns (uint256){
                return block.number % seed;
        }

}

// WARNING THIS CODE IS AWFUL, NEVER DO ANYTHING LIKE THIS

contract Lottery {

        struct Team {
                string name;
                string password;
                uint256 points;
        }
    struct LotteryDetails {
        uint endTime;
        uint seed;
    }

        address public owner;
        mapping(address => bool) public admins;

        Oracle private oracle;
        LotteryDetails public thisLottery;


        // public keyword (!!!)
        mapping(address => Team) public teams;
        address [] public teamAddresses;

        event LogTeamRegistered(string name);
        event LogGuessMade(address teamAddress);
        event LogTeamCorrectGuess(string name);
        event LogAddressPaid(address sender, uint256 amount);
        event LogResetOracle(uint8 _newSeed);

        modifier onlyOwner(){
                if (msg.sender==owner) {
                        _;
                }
        }

        modifier onlyAdmins() {
                require (admins[msg.sender]);
```

```
            _;
    }

    modifier needsReset() {
            if (teamAddresses.length > 0) {
                    delete teamAddresses;
            }
            _;
    }


    // Constructor - set the owner of the contract
    constructor() public {
            owner = msg.sender;
            admins[msg.sender] = true;
            admins[0x0e11fe90bC6AA82fc316Cb58683266Ff0d005e12] = true
            admins[0x7F65E7A5079Ed0A4469Cbd4429A616238DCb0985] = true
            admins[0x142563a96D55A57E7003F82a05f2f1FEe420cf98] = true
            admins[0x52faCd14353E4F9926E0cf6eeAC71bc6770267B8] = true
    }

    // initialise the oracle and lottery end time
    function initialiseLottery(uint8 seed)
external onlyAdmins needsReset{
            oracle = new Oracle(seed);
            uint endTime = block.timestamp + 7 days;
            teams[address(0)] = Team("Default Team", "Password", 5);
            teamAddresses.push(address(0));
    }

    // reset the lottery
    function reset(uint8 _newSeed) public view {
        uint endTime = block.timestamp + 7 days;
        LotteryDetails memory thisLottery =
    LotteryDetails({endTime : endTime, seed : _newSeed});
    }

    // register a team
    function registerTeam(address _walletAddress,
string calldata _teamName,
string calldata _password) external payable {
            // 2 ether deposit to register a team
            require(msg.value == 2 ether);
            // add to mapping as well as another array
            teams[_walletAddress] = Team(_teamName, _password, 5);
            teamAddresses.push(_walletAddress);
            emit LogTeamRegistered(_teamName);
    }

    // make your guess , return a success flag
    function makeAGuess(address _team,uint256 _guess) external
returns (bool) {
            // no checks for team being registered (???)
            emit LogGuessMade(_team);
            // get a random number
```

~~~
                uint256 random = oracle.getRandomNumber();
                if(random==_guess){
                        // give 100 points
                        teams[_team].points = 100;
                        emit LogTeamCorrectGuess(teams[_team].name);
                return true;
                }
                else{
                        // take away a point (!!!)
                    teams[_team].points -= 1;
                        return false;
                }
        }

        // once the lottery has finished pay out the best teams
        function payoutWinningTeam() external returns (bool) {

    // if you are a winning team you get paid double the deposit (4 ether
    for (uint ii=0; ii<teamAddresses.length; ii++) {
        if (teams[teamAddresses[ii]].points>=100) {
            // no gas limit on value transfer call (!!!)
            (bool sent ,)  = teamAddresses[ii].call.value(4 ether)("");
            teams[teamAddresses[ii]].points = 0;
            return sent;
        }
    }
        }

        function getTeamCount() public view returns (uint256){
                return teamAddresses.length;
        }

        function getTeamDetails(uint256 _num) public view
    returns(string memory ,address,uint256){
                Team memory team = teams[teamAddresses[_num]];
                return(team.name,teamAddresses[_num],team.points);
        }

        function resetOracle(uint8 _newSeed) internal {
            oracle = new Oracle(_newSeed);
        }

        // catch any ether sent to the contract
        fallback() external payable {
                emit LogAddressPaid(msg.sender,msg.value);
        }

        function addAdmin(address _adminAddress) public onlyAdmins {
                admins[_adminAddress] = true;
        }

}```
~~~