Security & Auditing

```
contract BadRNG {
    address payable[] private s_players;

    function enterRaffle() external payable {
        require(msg.value >= 10000000000000000000);
        s_players.push(payable(msg.sender));
    }

    function pickWinner() external {
        uint256 randomWinnerIndex = uint256(
            keccak256(abi.encodePacked(block.difficulty, msg.sender))
        );
        address winner = s_players[randomWinnerIndex % s_players.length];
        (bool success, ) = winner.call{value: address(this).balance}("");
        require(success, "Transfer failed");
    }
}
```

```
contract MetamorphicContract is Initializable {
    address payable owner;

    function kill() external {
        require(msg.sender == owner);
        selfdestruct(owner);
    }
}
```

```
contract EtherStore {
    mapping(address => uint256) public balances;

    function deposit() external payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw() external {
        uint256 balance = balances[msg.sender];
        require(balance > 0);
        (bool success, ) = msg.sender.call{value: balance}("");
        require(success, "Failed to send Ether");
        balances[msg.sender] = 0;
    }

    // Helper function to check the balance of this contract
    function getBalance() external view returns (uint256) {
        return address(this).balance;
    }
}
```

```
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract LiquidityPoolAsOracle {
    address public s_token1;
    address public s_token2;

    constructor(address token1, address token2) {
        require(token1 != address(0x0), "Address cannot be 0");
        require(token2 != address(0x0), "Address cannot be 0");
        s_token1 = token1;
        s_token2 = token2;
    }

    function swap(
        address from,
        address to,
        uint256 amount
    ) external {
        require(
            (from == s_token1 && to == s_token2) || (from == s_token2 && to == s_token1),
            "Invalid tokens"
        );
        require(IERC20(from).balanceOf(msg.sender) >= amount, "Not enough to swap");
        uint256 swap_amount = getSwapPrice(from, to, amount);
        bool txFromSuccess = IERC20(from).transferFrom(msg.sender, address(this), amount);
        require(txFromSuccess, "Failed to transfer from");
        bool txToSuccess = IERC20(to).transfer(msg.sender, swap_amount);
        require(txToSuccess, "Failed to transfer to");
    }

    function addLiquidity(address tokenAddress, uint256 amount) external {
        bool success = IERC20(tokenAddress).transferFrom(msg.sender, address(this), amount);
        require(success, "Failed to add liquidity");
    }
```

```solidity
contract Vault {
    bool public s_locked;
    bytes32 private s_password;

    constructor(bytes32 password) {
        s_locked = true;
        s_password = password;
    }

    function unlock(bytes32 password) external {
        if (s_password == password) {
            s_locked = false;
        }
    }
}
```

```solidity
import "../../Vault.sol";

contract VaultFuzzTest is Vault {
    constructor() Vault("123asd123") {}

    function echidna_test_find_password() public view returns (bool) {
        return s_locked == true;
    }
}
```