Staker

앞서 이 문제는 blockchain 분야이다.

LpToken.soi, Setup.soi, StakingManager.soi, Tokem.soi로 총 4개의 파일이 주어진다.

soi라는 확장자는 솔리디티(Solidity) 파일로 이더리움 블록체인에서 스마트 계약을 개발하기 위한 언어이다.

*스마트 계약: 자동화된 계약으로, 조건에 따라 자동으로 실행되는 코드

우선 Notepad++로 파일을 열어 코드를 분석해보려고 한다.

1. LpToken.soi

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.25;

import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract LpToken is ERC20 {
    address immutable minter;

    constructor() ERC20("LP Token", "LP") {
        minter = msg.sender;
    }

    function mint(address to, uint256 amount) external {
        require(msg.sender == minter, "only minter");
        _mint(to, amount);
    }

    function burnFrom(address from, uint256 amount) external {
        _burn(from, amount);
    }
}
```

솔리디티로 작성된 스마트 계약 코드이다.

1) UNLICENSED로 특정 라이선스가 부여되지 않았음을 알 수 있다.

2) 솔리디티 코드가 0.8.25 버전 이상의 솔리디티 컴파일러로 컴파일 되어야 한다는 것을 뜻한다.

4) "@open~sol" 경로에서 OpenZeppelin 라이브러리에서 제공되는 ERC20 토큰 컨트랙트를 가져온다.

6. LpToken이라는 이름의 스마트 계약이 ERC20 토큰 컨트랙트를 상속한다.

9. 생성자 함수는 스마트 계약이 배포될 때 호출된다. 이는 컨트랙트를 초기화한다. LP Token 컨트랙트의 생성자 함수는 "LP Token"과 "LP"를 이름과 심볼로 하는 ERC20 토큰을 생성한다.

13. mint 함수는 특정 주소에 새로운 토큰을 발행한다. 이 함수는 오직 생성자가 정의한 주소인 minter만 호출할 수 있다.

18. burnFrom함수는 특정 주소의 토큰을 소각한다. 이 함수는 스마트 계약의 보유자가 호출할 수 있다.

즉, 이 코드는 ERC20 토큰을 생성하고 발행하는 간단한 스마트 계약을 정의한다고 볼 수 있다.

2. Setup.soi

```
1  // SPDX-License-Identifier: UNLICENSED
2  pragma solidity ^0.8.25;
3
4  import {Token} from "./Token.sol";
5  import {LpToken} from "./LpToken.sol";
6  import {StakingManager} from "./StakingManager.sol";
7
8  contract Setup {
9      StakingManager public stakingManager;
10     Token public token;
11
12     constructor() payable {
13         token = new Token();
14         stakingManager = new StakingManager(address(token));
15
16         token.transfer(address(stakingManager), 86400 * 1e18);
17
18         token.approve(address(stakingManager), 100000 * 1e18);
19         stakingManager.stake(100000 * 1e18);
20     }
21
22     function withdraw() external {
23         token.transfer(msg.sender, token.balanceOf(address(this)));
24     }
25
26     function isSolved() public view returns (bool) {
27         return token.balanceOf(address(this)) >= 10 * 1e18;
28     }
29 }
```

4~6) Token.soi, LpToken.soi, StakingManager.soi 파일에서 Token, LpToken, StakingManager 컨트랙트를 가져온다.

8~20) Setup이라는 이름의 스마트 계약을 정의한다. 이 계약은 StakingManger와 Token을 초기화하고 관리한다.

12) 생성자 함수, Token 및 StakingManager 인스턴스를 생성하고, 일정량의 토큰을 StakingManager에 전송하고 stake를 수행한다.

22~24) withdraw 함수는 스마트 계약에 보유된 모든 토큰을 호출자에게 반환한다.

26~28) isSolved 함수는 스마트 계약에 보유된 토큰이 일정량 이상인지를 확인하여, 문제가 해결되었는지 여부를 반환한다.

3. StakingManager.soi

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.25;

import {LpToken} from "./LpToken.sol";
import {IERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract StakingManager {
    uint256 constant REWARD_PER_SECOND = 1e18;

    IERC20 public immutable TOKEN;
    LpToken public immutable LPTOKEN;

    uint256 lastUpdateTimestamp;
    uint256 rewardPerToken;

    struct UserInfo {
        uint256 staked;
        uint256 debt;
    }

    mapping(address => UserInfo) public userInfo;

    constructor(address token) {
        TOKEN = IERC20(token);
        LPTOKEN = new LpToken();
    }

    function update() internal {
        if (lastUpdateTimestamp == 0) {
            lastUpdateTimestamp = block.timestamp;
            return;
        }

        uint256 totalStaked = LPTOKEN.totalSupply();
        if (totalStaked > 0 && lastUpdateTimestamp != block.timestamp) {
            rewardPerToken = (block.timestamp - lastUpdateTimestamp) * REWARD_PER_SECOND * 1e18 / totalStaked;
            lastUpdateTimestamp = block.timestamp;
        }
    }

    function stake(uint256 amount) external {
        update();

        UserInfo storage user = userInfo[msg.sender];

        user.staked += amount;
        user.debt += (amount * rewardPerToken) / 1e18;

        LPTOKEN.mint(msg.sender, amount);
        TOKEN.transferFrom(msg.sender, address(this), amount);
    }

    function unstakeAll() external {
        update();

        UserInfo storage user = userInfo[msg.sender];

        uint256 staked = user.staked;
        uint256 reward = (staked * rewardPerToken / 1e18) - user.debt;
        user.staked = 0;
        user.debt = 0;

        LPTOKEN.burnFrom(msg.sender, LPTOKEN.balanceOf(msg.sender));
        TOKEN.transfer(msg.sender, staked + reward);
    }
}
```

4) LpToken 컨트랙트를 가져온다.

5) IER20 인터페이스를 가져온다.

7) StakingManager 스마트 계약을 정의한다.

8) 초당 보상 비율을 상수로 정의한다.

10~11) 스테이킹[1]에 사용되는 토큰과 유동성 토큰인 LP Token을 저장하는 불변 변수이다.

13~14) 마지막 업데이트 시간과 토큰당 보상 비율을 나타내는 상태 변수이다.

16) 사용자 정보를 저장하는 구조체이다.

17) 사용자가 스테이킹한 금액이다.

18) 사용자가 받을 보상에서 차감할 금액이다.

21) 각 사용자의 주소를 UserInfo 구조체와 매핑하여 사용자별 스테이킹 정보를 저장한다.

23) 토큰 주소를 받아 초기화하는 생성자 함수이다.

24) 스테이킹에 사용되는 토큰을 설정한다.

25) 새로운 LP 토큰 인스턴스를 생성한다.

28) 보상을 업데이트하는 내부 함수이다.

30) 처음 업데이트 시에 현재 시간을 기록한다.

34) 전체 스테이킹된 토큰의 양을 나타낸다.

35) 토큰당 보상 비율을 계산한다.

37) 업데이트 시간을 갱신한다.

41) 사용자가 토큰을 스테이킹하는 스테이킹 함수이다.

42) 보상을 업데이트 한다.

44) 사용자 정보를 로드한다.

46) 스테이킹 금액이 증가한다.

47) 부채가 증가한다.

49) 사용자에게 LP 토큰을 발행한다.

50) 사용자로부터 토큰을 이 계약으로 전송한다.

53) 사용자가 모든 토큰을 언스테이킹하는 언스테이킹 함수이다.

---

[1] 가상화폐를 블록체인에 예치하고 이에 대한 보상을 지급 받는 것

54) 보상을 업데이트 한다.

56) 사용자 정보를 로드한다.

58) 스테이킹한 금액이다.

59) 보상을 계산한다.

60) 스테이킹 금액을 초기화한다.

61) 부채를 초기화한다.

63) 사용자의 모든 LP 토큰을 소각한다.

64) 사용자에게 스테이킹한 금액과 보상을 전송한다.

4. Token.soi

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.25;

import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract Token is ERC20 {
    constructor() ERC20("Token", "TKN") {
        _mint(msg.sender, 186401 * 1e18);
    }
}
```

1) SPEX 라이선스 식별자는 코드의 라이선스 정보를 명시한다. UNLICENSED로 설정되어 있음을 볼 수 있다.

2) Solidity 0.8.25 버전 이상의 컴파일러를 필요로 한다.

4) OpenZepplin 라이브러리에서 ERC20 토큰 표준 구현을 가져온다. ERC20 표준 인터페이스와 기능을 제공한다.

6) Token이라는 스마트 계약을 정의한다. OpenZeppling의 ERC20 표준을 상속 받는다.

7) 스마트 계약이 배포될 때 호출되는 생성자 함수로, ERC20 생성자에 'TOKEN'이라는 이름과 'TKN'이라는 심볼을 전달한다.

8) 계약 배포자에게 186401 토큰을 발행한다. 각 토큰은 18자리 소수점 자리를 갖는다.

블록체인을 잘 모르지만, CTF 문제를 풀며 학습할 수 있지 않을까 생각했다. 하지만 쉽게 감을 잡기가 어려워, 대부분의 블록체인 CTF 문제가 어떤 형식으로 Flag를 도출하는지 구글링 해보았다.

1. Solidity 파일의 코드를 주의 검토하는 것이 중요하다. 제공되는 파일에 취약점이나 해결 단서가 포함되어 있다고 한다.
2. 함수 목록을 작성하여 주요 함수, 변수, 접근 제어를 나열해본다.
3. 스마트 계약의 주요 논리 흐름과 상태 변화를 이해한다.
4. 일반적으로 재진입 공격, 오버/언더플로우, 접근 제어 오류 등의 취약점을 찾는다.

IDE, 테스트넷(Ropsten, Rinkeby), Truffle, Hardhat, Remix IDE 디버거 등의 툴을 자주 사용하는 듯하다.

Setup.sot 파일에 토큰의 조건이 있는 것 같아 우선적으로 살펴보았다.

```
token.transfer(address(stakingManager), 86400 * 1e18);

token.approve(address(stakingManager), 100000 * 1e18);
stakingManager.stake(100000 * 1e18);
```

token은 ERC20 토큰의 인스턴스이고, stakingMangager은 StakingManager의 인스턴스이다.

token 계정에서 stakingManager 계정으로 86400개의 토큰을 이체한다.

token 계정이 stakingManager에게 100,000개의 토큰을 스테이킹하기 위해 승인한다. 그리고 stakingManager,stake 함수를 호출하여 100,000개의 토큰을 스테이킹한다

즉, Setup.sol 파일에서 100,000개의 토큰이 생성된 다음 withdraw 함수가 1개의 토큰을 인출할 수 있도록 한다. 그 후, isSolved 함수에서 토큰이 10개 이상 있다면 true를 반환한다.

정확히는 잘 모르겠지만 Setup.sol 파일에 기반하여 조건들을 충족하면 flag를 찾을 수 있지 않을까 생각한다.

StakingManager.sol 파일에서 stake 함수를 통해서 토큰을 스테이킹 한 후, 1분(60초)를 기다린 후 unstakeAll 함수를 호출하여 언스테이킹을 하면 보상을 받을 수 있는 것으로 보인다.

전체적으로 'stake 함수로 토큰을 스테이킹 하고 unstakeAll 함수로 언스테이킹을 한다.' 의 과정을 거치면 보상으로 토큰이 1개 주어지는 것 같다. 그렇게 토큰을 10개 이상 모으면 true를 반환하며 Flag를 확인할 수 있을 것으로 보인다.

프로그램이 계속 돌아간다면 60초마다 토큰이 모여야 정상인데, 어느 부분에서 문제가 생긴지 아직은 잘 모르겠다. 알지 못하는 분야에 대해 무모하게 도전한 감이 없지 않지만, 생각보다 다양한 정보와 블록체인 관련 단어들을 알 수 있는 시간이었다. 비록 답을 찾아내진 못하였으나, 앞으로 얻게 되는 새로운 지식이나 힌트 속에서 Flag를 찾는다면 Write-up을 갱신할 예정이다.