Heuristic Wave Follow
Oct 7, 2018 · 6 min read

Ethernaut Recovery Problem — 이더넛 18단계 문제 해설

문제 해설에 들어가기 전, 이더넛 내에서 콘솔창과 상호작용을 할 줄 알고 기본적인 리믹스 및 메타마스크 사용법이 숙지되어 있다는 가정 하에 해설을 진행합니다. 필자의 풀이 방법이 절대적은 풀이 방법은 아니므로 이 점 참고하시기 바랍니다.

The Ethernaut

by

Heuristic Wave

. . .

Recovery Problem

이번에도 역시 주어진 조건을 읽어보자. 필자의 초월해석이 담겨 있기 때문에, 원문을 직접 읽는 것이 제일 좋다.

계약 생성자는 아주 간단한 토큰팩토리 컨트랙트를 만들었다. 누구나 쉽게 새 토큰을 만들 수 있습니다. 첫 번째 토큰 계약을 배포한 후, 생성자는 더 많은 토큰을 얻기위해 0.5eth를 더 보냈습니다. 그 이후, 그들은 CA주소를 잃어버렸다. 이번 단계에서는 잃어버린 CA주소에 있는 0.5eth를 회복하면 성공이다.

11 ~ 14 문제까지 너무 어려워 고생을 했는데, 앞선 문제점을 해결할 실력이라면 이번 문제도 수월하게 해결할 수 있다. 평상시에 Etherscan에 기록되는 트랜잭션 결과들을 눈여겨 보았더라면 바로 문제를 해결 할 수 있다.

코드 분석

SimpleToken.sol

토큰을 만들고 원하는 계좌에 전송하는 기능이 있는 컨트랙트

```
contract SimpleToken {
  string public name; // public variables
  // 계좌 주소당 얼마만큼의 balances 토큰량이 있는지 맵핑
  mapping (address => uint) public balances;

  // constructor, 이름과 생성자의 주소와 초기생산량를 매개변수로 하여 생성
  constructor(string _name, address _creator, uint256
_initialSupply) public {
    name = _name;
    balances[_creator] = _initialSupply;
  }

  // collect ether in return for tokens, 이더를 보내면 송금액
(msg.value) 10배의 양에 해당하는 토큰을 줌
  function() public payable {
    balances[msg.sender] = msg.value*10;
  }

  // allow transfers of tokens, 토큰을 전송하는 함수
  function transfer(address _to, uint _amount) public {
    require(balances[msg.sender] >= _amount); // 토큰량보다 뽑아
가려는게 많으면 불가
    balances[msg.sender] -= _amount; // 총량에서 뽑아가는량 만큼 빼
기
    balances[_to] = _amount; // _to의 주소에 _amount 만큼의 양을
기록함
  }

  // clean up after ourselves, selfdestrut, 계약을 파기시키고 _to
주소로 돈을 보냄
  function destroy(address _to) public {
    selfdestruct(_to);
  }
}
```

> Self-destruct Document
>
> 코드를 블록 체인에서 제거 시킬 수 있는 유일한 방법이다. 계좌에 남아있는 ETH는 지정

> 된 traget에 보내지고 코드가 제거된다.

우리는 selfdestruct의 기능을 통해서 잃어버린 계좌로 보내진 ETH에 대한 계약을 파기시켜서 돈을 찾아오면 이번 문제를 해결 할 수 있다.

문제 풀이 과정

Etherscan(ROPSTEN)을 통해서 거래과정을 추적할 수 있다.

우선적으로, 발급받은 문제인 Instance Address를 이더스캔에 조회해보자!

위 사진과 같은 그림이 나오면, `Internal Txns`에서 트랜잭션 내역을 볼 수 있는데 `Contract Creation`을 통해 새로운 컨트랙트가 생겼다는 사실을 알 수 있다. 링크를 타고 들어가면 문제주소가 만든 SimpleToken 컨트랙트(아래 사진)를 볼 수 있다.

SimpleToken에서 트랜잭션 내역을 살펴보면 0x10ff~ 로 시작하는 Instance 주소로부터 컨트랙트가 만들어지고 Level Address 로 부터 SimpleToken의 주소로 0.5eth를 보낸 내역을 확인 할 수 있다.

다시 한번 문제를 정리해보자면, 문제의 주소(Instance Address)가 SimpleToken을 만들고 이 주소에 출제자(Level Address)가 0.5eth를 보냈다. 그런데, 출제자는 돈을 보낸 주소를 잃어버렸다. 우리는 SimpleToken CA주소에 묶여 있는 ETH를 되찾아와야 한다.

그렇기 때문에 우리는 리믹스에서 At Address에 이더스캔에서 취득한 SimpleToken의 주소를 불러오자.

이후, destroy 함수의 매개변수에 이 계약을 파기시키기 전 ETH을 보낼 나의 주소를 넣고 함수를 호출하면 0.5ETH를 되찾고 이번 단계를 해결하게 된다.

마지막으로 컨트랙트를 파기하고 나서 이더스캔에서 조회해보면 Code가 파기되었다는 기록이 위와 같이 존재하게 된다.

문제를 풀고나면, 'CA 주소는 EOA 주소와 nonce 값으로 부터 만들어 지기 때문에, nonce를 잃어버린다면 영원히 찾을 수 없다.'는 교훈을 보여준다. (사실 트랜잭션을 통해 만들어진 주소는 이더스캔뿐만 아니라, 자체적인 연산으로 알아낼 수 있는 방법이 이더리움 황서에 기록 되어 있다.)이번 문제에서는 체험한 Self Destruct에 대한 기능과 EtherScan을 읽을 수 있는 능력을 숙지하고 다음 단계로 넘어가자!

그럼, 다음번에는 19단계 MagicNumber에서 만나요!