Heuristic Wave Follow
Feb 6, 2019 · 11 min read

Ethernaut MagicNumber Problem — 이더넛 19단계 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

MagicNumber Problem

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

이번 문제를 풀기위해서는, `whatIsTheMeaningOfLife()`에 대한 올바른 숫자를 주는 `Solver`를 만드는 것이다. 솔버의 코드는 매우 단순(tiny)하다. 최대 10개의 opcode에 해당하는 매우 작은 크기이다.

힌트 : 아마도 잠시동안 솔리디티 컴파일러의 편리함을 냅두고, 손으로 빌드를 해야 할 것이다. 그래 Raw EVM 바이트코드다!

사실 19번 문제는 약 5개월 전에 해결하였다. 그러나, 22번까지의 해설을 마치고 19번의 해설을 포스팅하는 이유는 그동안 이더리움을 깊게 다루면서 19번문제의 메시지가 너무 중요하다는 사실에..... 내방식대로 쓰려던 해설을 포기하고, 출제자 포스팅을 기반으로 하여 해설을 진행하겠다. 한번에 19번의 출제자 의도를 완벽히 이해하기는 어렵지만, 이

해를 한다면 향후 이더리움을 깊게 다룰때 rawTransaction을 만들기 훨씬 수월 할 것이다.

코드 분석

MagicNum.sol

이번문제는 코드분석이라고 소제목을 작성하였지만, 딱히 설명할 내용이 없다. solver의 CA주소만 넣으면 해결할 수 있는 컨트랙트 (42가 MagicNumber인 것은 안비밀!)

```
pragma solidity ^0.4.24;

contract MagicNum {
  address public solver;    // 솔버의 CA 주소로 예상
  constructor() public {}

  function setSolver(address _solver) public {
    solver = _solver;
  }
  /*
    ____________/\\\_______/\\\\\\\\\_____
     __________/\\\\\_____/\\\///////\\\___
      ________/\\\/\\\____\///______\//\\\__
       ______/\\\/\/\\\______________/\\\/___
        ____/\\\/__\/\\\___________/\\\//_____
         __/\\\\\\\\\\\\\\\_____/\\\//________
          _\///////////\\\//____/\\\/__________
            __________\/\\\_____/\\\\\\\\\\\\\\\_
             __________\///_____\///////////////__
  */
}
```

문제에서 주석으로 처리된 42가 solver가된다!!

그렇다면 우리는 어떻게 42라는 값을 줄 수 있을까? 일단 solver를 주소로 받으니 새로운 컨트랙트가 필요하다! 또한 힌트에서 컴파일러가아닌 EVM 바이트코드를 활용하라고 했다. 14번 어셈블리문제에서 간접적으로 공부한 지식이 떠올랐다면, 그대로 실행에 옮기면 된다!

19번 문제의 출제자가 이번 문제에서 독자들에게 가장 알려주고 싶은 정보는 바로 이것

이다.

출제자의 포스팅을 통해 공부를 하면 더욱 좋다! 만약 이해가 잘 안된다면, 아래 내용을 반드시 이해하고 문제 풀이 과정으로 넘어가자!!

컨트랙트가 만들어 지는 과정

1. user가 고수준의 언어(Solidity Code)로 코드를 작성한다.
2. EVM 컴파일러가 컨트랙트의 코드를 저수준으로(byteCode)로 변경한다.

- 컨트랙트를 생성하는 바이트코드는 1) 초기화 코드와 2) 런타임 코드를 순차적으로 연결한다.

3. 바이트코드가 stack에 적재된다. 이때, 바이트 코드는 initialization 부분과 execution 로직 부분으로 나뉜다.

3.1 initialization 부분

컨트랙트가 생성되는 동안에, EVM은 스택에서 첫 번째 STOP 또는 RETURN 명령에 도달 할 때까지 초기화 코드 만 실행한다. 이 단계에서 컨트랙트의 constructor() 함수가 실행되고 컨트랙트는 주소를 가지게 된다.

3.2 실행로직 부분

초기화 코드가 실행된 이후에는 런타임 코드만이 스택에 남아있는데, opcode가 메모리에 복사되어 EVM으로 반환된다.

4. EVM은 리턴된 남은 코드를 새로운 컨트랙트 주소와 관련하여 state storage에 저장한다. 이것은 향후 새로운 계약에 대한 모든 호출에서 스택에 의해 실행될 런타임 코드다.

초보자가 위 4단계를 이해하기란 정말 어려울 것이다. 필자도 이부분을 이해하는데 1달이 걸린듯하다.....

(위 과정을 정확하게 이해했다면, 출제자 포스팅의 치명적인 오타도 찾을 수 있다!)

위 4가지 단계가 정확하게 이해된다면 remix 디버거에서 일어나는 일들을 이해할 수 있을 것이다.

필자는 위 단계를 이해하고 난 후, remix IDE를 설계한 사람에게 감탄을 금치 못했다.

문제 풀이 과정

이 단계를 해결하기 위해서는 2가지의 opcodes를 설정해야 한다.

- 초기화 opcodes
- 런타임 opcodes : 0x42를 반환하고 10개 미만의 opcode가 있어야 하는 너의 코드의 주요 부분이다.

Runtime Opcodes

리턴 값은 두개의 인수를 취하는 `RETURN` opcode에 의해서 처리된다.

- `p` : 값이 메모리에 저장된 위치, 우리는 임의로 0x80슬롯을 선택한다.
- `s` : 저장된 데이터의 크기. 값은 32bytes(슬롯하나의 크기)의 크기다. (16진수로는 0x20이다).

-> 값을 리턴받기 전에 우선 값을 저장해야 한다.

우리는 해당하는 opcode들을 바이트코드로 만들어야 하기 때문에 아래의 과정을 거친다.

1. 먼저 mstore(p, v)를 사용하여 메모리에 0x42의 값을 저장해야한다. 인자 p는 위치고 v는 16진수의 값이다.

```
6042 //  v: push1 0x2A (옵코드 push1은 16진수로 0x60에 해당한다, 2A는 42의 16진수 값)
6080 // p: push1 0x80 (memory slot is 0x80)
52   //  opcode mstore에 해당하는 16진수 바이트 코드
```

2. 그러면, 0x42를 반환 할 수 있다.

```
6020 // s: push1 0x20 (value is 32 bytes in size)
6080 // p: push1 0x80 (value was stored in slot 0x80)
f3 // return
```

위 과정을 거치면 opcode를 `602A60805260206080f3`라는 바이트 코드를 얻을 수 있다. 런타인 옵코드의 길이는 정확히 10개의 opcode(push1을 포함해서 총 10개)와 10 bytes의 길이를 가진다.

Initialization Opcodes

위에서 언급한 컨트랙트가 생성되는 과정을 정확히 이해했다면, 초기화 옵코드들은 EVM에 반환하기 전에 런타임 opcode를 메모리에 복제해야 한다는 사실을 알 수 있다. EVM은 자동으로 블록체인에 런타임시퀀스(602A60805260206080f3)를 저장함으로 마지막 부분을 처리 할 필요가 없다.

한 곳에서 다른 곳으로 코드를 복사하는 opcode는 `CODECOPY`라는 옵코드에 의해 행해진다. 코드카피는 다음과 같은 3개의 인수를 가진다.

- t : 메모리 내, 코드의 목적지 위치. 우리는 임의로 0x00 위치에 코드를 저장한다.
- f : 전체 바이트 코드를 참조한 `런타임 opcode`의 현재 위치. f는 `초기화 opcode`뒤에서 시작한다는 것을 기억하자! 이 값은 현재 우리에게 알려지지 않았다.
- s : 코드의 크기 (바이트).

3. 런타인 옵코드를 메모리에 작성하는 바이트 코드 작성하기

```
600a // s: push1 0x0a (10 bytes <- 런타임 옵코드의 길이)
60?? // f: push1 0x?? (현재 위치를 모르니 ??으로 표기)
6000 // t: push1 0x00 (destination memory index 0)
39 // CODECOPY에 해당하는 바이트코드(16진수)
```

4. 이후, 메모리 안에있는 런타임 옵코드를 EVM으로 반환하는 바이트 코드 작성하기

```
600a // s: push1 0x0a (런타임 옵코드의 길이)
6000 // p: push1 0x00 (0의 위치에 접근)
f3 // return to EVM
```

5. 여기까지 완성된 우리의 초기화 opcode를 점검해보자! 600a60?? 600039600a6000f3으로 총 12 바이트(0x0c) 에 이른다. 즉, 우리의 런타임 opcode는 인덱스 0x0c에서 시작하고 ??은 0c에 해당한다.

위 과정을 통해 초기화 코드의 바이트 코드는 600a600c600039600a6000f3가되고 런타임 바이트코드와 합친 전체적인 바이트코드는 0x600a600c600039600a6000f3602A60805260206080f3가 된다!

이후 콘솔창에서 다음과 같은 코드로 문제를 해결 할 수 있다.

그동안 우리가 메타마스크에서 GUI로만 트랜잭션을 발생시켜 자세한 과정을 잘 몰랐겠지만, 사실 트랜잭션을 만들때 sendTransaction라는 함수에 매개변수를 넣어 발생시킨다.

```
> var account = "your address here";
> var bytecode =
"0x600a600c600039600a6000f3602A60805260206080f3";
> web3.eth.sendTransaction({ from: account, data: bytecode
}, function(err,res){console.log(res)});
```

위 3줄의 코드로 트랜잭션을 만들면 txId값을 이더스캔에 조회하면 새로운 CA주소가 나올 것이다.

이후 발급된 CA주소를 아래코드와 함께 다시한번 setSolver함수를 호출하면 성공!

```
await contract.setSolver("contract address");
```

다른 풀이 (부분 고수준)

앞선 과정을 완벽하게 이해했다면 우리는 이런 응용도 할 수 있다.

초기화 옵코드를 고수준으로 작성해서 아래와 같은 코드로 트랜잭션을 발생시켜도 성공한다.

```
contract MagicNumAssembly {
  constructor() public {
    assembly {
      // 런타임 옵코드가 매개변수로 들어감

      mstore(0, 0x602a60805260206080f3)
      return(0x16, 0x0a)
    }
  }
}
```

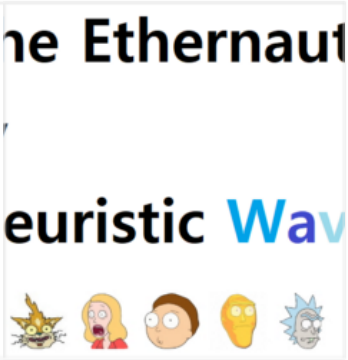
이건 출제자의 의도가 전혀 아닌 완전 고수준으로 푸는 방법이다.

```
// 힌트에서 언급한 whatIsTheMeaningOfLife()함수에 리턴값 42를 넣는다.
// 아래 함수를 포함하여 새롭게 컨트랙트를 작성하여도 통과할 것이다.

function whatIsTheMeaningOfLife() public pure returns
(bytes32) {
    return 42;
  }
```

이렇게 힘겹게 19단계를 마쳤다. 옵코드와 바이트코드에 대한 이해가 없다면 아마 최고로 어려운 난이도였을 것이다. 아무리 생각해도 19번 문제는 너무 아름답다ㅋㅋㅋ

20번, Alien Codex에서 만나요!



Ethernaut Alien Codex Problem — 이더넛 20단계 문제 해설

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

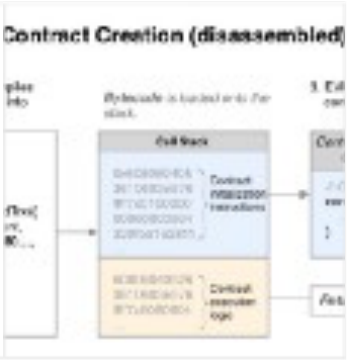
medium.com

참고자료

Ethernaut Lvl 19 MagicNumber Walkthrough: How to deploy contracts using raw assembly opcodes

This is a in-depth series around Zeppelin team's smart contract security puzzles. We learn key Solidity concepts

medium.com



추가로, 출제자 포스팅 하단에 있는 첨부자료를 이해한다면 더욱더 성장할 수 있다!