



Open in app

Get started



Heuristic Wave

Follow

Oct 4, 2018 · 17 min read



Save



Ethernaut Naught Coin Problem — 이더넷 15단계 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



. . .

Naught Coin

이번문제는 비교적 수월하게 문제를 풀 수 있었다. 필자는 이미 ERC20 토큰에 대한 이해와 배포경험 덕분에 오랜만에 빠르게 문제를 풀 수 있었다.

Naught 코인은 ERC20기반의 토큰이며 모든 토큰을 가지고 있다. 10년동안 locked이 된 이 토큰을 다른 주소로 옮기면 문제를 풀 수 있다. 아래에서 우선적으로 ERC 20을 공부하고 문제를 풀자!

- The [ERC20 Spec](#)
- The [OpenZeppelin](#) codebase

코드 분석

```
pragma solidity ^0.4.18;

import 'zeppelin-solidity/contracts/token/ERC20/StandardToken.sol';
/// StandardToken을 상속 받아서 구현했다.
contract NaughtCoin is StandardToken {

    string public constant name = 'NaughtCoin';
    string public constant symbol = '0x0';
    uint public constant decimals = 18;
    uint public timeLock = now + 10 years; // 이곳 때문에 10년의
    Lock이 걸려있다.
    uint public INITIAL_SUPPLY = 1000000 * (10 ** decimals);
    address public player;

    function NaughtCoin(address _player) public {
        player = _player;
        totalSupply_ = INITIAL_SUPPLY;
        balances[player] = INITIAL_SUPPLY;
        Transfer(0x0, player, INITIAL_SUPPLY);
    }

    function transfer(address _to, uint256 _value) lockTokens
    public returns(bool) {
        super.transfer(_to, _value);
    }
}
```

```

    }

    modifier lockTokens() {
        if (msg.sender == player) {
            require(now > timeLock);
            if (now < timeLock) {
                -;
            }
        } else {
            -;
        }
    }
}

```

힌트에서 주어진 ERC20의 기능들을 잘 살펴보면 ERC20 기반의 토큰에서는 돈을 보내는 방법이 `transfer` 과 `transferFrom` 이 2가지가 있다. ERC20의 기능을 하나의 포스팅으로 다뤄도 될정도의 양이니 여기서는 해당함수의 간단한 기능을 소개하겠다.

transfer

```

function transfer(address to, uint tokens) public returns
(bool success);

```

to 의 주소로 tokens 양만큼 보낸다.

transferFrom

```

function transferFrom(address from, address to, uint tokens)
public returns (bool success);

```

from에서 to로 tokens의 양만큼 보낸다. 그런데 transferFrom은 transfer과는 다르게 from의 주소가 있다. 이는 ERC20에 구현된 approve 함수와의 관계 때문에 그렇다. approve에서 승인해준 토큰의 양만큼을 transferFrom함수를 통해서 제 3자에게 보낼 수 있다.

즉 우리는 문제를 해결하기 위해서 approve 함수를 통해 제 2의 주소에 보낼수 있는 양을 허가하고 제 2의 주소에서 제 3(문제를 풀게된 나의 주소)의 주소로 토큰은 전송시키면 위와 같은 로직을 통과하고 토큰을 전송할 수 있다.

우리가 문제를 지급 받으면 NaughtCoin이 문제를 풀고있는 계정에 지급된다.

이는 이더넷 개발자도구에서 아래와 같은 명령어로 확인 할 수 있다.

```
await
contract.balanceOf('0x344fbee17c2d215d364ec5943bc4a0c7030cfaa1').then(x => x.toNumber())
```

Player의 주소의 balanceOf함수를 호출하여 toNumber함수를 통해 화면에 띄우면 $1e+24$ 이 나올 것이다. 문제에서 decimal을 18로 하고 처음 초기량이 10의 6제곱이기 때문에 $(10^{18}) * (10^6)$ 이기 때문에 24제곱이 나온다. decimal과 공급량에 대한 정보는 ERC20을 알고 있다는 전제하에 이 문제에서 논외로 한다.

우리는 remix에서도 확인 할 수 있다. 다른 문제와는 다르게 import문이 리믹스에서 불러올 수 없으므로 아래 필자가 붙여놓은 코드를 import문을 대체해서 사용하자!!

제플린의 표준코드가 트러플 기반으로 작성되었기 때문에 import문이 존재하는 것이다

```
library SafeMath {
    /**
     * @dev Multiplies two numbers, throws on overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns
    (uint256) {
        if (a == 0) {
            return 0;
        }
        uint256 c = a * b;
        assert(c / a == b);
        return c;
    }

    /**
     * @dev Integer division of two numbers, truncating the
```

```

quotient.
    */
    function div(uint256 a, uint256 b) internal pure returns
(uint256) {
        // assert(b > 0); // Solidity automatically throws when
dividing by 0
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in
which this doesn't hold
        return c;
    }

    /**
    * @dev Subtracts two numbers, throws on overflow (i.e. if
subtrahend is greater than minuend).
    */
    function sub(uint256 a, uint256 b) internal pure returns
(uint256) {
        assert(b <= a);
        return a - b;
    }

    /**
    * @dev Adds two numbers, throws on overflow.
    */
    function add(uint256 a, uint256 b) internal pure returns
(uint256) {
        uint256 c = a + b;
        assert(c >= a);
        return c;
    }
}

contract ERC20Basic {
    function totalSupply() public view returns (uint256);
    function balanceOf(address who) public view returns
(uint256);
    function transfer(address to, uint256 value) public
returns (bool);
    event Transfer(address indexed from, address indexed to,
uint256 value);
}

contract ERC20 is ERC20Basic {
    function allowance(address owner, address spender) public
view returns (uint256);
    function transferFrom(address from, address to, uint256
value) public returns (bool);
    function approve(address spender, uint256 value) public
returns (bool);

```

```

    event Approval(address indexed owner, address indexed
spender, uint256 value);
}

contract BasicToken is ERC20Basic {
    using SafeMath for uint256;

    mapping(address => uint256) balances;

    uint256 totalSupply_;

    /**
    * @dev total number of tokens in existence
    */
    function totalSupply() public view returns (uint256) {
        return totalSupply_;
    }

    /**
    * @dev transfer token for a specified address
    * @param _to The address to transfer to.
    * @param _value The amount to be transferred.
    */
    function transfer(address _to, uint256 _value) public
returns (bool) {
        require(_to != address(0));
        require(_value <= balances[msg.sender]);

        // SafeMath.sub will throw if there is not enough
balance.
        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        Transfer(msg.sender, _to, _value);
        return true;
    }

    /**
    * @dev Gets the balance of the specified address.
    * @param _owner The address to query the the balance of.
    * @return An uint256 representing the amount owned by the
passed address.
    */
    function balanceOf(address _owner) public view returns
(uint256 balance) {
        return balances[_owner];
    }
}

contract StandardToken is ERC20, BasicToken {

```

```
mapping (address => mapping (address => uint256)) internal  
allowed;
```

```
/**  
 * @dev Transfer tokens from one address to another  
 * @param _from address The address which you want to send  
tokens from  
 * @param _to address The address which you want to  
transfer to  
 * @param _value uint256 the amount of tokens to be  
transferred  
 */
```

```
function transferFrom(address _from, address _to, uint256  
_value) public returns (bool) {  
    require(_to != address(0));  
    require(_value <= balances[_from]);  
    require(_value <= allowed[_from][msg.sender]);  
  
    balances[_from] = balances[_from].sub(_value);  
    balances[_to] = balances[_to].add(_value);  
    allowed[_from][msg.sender] = allowed[_from]  
[msg.sender].sub(_value);  
    Transfer(_from, _to, _value);  
    return true;  
}
```

```
/**  
 * @dev Approve the passed address to spend the specified  
amount of tokens on behalf of msg.sender.  
 *  
 * Beware that changing an allowance with this method  
brings the risk that someone may use both the old  
 * and the new allowance by unfortunate transaction  
ordering. One possible solution to mitigate this  
 * race condition is to first reduce the spender's  
allowance to 0 and set the desired value afterwards:  
 *
```

```
https://github.com/ethereum/EIPs/issues/20#issuecomment-  
263524729
```

```
 * @param _spender The address which will spend the funds.  
 * @param _value The amount of tokens to be spent.  
 */  
function approve(address _spender, uint256 _value) public  
returns (bool) {  
    allowed[msg.sender][_spender] = _value;  
    Approval(msg.sender, _spender, _value);  
    return true;  
}
```

```

/**
 * @dev Function to check the amount of tokens that an
owner allowed to a spender.
 * @param _owner address The address which owns the funds.
 * @param _spender address The address which will spend
the funds.
 * @return A uint256 specifying the amount of tokens still
available for the spender.
 */
function allowance(address _owner, address _spender)
public view returns (uint256) {
    return allowed[_owner][_spender];
}

/**
 * @dev Increase the amount of tokens that an owner
allowed to a spender.
 *
 * approve should be called when allowed[_spender] == 0.
To increment
 * allowed value is better to use this function to avoid 2
calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * @param _spender The address which will spend the funds.
 * @param _addedValue The amount of tokens to increase the
allowance by.
 */
function increaseApproval(address _spender, uint
_addedValue) public returns (bool) {
    allowed[msg.sender][_spender] = allowed[msg.sender]
[_spender].add(_addedValue);
    Approval(msg.sender, _spender, allowed[msg.sender]
[_spender]);
    return true;
}

/**
 * @dev Decrease the amount of tokens that an owner
allowed to a spender.
 *
 * approve should be called when allowed[_spender] == 0.
To decrement
 * allowed value is better to use this function to avoid 2
calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * @param _spender The address which will spend the funds.
 * @param _subtractedValue The amount of tokens to
decrease the allowance by.

```



```

    */
    function decreaseApproval(address _spender, uint
    _subtractedValue) public returns (bool) {
        uint oldValue = allowed[msg.sender][_spender];
        if (_subtractedValue > oldValue) {
            allowed[msg.sender][_spender] = 0;
        } else {
            allowed[msg.sender][_spender] =
            oldValue.sub(_subtractedValue);
        }
        Approval(msg.sender, _spender, allowed[msg.sender]
        [_spender]);
        return true;
    }
}

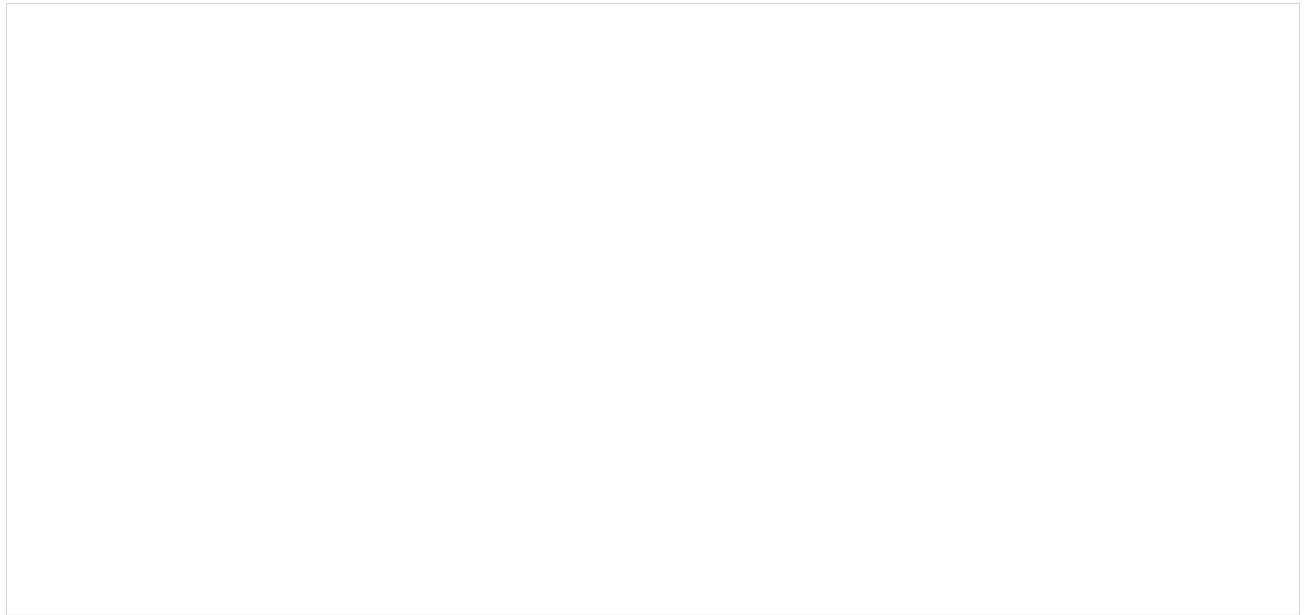
```

리믹스에서 위 코드와 주어진 코드를 한페이지에 넣고 문제주소를 불러오면 NaughtCoin을 불러 올 수 있다.

문제 풀이

위 사진을 확인해보면 balanceOf 함수로 필자의 계정을 조회해 보았을때 10^{24} 제곱 만큼의 양이 있다는 것을 알 수 있다. 우리는 approve 함수에 `_spender`에 제어할 수 있는 다른 계정(메타마스크에 위치한 나의 다른 계정)을 넣고 `_value`에 10^{24} 제곱 만큼의

양을 기입하고 트랜잭션을 생성시키자!



이후 우리는 `_spender`에 기입했던 계정으로 메타 마스크를 전환하여 `allowance` 함수의 인자 값으로 `_owner`와 `_spender`에 해당하는 각 값을 넣으면 정상적으로 10^{24} 제곱 만큼의 양을 허락 받은 것을 알 수 있다.

이제 `transferFrom` 함수로 `_from`에 코인을 가지고 있는 계정, `_to`에 보낼 주소와 양을 함께 기입하고 트랜잭션을 발생시키면 처음에 공급받은 모든 토큰을 전송하는데 성공한다.

이후 답안지 제출 버튼을 누르면 다음 단계로 넘어 갈 수 있다.

이번 문제에서는 ERC20의 기능들을 공부 할 수 있는 문제다. 또한, 본인이 발행한 토큰에 `lock` 함수를 구현해도 통과할 수 있는 방법이 있기 때문에 매번 신중하여야 한다. 또한 타인이 작성한 소스코드를 사용할때도 꼼꼼히 살펴보고 사용하여야 한다.

도움이 될만한 자료들

| ERC20 기능 정리 — 필자와 함께 블록체인을 공부하는 친구의 자료다.

그럼, 다음번에는 16단계 Preservation에서 만나요!

Ethernaut Preservation Problem — 이더넷 16단계 문제 해설

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

medium.com

ne Ethernaut

euristic Wav



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

Get the Medium app

