# HTB — NotADemocraticElection

3 min read · Oct 28, 2024

DSG Follow

Listen Share

So guys this will be my third solve from HTB blockchain.Anyways first lets understand the contract.Be patient and you can crack it :)

1)NotADemocraticElection.sol

```
pragma solidity 0.8.25;

contract NotADemocraticElection {
    // ****************************************************
    // ******* NOTE: THIS NOT A DEMOCRATIC ELECTION *******
    // ****************************************************

    uint256 constant TARGET_VOTES = 1000e18;

    struct Party {
        string fullname;
        uint256 totalvotes;
    }
    struct Voter {
        uint256 weight;
        address addr;
    }

    mapping(bytes3 _id => Party) public parties;
    mapping(bytes _sig => Voter) public voters;
    mapping(string _name => mapping(string _surname => address _addr))
        public uniqueVoters;
    bytes3 public winner;

    event Voted(address _voter, bytes3 _party);
    event VoterDeposited(address _voter, uint256 _weight);
```

```
    event ElectionWinner(bytes3 _party);

    constructor(
        bytes3 _partyAsymbol,
        string memory _partyAfullname,
        bytes3 _partyBsymbol,
        string memory _partyBfullname
    ) {
        parties[_partyAsymbol].fullname = _partyAfullname;
        parties[_partyBsymbol].fullname = _partyBfullname;
    }

    function getVotesCount(bytes3 _party) public view returns (uint256) {
        return parties[_party].totalvotes;
    }

    function getVoterSig(
        string memory _name,
        string memory _surname
    ) public pure returns (bytes memory) {
        return abi.encodePacked(_name, _surname);
    }

    function checkWinner(bytes3 _party) public {
        if (parties[_party].totalvotes >= TARGET_VOTES) {
            winner = _party;
            emit ElectionWinner(_party);
        }
    }

    function depositVoteCollateral(
        string memory _name,
        string memory _surname
    ) external payable {
        require(
            uniqueVoters[_name][_surname] == address(0),
            "Already deposited"
        );

        bytes memory voterSig = getVoterSig(_name, _surname);
        voters[voterSig].weight += msg.value;
        uniqueVoters[_name][_surname] = msg.sender;

        emit VoterDeposited(msg.sender, msg.value);
    }

    function vote(
        bytes3 _party,
        string memory _name,
        string memory _surname
    ) public {
        require(
            uniqueVoters[_name][_surname] == msg.sender,
```

```
            "You cannot vote on behalf of others."
        );

        bytes memory voterSig = getVoterSig(_name, _surname);
        uint256 voterWeight = voters[voterSig].weight == 0
            ? 1
            : voters[voterSig].weight;
        parties[_party].totalvotes += 1 * voterWeight;

        emit Voted(msg.sender, _party);
        checkWinner(_party);
    }
}
```

The `NotADemocraticElection` contract is a Solidity smart contract that facilitates a simple voting system where parties accumulate votes based on voters' deposits. The contract allows two parties to be initialized, and voters can deposit Ether to gain voting weight. Each voter must deposit a collateral to vote, and they can only vote for a party if their identity is registered. Votes are weighted by the amount of collateral deposited. When a party reaches a predefined target of votes, it is declared the winner through an emitted event. The contract includes mechanisms for voting, depositing collateral, and checking the winning party.

2)Setup.sol

```
pragma solidity 0.8.25;

import {NotADemocraticElection} from "./NotADemocraticElection.sol";

contract Setup {
    NotADemocraticElection public immutable TARGET;

    constructor() payable {
        TARGET = new NotADemocraticElection(
            bytes3("ALF"), "Automata Liberation Front",
            bytes3("CIM"), "Cyborgs Indipendence Movement"
        );
        TARGET.depositVoteCollateral{value: 100 ether}("Satoshi", "Nakamoto");
    }
```

```
        }
    }
```

The `Setup` contract deploys an instance of the `NotADemocraticElection` contract with two predefined parties: "Automata Liberation Front" and "Cyborgs Independence Movement." It deposits 100 ether as collateral for a voter named "Satoshi Nakamoto." The `isSolved` function checks if the winning party is "Cyborgs Independence Movement" by comparing the current winner stored in the `NotADemocraticElection` contract with the party's symbol. This setup ensures that the election's result can be verified based on the deployed contract's state

**THE EXPLOIT:**

Well initially by looking the contract that you can think that its an easy one.Like you can deposit some huge ether amount and just influence the vote bank to make CIM win. But the voters are provided only with 1 ETH so they would run out of gas for sure. So we cannot break it there and have to think it in a different way. So shifting the focus to getVoterSig() function,we can find that it uses abi.encodePacked(). This function is vulnerable to signature forgery attack.

We will register a voter as “S atoshiNakamato”. So now I will call the depositVoteCollateral() function so uniqueVoters[_name][_surname] where name would be “S” and _surname would be “atoshiNakamato”. Now replicate this with such sequences like Sa toshiNakamato and eventually you can gain enough weight to influence the winning party.

```
private-key $PRIVATE_KEY "depositVoteCollateral(string,string)" Sato shiNakamoto
```

change the strings and vote for 10 times. You can solve this challenge.

Blockchain Solidity