USING ETHEREUM SMART CONTRACTS TO DEMOCRATIZE CHARITIES

JESSIE BAI, WILIAM SUN, STANLEY WANG, AND ELAINE WONG

BACKGROUND

Why do charities need to be democratized?

Current Charities

- ➔ Generally have a mission statement identifying a broad cause they are contributing towards (i.e. saving wildlife, furthering education, etc.)
- ➔ Hard to know how funds are used:
  - ◆ Administrative costs
  - ◆ Different projects done by the charity
- ➔ **Problem:** Donating can feel low impact if you don’t know what the money is going towards; lower motivation

Other Existing Solutions

- ➔ Available sample voting smart contract:
  - ◆ Should limit the voting time to a certain period controlled by the creator
- ➔ Giveth, Alice - Platforms which use Ethereum to make charities' spending transparent
  - ◆ Transparent, allows donors to see the impact of their funds
  - ◆ Giveth: Can withdraw if they feel negatively about usage of funds, but no indication of what they want
  - ◆ Alice: Weighted voting on Project Update Proposals for existing projects

OUR SOLUTION

Using Ethereum smart contracts to vote on how money is spent

A Weighted Voting System

- ➔ Have a variety of potential projects up for voting (i.e. building homeless shelters, growing bamboo for pandas, etc.)
- ➔ Donations make a donor eligible for the voting process, since votes are weighted by donation
- ➔ The project that gets the most votes gets all of the balance donated to it
- ➔ Benefits:
  - ◆ Donors feel more impactful since they can choose projects, and pooled donations maximize the impact on the winning project
  - ◆ Encourages more donations via competition

Current Implementation

- ➔ Always open to donations, keeps a map of donor -> donation amount
- ➔ Owner of contract can add projects (projects each have their own wallet/address), create a voting period by calling startVoting(duration)
- ➔ After voting starts, donors can still donate, but now they can use their donations to vote (i.e. 100 wei = 100 votes)
- ➔ After voting ends, anyone can call disperse() to send the entire balance of the contract to the winning project

Ethereum Tools

- ➔ Truffle: Solidity development & testing framework
  - ◆ Used for writing/compiling smart contract
  - ◆ Ganache: Local development blockchain
  - ◆ Used for writing integration tests in JavaScript with Chai
- ➔ MetaMask: Extension for browser to connect to Ganache Ethereum wallets

DEMO!!

FUTURE WORK

Attacks/problems and potential solutions

Problems

- ➔ People who would benefit from a certain project could donate a huge amount of money to that project and get all of their money back + everyone else’s money
- ➔ Centralization: Owner manages the contract and projects
- ➔ Integer over/underflow
- ➔ Gas attacks
- ➔ Timestamp dependency
- ➔ Reentrancy

Security Considerations

Integer Over/Underflow

1] Should not happen in our case, since the amount of donation can hardly reach any bound (Overflow)

2] The algorithm always clears the donated() to zero instead of decreasing any value (Underflow).

```
mapping (address => uint256) public balanceOf;

// INSECURE
function transfer(address _to, uint256 _value) {
    /* Check if sender has balance */
    require(balanceOf[msg.sender] >= _value);
    /* Add and subtract new balances */
    balanceOf[msg.sender] -= _value;
    balanceOf[_to] += _value;
}

// SECURE
function transfer(address _to, uint256 _value) {
    /* Check if sender has balance and for overflows */
    require(balanceOf[msg.sender] >= _value && balanceOf[_to] + _value >= balanceOf

    /* Add and subtract new balances */
    balanceOf[msg.sender] -= _value;
    balanceOf[_to] += _value;
}
```

Security Considerations

Integer Over/Underflow

1] Should not happen in our case, since the amount of donation can hardly reach any bound (Overflow)

2] The algorithm always clear the donated() to zero instead of decreasing any value (Underflow).

```
contract UnderflowManipulation {
    address public owner;
    uint256 public manipulateMe = 10;
    function UnderflowManipulation() {
        owner = msg.sender;
    }

    uint[] public bonusCodes;

    function pushBonusCode(uint code) {
        bonusCodes.push(code);
    }

    function popBonusCode()  {
        require(bonusCodes.length >=0);  // this is a tautology
        bonusCodes.length--; // an underflow can be caused here
    }

    function modifyBonusCode(uint index, uint update)  {
        require(index < bonusCodes.length);
        bonusCodes[index] = update; // write to any index less than bonusCodes.leng
    }

}
```

Security Considerations

Integer Over/Underflow

1] Should not happen in our case, since the amount of donation can hardly reach any bound (Overflow)

2] The algorithm always clear the donations[msg.sender] to zero instead of decreasing any value (Underflow).

```
// Allows any individual to donate ethereum to this charity
function donate() public payable {
    donations[msg.sender] += msg.value;
    emit donated(msg.sender, msg.value);
}
```

```
// @param option: the index of the option to vote for
function vote(uint option) public {
    if (isVotingActive() && option < votingOptionsCount) {
        uint temp = donations[msg.sender];
        donations[msg.sender] = 0;
        votingOptionVotes[option] += temp;
    }
}
```

Security Considerations

```
function addVoteOption(bytes32 option, address optionAddress) public {
    if (msg.sender == creator && startTime == (2**256 - 1)) {
        // TODO: Make this less susceptible to gas attacks; we need a max # of charities
        if (votingOptions.length <= votingOptionsCount) {
            votingOptions.push("");
            votingOptionAddresses.push(0);
            votingOptionVotes.push(0);
        }
        votingOptions[votingOptionsCount] = option;
        votingOptionAddresses[votingOptionsCount] = optionAddress;
        votingOptionVotes[votingOptionsCount] = 0;
        votingOptionsCount++;
        emit optionAdded(option, optionAddress);
    }
}
```

**Not deleting options to save gas

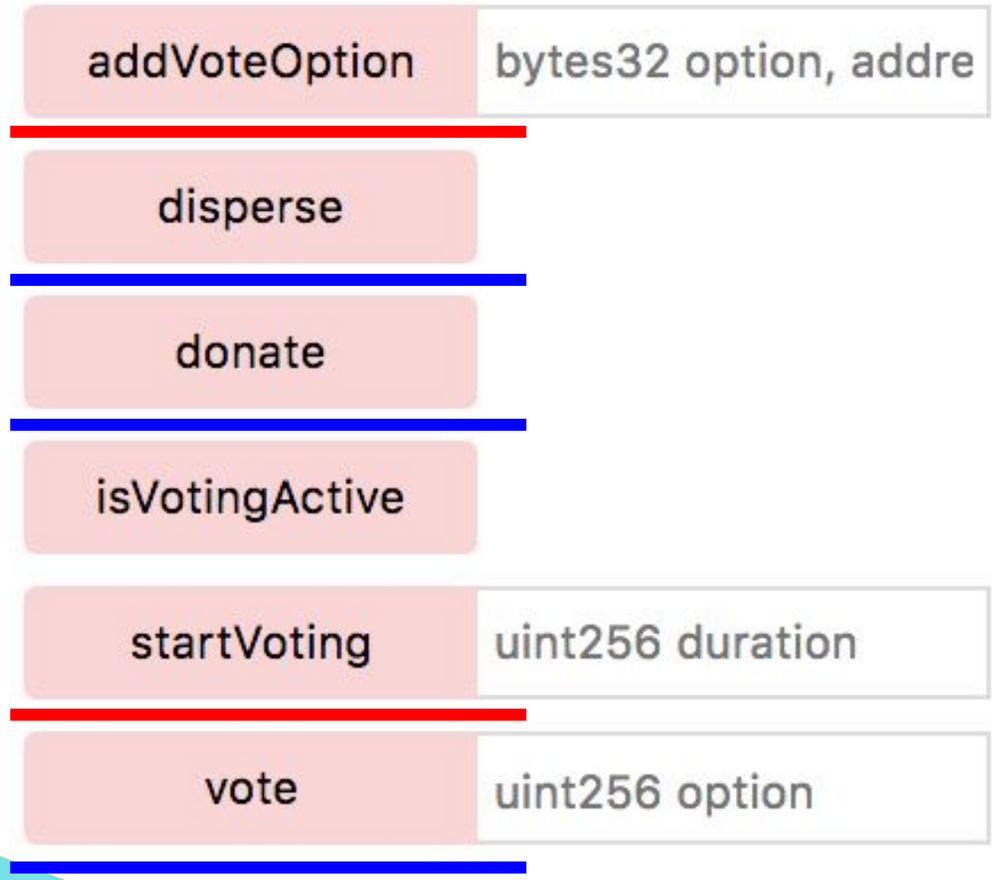
Block Gas Limit

1] The number of charity is created and therefore limited by the creator of the contract.

2] Donating to our account does NOT cost gas

3] There is no refund function in the contract so therefore no possibility for malicious request.

Security Considerations



Block Gas Limit

1] The number of charity is created and therefore limited by the creator of the contract.

2] Donating to our account does NOT cost gas

3] There is no refund function in the contract so therefore no possibility for malicious request.

: Cost gas from Creator

: Cost gas from Donors

Security Considerations

```
// If one donates money but hasn't voted, he/she can reclaim money back.
function returnDonation() public {
    if (donations[msg.sender] > 0) {
        donations[msg.sender] = 0;
        msg.sender.transfer(donations[msg.sender]);
    }
}
```

Block Gas Limit

1] The number of charity is created and therefore limited by the creator of the contract.

2] Donating to our account does NOT cost gas

3] There is no refund function in the contract so therefore no possibility for malicious request.

Security Considerations

Reentrancy

1] Since the algorithm sets the user's balance to 0 in the function before the donation is added into a certain charity vote, no second (and later) invocations will succeed, and the donation would not take place over and over again.

```
// INSECURE
mapping (address => uint) private userBalances;

function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
  1. require(msg.sender.call.value(amountToWithdraw)());
  2. userBalances[msg.sender] = 0;
}
```

2] This is more of a problem in the older version solidity: picking between send() instead of call.value()() to prevent any external code from being executed.
Solidity later introduces the new transfer() function; it is even safer than send() to prevent reentrancy.

```
function vote(uint option) public {
    if (isVotingActive() && option < votingOptionsCount) {
        uint temp = donations[msg.sender];
      1. donations[msg.sender] = 0;
      2. votingOptionVotes[option] += temp;
    }
}
```

Security Considerations

```
uint public startTime;
uint public endTime;

// Constructor
function Charity() public {
    creator = msg.sender;
    startTime = 2**256 - 1;
    endTime = 2**256 - 1;
}
```

1.
```
function vote(uint option) public {
    if (isVotingActive() && option <
```

2.
```
function addVoteOption(bytes32 option, address optionAddress) public {
    if (msg.sender == creator && startTime == (2**256 - 1)) {
```

3.
```
function startVoting(uint duration) public {
    if (msg.sender == creator && startTime == 2**256 - 1 && now < endTime) {
        startTime = now;
        endTime = now + duration;
```

4.
```
function isVotingActive() public returns (bool) {
    return (now >= startTime && now <= endTime);
```

5.
```
function disperse() public returns (bool){
    // https://ethereum.stackexchange.com/q
    if (now >= endTime) {
```

Timestamp Dependence

Because the timestamp of the block could be manipulated by the miner, we should consider all direct and indirect uses of the timestamp.

In this case we implement our own time tracking algorithm. Specifically, only the administrator/creator can start the voting and the duration of voting time would be set at the same time. All the functions depend on the status of the voting (before voting/ during voting/ after voting)

Selfish Adversary: Proposed Solutions

- ➔ Proposal: Have all donations split vote over x number of rounds (ex. 4 rounds, so each donation's weight is quartered)
  - ◆ Adversary will have to waste ¾ the money if they are only doing it for one project or pay more to overwhelm the vote
    - Problem: Won't pay more if everyone keeps their donations the same (i.e. everything is still proportional)
    - Problem: Donors may be discouraged if they need to trust admin to pick good projects for next round of voting
- ➔ Proposal: Mask votes (but this destroys key feature of transparency)
- ➔ Does preventing this even make sense? Is it possible? More thinking... (insert snide remarks about democracy here)

Centralization: Proposed Solutions

- ➔ Currently: can’t embezzle funds, but can pick bad projects rendering the donations useless
- ➔ Proposal: Could increase number of administrators, but this still results in centralized committee
- ➔ Not too much of a problem in our simple model since people can donate after projects are specified

THANK YOU!

https://github.com/wsun19/Blockchains-Project