pragma solidity ^0.5.16;

contract Auction {

TransactionLog Transactions;

address internal judgeAddress;

address internal sellerAddress;

address internal winnerAddress;

uint winningPrice;

mapping (address => uint) balances;

constructor(address \_sellerAddress,

address \_judgeAddress) public {

judgeAddress = \_judgeAddress;

sellerAddress = \_sellerAddress;

if (sellerAddress == address(0))

sellerAddress = msg.sender;

Transactions=TransactionLog(msg.sender);

}

function getWinner() public view returns (address winner) {

return winnerAddress;

}

function finalize() public {

require(((msg.sender==judgeAddress || msg.sender==getWinner()) || judgeAddress==address(0)) && getWinner()!=address(0));

//winningPrice=msg.value;

if (msg.sender==judgeAddress || msg.sender==sellerAddress){

refund();

balances[sellerAddress]=winningPrice;

withdraw();

}

else{

//balances[getWinner()]=winningPrice;

balances[sellerAddress]=winningPrice;

withdraw();

}

}

function refund() public {

require((msg.sender==judgeAddress || msg.sender==sellerAddress) && getWinner()!=address(0));

//winningPrice=msg.value;

balances[getWinner()]=winningPrice;

}

//vulnerable

function withdraw() public {

uint tosend=balances[msg.sender];

//msg.sender.transfer(tosend);

if(msg.sender.call.value(winningPrice)())

{

balances[msg.sender]=0;

Transactions.DisplayMess(msg.sender, “Withdrawing Money”);

}

}

}

contract TransactionLog

{

struct Rec

{

address senderAddress;

string messageData;

uint timestamp;

}

Rec currTrans;

Rec[] public TranLog;

function DisplayMess(address \_senderAddress, string \_data)

public

{

currTrans.senderAddress = \_senderAddress;

currTrans.messageData=\_data;

currTrans.timestamp=now;

TranLog.push(currTrans):

}

}