

**CYBERSECURITY
J-COMPONENT
FINAL REPORT
ON
AUCTION WINNER DETERMINER WITH
SOLIDITY AND REMIX IDE**



VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

NAME: KARTHIK K

REG.NO: 17BEC0695

SCHOOL: SENSE

FACULTY: PROF.LAVANYA K

SUBMISSION DATE:27/04/2020

ABSTRACT:

As the above title in front page suggest it is all about conducting an auction without any cause of ruckus and in a fair method for everyone and this can be done with the help of a blocks of code which constitutes for the whole process in a fair method, this will be elaborated in following report

The main reason for going with this method is that auction is cannot be corrupted as the bids can be seen by everyone and this is being more transparent throughout the process and also transactions are done then and there when everyone is there to witness the process, it is a genuine process where auction is transparent for everyone so even if the auction conductor cannot act biased for his own purpose of gain.

BRIEFING THE PROCESS

This process starts by having number distinct items that are for auction each have their own item id and own item token to identify them and at the same time we have address, remaining token and person Id for each bidders or person

Here the owner of smart contract will have his own details such as address id, etc. this will be distributed publicly for the transaction to happen

Now each or any number of persons can bid for any item but before that have to register for that particular item to bid for the item or to compete with other bidders, so for each bidders of the particular item that they are competing will be given the smart contract account details and the bidders for each will be given 5 tokens so each token will have unique values issued by the smart contract owner

For bidding we need two parameters those are item id and total count of the item(price of the item)

Here balance will be remaining token out of total count this balance will be updated to the bidders account and the token items detail will be sent to the person id

Now for revealing the winners we will access for the particular item with its item id after that there is a

calculation for determining the token id after determining the winning token id we map it with the address of the person it will show the address of the transaction of the winner id

PROGRAM CODE

```
pragma solidity ^0.4.17;

contract Auction {

    struct Item {
        uint itemId; // id of the item
        uint[] itemTokens; //tokens bid in favor of the item
    }

    struct Person {
        uint remainingTokens; // tokens remaining with bidder
        uint personId; // it serves as tokenId as well
        address addr;//address of the bidder
    }

    mapping(address => Person) tokenDetails; //address to person
    Person [4] bidders;//Array containing 4 person objects

    Item [3] public items;//Array containing 3 item objects
    address[3] public winners;//Array for address of winners
    address public beneficiary;//owner of the smart contract

    uint bidderCount=0;//counter

    //functions

    function Auction() public payable{ //constructor
        beneficiary = msg.sender;
    }
}
```

```
uint[] memory emptyArray;

items[0] = Item({itemId:0,itemTokens:emptyArray});

items[1] = Item({itemId:1,itemTokens:emptyArray});
items[2] = Item({itemId:2,itemTokens:emptyArray});

}

function register() public payable{

    bidders[bidderCount].personId = bidderCount;

    bidders[bidderCount].addr = msg.sender;

    bidders[bidderCount].remainingTokens = 5; // only 5 tokens
    tokenDetails[msg.sender]=bidders[bidderCount];
    bidderCount++;
}

function bid(uint _itemId, uint _count) public payable {

    if (tokenDetails[msg.sender].remainingTokens < _count || bidders[bidderCount].remainingTokens == 0) revert();
    if (_itemId > 2) revert();

    uint balance = tokenDetails[msg.sender].remainingTokens - _count;
    tokenDetails[msg.sender].remainingTokens=balance;
```

```
bidders[tokenDetails[msg.sender].personId].remainingTokens=balance;//updating the same  
balance in bidders map.
```

```
Item storage bidItem = items[_itemId];  
  
for(uint i=0; i<_count;i++) {  
  
    bidItem.itemTokens.push(tokenDetails[msg.sender].personId);  
  
}  
  
}
```

```
modifier onlyOwner {
```

```
_;  
}
```

```
function revealWinners() public onlyOwner{
```

```
for (uint id = 0; id < 3; id++) {  
  
    Item storage currentItem=items[id];  
  
    if(currentItem.itemTokens.length != 0){  
  
        // generate random# from block number  
  
        uint randomIndex = (block.number / currentItem.itemTokens.length)%  
currentItem.itemTokens.length;  
  
        // Obtain the winning tokenId
```

```
        uint winnerId = currentItem.itemTokens[randomIndex];
```

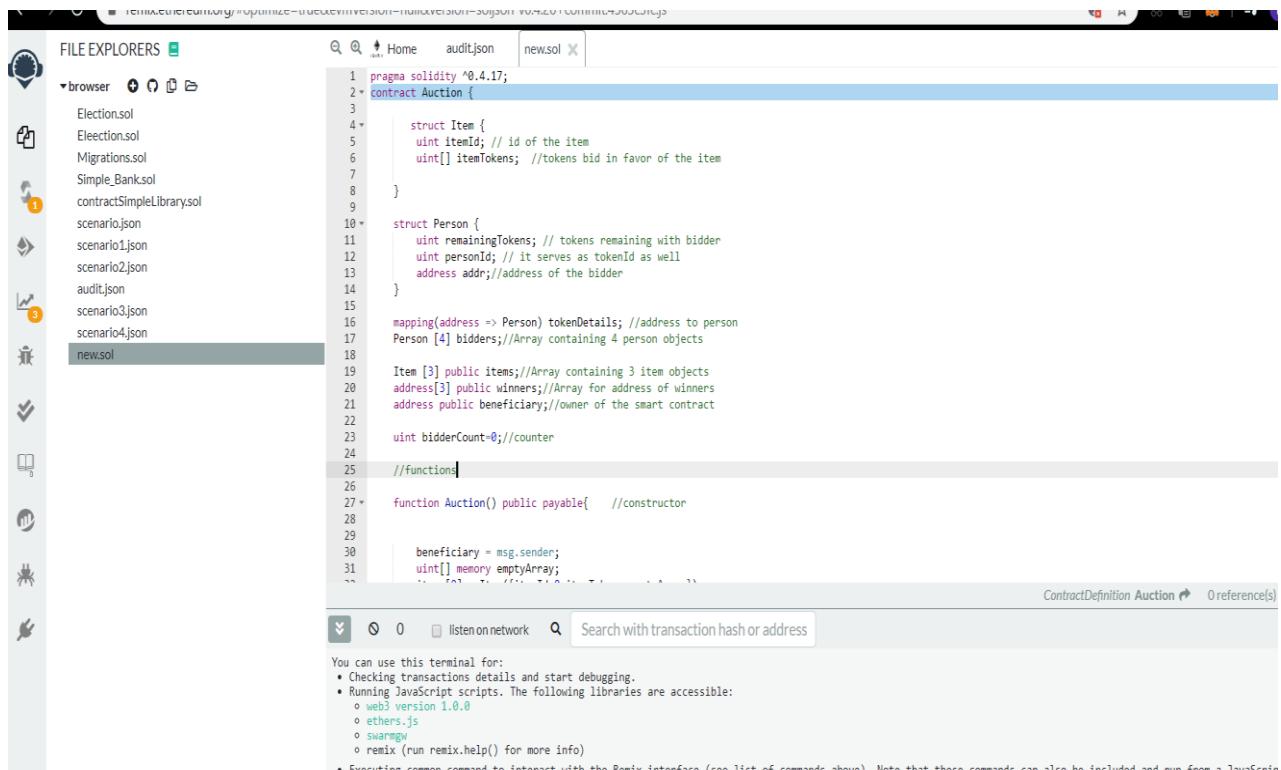
```
        winners[id] = bidders[winnerId].addr;
```

```
}
```

```
}}}
```

SCREENSHOTS

PROGRAM CODE



The screenshot shows the Remix IDE interface. On the left, there's a file explorer titled "FILE EXPLORERS" containing several Solidity files: Election.sol, Elecction.sol, Migrations.sol, Simple_Bank.sol, contractSimpleLibrary.sol, scenario.json, scenario1.json, scenario2.json, audit.json, scenario3.json, scenario4.json, and new.sol. The "new.sol" file is currently open in the main editor area. The code is as follows:

```
1 pragma solidity ^0.4.17;
2 * contract Auction {
3
4     struct Item {
5         uint itemId; // id of the item
6         uint[] itemTokens; //tokens bid in favor of the item
7
8     }
9
10    struct Person {
11        uint remainingTokens; // tokens remaining with bidder
12        uint personId; // it serves as tokenId as well
13        address addr; //address of the bidder
14    }
15
16    mapping(address => Person) tokenDetails; //address to person
17    Person [4] bidders; //Array containing 4 person objects
18
19    Item [3] public items; //Array containing 3 item objects
20    address[3] public winners; //Array for address of winners
21    address public beneficiary; //owner of the smart contract
22
23    uint bidderCount=0; //counter
24
25    //functions
26
27    function Auction() public payable{ //constructor
28
29
30        beneficiary = msg.sender;
31        uint[] memory emptyArray;
```

Below the code editor, there's a terminal window with the following instructions:

- You can use this terminal for:
- Checking transactions details and start debugging.
- Running JavaScript scripts. The following libraries are accessible:
 - o web3 version 1.0.0
 - o ethers.js
 - o swarmify
 - o remix (run remix.help() for more info)
- Executing common command to interact with the Remix interface (see list of commands above). Note that these commands can also be included and run from a JavaScript

At the bottom, there are buttons for "ContractDefinition Auction" and "0 reference(s)".

OUTPUT

RUNNING THE TRANSACTION

0x33c...A6546 (1.999181991 ether) ⚙️ ⓘ

GAS LIMIT	<input type="text" value="3000000"/>
VALUE	<input type="text" value="0"/> ether ⚙️ ⓘ
CONTRACT	Auction - browser/new.sol ⚙️ ⓘ

Deploy

PUBLISH TO IPFS

OR

At Address Load contract from Address

Transactions recorded ⓘ

All transactions (deployed contracts and function executions) in this environment can be saved and replayed in another environment, e.g. Transactions created in Javascript VM can be replayed in the Injected Web3.

CALLING AUCTION REGISTER

0x33c...A6546 (1.999181991 ether) ⚡

SAS LIMIT

3000000

/VALUE

0 ether ⚡

CONTRACT

Auction - browser/new.sol ⚡ ⓘ

Deploy

PUBLISH TO IPFS

OR

At Address Load contract from Address

Transactions recorded 3

All transactions (deployed contracts and function executions) in this environment can be saved and replayed in another environment, e.g. Transactions created in Javascript /Can be replayed in the Injected Web3.

CALLING AUCTION BENEFICIARY

ACCOUNT

GAS LIMIT

VALUE

ether

CONTRACT

Deploy

PUBLISH TO IPFS

OR

At Address
Load contract from Address

call to Auction.beneficiary

```
call  [call]  from:0x33c480606C805e25F3ba86a4B9851EF0103A6546 to:Auction.beneficiary() data:0x38a...f3eed
```

transaction hash	call10x33c480606C805e25F3ba86a4B9851EF0103A65460xAc728d6Cf961577Cb029c5A5ff15456f935f090a0x38af3eed
from	0x33c480606C805e25F3ba86a4B9851EF0103A6546
to	Auction.beneficiary() 0xAc728d6Cf961577Cb029c5A5ff15456f935f090a
hash	call10x33c480606C805e25F3ba86a4B9851EF0103A65460xAc728d6Cf961577Cb029c5A5ff15456f935f090a0x38af3eed
input	0x38a...f3eed
decoded input	{ }
decoded output	{ "0": "address: 0x33c480606C805e25F3ba86a4B9851EF0103A6546" }
logs	[]

CALLING AUCTION ITEMS

AUCTION WINNERS REVEALING

The screenshot shows the Ropsten etherscan interface. On the left, there's a form for deploying a contract. It includes fields for GAS LIMIT (3000000), VALUE (0 ether), and CONTRACT (Auction - browser/new.sol). Below these are buttons for Deploy, PUBLISH TO IPFS, and AT Address (selected). A note says "Transactions recorded" and describes how transactions can be saved and replayed in another environment. On the right, the transaction details for [block:7796255 txIndex:107] are shown. The transaction hash is 0xa65724f59153daac918b46ddc6c43ac706bbf0b1326c7e53378f76a88f48f4ed. The status is "true Transaction mined and execution succeed". The transaction details include: from 0x33c480606C805e25F3ba86a4B9851EF0103A6546, to Auction.revealWinners(), gas 24127, gas cost 24127, hash 0xa65724f59153daac918b46ddc6c43ac706bbf0b1326c7e53378f76a88f48f4ed, input 0x952...587d6, decoded input {}, decoded output {}, logs [], value 0 wei.

CALLING THE AUCTION WINNERS

The screenshot shows the Ropsten etherscan interface. On the left, there's a form for calling a function. It includes a field for call to Auction.winners(). On the right, the transaction details for [call] from:0x33c480606C805e25F3ba86a4B9851EF0103A6546 to:Auction.winners(uint256) are shown. The transaction hash is 0xa65724f59153daac918b46ddc6c43ac706bbf0b1326c7e53378f76a88f48f4ed. The transaction details include: from 0x33c480606C805e25F3ba86a4B9851EF0103A6546, to Auction.winners(uint256) 0x05211D291440e3D8022c24f8E8438880384829440xa2fb11750000000000, hash 0xa65724f59153daac918b46ddc6c43ac706bbf0b1326c7e53378f76a88f48f4ed, input 0xa2f...00000, decoded input [{"uint256": {"_hex": "0x00"}}], decoded output [{"0": "address: 0x00"}], logs [].

CONCLUSION

As we can see the results from the above screenshots it is evident that this process is highly secured and transparent as corruption rate is very low in this model and this can be adapted anywhere anytime anyplace as this model has no restriction and the whole process takes place quite swiftly in a transparent manner this model is also implemented in many banking sectors and also certified by several governments around the world another takeaway from this model is that the transaction cost very low even for big amount of transaction that is why many business companies changing to this platform.

REFERENCES

https://www.youtube.com/watch?v=SSo_ElwHSd4

<https://www.youtube.com/watch?v=ydp3sj0N0GQ>

<https://github.com/hillarymangena/Smart-Contract-for-a-Conventional-Auction/blob/master/Solidity%20source%20code>