Programming Homework 3

Foundations of Blockchain Technology and cryptocurrencies

Fall 2019

Sharif Blockchain Lab
Sharif University of Technology
Department of Electrical Engineering

Deadline: 20th Day 23:55

- Read each problem and code comments completely before coding. Provide exactly what the questions ask for.
- In the template exercise files that you are asked to complete:
  - You are allowed to add functions, variables and modifiers.
  - You are also allowed to add/delete "payable" keyword to/from functions and variables
  - You are **not** allowed to remove or rename functions
  - You are **not** allowed to change arguments passed to the functions
- Deliverables of this homework consist of a report and a set of smart contracts, and are specified in each section.
- Recommended editor for this exercise is Remix and you can compile and test your codes using Javascript VM as environment unless you're explicitly asked to use another environment (Problem 3 and Problem 5)
- If you have difficulties in submitting your answers, ask your questions in Quera.
- The policy for delayed submission is as follows: up til 24 hours after the deadline, 75 percent of the grade is given. Between 24 hours and 48 hours after the deadline, 50 percent of the grade is given. 48 hours after the deadline, submission is closed.

Problem 1 - Hello World (50 points)

Let's say Hello to the world of ethereum smart contracts. In this section, you will write a smart contract which its sole purpose is to greet people. Create a file in remix, paste Greeter.sol in it and Deploy the contract using "Hello From #YOUR_STUDENT_NUMBER" as argument (replace #YOUR_STUDENT_NUMBER with your student number).

Deliverable: Call greeting method and place a screenshot of the result in your report.

Problem 2 - Getting Familiar with ERC20 Token (200 points)

In this section you are going to implement a custom version of ERC20 token. First of all, What is an ERC20 token?

ERC20, which stands for Ethereum Requests for Comment, is a set of programming rules that all Ethereum-based tokens are expected to follow. The Ethereum Developers agreed on these three (optional) variables, six functions, and two logging events as protocol standard for the minimal viable token, in order to normalize expected behaviors while communicating across the Ethereum network. By establishing this protocol, Ethereum developers are able to more easily integrate and work with token contracts already published to the network.[1]

```
interface CustomERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount, uint expireTime) external
    returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external
    returns (bool);
    function pause() public returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
    event TimeApproval(address owner, address spender, uint256 expireTime);
    event Paused(address account);
    event Unpaused(address account);
}
```

- Approve: same as ECR20 approve with one exrta argument. Expire time is the duration (in seconds) that token owner gives allowance to spender.
- Pause: Pauses contract all external and public functions.

Complete CutomERC20.sol in order to satisfy requirements.

Deliverable: CutomERC20.sol

Problem 3 - eBook Library (300 points)

The manager of an eBook library has decided to lend his books through Ethereum smart contract to show that he follows the latest technology. He wants this smart contract to support the following functionality:

- The eBook library doesn't accept Ethereum and has its own token named "ELT" . all users need to exchange their Ether for ELT in order to use the library services.
- The admin can add books to the smart contract.
- A book is accessible by 1 user at the same time and until the access of the last user to the book hasn't been revoked, no one else can access this book even though he/she pay for the book.
- All the books are determined by a unique ID. The contract keeps a list of all books and their specifications including their validity (means that if this bookId has been added by admin before), their genre and their access fee. All the prices are in ELT.
- Any user that wants to access a book, should call "access" function and determine the Id of the book he/she wants to access and pay the access fee of that book. If the Id is valid and the book is available (not in access by someone else) access to this book will be granted to the user for an "accessDuration".
- The smart contract keeps a mapping of all rented books to the information of the user which has access to them and the "accessStartTime" and if the book is available or not.
- The "accessDuration" is defined by the admin, and after this passes the admin can revoke the user's access to the book.
- A user can call "changeToPremium" function and pay "premiumFee" in SLTs and become a premium user.
- "premiumFee" is determined by admin in the constructor.
- Premium users have a special privilege. if they want access to a book that is rented by a regular user, the access of that regular user to that book is revoked and the access fee is transferred to his/her ELT account and the access of the book is granted to the premium user. But if the current renter also has a premium account, the new user can't access the book. In other words, premium users have a priority over regular users but they don't have any priority over other premium users. notice that premium users also should pay for any book they rent like regular users.
- "EBookLibrary" contract is a child of the "CustomERC20" contract which you completed in problem 2. Inheritance in solidity study this tutorial.

  https://www.tutorialspoint.com/solidity/solidity_inheritance.htm
- There is a function "deposit" in the "EBookLibrary" contract which accepts Ether and allocates the sender the same amount of ELTs.

– Any user who wants to exit the library, can call the "withdraw" function and withdraw his/her unspent ELTs in Ether.

Example scenario:

1. Sara creates the smart contract by this input: ("600","EBookLibraryToken" , "ELT","18","10")
2. Sara adds book ("1", "1", "4") as admin.
3. Ali deposits 4 ELTs and then access the book "1".
4. Mohammad deposits 20 ELTs and then change his account to premium.
5. Mohammad wants to access the book "1". The access duration for Ali hasn't ended but because Mohammad is a premium user he can access this book and Ali's ELTs are refunded to him.

Complete EBookLibrary.sol to satisfy the properties above. Use "require" or "modifier" to ensure that a transaction reverts in case of wrong executions.

Deliverable: EBookLibrary.sol

Problem 4 - Deploy to a testnet (50 points)

In this section, we are going to deploy the EBookLibrary app to Reposten public testnet.

Prerequisites:

- Download MetaMask extension from Chrome Web Store.
- Go through MetaMask setup steps
- After setup select Ropsten Test Network from the drop down on top of the extension.
- Click on deposit and then GET ETHER to get your first test net ethers.
- in remix change the environment to Injected Web3.
- if everything goes right you should be able to see your account and its balance.

Deploy the contract to Ropsten network:

- Deploy the EBookLibrary to Ropsten Network using remix.
- MetaMask catches your request and asks you to confirm it. You can tweak some variable here too.

If everything goes right, after the confirmation your contract will be deployed to Ropsten and you can trace your transaction on Etherscan using the links provided in remix as well as MetaMask. Make two more transaction as the admin adds books ("1","0","3") and ("2","2","10")

Deliverable: Stipulate the address of your contract and the hash of your transactions in your report.

Problem 5 - Let's Do Some Hacking (300 points)

Part 1 (150 points)

You are provided with the code for a voting voting smart contract. This smart contract suffers from a security hole which can cause it to show wrong number of votes for a candidate in certain conditions. (Hint: there is no restriction on the number of users who can vote)

Deliverable: A description of the security hole and a proposed solution to prevent it in your report (no coding).

Part 2 (150 points)

"withdraw" function of "EBookLibrary" contract is vulnerable to an attack, and an attacker can drain out the balance stored in the contract. Exploit it and then change this function to fix it.

Deliverable: EBookLibrary.sol, and an explanation about the attack and your solution in your report.

Problem 6 - Example Dapp (100 points)

In DApp folder you can find an application which connects a user interface to blockchain using ethereum's web3.js library. In order to test our DApp we are going to use a blockchain emulator called Ganache.

Prerequisites:

- Download Ganache from here and start it.
- In remix change the environment to Web3 Provider and accept the default endpoint (if you can't connect to the default endpoint try http://localhost:7545).
- If everything goes right you should be able to see ten accounts with balance 100 in the balance drop-down in remix.

Deploy the contract to Ganache:

- Deploy the voting contract (given in problem 4) to Ganache using remix.

Connect the DApp to your contract:

- Open DAPP/js/index.js

- Copy the contract address and paste it in the contractAddress field (line 12).
- Copy the contract ABI and past it in the abi field (line 16). You can copy it from "SOLIDITY COMPILER" tab in remix.
- Make sure web3 provider port is correct (line 9),
- Open voting.html

You should be able to see a working voting DApp.

Deliverable: vote to the candidates several times using the DApp UI and place a screenshot of the DApp (after your votes) in your report.

References

[1] Advanced Cryptocurrency Topics: ERC20 Interface
`https://medium.com/setocean/advanced-cryptocurrency-topics-erc20-interface-7372e97f4a42`