CS 5594
Homework Assignment 3

Given: Mar 28, 2024 **Due:** Apr 11, 2024

General directions. The point value of each problem is shown in [ ]. Each solution must include all details and an explanation of why the given solution is correct. The completed assignment must be submitted on Canvas as **a ZIP containing your answers (in a single PDF) and all code files that you developed** by 11:59 PM on Apr 11, 2024. **No late homework will be accepted.**

Digital preparation of your solutions is mandatory. No matter how you prepare your homework, **Please include your name.**

[20] 1. Each smart contract function costs gas (i.e. a transaction fee) to be executed. For example, a function with 1000 loop iterations will cost much more than a simple addition function.

A. Why do some blockchains (e.g., Ethereum) have gas fees? What determines how gas is calculated?

B. Provide some examples of design choices that will affect gas fees.

---

[25] 2. All smart contracts for a blockchain are executed on its virtual machine. For the Ethereum blockchain, this is the Ethereum Virtual Machine (EVM).

A. At a high level, describe how smart contracts get executed on blockchains. How do these virtual machines work?

B. How many blockchain nodes execute each smart contract function? Does only one blockchain node execute each smart contract function and broadcast the result? Do all blockchain nodes execute every function?

C. What happens if a malicious node executes a smart contract function incorrectly and broadcasts it? Why can users of a blockchain trust that all smart contracts are executed correctly?

---

[40] 3. Follow the instructions below to complete the skeleton code for an Auction implementation of a smart contract in Solidity.

Complete Course #1: Making the Zombie Factory and Course #2: Zombies Attack Their Victims at https://cryptozombies.io/en/course/ to learn the basics of coding smart contracts with the Solidity language (which is the one used for the Ethereum blockchain).

We have provided template code for an auction smart contract on Ethereum which uses the Hardhat (https://hardhat.org) framework. Create a new directory and decompress the skeleton.tar into that directory. Install dependencies using a package manager (npm install or yarn install). If you are unfamiliar with how to do this, there are plenty of resources available online.

In your terminal, navigate to the location of the decompressed skeleton.tar file. To check that everything is installed correctly, you should be able to execute npx hardhat --version and have the Hardhat version printed.

We also highly recommend downloading and using a Solidity syntax highlighting extension for your IDE (such as Solidity - Visual Studio Marketplace for VS Code). Within this repository, the skeleton code for the auction smart contract can be found at contracts/Auction.sol in the figure below.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.6.0;

contract Auction {
    address public winningAddress;
    uint256 public AUCTION_DEADLINE;

    //TODO: You may add any other variables here, if necessary.

    constructor() public {
        AUCTION_DEADLINE = block.timestamp + 30 days;
    }

     # Ignore any bids that are submitted when block.timestamp is greater than
AUCTION_DEADLINE
    function submitBid(uint amount) public {
        //TODO:
    }

    # sets winningAddress to the address that called submitBid(uint amount) with
the highest input amount
    # Do not allow winningAddress to be set at a time before AUCTION_DEADLINE
    function calculateWinner() public {
        //TODO:
        // winningAddress = //TODO:;
    }
}
```

A. Complete the missing logic in this skeleton code to implement a simple auction that supports the functionality below. Only add code where there are TODO comments and do not change any function names, variable names, parameters, etc. Leave all other files in this repository unchanged. You should only make edits to contracts/Auction.sol.

Functionality of the auction:

- **When calculateWinner() is called, it sets winningAddress to the address that called submitBid(uint amount) with the highest input amount.**
- **Ignore any bids that are submitted when block.timestamp is greater than AUCTION_DEADLINE.**
- **Do not allow winningAddress to be set at a time before AUCTION_DEADLINE.**

<u>Hints:</u> **Learn to use block.timestamp, msg.sender, and require().**

- **To compile the contract, you can run npx hardhat compile**
- **To unit test the contract, you can run npx hardhat test ./test/Auction.ts**

Submit your completed Solidity smart contract file named as "Auction.sol".

---

[40] 4. The smart contract in the figure below is to be used as a third party to donate funds to other users.

A. Research common smart contract vulnerabilities and explain in detail why this contract is not secure and how an exploiter can exploit this contract to steal all the funds

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

contract Vulnerability {
  mapping(address => uint) public balances;

  function donate(address _to) public payable {
    balances[_to] = balances[_to] + msg.value;
  }

  function balanceOf(address _who) public view returns (uint balance) {
    return balances[_who];
  }

  function withdraw(uint _amount) public {
    if(balances[msg.sender] >= _amount) {
      (bool result,) = msg.sender.call{ value:_amount }("");
      if(result) {
        _amount;
      }
      balances[msg.sender] -= _amount;
    }
  }

  receive() external payable {}
}
```

deposited in the contract. We have provided the exploiter contract at contracts/ VulnerabilityExploit.sol. Try to analyze the logic of this exploiter contract and how it might interact maliciously with the vulnerable contract (contracts/Vulnerability.sol).

Hints: If a function from one contract is called from another contract, msg.sender will be the calling contract's address. The unnamed receive() external payable function is a fallback function (if you are unfamiliar with a fallback function, research what a fallback function is).

(Include the answer to this question in the file named Answers.PDF).

B. Similar to the auction question, we have provided the source code for this contract in the repository at contracts/Vulnerability.sol. Fix the contract to make it secure and not prone to the vulnerability described in part (A). Only edit contracts/Vulnerability.sol and do not change any function names, variable names, parameters, etc. Leave all other files in this repository unchanged. You should only make edits to contracts/Vulnerability.sol.

- **To compile the contract, you can run npx hardhat compile.**
- **To unit test, you can run npx hardhat test ./test/Vulnerability.ts.**

(Submit your completed Solidity smart contract file named as "Vulnerability.sol").