14-741/18-631: Homework 4

Assigned: Monday Nov 12, 2018
Due: Wednesday Nov 28, 2018

Name:

Andrew ID:

Scores

Problem 1 (50 pts max):

Problem 2 (50 pts max):

Total (100 pts max):

Guidelines

- Be neat and concise in your explanations.
- You must use at most one page for your explanation for each Problem (code you wrote may be on additional pages). Start each problem on a new page. You will need to map the sections of your PDF to problems in Gradescope.
- Proofs (including mathematical proofs) get full credit. Statements without proof or argumentation get no credit.
- There is an old saying from one of my math teachers in college: "In math, anything that is only partially right is totally wrong." While we are not as loathe giving partial credit, please check your derivations.
- **This is not a group assignment. Feel free to discuss the assignment in general terms with other people, but the answers must be your own.**
- Write a report using your favorite editor **Only PDF submissions will be graded.**
- Submit to **Gradescope** a PDF file containing your explanations and your code files before 7:30pm Eastern on the due date. Late submissions incur penalties as described on the syllabus (first you use up grace credits, then you lose points).
- Post any clarifications or questions regarding this homework in Piazza.
- Good luck!

1 Using Tor (50 pts)

Install Tor on your machine. Download instructions are available from `https://www.torproject.org/projects/torbrowser.html.en`. Do **not** configure your Tor machine as a server (relay).

This exercise will make use of the Stem library (`https://stem.torproject.org/`), a Python library that allows you to directly talk to the Tor controller. If you do not have any familiarity with Python, relax: it is really an easy language to pick up, nothing in this problem requires very advanced programming skills in Python, and there are tons of online resources to help, such as the Python tutorial `https://docs.python.org/2/tutorial/`. You'll find the PycURL library of use.

1.1 Preliminaries

1. (10 pts) (Before starting Tor) What is your public IP address? Show the output of a `whois` query on that address. Google how to do this.

2. (10 pts) Provide a Python script that, using the Stem library, displays:

   (a) A list of all Tor circuits your machine currently uses, as specified by the list of three Tor relays. An entry should look like "entry node–middle node–exit node."

   (b) For each circuit:

   - The IP address of each relay in the circuit
   - The location (country) of each relay
   - The current bandwidth of each relay

   Provide both your python script **and** its output.

1.2 Using exits to circumvent censorship

1. (10 pts) Try to connect to `http://ctf.martincarlisle.com`. If you connect directly, you will likely to get 403 error. To circumvent this "block," you will attempt to force the use of specific exit nodes. Write a Stem script to constrain Tor to specific exits in different countries. Provide your Python/Stem script in your handout. What is the secret code?

2. (10 pts) Use your script to find as many countries as possible (at least, five) in which the website `http://ctf.martincarlisle.com` is *not* blocked. Provide a list of these countries, the IP address of the exit node that you used, as well as the exact date/time at which you made each request verifying that each of these countries was not blocked.

3. (10 pts) You could also access the blocked site using a VPN. VPN also encrypts traffic and provides some level of anonymity. Explain why one would prefer Tor over VPN. Please base your explanation on security properties that users aim to achieve and/or TCB of the system to achieve those properties.

   Important note: We are logging all requests to the server and have reasonably good fingerprinting techniques. So, we can probably tell whether or not you are cheating on the exercise, so don't cheat! (One program per person, and don't run your program for friends). Also try to be considerate of others and do not flood the server with requests. (Attempting to hack into the server, it goes without saying, would be a major academic integrity offense, and would be treated as such). This should also act as incentive for you to start this problem early.

2 Smart Contracts (50 pts)

Smart contracts are a way to enforce and verify contracts in a transparent, decentralized way that avoid conflicts and trusted third parties. Contracts primarily deal with cryptocurrencies, but they can be extended to deal with other goods as well. Formally, a contract defines the rules of an agreement. They are implemented as programs and stored on a blockchain. Once they are stored, they cannot be modified.

To understand how these work, let's take a rental agreement between a landlord and a tenant for example. The tenant pays a security deposit when they first start their tenure. This deposit is stored within the contract. The contract collects monthly rent (in terms of bitcoin, ether, etc.) from the tenant and allows the landlord to withdraw the rent amount. If a tenant fails to pay rent for a particular month, the security deposit is handed over to the landlord. When a tenant's tenure is over, if they've paid their rent regularly, the security deposit is handed back to them. The contract ensures that landlord gets the deposit if and only if a tenant defaults on their payment. Likewise, if a tenant clears all their dues, the tenant is guaranteed to get back their security deposit. All transactions are logged on the blockchain and cannot be refuted. This goes to illustrate the power of smart contracts. In this homework, we'll write some smart contracts and study their attacks/defenses.

We will use Ethereum smart contracts for this homework. You can read about Ethereum here:

`https://github.com/ethereum/wiki/wiki/White-Paper`.

`https://solidity.readthedocs.io/en/v0.4.25/introduction-to-smart-contracts.html`

Note: Please use the VM that we've provided you. Setting up a development environment can be painful and there are several inconsistencies between different versions of Node, Truffle, etc. Blockchain development is still at a nascent stage and finding help with debugging/troubleshooting can be a challenge. The VM is available on Canvas in the hw4 folder. It is called **infosec-hw4-vm.zip**. The contracts for the homework are already in the VM, but have also been placed in the hw4 folder for your convenience. The VM username is *ctfuser* and password is *inictfroxftw*.

2.1 Development Toolchain (10 pts)

Smart contracts in Ethereum are written in a langauge called Solidity. We will compile our contracts using Truffle, which is a development framework for Ethereum. Since deploying our contracts on a real blockchain would take time and cost money, we'll use TestRPC, which is used for testing deployments locally. We're now going to write and test a simple contract.

1. Lets make a new directory for our Ethereum project at a location of your choice.

   ```
   $ mkdir smartcontracts
   $ cd smartcontracts
   ```

2. Initialize the Ethereum project.

   ```
   $ truffle init
   ```

   You should see three directories being created.

   - contracts - stores all your contracts.
   - migrations - stores migration files which contain information on how to deploy contracts onto a blockchain
   - test - used for writing test cases

3. We've provided you with three files called `Vault.sol`, `VacationFund.sol`, and `AttackVacationFund.sol` on the Desktop of the VM. Copy *all three* files to the contracts directory. Carefully study the contents of `Vault.sol`. You are expected to Google Solidity syntax/-functions and understand what's going on.

4. Run the following command from the project root directory (the smartcontracts folder you created) to check if your contract compiles.

   ```
   $ truffle compile
   ```

   You should see "`Writing artifacts ./build/contracts`" if the compilation was successful.

5. Now we need to be able to specify how to deploy our contract on the TestRPC blockchain. Copy the file called `2_deploy_contracts.js` from the Desktop to the migrations directory.

6. Replace `truffle.js` file in the project root with the file we've provided on the Desktop. This will setup the development network.

7. Now, without closing your current terminal window, open a new terminal and run the `testrpc` program. Go back to the original terminal.

8. We now run the truffle development console.

   ```
   $ truffle console
   ```

We shall now deploy the contract onto the TestRPC blockchain. Then, using test accounts provided by TestRPC, we'll send funds to and from the contract. This exercise should help you become more familiar with Ethereum smart contracts. Now answer these questions by carefully following the instructions provided. These commands must be run in the development console.

1. (3 pts) You have to first deploy your contract onto TestRPC. This is done by typing `migrate`. What output do you see? Do you see any output on the terminal running TestRPC? Show us a screenshot of both the outputs. Why is both a block hash and a transaction hash included?

2. (1 pts) On the development server, we are provided with 10 accounts to play with. We now assign the first two accounts that TestRPC creates for us to variables so they're more easy to access. Run

   ```
   truffle(development)> acc0 = web3.eth.accounts[0];
   truffle(development)> acc1 = web3.eth.accounts[1];
   ```

   Next, we will construct a function that can be used to get the balance in Ether of an account. This function takes an address and returns the Ethereum balance of that address as a string.

   ```
   truffle(development)> getEthBal = addr => web3.fromWei(web3.eth.getBalance(addr).toString());
   ```

   You can read the balance of acc0 in ether by using the following command,

   ```
   truffle(development)> getEthBal(acc0);
   ```

   What are the balances of the acc0 and acc1?

3. (1 pts) Now we will deposit 1 ether from acc0 into the Vault contract. First, let us assign the deployed contract to the variable `thevault`.

   ```
   truffle(development)> Vault.deployed().then(contract => thevault = contract);
   ```

Note that the variable `thevault` is not directly bound to the address of the deployed Vault contract; thus, to get the contract's balance, we have to do this:

```
truffle(development)> getEthBal(thevault.address);
```

Here is how we can call methods defined on Vault contract:

```
truffle(development)> thevault.deposit({from:acc0, value:web3.toWei(1, "ether")});
```

What output do you see? What is the balance of acc0? What is the balance of the Vault contract?

4. (2 pts) Observe the command from 3 closely and try modifying it such that you now send 5 ether to the contract. What happens? Which line of Vault.sol caused this?

5. (1 pts) Now try to withdraw some ether from the account. Run the following command to withdraw 1 ether from the contract and deposit it into acc0.

```
truffle(development)> thevault.withdraw(web3.toWei(1, "ether"), {from:acc0});
```

What output do you see? What's the new balance of acc0?

6. (2 pts) Contracts typically store values in Wei. What is the conversion rate between Wei and Ether? Tell us how you figured this out (give us a reference, command, etc).

2.2 Attacks and Defenses (40 pts)

Attackers have mounted a number of high-profile attacks on Ethereum smart contracts, which resulted in millions of dollars worth of ethers being stolen. The goal of this assignment is to demonstrate how such attacks could happen and how to defend against them. You can Google these attacks and propose defenses to complete this assignment. A very helpful page is at `https://solidity.readthedocs.io/en/v0.4.25/security-considerations.html`

1. Alice wants to organize a vacation with her friends. She reads about smart contracts and decides that it's a great way to collect funds for the trip. She writes a contract for this purpose; the VacationFund contract you saw earlier. (Recall that you can access this contract in the truffle console by assigning it to a variable; i.e. `VacationFund.deployed().then(contract => myvf = contract);`)

   (a) (5 pts) What does each function in this contract do? Explain in detail.

   (b) (1 pts) What is the target amount of funds that Alice is trying to collect for the trip?

   (c) (3 pts) What happens if fewer than 10 people sign up for the event?

   (d) (3 pts) What happens if more than 10 people sign up?

   (e) (2 pts) What does selfdestruct() do?

2. (12 pts) VacationFund is vulnerable. We've provided you with an outline of the attacker's contract called AttackVacationFund. Fill in the fallback function (marked as TODO) with code that would help you receive **more funds than you originally contributed**. You not may modify AttackVacationFund besides filling in the fallback function. You also may not modify VacationFund.

   Perform your attack in the VM. To do so, first recompile the contracts and perform the initial setup again:

   (a) Disconnect the Truffle console from TestRpc (press `ctrl+d` at the Truffle console)

(b) Recompile the contracts with `$ truffle compile`

(c) Restart TestRpc by quitting the program (press `ctrl+c`) and running the `testrpc` command

(d) Restart the truffle console; `$ truffle console`

(e) Perform the migration; `truffle(developement)> migrate`

Note that all account balances will be reset and all variables will need to be redefined every time you perform this reset.

List **all of the commands** necessary to perform your attack, along with an explanation of how your attack works.

Hint: Google the DAO attack from 2016.

3. (10 pts) How would you fix VacationFund so that it is not vulnerable to your attack? You do not need to write code to answer this, but your answer must be clear and precise.

4. (4 pts) List two best practices for writing smart contracts.