14-741/18-631: Homework 8

All sections (A/B/SV) Due: Thursday May 6, 2021 by 10:39am EST

Name:

Andrew ID:

Total (100 pts max):

Guidelines (Please read before starting!)

- 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.
- To access the CTF problems, create the SSH proxy as per the guide on Canvas.
- For CTF problems, **you must use the following format in your explanation**:
  - CTF Username
  - Flag
  - Explain the vulnerability in the program, and explain conceptually how that vulnerability can be exploited to get the flag.
  - How did you exploit the vulnerability? List the steps taken and the reasoning behind each step. The TA grading should be able to replicate the exploit following the steps. Feel free to make references to your code! **Note that *"use XYZ online solver"* is not sufficient - you must explain how the online solver derived the answer for full credit.**
  - Append your source code in the same writeup. Your source code should be readable from the writeup PDF itself. Note that this does not count towards the page count above.

  Omitting any of the above sections would result in points being deducted.
- Some questions are marked as **Team work**. For those, you will be asked to come up with a joint write up/solution. Only one needs to put the solution in the write up, and the other simply write "*see [teammate andrewid]*". However, individual CTF username and flag still need to be put in the write up for CTF questions. This only applies to questions marked as "Team work".
- It is highly recommended that you use Python for your assignment. You may use other languages that you are familiar with, but the teaching team will not be able to support or debug language specific errors.
- Please check your English. You won't be penalized for using incorrect grammar, but you will get penalized if we can't understand what you are writing.
- 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 partially right is totally wrong.” While we are not as loathe to give partial credit, please check your derivations.
- Write a report using your favorite editor. Note that **only PDF submissions will be graded.**
- Submit to Gradescope a PDF file containing your explanations and your code files before 10:39am Eastern Standard Time on the due date. You can find the timing for EST here: `https://time.is/EST`. Late submissions incur penalties as described on the syllabus (first you use up grace credits, then you lose points).
- If you choose to use a late day, you do not have to inform the instructors. We will calculate the number of late days used at the end of the semester based on the time of submission on Gradescope.
- Post any clarifications or questions regarding this homework to Piazza.
- **Team work** As a team, beyond designated team questions, you are encouraged to shared resources (e.g., TA’s help, online resources you found helpful); you are encourages to set up virtual study sessions with your teammate(s) to check each other’s progress and discuss homework assignment solutions.
- **This is not a group assignment. Beyond your teammate, feel free to discuss the assignment in general terms with other people, but the answers must be your own.** Our academic integrity policy strictly follows the current INI Student Handbook `http://www.ini.cmu.edu/current_students/handbook/`, section IV-C.
- Good luck!

1 Smart Contracts (80 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 have paid their rent regularly, the security deposit is handed back to them. The contract ensures that the 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 their security deposit back. All transactions are logged on the blockchain and cannot be refuted. This goes to illustrate the power of smart contracts. In this homework, we will 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/v0.5.0/introduction-to-smart-contracts.html`

Note: Please use the VM that we have provided to you (the VM is different from the one provided for previous homeworks). 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 will be available on CMU Box (the link will be provided on piazza).

The VM username is *ctfuser* and password is *inictf2020iirc!*. Please check piazza for the latest information.

1.1 Development Toolchain (20 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 will use TestRPC, which is used for testing deployments locally. We are now going to write and test a simple contract.

1. Let's 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 already 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 have 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-config.js` file in the project root with the file we have 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 will 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. (6 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. (2 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 are more easy to access. Run

   ```
   truffle(development)> web3.eth.getAccounts().then(function(result) {
   acc0 = result[0];
   acc1 = result[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 = async addr => {
   var balance = await web3.eth.getBalance(addr);
   return web3.utils.fromWei(balance, "ether"); }
   ```

   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. (2 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.utils.toWei("1", "ether")});
```

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

4. (4 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. (2 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.utils.toWei("1", "ether"), {from:acc0});
   ```

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

6. (4 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).

1.2 Attacks and Defenses (60 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/v0.5.0/security-considerations.html`

1. (20 pts) Alice wants to organize a vacation with her friends. She reads about smart contracts and decides that it is 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) (6 pts) What does each function in this contract do? Explain in detail.

   (b) (3 pts) What is the target amount of funds that Alice is trying to collect for the trip?

   (c) (4 pts) What happens if fewer than 10 people sign up for the event?

   (d) (4 pts) What happens if more than 10 people sign up?

   (e) (3 pts) What does selfdestruct() do?

2. (15 pts) VacationFund is vulnerable. We have 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. (15 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. (10 pts) List two best practices for writing smart contracts.

2 Almost Ready to deploy - Management & Assurance (20 pts)

TA Arjun is trying to setup a Course Website for Information Security Spring 2021 session. He chooses to use Flask because he is very familiar with it. However, since his term is coming to an end, he is looking for someone to take over.

Team Work. Feel free to discuss this question with your teammates. Please send in your own write up.

You have volunteered to help setup the website for the next semester. You replaced the content on the website with the correct & super secure content, sent it for approval to professor Hibshi. Professor Hibshi informs you that she deployed the website and it has a vulnerability. She leaves it for you to discover and fix it. She mentions the following xkcd post that she really likes and may be relevant to this issue during the conversation. `https://www.explainxkcd.com/wiki/index.php/2347:_Dependency`

Note: Arjun's starter code can be found in the *14741PatchingHW.zip* located on the Desktop of the problem VM. Please refer to the included README.md (in the zip) for instructions for deployment.

2.1 Find Vulnerability (6 pts)

Find the vulnerability in Arjun's base setup:

1. Explain the vulnerability. (Look for a RCE Vulnerability)
2. Explain how you found it.
3. Explain how to exploit it. (Exec something like 'ls')

Note: In order to find the vulnerability, you don't need to go and read any of the python dependency source code or the included HTML code (look at the dependency from a higher level). The vulnerability lies in the flask server code. Take a look at question 1.2 and that might help you with the scope.

2.2 Possible Fixes (8 pts)

Since you recognize the vulnerability, now consider the following scenarios:

1. You have access (R/W) to the source of the website (including app.py). However, you cannot update anything else.
   (a) Patch the source to remediate the vulnerability. Provide the changes you made.
   (b) Explain how your Patch works.
2. You do not have access to the source code of the website. However, you can update any python packages installed on the deployment machines (can edit requirements.txt).
   (a) Which packages would you update to remediate the vulnerability?
   (b) Explain the effects of applying this updated dependency and how it fixes the vulnerability.

2.3 Lessons Learnt (6 pts)

1. What kind of testing did you perform to find the flaw? Explain you response. (WhiteBox, BlackBox, GreyBox etc...)
2. In real life situations, which techniques can the developers employ to prevent such bugs from slipping by? (List at least 2 techniques)