COMSW 4182: Security II **(Due: 02/19/21)**

Homework Assignment #1

Instructor: Suman Jana *Name: , UNI:*

Course Policy: Read all the instructions below carefully before you start working on the assignment, and before you make a submission.

- Late policy: a late submission on the due date will lose 5%; on the next day 10%; two days late 20%, three days late 30%; after that, zero credit.
- Due date: **02/19/21 11:59PM(EST)**. After that submissions will be counted late. It does not make any difference if it's only 10 minutes or 23 hours.
- You are not allowed to discuss the solutions to homework problems/programming assignments with your fellow students.

Submission: You should tar your code and documents into a tar.gz file named as `hw1_UNI.tar.gz`. Also don't forget include your name and UNI in the `README` file.

Google cloud: All the assignments are designed to run under Ubuntu 20.04. To solve the environment issues, we use google cloud for the running environment. The detailed setup can be found in coursework announcements.

Problem 1: Password Cracker

(20 + 20 = 40 points)

An attacker exploited an unknown zero-day vulnerability to steal some entries of `/etc/shadow` from a remote server. Meanwhile, he or she happened to know the length of these leaked passwords and their possible character sets through some sophisticated social engineering. The victims on that remote server were lazy and refused to set up long passwords with some special characters, which is a bad yet common practice for many people in real world. However, this gives attacker a great chance to quickly crack their passwords. You task is to write a simple password cracker in C or C++ to find out their passwords.

(a) The passwords are to be 8 numerical digits(i.e., 0-9), representing users' birth date. The encryption algorithm used to compute the hash is sha512. So the equation to compute the hash is $sha512(password|salt)$, where | denotes the string concatenation.

(b) The passwords are to be 6 characters(including all possible upper and lower case letters i.e., a-zA-Z). The encryption algorithm used to compute the hash is sha3_512. So the equation to compute the hash is $sha3\_512(password|salt)$, where | denotes the string concatenation.

Hint:

- You will be given a `/etc/shadow` which contains user name, salt, password hash and other informtaion. Figure out the format of `/etc/shadow` before you get started.
- You might notice that the generated hash may contain some non-printable characters. In order to represent them into a text file, they are converted into *base64* encoding. In `/etc/shadow`, all password hashes are represented in base64 format. The padding characters "=" at the tail of base64-encoded hashes are removed.
- Use encryption function of OpenSSL library to compute the hash.
- Consider to use parallel computing to boost your password cracker by splitting the task into multiple sub-tasks.

Deliveries:

- source code of your password cracker

- a Makefile to build your tool
- a README document to explain your design
- a script to **automatically compile, run and print out** the result

Problem 2: OpenSSL Server and Client (20 + 20 = 40 points)

OpenSSL is a library to provide secure network communication. It also provides many standalone tools to perform encryption/decryption, generate public/private key and certificates. You have exercised OpenSSL api to compute password hash in Problem 1. In this task, you will use OpenSSL tools to establish a secure connection between a server and client , then transfer a file along with its signature, in the end verify the file against its signature.

(a) SSL connection is based on trusted certificates. In this problem, our connection is based on certificates which are created by yourself. Specifically, you need to first create a root certificate, then use it to create and sign an intermediate certificate. Further, you will create and sign a client and server certificates using the intermediate certificates you just created.

(b) Once the certificates are generated, we can establish secure communicate between client and server. You will use OpenSSL tools `s_server` and `s_client` to launch secure SSL connection with the certificates you have created in part (a).

- On the server side, create a directory named `server`, enter into the directory. Next, create a file `test.txt` containing your name and uni. Then generate a signature file `test.sign` for that file using server's private key.
- On the client side, create a directory named `client`, enter into the directory. Next, send a simple HTTP request to file `test.txt` and `test.sign`. In the end, verify the file against the received signature using OpenSSL tool `dgst`

Hint:

- There are many materials online about how to set up self-signed certificates. Here is a nice blog https://jamielinux.com/docs/openssl-certificate-authority/introduction.html.
- Use HTTP request method to obtain files from server. You can directly input HTTP request message inside `s_client`.
- The file signature may contain many non-printable characters and is not encoded into some text format like base64. Hence you cannot directly view or examine its content from stdout of `s_client`. Think of IO redirection.

Deliveries:

- two bash scripts to reproduce the each tasks
- the bash scripts should include basic comments
- a README document to explain your design
- a script to **automatically run and print out** the result