COMP8760 Week 17 Class Worksheet

Budi Arief (b.arief@kent.ac.uk) – Based on materials by Shujun Li and Pierre Mondon
Class Supervisors: Pierre Mondon (ppm5@kent.ac.uk) and Adel ElZemity (ae455@kent.ac.uk)

Friday 1 December 2023

In this week's class, you will work on some exercises beyond passwords to gain a better understanding of user authentication as a whole. The first four exercises are easier and mandatory (and should be included in your CW2 practical report). The fifth exercise is more advanced and optional.

Like in the last week's class, for this week's class exercises, **you are encouraged to use existing third-party libraries and tools as much as you can**, as long as you will still write some source code yourself to get the exercises done. You should use Google and/or other search engines to get help and look for hints. Work independently first before asking for help.

For all exercises, feel free to show the work you have completed to class supervisors; but if you are confident with your results and do not feel the need to get them checked, that is also fine.

Note: This class is the second class that you will need to include in your CW2 practical report (i.e. the log book). *Please keep any evidence of the work you achieved (e.g. as screenshots or answers to specific exercises), and include them in the practical report that you will submit as part of CW2.*

Exercise 1 – Password with emojis

Based on Exercise 1 from last week's class, add the support of emojis in your passwords. Note that emoji characters are defined in the Unicode standard (for more information, have a look at https://en.wikipedia.org/wiki/Emoji#Unicode_blocks) so you need to use a character encoding scheme such as UTF-8 to encode them and display them properly. If your command line does not work, consider building a simple web-based application (e.g., an application producing a UTF-8 encoded HTML page) to demonstrate entering a password with emojis.

Hint: What emojis can be shown on your system depend on what emojis are covered by the default font you are using. If you want to support emojis not covered by your font, consider installing a new font or using pictures of such emojis instead. For the latter, you will need to have a way to allow a picture added to a textual password, e.g., by representing each emoji picture as a specially encoded file path. This will effectively lead to a mini-version of PassInfinity (https://www.passinfinity.com/).

Exercise 2 – Password as a file on your computer

Write two functions and a script to demonstrate how a user can select a local computer file as her/his password: one function for user registration, the other function for login, and the script to invoke the two functions to demonstrate how a local computer file can be used as the password for both registration and login. For login, try a wrong computer file to demonstrate that the user will be rejected.

Reference: Mohammad Mannan and P.C. van Oorschot, "Digital Objects as Passwords," HotSec 2008, USENIX Association,
https://www.usenix.org/legacy/event/hotsec08/tech/full_papers/mannan/mannan_html/

Exercise 3 – A toy one-time-password (OTP) generator

Using any of your password hashing program implemented so far, implement a toy OTP scheme as follows:

- Immediately after user registration

    - The server stores the hashed and salted password as HSP=H^n(password || salt) as usual, where H(.) is a hash function and || indicates byte/string concatenation.
- For each login
    - The server updates the HSP as follows: HSP = H(HSP || t), where t is the current date and time in any format you decide.
    - The server calculate an OTP as the last 6 bytes of HSP and send its hexadecimal representation to the user via SMS.
    - The user enters the received OTP to log in.
    - The server checks the user's input to decide accept or reject the user.

In the above scheme, you don't need to implement the real SMS feature, but just assume the user get the OTP and write a script to demonstrate that if the right OTP is given the user will be accepted; otherwise (s)he will be rejected.

Additional questions (optional): Think about how secure the above scheme is if a hacker managed to hack into the server once but loses access later (e.g., the original hack was due to a software vulnerability which is now fixed, but the server administrator is unaware of the original hack). Assuming the server setting does not change since the hack, will (s)he still have a better chance to impersonate users on the server without breaking into the server again? If yes, can you think of a way to improve the security of the OTP scheme? Does the user really need to provide a password to register for an OTP scheme to work?

Exercise 4 – Evaluating performance of a mock biometric-based authentication system

Imagine you are using a biometrics-based user authentication system and would like to evaluate its performance for a specific user (let's say, Alice). Below are some data you obtained from some earlier experiments:

- For 100 biometric templates Alice produces for login attempts, the distance between those templates and the enrolled template of Alice is as follows:
  [0.27, 0.079, 0.27, 0.36, 0.25, 0.3, 0.27, 0.17, 0.23, 0.12, 0.29, 0.085, 0.093, 0.12, 0, 0.34, 0.23, 0.12, 0.34, 0.029, 0.19, 0.18, 0.23, 0.23, 0.11, 0.2, 0.18, 0.26, 0.31, 0.31, 0.11, 0.21, 0.079, 0.089, 0.2, 0.35, 0.12, 0.24, 0.18, 0.31, 0.091, 0.2, 0.26, 0.31, 0.35, 0.21, 0.051, 0.13, 0.094, 0.44, 0.14, 0.27, 0.18, 0.29, 0.12, 0.06, 0.058, 0.25, 0.18, 0.18, 0.34, 0.23, 0.22, 0.36, 0.12, 0.27, 0.28, 0.18, 0.22, 0.083, 0.085, 0.21, 0.27, 0.46, 0.13, 0.22, 0.19, 0.0067, 0.16, 0.021, 0.28, 0.11, 0.21, 0.15, 0.23, 0.14, 0.25, 0.27, 0.37, 0.18, 0, 0.12, 0.34, 0.093, 0.3, 0.21, 0.34, 0.0039, 0.18, 0.079]
- For 100 biometric templates Eve (a mock attacker who pretended to impersonate Alice) produced, the distance between those templates and the enrolled template of Alice is as follows:
  [1.2, 0.77, 0.88, 0.39, 0.51, 0.55, 0.82, 0.54, 0.74, 0.19, 0.53, 0.44, 0.28, 0.7, 0.66, 0.61, 0.33, 0.83, 0.67, 0.54, 0.6, 0.55, 0.25, 0.54, 0.43, 0.4, 0.37, 0.49, 0.2, 0.79, 0.7, 0.6, 0.59, 0.44, 0.8, 0.57, 0.46, 0.87, 0.56, 0.48, 0.54, 0.43, 0.38, 1.1, 0.93, 0.66, 0.35, 0.43, 0.56, 0.76, 0.33, 0.13, 0.31, 0.67, 0.68, 0.69, 0.57, 0.64, 0.5, 0.77, 0.33, 0.69, 0.43, 0.53, 0.71, 0.81, 0.38, 0.85, 0.73, 0.59, 0.56, 0.56, 0.54, 0.6, 0.61, 0.77, 0.91, 0.69, 0.56, 0.73, 0.64, 0.39, 0.79, 0.66, 0.63, 0.7, 0.65, 0.41, 0.57, 0.57, 0.49, 0.94, 0.42, 0.5, 0.46, 0.37, 0.56, 0.55, 0.91, 0.55]

Based on these data, now you need to calculate how to set a threshold to accept Alice with a small false reject rate (FRR) and reject Eve with a small (if not zero) false accept rate (FAR).

Pick up a number of typical threshold values between the means of Alice's and Eve's distance values, write a programme to calculate the FAR and FRR. Observe how they change with respect to the value of the threshold. A recommended set of thresholds are [0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55].

Hint 1: A smaller distance means two templates being more similar. Assume a user will be accepted if the distance is smaller (<) than the threshold, but will be rejected if greater than or equal to (>=) the threshold. Moving the "equal to" (=) to the former will change the results to some extent but not significantly.

Hint 2: If you are really struck with writing a program to do this exercise, use Excel to do the calculation instead.

Exercise 5 (Optional) – Playing with a biometric system

Find a biometric (e.g., face or speech recognition) library that will allow you to test biometrics-based user authentication.

Try to enrol yourself and a second user who may impersonate you, and evaluate the performance of the system with the two users in terms of FAR and FRR.

Show a printed image of your face or a recorded speech of yourself to see if the system is robust against this simple type of spoofing.

One example facial recognition library in Python: https://pypi.org/project/face-recognition/.