CS 5460: Computer Security I
Fall 2020
Assignment 2: Secure Authentication
Total Marks: 150

Assume the following scenario:
The authorities in your organization have recently decided to upgrade their authentication system through a multi-step process:

- Developing a smart password meter/checker to help their customers with creating strong passwords that are not vulnerable to dictionary and targeted guessing attacks
- Implementing lock-out rule to gain better protection against online guessing attacks

Now, considering your knowledge and expertise in cybersecurity, you have been assigned to complete the following tasks as a part of the prototype development. If your prototype works successfully, the organization would allocate a full-scale budget to develop the complete version of a secure authentication system.

You can use any programming language of your choice for this assignment. You can use either a Graphical User Interface (GUI) or Console/Command Line to take user inputs and show the outputs. Inputs/outputs from/to file would be required in some cases (see below for details). You should carefully read through the assignment description to identify which programming language would work best for you.

A. Smart Password Meter / Checker

Task 1
- Identify a list of 30 dictionary words commonly used to create passwords. **Submit** this list in a pdf file.

- Users often create passwords using a variation of dictionary words, which include but are not limited to the following strategies:

- Password based on repeating a dictionary word twice or thrice (e.g., catcat, dogdogdog).
- Password based on a dictionary word spelled backwards (e.g., sttesuhcassam).
- Password based on quick adjustments, like making the first letter of a word capital, adding a digit as the second last or last character, and/or adding a common special character (e.g., !, @, #, $, %, &) at the end.
- Password based on replacing letters in a dictionary word with similar-looking numbers or special characters, like replacing ‘a’ by ‘@’, ‘i’ by ‘!’, ‘S’ by ‘$’, ‘S’ by ‘5’, etc. [Find more of such possible replacements, and make a list of the replacements you have considered in your program. **Submit** this list in a pdf file].
- Password based on combining any of the above techniques, like ‘T@ctac2%’ [cat -> catcat -> tactac -> Tactac -> Tactac2 -> Tactac2% -> T@ctac2%]

Write a computer program with the following input and outputs:

Console / GUI Input: A dictionary word
Console / GUI Output: A list of common passwords, which could be created by using different variations of the dictionary word (given as input). Such variations should include at least the above-noted techniques commonly used by people in creating passwords.

Use this program to enhance your list of common dictionary words:

- Take inputs from your file consisting of 30 dictionary words
- Store outputs in another file, containing your list of 30 dictionary words and their variations. You can call it Enhanced List of Common Passwords 1.0 (ELCP 1.0).

Task 2

Assume that a user needs to provide the following personal information during creating his/her online account with your organization (before password creation for that account):

i-a) First Name
i-b) Last Name
ii) Date of Birth (*Format*: MM/DD/YYYY)
iii) Telephone Number (*Format*: xxx-xxx-xxxx)
iv) Mailing Address
Format:
Street Information (Street number and name): _____, APT No. (if applicable): _____
City: _____ , State: _____, Zip Code (first 5-digit): _____
v) Email ID (it will be also used as a "username" for the account).

Users often create password based on their personal information, which include but are not limited to the following:

- First Name
- Last Name
- Date of Birth (DOB)
- A part of DOB, like the year of birth (1992 or just 92)
- Telephone Number
- A part of Telephone Number, like the first 6 digits or last 4 digits
- Street Number from Mailing Address
- Street Name from Mailing Address
- Apartment Number from Mailing Address
- Name of City from Mailing Address
- Name of State from Mailing Address
- Zip Code from Mailing Address
- Part of Email ID created by user, like for person@gmail.com: "person" could be a part of password; for person_gemini@gmail.com: "person" *and/or* "gemini" could be a part of the password.

Sometimes, users use a different variations of the above personal information to create their password, which include but are not limited to the following: Password based on replacing letters in a word with similar-looking numbers or special characters, like replacing 'a' by '@', 'i' by '!', 'S' by '$', 'S' by '5', etc. [Find more of such possible replacements, and make a list of the replacements you have considered in your program. **Submit** this list in a pdf file. *To be clear, you will need to submit just one such list for Task 1 and Task 2*].

Write a computer program with the following inputs and outputs:

Console / GUI Input: User's Personal Information (needed to provide for creating an online account with your organization: see above)
Console / GUI Output: A list of common passwords for this user, which could be created by using different variations of his personal information. Such variations should include at least the above-noted techniques commonly used by people in creating passwords. You can call it Enhanced List of Common Passwords 2.0 (ELCP 2.0) for this user.

For Task 2, it is optional to store the outputs in a file.

Task 3
Write a program for online account creation (i.e., registration) with your organization that asks users to provide personal information noted in Task 2, and create a password.

Console / GUI input: User's information during registration (see Task 2 above), and the password.

Store ELCP 2.0 for this user in a file. Once the password is created, your program checks if it matches with any common password included in ELCP 1.0 (from Task 1) or ELCP 2.0 for this user.

Console / GUI output: If a match is found, your program alerts users about the vulnerability of their password to dictionary attack or targeted guessing attack with *easy-to-understand explanation* of why their password is vulnerable to such an attack [Example: "Your password is vulnerable to dictionary attack since you used the dictionary word: *'cat'*, or a variation of this word in your password"; OR "Your password is vulnerable to targeted guessing attack since you used your *mailing address* or a part of it in your password"]. Then, the program prompts the user to create a new, stronger password.

The user-created password that is included neither in ELCP 1.0 nor in ELCP 2.0 would be accepted by the system, which would lead to the completion of the registration process for the user. Then an output message will be shown, e.g., "Registration is successfully completed."

B. Secure Login (Task 4)

Write a program for login where a user would enter his/her username (email ID) and password. Identify if a user's password entered during login matches with his/her password created during

registration. For this assignment, you can store users' passwords in plaintext in any type of file or database of your choice [Note to Remember: The more secure approach is to concatenate a password with a randomly generated salt, and the resulting value is cryptographically hashed before storing in the system].

If a user enters a wrong password during login, he/she could try again. After a certain number of consecutive failed login attempts (3 consecutive failed login attempts for this assignment), the user would be locked out for a certain period of time (2^i [i = 0, 1, 2,..] minutes for this assignment) before he/she could attempt to log in again. See the login sequence below for a clearer understanding of lockout rules for this assignment:

1. Failed attempt
2. Failed attempt
3. Failed attempt
   Locked out for 2^0 = 1 minute
4. Failed attempt
5. Failed attempt
6. Failed attempt
   Locked out for 2^1 = 2 minutes
7. Failed attempt
8. Failed attempt
9. Failed attempt
   Locked out for 2^2 = 4 minutes
10. Failed attempt
11. **Successful attempt**
12. Failed attempt
13. Failed attempt
14. Failed attempt
    Locked out for 2^0 = 1 minute

Submission Instructions

- The program without required input fields and visible output is not acceptable. A program that does not run (e.g., due to errors/bugs in code) is not acceptable.

- You are required to do your own work. Individual submission is needed from each student.

- You will need to submit the pdf files [see Task 1 and 2] and the working version of your code via Canvas before **11:59 PM on Monday, November 02**. Add necessary instructions for running your code in a 'Read Me' file. For multiple files, you can zip them before submission.

Grading Rubric

Tasks	**Marks**
A. Smart Password Meter/Checker	
Task 1	40
Task 2	40
Task 3	35
B. Secure Login (Task 4)	35
Total Marks	**150**

See below for further information on the grading criteria for each of the above tasks.

Criteria	**What does *excellent* performance represent for each criterion?**	**Percentage of Total Marks (for a Task)**
Requirements, Delivery, and Efficiency	● Includes name, date, and assignment title. ● Completed all of the tasks and requirements. ● Delivered on time, and in correct format. ● Thorough and organized testing or input validation has been completed. ● Solution is efficient, and easy to understand.	70%
Coding Standards and Presentation	● Excellent use of white space. ● Creatively organized work. ● Excellent use of variables. ● Excellent user prompts, good use of symbols, spacing in output	15%
Documentation	● Clearly and effectively documented including descriptions of all variables. ● Specific purpose noted for each function, control structure, input requirements, and output results.	15%