CS 645 – Fall 2022 (section 101)
Project 1 (due October 21, in the beginning of class at 6:00pm)

For any questions regarding the assignment, please email the instructor at cong.shi@njit.edu

- For Problem 1 (Part 1 and Part 2) you need to upload only the Java source files of your programs (i.e., files with .java extension) onto the Canvas website for this course.
- For Problem 1 (Part 3), Problem 2, and Problem 3 you need to upload a document with your answers onto the Canvas website for this course. The document needs to be in Microsoft Word or PDF format.
- You are allowed to work in **teams of up to three students**. Only one submission per team is needed, indicating the name(s) of the team members. Include the name(s) of your team members in every file you submit (including in the document with your answers, and in the beginning of the Java source code files as a comment).

Note: If you decide to work in a group, you must work on ALL problems in the same group. Thus, there must be only one submission for ALL 3 problems. For example, you cannot submit problem 1 as a group, and problems 2 and 3 as individuals.

Problem 1 (40 points)

For this problem, you will play the role of an attacker who has captured a password file and is performing an offline dictionary attack in order to find user passwords.

Part 1 (15 points).

As a warm-up, we will be using first a stripped-down version of a password file. You are given a shadow password file (named "shadow-simple") and one file with a list of common passwords (named "common-passwords.txt").

The shadow file is a text file in which every line corresponds to one user. The format of each line is:
username:salt:hash

where "username" is the user's ID, "salt" is a 8-character salt, and "hash" is the MD5 cryptographic hash of the string obtained by concatenating the salt with the user's password. These fields are separated by the colon (:) character.

Your task is to write a Java program that determines whether the shadow file contains any commonly used passwords. To obtain the cryptographic hash of a string of characters in Java, you can use the object MessageDigest. Because the hash is a string of characters, which may contain non-printable characters, the shadow file actually contains the hexadecimal representation of the characters. You can use the following function to convert a byte array into a String that contains the hexadecimal representation of the byte array:

```
public static String toHex(byte[] bytes)
    {
        BigInteger bi = new BigInteger(1, bytes);
        return String.format("%0" + (bytes.length << 1) + "X", bi);
    }
```

Note that the "shadow-simple" file is not a password shadow file as found on a real Linux system. The hashes are computed using a regular MD5 hash (whereas the hashes in a real shadow file on a Linux system are generated using a more complex hashing mechanism, as shown in Part 2 of this problem).

Your program should be named SimpleCracker.java and should print on the screen the user names and their passwords for those passwords that were found using the dictionary attack (one username and password per line), in the format:
username:password
(it is assumed that the files "shadow-simple" and "common-passwords.txt" are in the same directory with your program)

Part 2 (20 points).
You are now given a real Linux-style shadow file (named "shadow") and the same file with a list of common passwords (named "common-passwords.txt"). The shadow file is a text file in which every line corresponds to one user. The format of each line is (fields are separated by the colon ":" character):

username:shash:……(several other fields)
The "username" field is the user's ID. The salted hash "shash" field contains the following subfields (separated by the "$" character):
1salt$hash

Your task is to write a Java program that determines whether the shadow file contains any commonly used passwords. The hash has been generated this time using a more complex algorithm that is actually used in the Ubuntu Linux distribution. You are also provided with the file "MD5Shadow.java", which contains a class that has the MD5Shadow.crypt() method, which receives a password and a salt, and generates the shadow-style hash.

Your program should be named Cracker.java and should print on the screen the user names and their passwords for those passwords that were found using the dictionary attack (one username and password per line), in the format:
username:password
(again, it is assumed that the files "shadow" and "common-passwords.txt" are in the same directory with your program)

Part 3 (5 points).

The setting is identical as in Part 2 of this problem. One of the users in the file "shadow" has a password that is a common word, but this word is not in the file "common-passwords.txt". Your task is to find that one password. One way to solve this part is to search on the web files with common words that can be used in a dictionary attack.

This means that after adding the password you found to the file "common-passwords.txt", your program Cracker.java should find it in the "shadow" file.

Write down the password you found and describe how you found the file you used for the dictionary attack (this includes writing the URL where you found this file).

General remarks for Problem 1:
Your programs will be tested on the given shadow files and also on another set of shadow files which are not given to you.

Please upload only the Java source files (i.e., files with .java extension) of your programs onto the Canvas website for this course. Include the name(s) of your team members as a comment in the beginning of every source file you submit.

Problem 2 (40 points)

Set-UID is an important security mechanism in Unix-based operating systems. When a Set-UID program is run, it assumes the owner's privileges. For example, if the program's owner is root, then when anyone runs this program, the program gains the root's privileges during its execution. Set-UID allows us to do many interesting things, e.g., regular users can update their passwords by running the Set-UID "passwd" program. Unfortunately, Set-UID could also be the culprit for many bad things, and in this problem we will understand some of its potential security problems.

For this problem, you will need to have root access on a system with Linux OS. If you don't have root access on a Linux system, you need to create one such environment by installing a Linux OS in a virtual machine based on the instructions in the Syllabus under "Special Software Installation Requirements".

(a) Figure out why the "passwd" command needs to be a Root Set-UID program. What will happen if it is not? Login as a regular user and copy this command to your own home directory (usually "passwd" resides in /usr/bin); the copy will not be a Set-UID program. Run the copied program, and observe what happens. Describe your observations and provide an explanation for what you observed.

(b1) zsh is an older shell, which unlike the more recent bash shell does not have certain protection mechanisms incorporated.
Login as root, copy /bin/zsh to /tmp, and make it a Set-UID program with permissions 4755. Then login as a regular user, and run /tmp/zsh. Will you get root privileges? Please describe and explain your observation.

If you cannot find /bin/zsh in your operating system, please run the following command as root to install it:

- For Fedora: yum install zsh
- For Ubuntu: apt-get install zsh

(b2) Login as root and instead of copying /bin/zsh, this time, copy /bin/bash to /tmp, make it a Set-UID program. Login as a regular user and run /tmp/bash. Will you get root privilege? Please describe and provide a possible explanation for your observation.

(c1) In most Linux distributions (Fedora and Ubuntu included), /bin/sh is actually a symbolic link to /bin/bash. To use zsh, we need to link /bin/sh to /bin/zsh. The following instructions describe how to change the default shell to zsh:

- login as root
- cd /bin
- rm sh
- ln –s zsh sh

The system(const char *cmd) library function can be used to execute a command within a program. The way system(cmd) works is to invoke the /bin/sh program, and then let the shell program to execute cmd. Because of the shell program invoked, calling system() within a Set-UID program is extremely dangerous. This is because the actual behavior of the shell program can be affected by environment variables, such as PATH; these environment variables are under user's control. By changing these variables, malicious users can control the behavior of the Set-UID program.

The Set-UID program below is supposed to execute the /bin/ls command; however, the programmer only uses the relative path for the ls command, rather than the absolute path:

```
int main()
{
    system("ls"); return 0;
}
```

Login as root, create a new directory /tmp1 and set it to have the same permissions as /tmp, write this program into a file named bad_ls.c, compile it (using gcc –o bad_ls bad_ls.c) and copy the executable as a Set-UID program into /tmp1 with permissions 4755.

Is it a good idea to let regular users execute the /tmp1/bad_ls program (owned by root) instead of /bin/ls ? Describe an attack by which a regular user can manipulate the PATH environment variable in order to read the /etc/shadow file.

Note: this part is not related to part (b), which means that you do not need to have zsh or bash copied into /tmp1. In other words, zsh is not available as a SetUID program anymore.

(c2) Now, change /bin/sh so it points back to /bin/bash, and repeat the above attack. Can you still get the root privilege and list the contents of the /etc/shadow file? Describe and explain your observations.

Note: in this part, unlike in (b2), bash is not available as a SetUID program.

(c3) Specify what Linux distribution you used for Problem 2 (distribution & kernel version). You can find this information by running the command "uname –a".

Problem 3 (20 points)

Consider the following security measures for airline travel. A list of names of people who are not allowed to fly is maintained by the government and given to the airlines; people whose names are on the list are not allowed to make flight reservations. Before entering the departure area of the airport, passengers go through a security check where they have to present a government-issued ID and a boarding pass (the check done here is based on visual inspection: the person must match the picture on the ID and the name on the ID must match the name on the boarding pass). Before boarding a flight, passengers must present a boarding pass, which is scanned to verify the reservation (the check done here is to ensure the scanned information from the boarding pass matches an existing reservation in the system).

Show how someone who is on the no-fly list can manage to fly provided that boarding passes could be generated online (as an HTML page) and then printed. Please provide a step-by-step description of the attack.

Which additional security measures should be implemented in order to eliminate this vulnerability?

HINT: remember that airplane-boarding passes contain a simple two-dimensional barcode.