CS 340: Cyber Security

Password Cracking (30 points)

DUE: Friday, February 6, 2015 (End of Day)

**Password Cracking: Dictionary Attacks**

On a traditional Unix system, passwords are stored in encrypted form in a world-readable file **/etc/passwd**. Moreover the encryption algorithm is widely known. This means that an attacker can attempt to discover one or more passwords on the system by encrypting a sequence of strings and comparing the results against the stored encrypted passwords of the system. If any of the trial encyrptions match stored encrypted passwords, the attacker will know the corresponding cleartext password for that user and can then use it to access the user's account. This is a classic *dictionary attack* and explains why many systems enforce rules to ensure that user-generated passwords are not easily guessed words.

Systematic password guessing involves both cleverness and brute force. Dictionary attacks are so named because a word list, or dictionary, is used to generate password guesses. For example, most public Linux systems have such dictionary file at **/usr/share/dict/words**. A more sophisticated dictionary attack, not only uses common words and phrases, but also attempts users' surnames, common pet names, etc. Such words and phrases may be prepended to the dictionary and then become available in the attack.

A user may attempt to render his or her password unguessable by "mangling" the plaintext password in some algorithmic way. Some common "mangles" (ways to take a password and make it less easily guessable) are listed below. Assume the plaintext password is "string". You might:

* prepend a character to the string, e.g., @string;
* append a character to the string, e.g., string9;
* delete the first character from the string, e.g., tring;
* delete the last character from the string, e.g., strin;
* reverse the string, e.g., gnirts;
* duplicate the string, e.g., stringstring;
* reflect the string, e.g., stringgnirts or gnirtsstring;
* uppercase the string, e.g., STRING;
* lowercase the string, e.g., string;
* capitalize the string, e.g., String;
* ncapitalize the string, e.g., sTRING;
* toggle case of the string, e.g., StRiNg or sTrInG;

You need only consider characters that you could type from your keyboard. Weird control characters don't usually occur in passwords. A list of words that you can use as a dictionary (alternatively, you could use the Unix system dictionary at **/usr/share/dict/words** of ~100,000 words, but expect things to run pretty slowly).

**Encryption Specifics**

On traditional UNIX system, passwords are encrypted and stored in the file **/etc/passwd**. The stored value is actually the result of encrypting a string of zeros with a key formed from the first eight characters of your password and a two-character "salt". The "salt" is a two-character string stored with a user's login information. Salt is used so that anyone guessing passwords has to guess on a per-user basis rather than a per-system basis. Also, in the case that two users have the same password, as long as they have different salt, their encrypted passwords will not be the same. For example, if a user's plain text password is "amazing" and the salt is "(b", then it would return "(bUx9LiAcW8As".

Lines in /etc/passwd have the following format, with fields separated by colons:  
account:encrypted password data:uid:gid:GCOS-field:homedir:shell

For example, this line represents the account for Tyler Jones. The salt is "<q":  
tyler:<qt0.GlIrXuKs:503:503:Tyler Jones:/home/tyler:/bin/tcsh

The encrypted password data field is thirteen characters long. The first two characters are the salt, and the next eleven characters are the encrypted password (actually, a string of zeros encrypted with the salt and the password). Newer systems make dictionary attacks more difficult by employing "shadow passwords." In a shadow password system, the password field in /etc/passwd is replaced with an 'x'. Actual encrypted passwords are stored in a file /etc/shadow which is not world-readable. Your program should stop searching with respect to a given user if you have cracked that password. Consider whether to use a breadth-first or depth-first search. The algorithm only considers the first 8 characters of a password, but the user might or might not take that into account. You do not have to break all passwords, but you should break at least the simple passwords (generated from words in the dictionary using one mangle).

**Your task:**

1. Research and comprehend the Unix user password encryption scheme. Dr. Wahjudi have been kind enough to provide the JCrypt program to assist in the encryption.
2. Devise your own algorithm to perform dictionary attack on a Unix password file.
3. Implement a program to guess one or more Unix passwords. Input to your program will be a "captured" **/etc/passwd** file from a system with 20 users. Your aim is to crack as many passwords as possible. ***But don't expect to crack them all,*** there are a few passwords included generated from random strings. You should be able to get 15 or so passwords, within a few minutes.
4. How do you know when to stop? You don't! Write the program to run until it finds all of the passwords. We will stop it when we are tired of waiting ☺. Realistically, your program should find a majority of the passwords (12 or so) without problem.

**Grading:**

1. (20 points) You program should crack >= 80% passwords to get full points (efficiency is important).
2. (10 points) Write a short and concise report (2 pages max) on how you approached the problem, explain the algorithm used, describe some of the methods developed and show some examples. The idea of the report is to provide an overview of your algorithm and the program without having to go through the source code.

**Requirements & Deliverables:**

1. Your program must be written in JAVA.
2. Prompt the user for the input filename.
3. Output the cracked passwords to the screen.
4. A test file is provided in MUOnline, you can also come up with your own test file.
5. Submit your java file and the report document (Word or PDF) through MUOnline.
6. Be sure to provide detailed instructions and to comment the code sufficiently. Any code without proper comments will receive deductions up to 10 points (proper comments != over commenting).
7. You are allowed to use existing libraries (JCrypt) to perform the encryption. However, the attack itself must be of your own code. If you do use external “resources” be sure to give credit to the author.
8. Add a timer to your code to determine the time needed to crack all the passwords. Timer starts when the input file is read until the entire process is done, dictionary is pre-loaded.
9. Add a summary print statement to show how many passwords was cracked and how many was not cracked (example: 15 out of 20 passwords cracked).

**Bonus:**

1. **+3 Points**: Program that can solve the most passwords. Minimum 18 out of 20 passwords cracked to qualify.
2. **+3 Points**: Program that can crack the passwords the fastest and below the expected/standard time. Minimum 16 out of 20 passwords cracked to qualify.