Password Cracking

The goal of this lab is to familiarize students with password files and some elementary password cracking schemes.

Task 1: Password Files

In this task, you will briefly examine how your Linux system manages and stores user passwords.

1. Use the more command, as shown below, to view the `/etc/passwd` file.

   ```
   more /etc/passwd
   ```

   You should see a list of all the users that potentially have login access to your machine. At the bottom of this file you should see a line for your account (e.g., the "student" account). There was a time when `/etc/passwd` was the place where UNIX-based operating systems stored password digests, but you'll notice that no digests exist in this file. Your Linux distribution uses what is called *shadow passwords*: a special file managed by the operating system to store and protect password digests.

2. Your shadow password file exists at `/etc/shadow`. Use the more command, as shown below to try to view its contents.

   ```
   more /etc/shadow
   ```

   Record in item #1 of the worksheet the error message you received when you tried to view the shadow password file.

   Note that item #2 asks a follow-up question.

3. Try opening the shadow password file with root privileges. You can gain root privileges on Ubuntu by preceding your command with `sudo`. For example:

   ```
   sudo more /etc/shadow
   ```

   After entering your password you should be able to see the contents of the shadow password file.

4. Go to the bottom of the shadow password file to see the entry for your account (which may be so long it wraps around to the next line).

5. Each line in the shadow file is separated into different fields by a ':'. Reading left-to-right, the fields are:

   a. Login name
   b. Digest (referred to horribly as the "encrypted password")
   c. Date of last password change
   d. Minimum password age
   e. Maximum password age
   f. Password warning period
   g. Password inactivity period
   h. Account expiration date
   i. Reserved

6. Within the field designated for the digest, it is further broken up into other fields that are separated by a '$'. These fields are:

   IDsalt$digest

   The "ID" field contains a number that corresponds to the hash function/algorithm used to generate the digest. The following table shows the interpretation of that number:

ID	Hash Function
1	MD5
2	Blowfish
3	NT-hash
4	(not used)
5	SHA-256
6	SHA-512

   Using the information provided above, determine the hash function that was used to generate the value currently stored in your entry of the shadow password file.

   In item #3 of the worksheet, enter the hash function used on your password.

   In item #4 record the salt value used in the generation of the stored digest.

7. Execute the following command to list some account information (where the "`-l`" is the letter ell, not the number one):

   ```
   chage -l student
   ```

   In item #5 of the worksheet record the date when your password was chosen.

   Note that item #6 asks a follow-up question.

Task 2: Dictionary Attacks

In this task, you will try a dictionary attack to crack the contents of a password file.

Getting started

1. Use the `cat` command to view the contents of `htpasswd-sha1`.

2. You will notice that this is not, in fact, a UNIX-like password file. It is actually an htpasswd format, which can be used by an Apache web server to provide password-based access control to portions of a web site. The line:

   ```
   alice:{SHA}A9Z8JjwnpFPvZbKeMDNHJzM8y80=
   ```

   says the user "alice" has had her password hashed by the SHA1 hash function, and that the digest is stored in a base64 encoded digest of "A9Z8JjwnpFPvZbKeMDNHJzM8y80=".

   Examine the digest values for each user.

   Record in item #7 of the worksheet the users that selected the same password.

Simple Dictionary Attack

In this task you will be performing a simple and literal dictionary attack because the list contains only about 109,000 English words, and because no variations of each word is attempted.

1. Execute the following command to execute a dictionary attack on `htpasswd-sha1`, using the list of English words:

   ```
   ./crackSHA.py htpasswd-sha1 tinylist.txt
   ```

   Record in item #8 of the worksheet the username(s) and password(s) of the accounts that were cracked.

Common Password Dictionary Attack

By now, you should have cracked some passwords, but not all. This is because the dictionary you used contains only a small set of English words, with no common passwords or variants. Instead of the rather small list of words used above, you will now use `biglist.txt` that you downloaded earlier, which contains 2.2 million words and commonly used passwords.

1. Execute the `crackSHA.py` script again with the new dictionary, as shown below:

   ```
   ./crackSHA.py htpasswd-sha1 biglist.txt
   ```

   Record in item #9 of the worksheet the username(s) and password(s) of the accounts that were cracked when using `biglist.txt`.

   Record in item #10 of the worksheet the number of words that were attempted, the number of passwords that were cracked, and the number of seconds[1] it took (as reported at the end of the output).

   Note that item #11 asks a follow-up question.

[1] Note that there are at least two things slowing down the password cracking: 1) running in a VM; and 2) executing via an interpreted script (rather than a compiled program). Otherwise the rate should be higher.

Task 3: Considering Execution Time

In this task you will be comparing the execution time of various hash functions.

1. As a speed comparison between different hash functions, execute the following script on a file that hashed the passwords using MD5 (an older hash function):

   ```
   ./crackMD5.py htpasswd-md5 biglist.txt
   ```

   Record in item #12 of the worksheet the number of words that were attempted, the number of passwords that were cracked, and the number of seconds it took.

   Note that item #13 asks a follow-up question. [You may need to refer to the lecture slides on APR1, but do not use the example time from the slide because it does not apply in this case.]

   Note that item #14 asks a follow-up question. [You may need to refer to the benefits of salt values, as discussed in lecture.]

2. As another point of reference in the speed comparison, execute the following script on a password file that used SHA512:

   ```
   ./crack512.py htpasswd-sha512 biglist.txt
   ```

   Record in item #15 of the worksheet the number of words that were attempted, the number of passwords that were cracked, and the number of seconds it took.

   Note that item #16 asks a follow-up question.

3. Refer to the numbers recorded in item #10 of the worksheet. Enter into the downloaded spreadsheet (where it is highlighted in red) the "Words processed" and the "Seconds to process". The spreadsheet will then show you the estimated amount of time the python script would take (on your VM) if it were modified to do a brute force attack on 15-character passwords using SHA1.

4. Now enter 10,000,000,000 as the number of words processed, and then enter 1 as the number of seconds. In other words, you are specifying what the time would be if you could try 10,000,000,000 passwords per second. **Save the results**.

   Note that item #17 asks a follow-up question.

In the next step below you will execute a script that will make use of pre-calculated digests . These pre-calculated digests were created by hashing each password in `biglist.txt` with SHA1, and then sorting the words into files whose names were the first two hex digits of the digest. No digests were saved.

To use the presorted passwords, the logic of the password-cracking script is roughly as follows:

a. Get a digest from the password file and look at the first two hex digits.
b. Open the file that has the same name as those two hex digits.
c. Hash each word in the opened file to see if one of the words will hash to the same full digest as was seen in step 'a'.

5. Do the following to see all the **file names** for the sorted dictionary words:

```
ls calc
```

6. Do the following to see the contents of **one** of the saved files:

```
more calc/a9
```

   This file contains all the words in `biglist.txt` that hash to a digest that starts with "a9". If a digest in a password file starts with "a9", then the script only needs to hash the words in this file to **potentially** find a match.

7. Execute the following command to make use of the pre-calculated digests:

```
./crackPre.py htpasswd-sha1 calc
```

   Record in item #18 of the worksheet the number of words that were attempted, the number of passwords that were cracked, and the number of seconds it took.

   Note that item #19 of the worksheet asks a follow-up question.

Task 4: Personal Experimentation

In this task you will experiment with passwords of your choice.

1. As described below, create your own password file (named `htpasswd-me`) in the htpasswd format, with an entry for "alice". You will be prompted for the password.

   ```
   htpasswd -sc htpasswd-me alice
   ```

   You can **add** other entries by doing the following (slightly modified) command:

   ```
   htpasswd -s htpasswd-me bob
   ```

   As a security exercise, you may want to add passwords you commonly use (or have used) to see if they can be cracked using this relatively small list of passwords.

2. Display your `htpasswd-me` file by using the `cat` command.

3. Perform the pre-calculated attack as follows:

   ```
   ./crackPre.py htpasswd-me calc
   ```

   Record in item #20 of the worksheet the results of your experiments. [Do not write down any actual passwords you have used.]

   Complete items #21 and #22.

Submission

Post the completed Lab Report to Sakai by the deadline.

Appendix – Some Unix Commands

cd — Change the current directory.

```
cd destination
```

With no "destination" your current directory will be changed to your home directory. If you "destination" is "..", then your current directory will be changed to the parent of your current directory.

cp — Copy a file.

```
cp source destination
```

This will copy the file with the "source" name to a copy with the "destination" name. The "destination" can also include the path to another directory.

clear — Erase all the output on the current terminal and place the shell prompt at the top of the terminal.

less — Display a page of a text file at a time in the terminal. (Also see more).

```
less file
```

To see another page, press the space bar. To see one more line, press the Enter key. To quit at any time press 'q' to quit.

ls — List the contents and/or attributes of a directory or file

```
ls location
ls file
```

With no "location" or "file" it will display the contents of the current working directory.

man — Manual

```
man command
```

Displays the manual page for the given "command". To see another page, press the space bar. To see one more line, press the Enter key. To quit before reaching the end of the file enter 'q'.

more — Display a page of a text file at a time in the terminal. (Also see less).

```
more file
```

To see another page, press the space bar. To see one more line, press the Enter key. To quit at any time press 'q' to quit.

mv — Move and/or Rename a file/directory

```
mv source destination
```

The "source" file will be moved and/or renamed to the given "destination."

pwd — Display the present working directory

```
pwd
```