CS4000

Homework # 5: Hacking Salted Passwords with MPI

due Monday, April 1st, 2019, 11:59 p.m.

(50 pts.)

Introduction

Storing passwords is one of the most crucial aspects of Internet security, and one of the easiest things to do incorrectly. If you have a "secure" web-site, i.e., one that requires passwords for a user to gain access to some resources, one of the worst things that you could do would be to store the passwords in a database as plaintext. Then, if a hacker was able to access the database and download the passwords, they would have complete control of all user accounts in the system. To get around this vulnerability, computer systems often store passwords in an encrypted format using a hash function such as MD5, Blowfish, or DES. The problem with this approach is that hackers, if they have access to the hashed version of the passwords, can use these common hash functions to build a database of common password hashes to find whether users are using simple passwords. To get around this potential vulnerability, system designers invented the concept of *salted* passwords. The concept is somewhat straightforward. Passwords are encrypted using a random *salt*, and both the encrypted version of the password and the *salt* are stored in the password file/database. This approach stops the database driven (as well as *rainbow table*) password hacking techniques. However, like all such password storage approaches, they are not fool proof.

Salted passwords are used in UNIX/Linux to store system passwords. System passwords are usually stored in `/etc/passwd` or `/etc/shadow`.

Generating Salted Passwords using Crypt

In the UNIX `passwd` file, the system stores your username followed by a colon separated list of information about the username. The first item in this list is an encrypted version of the password for the account. For example, you might see

```
user1:$6$mk3DMfWk3hwXSva1$70DFGsRukpYQ4UhoIQ2mM9dTfimiJHt9o939a04nFZ/AK112OZa7
Sw6WxKqty1ZQ386Jc431Thf5xPcCajxNt/:
```

The encrypted password was generated using the Unix `crypt`. The first part of the encrypted password (`$6`) tells the `crypt` function which encryption algorithm to run. The second part of the encrypted password (up until the third $), `mk3DMfWk3hwXSva1` is a random string that is used as the salt for the encryption algorithm. The rest of this line, up to the final : stores the encrypted password. This line was generated by the following short `C++` code:

```
#include <unistd.h>
......
  string t;
  string salt = "$6$mk3DMfWk3hwXSva1";
  string pass = "dumbpassword";
  t = crypt(pass.c_str(),salt.c_str());
  cout << t << endl;
```

Notice that `crypt` returns the salt along with the encrypted version of the password.

Cracking Salted Passwords

For this project, you will be cracking (trying to discover the real password) salted passwords using a dictionary based attack and `OpenMPI`. You will receive a list of salted passwords (e.g., see `password_file`) and a dictionary (see `words`). You will try to crack the salted passwords by trying passwords formed by concatenating one or two words from the dictionary together with (or without) a short ($\leq$3) string of digits. For example, if "orthogonal123ant" is your password, this program should be able to crack it.

You will know whether you guessed a password correctly if, when you supply the function `crypt` with the salt and the guessed password, you get the original encrypted password.

Implementation

Write an OpenMPI program that takes command line parameters: a filename for the list of salted/encrypted passwords, and a filename for the dictionary. Your program will try all possible (one or two word) combinations of words from the dictionary to try to *crack* the passwords. Your program will be produce one line for each encrypted password in the password file. It will either print "Password:XXXX", where XXXX is the correct password, or it will print "Password not found" Try to make your program as fast as possible. Your program should be able to achieve 75% efficiency when run on 10 of the machines in the 307 lab.

Make sure to properly document your program.