CS3780

Project 2: Passwords and Authentication

Due date: Nov 12th

Overview:

In this project you will be both saving and storing passwords, as well as trying to “crack” a password file. This project is divided up into two parts. This project will require the use of a cryptographic library to do hashes. That information is linked at the end.

While you can do this project by yourself, I will allow you to work in pairs, as this project tends to have a fairly straightforward division of labor for at least part of the project. Note that these parts might not be equally difficult and certainly you will have to come to some agreement on format and encoding before starting, so do not assume you will be able to code your part without input from the other.

Task 1:

Write a program that allows a user to create an account and then authenticate whether or not they have an account. However, the trick is that this is done through three different password files. So when the program starts, it should ask the user which one of those they should like to either log in or authenticate through. Once they have done that, your program will ask them if they want to make an account or log in. If they want to log in, your program should accept their username and password and verify that using the appropriate password file. If they want to create an account, the program should create the appropriate data in the appropriate password file. Note the primary purpose of this program is to allow testing of parts 2 and 3.

For ease of cracking later, I want your passwords to be restricted to only lowercase letters (a..z) of configurable length. The length of the salt should be one byte for this project. In practice, salt should be of much greater size than this. We have restricted the size for this project in order to make cracking it easier.

Usernames should be restricted to 8 alphanumeric characters.

The three password files should be as follows:

A plaintext username password pair, stored in text in a file

A username and a hashed password, stored in some format in the file

A username, a salt and the result of the hashed (password + salt), stored in some format in the file

Task 2:

Extend your previous program such that you can get it to, when given bounds on password size (for example, all passwords between length 3 and 8) and a number of accounts (100 for example), will then create usernames and passwords for 100 accounts in the specified ranges for each of the 3 password files. For this purpose it is probably useful for this program to create usernames of a particular format (user00001) for example.

Task3:

In this task we are replicating a situation where you have somehow gotten ahold of a password file and you wish to crack it. You can assume that you have full knowledge of the format of the particular file you are working with. This program should take as an option one of the password files above (only 2 or 3, 1 is trivial), as well as a maximum password size. It should then try to figure out one of the passwords in that file by brute force. That is, if ran with a password file of type2, it would attempt to find a password that hashed to one of the passwords of a username in the file, trying all combinations of passwords up to some specified length. If called with a password file of type3, it would attempt to find a password that when hashed with the salt would match one of the usernames.

Note that trying to crack a file of type 2, you can compare your hash against all of the usernames in the file. This you can do by loading this file into some structure. As you are searching for a particular match, this is done much more efficiently if it is stored in a binary search tree or similar structure. For files of type 3, whether your need to do something like this or not is something I leave up to you to consider.

Testing and Analysis:

You now have a program that creates password files and is able to test some of those passwords and a program that is supposed to try and crack some of those passwords.

For the final part of this project, I want you to experiment and find for me how long it takes your password cracker to crack one of the passwords in the files using type 2 and type 3 for various bounds of random files (ex: file of 1000 passwords of length 3 to 8). I want you to be able to describe to me which format is more secure and give some concrete numbers on how much more secure it is. I would like you to tell me what password length would be the minimum to be secure on the system that you ran the code on. Discuss these conclusions in your documentation.

Cryptographic library:

For this project you will need some cryptographic library that lets you do hashes. If you are using Java, it is relatively straightforward to use the Java Cryptography library.

http://tutorials.jenkov.com/java-cryptography/index.html

If you are using Python, it is also possible to use its cryptography library.
https://cryptography.io/en/latest/

The above options are probably the easiest. If you wish to use C++, then it is a bit less standard. You can check your own environment and see if you have a library. If you are using delmar, you can install yourself the OpenSSL library.

https://www.openssl.org/

You will then need to install it (locally) from source on delmar,

https://wiki.openssl.org/index.php/Compilation_and_Installation

The openssl files themselves are available through that link but also at

/home/hauschildm/Crypto

To install it locally, you unzip the file in some location, then you have to configure it to install in another directory. So suppose you wanted to install it to your subdirectory /home/yourusername/usr/local/ssl, then you would then run in the directory that you unzipped the files:

```
./config --openssldir=/home/yourusername/usr/local/ssl --prefixdir=/home/yourusername/usr/local/ssl
```

Then you have to compile it, so you do:

```
make
```

Then you have to install it:

```
make install
```

You should then test the environment by writing a program that does some basic operation. I have example code to do this on delmar, located at

/home/hauschildm/Crypto/proj2test

which has a Makefile and some source files for a program that takes in a string from the user and hashes it in C. Remember that you will also have to set up your LD_LIBRARY_PATH and change the makefile for your installation directories.

Submission:

For turnin I want the source files of each of the three tasks uploaded separately (not zipped). I then want a pdf document explaining your results, including environment chosen, results obtained, etc. At the end of this document I want you to also put your source code so I have it also available in the same location. Each person is required to submit their own copy of all the data, please be clear in the documentation and the submission who your partner is.