02230: Computer Security
Lab 0: Passwords

Robin Sharp

Autumn 2005

This lab. consists of a number of small tasks to increase your awareness of security issues associated with passwords. The first main task requires you to do some C-programming on a Linux- (or other Unix-) based system. The second one you can try out on any system which has a web browser.

1 Unix password encryption

In Unix systems, passwords are commonly encrypted before being stored in a password file which (basically) contains a list of user identifiers and the corresponding passwords. Two encryption functions are in particularly common use:

crypt: A standard C-library function which, given a user password and a 2-character *salt* value, will return the password in standard encrypted form. The encryption is based on the DES symmetric encryption algorithm, and produces a 13-byte result, of which the first two bytes contain the unencrypted salt itself. For further details, see the Unix `man` page for the function `crypt` (in section 3 of the `man` pages).

MD5: A standard C-library function which, given a user password and its length in characters, will return a cryptographic hash value (message digest) for the password, evaluated using the MD5 algorithm. A cryptographic hash function performs an irreversible transformation on its argument, usually producing a fixed-length result, which in the case of the MD5 algorithm is 128 bits (16 bytes) long. For further details, see the Unix `man` page for the function `MD5` (in section 3 of the `man` pages).

You may find it helpful to know that under Linux a linkable version of the `crypt` function can be found in the library `libcrypt.a`, and a linkable version of `md5` in `libssl.a`.

Your task consists in writing a program which can test some of the properties of these functions. Your program should allow the user repeatedly to:

1. Read in a sequence of characters which make up a proposed "user password"
2. Read in a pair of characters to make up a *salt* value.
3. Print out the result of encrypting this user password using
   (a) `crypt`, with the given *salt* value.
   (b) `MD5`.

   (The results are really binary values, so it is best to print them out as hexadecimal numbers.)

Test out your program in a systematic manner, for example by trying the following:

1. Try a series of alphabetically consecutive passwords: A, B, C,..., AA, AB, AC,...
2. Try a series of passwords ending in consecutive digits: XY1, XY2, XY3, XY4,...
3. Try a password and its reverse: ALEPH, HPELA.

In principle, the functions are intended to remove the redundancy which is inherent in any text, so that closely related passwords give markedly different encrypted values. Try to decide whether this is in fact the case.

Extra Challenges:

1. When `crypt` is used with the value `xy` as salt, a certain password gives the encrypted value in hexadecimal form (where the unencrypted salt has been removed from the beginning!):

   ```
   774b344c4855717162766b
   ```

   What is the password?

2. When `crypt` is used with a salt value which you are not told, a certain password gives the encrypted value (again with the unencrypted salt removed from the beginning):

   ```
   57744c4e55383658664755
   ```

   What is the password?

2 Password checking

It is important to use reasonably strong passwords for user authentication, where *strong* here means that it is difficult to discover the password by making intelligent guesses, possibly also exploiting a dictionary. Many companies have a definite policy in this area, which all employees are expected to follow.

You can find a freely available password checking service at the Web site:

`http://www.securitystats.com/tools/password.php`

The web page also contains useful advice on how to create strong passwords.

Experiment with this checking tool, using passwords which you choose for yourself. To find out what the tool can do for you, you should try first with simple words, then increase the length and complexity of your trial password until it is declared strong.

In modern Linux systems it is possible to build this type of checker into the password changing mechanism, so that proposed new passwords are checked to ensure that they have a certain minimum strength before you are allowed to replace your existing password. This is known as *proactive control*. Facilities for including this in Linux are described on the Web page:

`http://www.openwall.com/passwdqc/`

The computers in the Linux databar use this system, so if you try to change your password it will be checked for strength. On the other hand, it is not very convenient to test the checker, since the only way of doing it at present is to try to change your password, which you may not want to do!