Home exam for PG3400

Kjetil Raaen

Instructions

This assignment is graded A-E with F for fail and counts 40% of your final grade. Competent and idiomatic use of the programming language C is more important than fancy features while grading. A short description of how the program and your thought processes have worked will help grading. Try to avoid memory leaks and buffer overflows. The product should compile with GCC 7.3.0, and will be tested under Ubuntu 18.04.

The delivery should contain the code and a readme.txt which explains any decisions made in the assignment, as well as instructions about how to compile and run the program.

Cracking passwords

We regularly read about large caches of passwords being leaked on the internet. Such a leak happens in two steps: First, a cracker needs to access the password file. Such files are only accessible by root users, so this requires full access to the system. But even when you have downloaded a password file, the passwords are not stored in cleartext. Passwords are run through a hashing algorithm multiple times, to produce a result that cannot be decoded to the original password. The purpose of this assignment is creating a program that cracks such hashed passwords.

The trick to this is in principle simple: Use the same algorithm used to hash the passwords on a long series of guesses. There are two ways of guessing: The simplest is going through a long list of common passwords, called a "dictionary", and guessing each one. Many such lists are available online, and one is attached to this assignment. If this fails, you have to try any possible combination of the symbols allowed in passwords. If the possible alphabet for the password is 'A','B' and 'C', and the maximum length of a password is two characters, this system would guess 'A', 'B', 'C', 'AA', 'BA', 'CA', 'AB', 'BB', 'CB', 'AC', 'BC', 'CC'. As you can see, the time taken to crack such a password is proportional to $O(N^M)$ where N is the number selection of characters the password is selected from, and M is the number of characters in the password. This means the cracking will be slow for long passwords using a varied set of characters.

Technical details

Hashing passwords is handled by the function "crypt (3)" ("see man" 3 crypt for details). I have attached a small utility (crypto.c) that uses this function to hash passwords. Compiling this program requires using the "-lcrypt" flag to the linker.

Use this program to create your own hashed passwords for testing the cracker. The hashes look like this:
1OxMvna1o$M.mMB9Jy9H6YfJeet1ICc.
It starts by a dollar sign, followed by a 1. This indicates the hashing function used. All passwords in this assignment will use hash function 1, MD5. You may play with other hash functions if you like. After the next dollar sign are eight character called the "salt". These are random characters added to the password before encrypting. This makes sure that identical passwords do not look identical in the password file, and must be cracked individually. After the next dollar sign, is the hash itself.

The goal of this assignment is making a program that lets you write on the command line:
```
./cracker '$1$aTWHXMtg$akUOXns69Uzn44CGTLQ7S0'
```
Giving the output:
```
Searching for hash: $1$aTWHXMtg$akUOXns69Uzn44CGTLQ7S0
Found password: oi
```
Note the single quotes around the hash value. Bash will interpret dollar signs as variables unless quoted. As an alternative the program can expect a file name of a file where the hashes are stored one per line. I have attached one such file with example passwords.

Requirements

- The program should crack passwords hashed with the Linux "crypt 3"-fucntion.
- It should first go through a dictionary file.
    - Optional: Let user select dictionary file by command line parameter.
- Next, it should guess systematically any combination of allowed characters.
- Allowed characters are:
  ```
  const char passchars[] =
  "abcdefghikjlmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXY
  Z1234567890+\"#&/()=?!@$|[]|{}";
  ```
- The number of characters tried should be configurable, and the code should support any number of characters.
- When the password is found, it should be reported to console.
- The code should be as efficient as possible.

Advanced features (Note: These will only be considered for the highest grades.):

- Cracking should utilise any number of CPU cores. (Note that "crypt" is not thread-safe, and you should use "r_crypt" for multithreaded implementations. See man page.
- Cracking should work on multiple machines communicating over a network.

Hints

- Consider a recursive implementation of the guessing system.
- Try with a short password and a short alphabet first, printing out the guesses. Then you can see if it works.
- Do not free the string returned by “crypt”. The system handles this internally and it will lead to strange behaviour.