Homework : rainbow attack
SSD & WS

R. Absil

2020 - 2021

The objective of this group homework (max. 3 students per group) is to implement an attack on password tables with a rainbow table. The deadline is set on 18th October at 23:59.

Minimal objectives

From a table of passwords stored as pairs "(login,hash)" with the help of some cryptographic function H, you must implement a rainbow attack.

For academic reasons (mainly simplicity),
— passwords are *not* salted,
— passwords are stored after a single pass through the hash function,
— passwords are alphanumeric with length at least 8 and at most 12,
— the hash function H is SHA-256.

The choice of language is left to your discretion (but that choice is your responsibility), as long as the required libraries are in updated Debian packages. Should you use custom libraries, their code *must* be open-source.

Please note that you must at least submit two scripts and one text file :
— a preprocessing script allowing to generate a "sufficiently large" [1] rainbow table RT,
— an attack script allowing to exploit RT in order to find passwords from their hashes.
In *no way* you have to submit the rainbow table RT, which can be quite large.

For the sake of uniformity, your attack script must allow to input hashes stored in a text file, one hash per line.

Should you find it useful, two scripts are provided for you :
— `gen-passwd`, generating passwords accepted by the policy, storing them in one text file, and their hashes in another file,
— `check-passwd`, checking whether passwords stored in one file match hashes stored in another file.

1. The user can decide what is "sufficiently large".

You can compile them using the commands

```
g++ -o gen-passwd -std=XXX random.hpp sha256.cpp gen-passwd.cpp passwd-utils.hpp
g++ -o check-passwd -std=XXX random.hpp sha256.cpp check-passwd.cpp passwd-utils.hpp
```

where `XXX` is assumed to refer at least `c++17`. Running these programs without command line arguments will provide further information about how to use them.

You will also find an open source certified `C++` implementation of SHA-256. A `main` file also shows how to use this implementation.

Submission and grade

This homework is a *group project*, a single submission per group of 3 students is enough. Your work has to be submitted through a gitlab link sent by email. The deadline is set on 18th October at 23:59.

In order *not to get* a grade of 0/20, you must
— submit by email (`rabsil@he2b.be`) and at most on 8th October a link to a gitlab repository where your homework will be hosted,
— add me (`rabsil`) as *maintainer* of your repository,
— push your project on time (18th October at 23:59) on your gitlab repository,
— at least cover the minimal objectives,
— provide a `MAKEFILE` in order to build your project [2],
— provide a `README` file detailing
  — the list of the members of your group,
  — how to launch your project (that is, the set of commands necessary to build the rainbow table and run the attack script).

In order to get a grade of at least 10/20, you must be able to
— generate a sufficiently large rainbow table under one night [3] of user time on a laptop [4],
— not to generate a rainbow table bigger than 12 Gb,
— to be able to successfully crack 60% of a set ($\simeq$ 100) of hashes of passwords of length 8 provided as a text file [5] under 45min of CPU time on a laptop[4].

2. This includes compiling your scripts as well as installing third party missing libraries. If you're unfamiliar with makefiles, you can also provide the list of *shell commands* needed to compile your project and install third party libraries.

3. One night is not a well defined time unit. Consider maximum 12h of user time.

4. That is, you cannot reasonably assume I have a computing cluster at my disposal, nor that my machine will behave fairly if you load computations on the GPU.

5. Recall that passwords are alphanumeric (lower and upper case) and are stored unsalted after a single pass to the SHA-256 hash function.

Advisable choices

To increase your chances of successfully complete this homework, I would strongly advise to implement various algorithmic and programming optimisations, such as using dedicated data structures, using dynamic programming and caching, avoiding passing function parameters by value, avoiding runtime decision making such as polymorphism, etc.

Also, considering that running the project is time consuming, it is probably wise to
— use a programming language that outputs a binary file directly executed by the processor,
— test your implementation on small passwords and short hash chains, where you can actually see the entire table (including the "middle" of the chains), and manually check whether it fails or not, where chains might merge, etc.

On the other hand, choosing a good reduction function is usually a key feature in such implementations. Consequently, running some form of statistical analysis to check the uniformity of its output (on input spaces of "small" size) is probably a good idea.

"You will not laugh ! You will not cry ! You will learn by the numbers !"