SSD & WS
Homework : rainbow attack

R. Absil

2024 - 2025

The objective of this group (3-5 students) homework is to implement an attack on password tables with a rainbow table. The deadline is set on October 13 at 23h59.

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[1] at least 6 and at most 10,
- the hash function H is SHA-256.

The choice of language is left to your discretion (but that choice is your responsibility). Should you use custom libraries, their code *must* be open-source.

Please note that you must at least submit two scripts:

- a preprocessing script allowing to generate a "sufficiently large"[2] rainbow table RT,
- an attack script allowing to exploit RT in order to find passwords from their hashes.

In *no way* you are allowed to submit the rainbow table RT, which can be quite large.

Should you find it useful[3], two scripts are provided for you:

[1]You are allowed to build a rainbow table per password length, for simplicity reasons.
[2]The user can decide what is "sufficiently large".
[3]Check the input/output modalities in the last Section.

- `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.

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 implementation of a thread pool[4] should you find it useful[5], as well as an open source `C++` implementation of SHA-256. A `main` file also shows how to use this implementation.

Submission modalities

Projects have to be implemented in groups of 3 to 5 students, and submitted with the help of a gitlab[6] repository[7]. For that purpose, send us an email on September 29 at 23h59 at the latest with the ssh URL[8] to your repository[9], and the name and matricule of your group members.

You have to submit your work on October 13 at 23h59 at the latest. The minimal requirements for submitted projects are as follows:

- projects have to be submitted on time,
- projects have to provide a `README` file
  - mentioning the name and matricule of your group members,
  - explaining how to build[10] your project on a ubuntu 22.04 distribution (we recommend here to either provide a Makefile, or a shell script to install missing dependencies, compile the project and run relevant scripts),
  - explaining how to use your project (for example, "to launch the attack, type the following command in a shell").

Projects failing to meet these requirements will not be graded (that is, they will get 0/20). In particular, projects that do not compile according to your *exact* instructions will not be graded. Furthermore, note that we shall in *no way* build or run your projects in an IDE.

[4] I suspect my implementation is vulnerable to spurious wakeups. That shouldn't be a problem.
[5] It is unlikely that you will meet the efficiency requirements detailed later without multithreading.
[6] The only two allowed instances are `gitlab.com` and `git.esi-bru.be`.
[7] Create the repository yourself, add your teachers (`rabsil`) as maintainer.
[8] A gitlab ssh URL looks like `git@instance.com:username/projectname.git` .
[9] The automatic email notification is *not* enough.
[10] It is expected that your script installs missing dependencies in addition to compiling your code.

Furthermore, your attack script

- must allow to input hashes stored in a ASCII-encoded text file, one hash per line (the last line being blank), written as a base-16 string;
- must output a text file with the cracked passwords (that is, the passwords that match the input hashes), one password per line (the last line being blank). The passwords that could not be cracked have to be replaced with a line only containing the character '?'.

Note that these above conditions are necessary but clearly not sufficient to get 10/20. To increase your chances of successfully complete this homework, I would strongly advise

- to be able to generate a sufficiently large rainbow table under one night of user time on a laptop[11],
- not to generate a rainbow table bigger than 12 GB,
- to be able to successfully crack 50% of a set of hashes ($\simeq$ 100) provided as a text file[12] under 45min of CPU time on a laptop[11].

It is forbidden to cry and forbidden to laugh.

[11] 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.

[12] Recall that passwords are alphanumeric (lower and upper case) with length at least 6 and at most 10, are stored unsalted after a single pass to the SHA-256 hash function.