Groupe 1 Année 2017-2018

Bruteforce SHA256

Introduction

Conception d’un logiciel permettant de déterminer un mot/phrase à partir de son hash SHA256.

Le principe repose sur la génération de toutes les chaînes possibles et de comparer leur hash SHA256 avec le hash reçu en paramètre lors de l’appel du programme.

On découpera l’exécution du programme dans des threads, dont le nombre sera identifié avant l’exécution du bruteforce, afin d’accélérer le calcul des chaines.

Le dictionnaire est constitué des caractères suivants: [a-z][A-Z][0-9] et espace.

La longueur maximale de la chaîne à déterminer ne peut excéder 100 caractères.

Définition d’un hash SHA256:

SHA-2 [(Secure Hash Algorithm)](https://fr.wikipedia.org/wiki/Secure_Hash_Algorithm) est une famille de [fonctions de hachage](https://fr.wikipedia.org/wiki/Fonction_de_hachage) qui ont été conçues par

la [National Security Agency](https://fr.wikipedia.org/wiki/National_Security_Agency) des États-Unis (NSA), sur le modèle des fonctions [SHA-1](https://fr.wikipedia.org/wiki/SHA-1) et [SHA-](https://fr.wikipedia.org/wiki/SHA-0)0, elles-mêmes fortement inspirées de la fonction [MD4](https://fr.wikipedia.org/wiki/MD4) de [Ron Rivest](https://fr.wikipedia.org/wiki/Ron_Rivest) (qui a donné parallèlement [MD5)](https://fr.wikipedia.org/wiki/MD5).



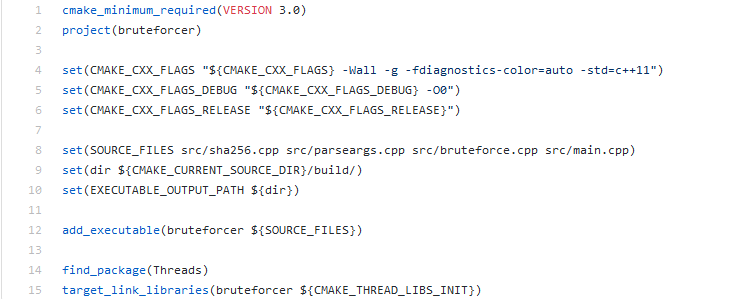
Utilisation:

./bruteforce nbCore hash\_du\_mot

Exemple

./bruteforcer 5 ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad

CMake:



CMake est un « moteur de production » multiplateforme.

Il est comparable au programme Make dans le sens où le processus de construction logicielle est entièrement contrôlé par des fichiers de configuration, appelés CMakeLists.txt dans notre cas.

Mais CMake ne produit pas directement le logiciel final, il s'occupe de la génération de fichiers de construction standards : makefiles sous Unix, et fichiers de projet Visual Studio sous Windows. Cela permet aux développeurs d'utiliser leur environnement de développement préféré comme à leur habitude.  
  
Le nom « CMake » est l'abréviation de « cross platform make ». Malgré l'utilisation de « make » dans son nom, CMake est une application séparée et de plus haut niveau que l'outil make.

Dans le fichier CMakeLists.txt il est possible de spécifier plusieurs flags, nous allons expliquer ceux que nous avons utilisés :

* **CMAKE\_CXX\_FLAGS** : utilisé pour indiquer que l’on doit compiler du C++
* **CMAKE\_CXX\_FLAGS\_DEBUG** : utilisé pour la stacktrace de débug lors de la compilation
* **CMAKE\_CXX\_FLAGS\_RELEASE** : utilisé pour les release

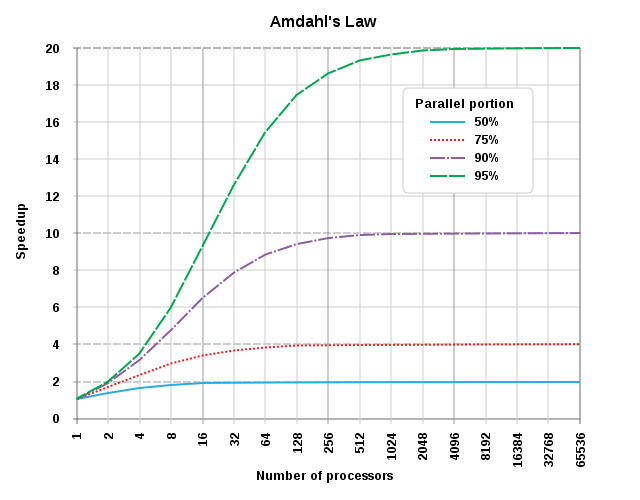
Stratégie de Parallélisation :

En ce qui concerne la stratégie que nous avons adopté, nous avons décidé de découper notre dictionnaire de caractères disponibles en fonctions du nombre de threads définit dans les options de lancement du bruteforcer.

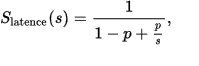
En effet, chaque thread possède alors une partie du dictionnaire et peut donc le parcourir pour trouver des similitudes avec le hash fourni.

La parallélisation permet alors de parcourir le dictionnaire très rapidement pour chaque caractère et ainsi trouvé le mot de passe très rapidement.

Loi d’Amdahl :



La loi d'Amdahl peut être formulée de la façon suivante :



où

* Slatence est l'accélération théorique en latence de l'exécution de toute la tâche ;
* s est l'accélération en latence de l'exécution de la partie de la tâche bénéficiant de l'amélioration des ressources du système ;
* p est le pourcentage du temps d'exécution de toute la tâche concernant la partie bénéficiant de l'amélioration des ressources du système avant l'amélioration.

|  |  |  |
| --- | --- | --- |
| Nombre de Threads | Speedup | % Code Parallèle |
| 1 | 1.0 | 94% |
| 2 | 1.88679245 | 94% |
| 4 | 3.38983051 | 94% |

Difficultés Rencontrées :

La principale difficulté à été d’appliquer la parallélisation dans notre algorithme, à la base, séquentiel.

En effet, nous avons dû passer par 3 algorithmes différents afin de pouvoir obtenir un temps de bruteforce convenable.

D’abord notre premier algorithme ne pouvait exécuter un bruteforce sur plus de 3 caractères, du moins nous abandonnons rapidement l'exécution car cela prenait trop de temps.

Ensuite, l’algorithme suivant était plus rapide et plus efficace, en effet il permettait le bruteforce de plus de caractères que le premier, et plus rapide. Mais plus le mot de passe à trouver était long, plus l’algorithme surchargeait et risquait de finir en erreur.