SINF1252 - P2 - Password cracker

2017

1 Énoncé

L'objectif de ce projet est de construire et d'implémenter une architecture résolvant un problème de calcul intensif grâce à la parallélisation. Cette parallélisation se réalise en utilisant des *threads*. Dans ce projet, vous commencerez par spécifier votre architecture à haut niveau. Après validation par les assistants, vous vous attaquerez à l'implémentation de votre architecture. Vous écrirez des tests unitaires afin de valider le bon fonctionnement de votre implémentation. Après avoir remis votre projet accompagné d'un rapport, vous effectuerez une review de deux autres projets.

1.1 Description

Ayant eu vent de votre talent en informatique, une de vos connaissances un peu louche vous demande un service très particulier. Elle a obtenu, d'une manière que nous tairons, un fichier contenant une liste de mots de passe. Un ou plusieurs d'entre eux donnent accès, dit-elle, à d'importantes informations. Seulement, ces mots de passe ne sont pas stockés en clair: ils sont *hashés* [1]. Vous savez seulement que la fonction de hash utilisée est SHA-256. Une autre information capitale est le critère de sélection des mots de passe recherchés. Vous apprenez que les mots de passe d'intérêt ont tous une caractéristique particulière: ils ont le plus grand nombre d'occurrences d'une lettre donnée par rapport aux autres mots de passe de la liste. Cette lettre, vous ne la connaîtrez qu'au dernier moment. Votre connaissance vous propose une très forte récompense si vous parvenez à découvrir le plus vite possible les mots de passe demandés.

Vous l'aurez compris, votre programme a pour objectif d'inverser les hash que vous avez en entrée et d'en sortir le sous-ensemble qui correspond au critère de sélection. La principale difficulté du problème est qu'une fonction de hash est réalisée de telle manière à ce qu'elle ne soit pas inversible. La seule possibilité pour en trouver l'inverse est de générer des candidats, de calculer le hash de ces candidats et de comparer ces hash avec celui que l'on désire inverser. Si le hash est le même, alors l'inverse a été trouvé. **Cette opération est souvent coûteuse en temps de calcul**. Étant donné qu'une fonction de hash accepte en entrée un nombre indéfini de bytes pour en sortir un nombre fini et fixé, il existe théoriquement des collisions. Autrement dit, il se peut que pour deux inputs différents, le même hash soit généré. Cependant, la probabilité que cela se produise est excessivement faible. En pratique, aucune collision n'a encore été découverte pour SHA-256. Pour ce projet, vous pouvez donc considérer que cela n'arrivera pas.

1.2 SHA-256

La fonction de hash SHA-256 génère un hash de 32 bytes de long (donc 256 bits). Lorsqu'un hash est représenté de manière textuelle, on l'affiche comme une concaténation de la représentation hexadécimale de chaque byte. Par exemple, le hash SHA-256 de la string `coucou` est

`110812f67fa1e1f0117f6f3d70241c1a42a7b07711a93c2477cc516d9042f9db`

Chaque groupe de deux caractères représente un byte du hash sous forme hexadécimale. Le hash est donc composé des bytes `0x11`, `0x08`, `0x12`, `0xf6`, etc.

Vous pouvez générer vous-même des hash grâce à la commande suivante:

```
echo -n string | sha256sum
```

L'option `-n` est importante afin d'éviter que `echo` ne rajoute automatiquement un caractère de passage à la ligne à la fin de la string.

1.3 Spécifications

1.3.1 Exécution

L'exécutable de votre programme doit s'appeler `cracker`. Il s'utilise comme suit:

```
./cracker [-t NTHREADS] [-l LETTRE] FICHIER1 [FICHIER2 ... FICHIERN]
```

Ce qui est entre `[]` est optionel. L'argument `-t` spécifie le nombre de threads de **calcul** à utiliser (les threads qui vont appeler la fonction d'inversion de hash). Le nombre de threads doit être un entier positif non nul. Si l'argument `-t` n'est

[1] `https://fr.wikipedia.org/wiki/Fonction_de_hachage`

pas spécifié, alors le nombre de threads par défaut est de 1. L'argument `-l` spécifie le critère de sélection des mots de passe. Si l'argument est absent, alors la lettre par défaut doit être la lettre `a`. Ensuite, il doit y avoir au moins un nom de fichier spécifié. D'autres fichiers (optionels) peuvent être spécifiés à la suite. Ces fichiers contiennent les hash de mots de passe.

Voici un exemple d'exécution.

```
./cracker -t 4 -l z passwords.txt
```

Cette commande exécute `cracker` avec 4 threads de calculs, la lettre de sélection `z`, et un fichier d'input nommé `passwords.txt`. Autre exemple:

```
./cracker pass1.txt pass2.txt pass3.txt
```

Cette commande exécute `cracker` avec 1 thread de calcul (valeur par défaut), la lettre de sélection `a` (valeur par défaut), et trois fichiers d'input.

1.3.2 Format des fichiers d'input

Chaque fichier d'input contient des hash de mots de passe sous forme **binaire** et pas textuelle. Les 32 premiers bytes d'un fichier forment donc le premier hash, les 32 suivants le deuxième, et ainsi de suite. Un fichier aura toujours une taille multiple de 32 bytes.

Les mots de passe à l'origine des hash ne sont composés que de **lettres minuscules**. Leur longueur ne dépasse pas 16 lettres.

1.3.3 Helper

Le but du projet n'étant pas d'implémenter une inversion de hash mais plutôt de correctement paralléliser ce travail, nous vous fournissons une fonction réalisant cette inversion. Le prototype de cette fonction est le suivant:

```
bool reversehash(const uint8_t *hash, char *res, size_t len);
```

Le paramètre `hash` est un pointeur vers un tableau de 32 bytes représentant un hash SHA-256. Le paramètre `res` est un pointeur vers une string, où sera écrit l'inverse du hash, s'il est trouvé. Le paramètre `len` indique la longueur maximale de l'inverse. La valeur de retour est `true` si un inverse est trouvé, `false` dans le cas contraire.

1.3.4 Sélection des mots de passe

Pour chaque hash, votre programme doit effectuer les deux opérations suivantes:

1. Appeler la fonction `reversehash` (**coûteux**)
2. Décider si le mot de passe obtenu est un candidat

Pour qu'un mot de passe donné soit candidat, il faut que son nombre d'occurrences de la lettre de sélection soit au moins aussi élevé que les autres candidats actuels.

Afin d'illustrer, imaginons que la lettre de sélection est la lettre `c` et que vous traitez les mots de passe suivants, dans l'ordre donné:

1. `coucou`
2. `calotte`
3. `cacao`
4. `poulet`
5. `cocorico`

L'ensemble des candidats est initialement vide. Le premier mot `coucou` est donc automatiquement considéré comme candidat avec deux occurrences de la lettre de sélection. Ensuite, le mot `calotte` n'est pas considéré comme candidat (une seule occurrence de la lettre). le mot suivant, `cacao`, a deux occurrences de la lettre. Il est donc considéré comme candidat. L'ensemble des candidats est à présent composé de deux mots de passe, `coucou` et `cacao`. Le mot de passe suivant est ignoré avec zéro occurrences de la lettre. Le dernier mot `cocorico` compte, quant à lui, trois occurrences de la lettre de sélection, ce qui est plus que les candidats actuels. Ceux-ci sont ainsi éliminés et `cocorico` devient le seul candidat.

1.3.5 Format de sortie

Lorsque tous les hash ont été traités, votre programme devra écrire sur la sortie standard (`stdout`) une ligne par candidat restant. Cette ligne sera composée uniquement du mot de passe (en clair) en question.

2 Délivrables

Le projet se déroule en trois phases: architecture, implémentation, reviews.

2.1 Architecture

Durant cette phase, vous devez spécifier à haut niveau l'architecture de votre programme. Il faut au moins définir:

- Structures de données (représentation des mots de passe, etc.)
- Types de threads entrant en jeu (quelle(s) tâche(s) ils réalisent)
- Méthode(s) de communication entre les threads
- Informations communiquées entre les threads

Afin de valider votre architecture, vous effectuerez une interview de 10 minutes avec les assistants. Cela vous permettra de ne pas partir sur une mauvaise piste. Les interviews se feront le **jeudi 30 mars** et **vendredi 31 mars**. Lors de votre interview, vous devez amener un **mini-rapport de une à deux pages** expliquant les quatre points d'architecture décrits ci-dessus.

2.2 Implémentation

Dès que votre architecture sera validée, vous pourrez commencer à implémenter. La remise de l'implémentation complète est fixée à la S12. Les modalités détaillées seront annoncées en temps voulu.

2.3 Reviews

Immédiatement après la remise de l'implémentation, vous recevrez chacun (individuellement) 5 projets d'autres groupes. Parmi ces projets, vous devrez en sélectionner deux qu'il vous faudra reviewer. La deadline des reviews est fixée à la S13. Les modalités suivront.