

Cassage de mot de passe par la méthode « brute force » en OpenMP & MPI

■■■■ Évaluer la qualité des mots de passe

La problématique du **cassage des mots de passe par la méthode « brute force »**, c-à-d trouver le mot de passe par une recherche plus ou moins exhaustive des différents mots de passe possibles, est essentielle dans l'administration d'un système :

- ▷ L'administrateur doit surveiller régulièrement le système dont il est le « responsable de la sécurité » afin de découvrir les mots de passe « faibles » :
 - ◇ soit c'est l'administrateur qui le fait en premier et il peut ensuite avertir l'utilisateur maladroit en fermant l'accès à son compte au besoin ;
 - ◇ soit c'est un attaquant et la sécurité du système complet peut être remise en cause.
- ▷ Un mot de passe est dit « faible », s'il est facilement « devinable » sans avoir forcément à connaître l'utilisateur auquel il est associé (inutile de faire du « Social Engineering » comme chercher le nom des enfants, la date de naissance, le nom du chien *etc.*) :
 - ◇ on essaie les mots du dictionnaire ou des variations sur ces mots (changement des o en 0, des i en 1, *etc.*) ;
 - ◇ on essaie les mots de passe choisis le plus souvent (*il faut pouvoir en obtenir la liste, ce qui n'est pas évident... mais ce qui arrive lorsqu'un « hacker » laisse fuiter le recueil de ses exploits...*)
- ▷ Il existe des outils « tout faits » comme « John the Ripper password cracker » qui sont optimisés et configurés pour réaliser ce travail.

■■■■ Présentation de la méthode de connexion sécurisée à attaquer

On utilise la méthode suivante pour la **gestion des accès à une machine** :

- a. l'utilisateur qui veut se connecter saisit un couple « login/mot de passe » ;
- b. le mot de passe subit une **fonction de « transformation »** qui donne un **résultat** ;
- c. ce résultat est comparé à une valeur enregistrée dans le système de gestion d'accès :
 - ◇ si le résultat est **identique** à la valeur enregistrée : la connexion est **acceptée** ;
 - ◇ si le résultat est **différent** : la connexion **échoue**.

On remarquera que, contrairement à certains sites Web mal gérés, les mots de passes ne sont pas stockés en clair dans la machine, c-à-d que l'on ne peut/doit pas, pouvoir vous renvoyer votre mot de passe par courrier en cas de perte !

La **fonction de « transformation »** est la suivante :

- on choisit une valeur entière i entre 1 et 5 : cette valeur *configure* le système : toutes les utilisations de la fonction de transformation dans le même système utilisent la même valeur ;
- on utilise le mot de passe entré par l'utilisateur et on calcule une « chaîne de hachés » ;
Exemple avec $i = 5$:
 - ◇ $valeur_initiale = mot_de_passe_entré$
 - ◇ $valeur_haché_1 = hash_{md5}(valeur_initiale)$
 - ◇ $valeur_haché_2 = hash_{md5}(valeur_haché_1)$
 - ◇ $valeur_haché_3 = hash_{md5}(valeur_haché_2)$
 - ◇ $valeur_haché_4 = hash_{md5}(valeur_haché_3)$
 - ◇ $résultat = hash_{md5}(valeur_haché_4)$.

Le **résultat** est :

- ▷ enregistré dans un fichier lors de la définition initiale du mot de passe de l'utilisateur ;
- ▷ recalculé et comparé à la version enregistrée lors d'une tentative de connexion de l'utilisateur.



■ ■ ■ Présentation de la méthode de cassage par « brute force »

Il s'agit de découvrir le mode de passe de l'administrateur « root » :

1. on récupère la liste des mots de passe les plus fréquemment utilisés ;
2. on récupère le fichier des résultats des chaînes de hachés des mots de passe d'un système ;
3. on « découvre » la valeur i utilisée sur le système : chercher un mot de passe faible d'un utilisateur ;
4. grâce à la valeur i trouvée, chercher le mot de passe de « root ».

Soit l'algorithme suivant :

- ▷ découverte de la valeur i qui configure le système, en réduisant la recherche sur toute cette liste aux 3 premiers mots de passes les plus utilisés :
 - ◇ pour chacun de ses mots de passe les plus utilisés :
 - * on construit la liste de « résultats » par la fonction de transformation pour les différentes valeurs de i possibles : {1, 2, 3, 4, 5} ;
 - * on recherche chacun de ces résultats dans la liste des mots de passe transformés :
 - ▷ si on trouve une correspondance : on a trouvé un mot de passe et également le i de configuration du système ;
 - ▷ si on ne trouve pas de correspondance on passe au mot de passe le plus utilisé suivant.
 - ◇ Si aucun des mots de passe les plus fréquemment utilisés ne correspond : « *Youpi ! soit votre système est « secure », soit vous avez fait une erreur !* »
- ▷ une fois la valeur de i découverte, on va chercher le mot de passe du compte « administrateur » :
 - ◇ pour chaque mot de passe de la liste des mots de passes les plus fréquemment utilisés :
 - * on construit le « résultat » par la fonction de transformation avec la valeur de i trouvée ;
 - * on compare ce résultat à la valeur du mot de passe « transformé » du fichier :
 - ▷ si on trouve une correspondance : on affiche le mot de passe et *on se moque de l'administrateur* ;
 - ▷ sinon ...

■ ■ ■ Contexte de l'implémentation

Dans le cadre du projet, vous trouverez sur la page <https://p-fb.net/master1/parapp/> :

- * une liste de mots de passe les plus fréquemment utilisés (un mot de passe par ligne) ;
- * une liste de mots de passe *transformés* (un mot de passe par ligne) pour **cinq systèmes différents** ;
Chaque liste contiendra 10 mots de passe transformés :
 - ◇ le premier de la liste est celui du compte administrateur ;
 - ◇ pour la valeur de i , et compte tenu de la puissance actuelle des machines, un coefficient multiplicateur de 40.000.000 a été appliqué, c-à-d que vous chercherez la valeur de i parmi les valeurs :
 - * $m = 40000000$
 - * $i \in \{1 * m, 2 * m, 3 * m, 4 * m, 5 * m, 6 * m, 7 * m, 8 * m, 9 * m\}$

■ ■ ■ Utilisation de la méthode « brute force » dans le contexte d'implémentation

La méthode « brute force » va être utilisée :

- * une première fois pour la recherche de la valeur i associée à chaque système :
 - ◇ *recherche des 3 mots de passe les plus fréquents dans le fichier complet de mot de passe d'un système avec les différentes valeurs possibles de i :*
 - * on essaye pour chaque valeur possible de i , chaque valeur de mdp possible parmi ces 3 :
 - ▷ soit on trouve le résultat obtenu dans le fichier du système la valeur obtenue : *Bingo !* on a trouvé i ;
 - ▷ soit on ne trouve pas le résultat obtenu dans le fichier et on passe à l'essai suivant.
*On notera que l'on fera au plus : $5 * 3 * 5 = 75$ calculs de « chaîne de hachés » pour 3 mdp à essayer multiplier par 5 valeurs de i et par 5 fichiers systèmes.*
- * une seconde fois pour la recherche du mot de passe de l'administrateur de chaque système :
 - ◇ *recherche de tous les mots de passes les plus fréquents en considérant **uniquement** le premier mot de passe contenu dans le fichier du système :*
 - * on calcul un résultat pour tous les mots de passe les plus fréquents avec la valeur i trouvée précédemment ;
 - * on compare au premier mot de passe du fichier système qui correspond au mot de passe associé à l'administrateur.
 - ▷ soit il y a correspondance, *Bingo !* on a trouvé le mot de passe administrateur ;
 - ▷ soit il y a différence et on passe à l'essai suivant.
*On notera que l'on fera au plus : $10 * 5 = 50$ tentatives pour 10 mots de passe les plus fréquents et 5 fichiers système.*

Travail à réaliser

Vous trouverez sur la page de l'UE sur <https://p-fb.net/master1/parapp/> un module de calcul de hash avec la méthode MD5 implémentée par Christophe Devine, c.devine@cr0.net, constitué des fichiers `md5.c` et `md5.h`.

Exemple d'utilisation :

```
#include <stdio.h>
#include <stdlib.h>
#include "md5.h"

void main()
{
    int i;
    unsigned char *ma_chaine = "toto";
    unsigned char mon_hash[MD5_HASHBYTES]; /* le MD5 renvoie 16 octets */

    calcul_md5(ma_chaine, 4, mon_hash);
    printf("le hash de %s est ", ma_chaine);
    for(i=0; i < MD5_HASHBYTES;i++)
        printf("%2.2x ", mon_hash[i]); /* Affichage sur deux caracteres*/
    printf("\n");
}
```

Attention

Lors du calcul de la « chaîne de haché », en recopiant l'ancienne valeur de haché pour la soumettre à la fonction de hachage, **n'utilisez pas les fonctions de gestion de chaîne de caractères**, comme `strncpy` qui s'arrêtent en présence d'un caractère à 0 qui peut être pris par un hash (celui-ci indiquant la fin pour une chaîne de caractère,). **Utilisez `memcpy`**.

- Écrire un programme réalisant la **parallélisation de la méthode « brute force »**, en vous servant des différentes possibilités d'exploitation de parallélisme offertes par OpenMP et MPI que vous pourrez combiner :
 - en exploitant le nombre de systèmes à casser (les 5 machines);
 - en exploitant le nombre de mot de passe à chercher (les 3 premiers au début pour découvrir i , puis les 10 ensuite pour le mdp root);
- Vous présenterez votre choix de parallélisation dans le document PDF joint :
 - vous le discuterez,
 - vous le justifierez.
- Vous mesurerez et noterez le temps pris par cette version en prenant un nombre fixe de threads et de nœuds de cluster comme référence.
- Vous comparerez avec une évaluation séquentielle (pour tout ou partie de votre travail).

Remise du travail

Le travail devra être remis sur le dépôt sur Community-Science associé à l'UE.

Pour les mesures et commentaires sur vos programmes vous joindrez un document PDF (pas un rapport) à cette archive avec des copies d'écran des résultats et la liste des mots de passe trouvés pour chaque système.