SECURITE SYSTEMES ET RESEAUX - LAB 1

Attaques sur les mots de passe

# Etude du fichier SHADOW

## Question 3.1.1

Structure du fichier :

Le fichier shadow se compose de 9 champs (séparés par des deux-points) :

**« username : passwd : last : may : must : warn : expire : disable : reserved »**

En détail :

* **USERNAME** : Le nom de l’utilisateur
* **PASSWD** : le mot de passe encodé
* **LAST** : Date de la dernière modification (en nombre de jours depuis le 1er janvier 1970)
* **MAY** : Nombre de jours avant que le mot de passe puisse être modifié
* **MUST** : Nombre de jours avant que le mot de passe doive être modifié
* **WARN** : Nombre de jours durant lesquels l’utilisateur est prévenu de l’expiration de son mot de passe
* **EXPIRE** : Nombre de jours entre l’expiration du mot de passe et la fermeture du compte
* **DISABLE** : Date de la fermeture du compte (en nombre de jours depuis le 1er janvier 1970)
* **RESERVED** : Champ réservé

Ainsi, en reprenant le fichier shadow fourni, un exemple serait :

« Root : $1$934b4a210c17493f68bf6bfe74bff77a : 16749 : 0 : 99999 : 7 : : : »

Algorithme utilisé :

Afin de déterminer l’algorithme utilisé pour générer les empreintes des mots de passe nous utilisons le référentiel suivant :

|  |  |
| --- | --- |
| **ALGORITHMES** | **IDENTIFIANTS** |
| **MD5** | $1 |
| **Blowfish** | $2 |
| **SHA256** | $5 |
| **SHA512** | $6 |

Ainsi nous trouvons une correspondance avec notre fichier shadow : $1 situé dans le champ du mot de passe. L’algorithme utilisé est le **MD5**.

# MISE EN ŒUVRE D’UN SCRIPT D’ATTAQUE PAR FORCE BRUTE

## Question 3.2.1

Attaque par force brute

L’attaque par **force brute**, également nommée « BruteForce » ou « Recherche exhaustive » consiste à construire et tester toutes les combinaisons possibles. Ceci, à partir d’une longueur de mot de passe et d’un alphabet de caractères (définis au préalable).

Ce type d’attaque s’avère très efficace ; en effet son taux de réussite est de 100% mais son utilisation peut s’avérer très longue dès lors que le mot de passe s’allonge (supérieur à 8 caractères). Néanmoins ce temps de traitement peut être optimisé en utilisant les « chaînes de Markov ».

## Question 3.2.2

Mise en œuvre du script

Fichier :

shadowBreaker.py

Lire le contenu du fichier shadow :

Afin de lire le contenu du fichier shadow ligne par ligne nous avons écrit une fonction intitulée **extractShadowPasswords**.

Cette fonction a pour objectif d’ouvrir le fichier shadow, de le lire, d’extraire les mots de passe et de les afficher à l’écran (si l’extraction s’avère possible).

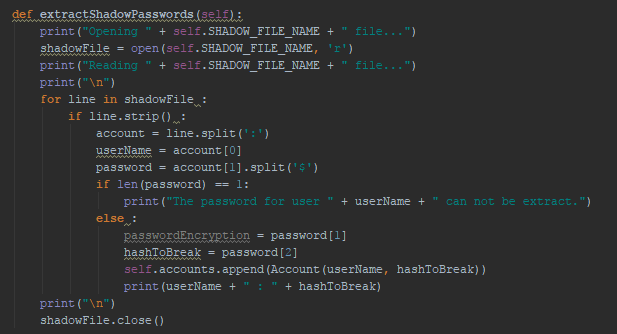


Figure 1 - Fonction extractShadowPasswords

Tentative d’attaque par BruteForce :

Ainsi, grâce à la méthode d’attaque par BruteForce, nous avons découvert 3 mots de passe dissimulés dans le fichier shadow :

* Utilisateur : ROOT – mdp : t3\_p@hEN\_v12
* Utilisateur : GISELLE – mdp : brazil
* Utilisateur : FRED – mdp :

Les autres mots de passe n’étaient pas extractibles du fichier shadow.

# MISE EN ŒUVRE D’UN SCRIPT D’ATTAQUE PAR DICTIONNAIRE

## Question 3.3.1

Attaque par dictionnaire

L’attaque par dictionnaire consiste à comparer une liste de mots de passe prédéfinis dans un fichier, le dictionnaire, et les comparer un a un au mot de passe souhaitant être découvert.

Pour ce faire, on lit le fichier dictionnaire ligne par ligne, puis on encode chaque ligne, afin de les comparer au mot de passe chiffré. Si une correspondance est trouvée, on décode la ligne correspondante du dictionnaire : on trouve alors le mot de passe.

Cette méthode est très rapide mais nécessite un dictionnaire le plus exhaustif possible et dépend de la prédictibilité du mot de passe.

Pour améliorer les chances de résultats, on adapte ce fichier à la langue par exemple : il existe des dictionnaires pour le français, l’anglais...etc. Cette méthode ne donne bien évidemment pas un résultat positif pour tous les mots de passe, mais s’avère efficace car la plupart des utilisateurs utilisent souvent les mêmes.

## Question 3.3.2

Mise en œuvre du script

Fichier :

shadowBreaker.py

Lire le contenu du fichier shadow :

Afin de lire le contenu du fichier shadow, nous utilisons de nouveau la fonction **extractShadowPasswords** (réf. Figure 1)

Tentative d’attaque par dictionnaire :

Nous utilisons la fonction **dictionary** pour ouvrir le fichier dico\_mini\_fr et tester l’occurrence des mots de passe présents dans le fichier shadow.

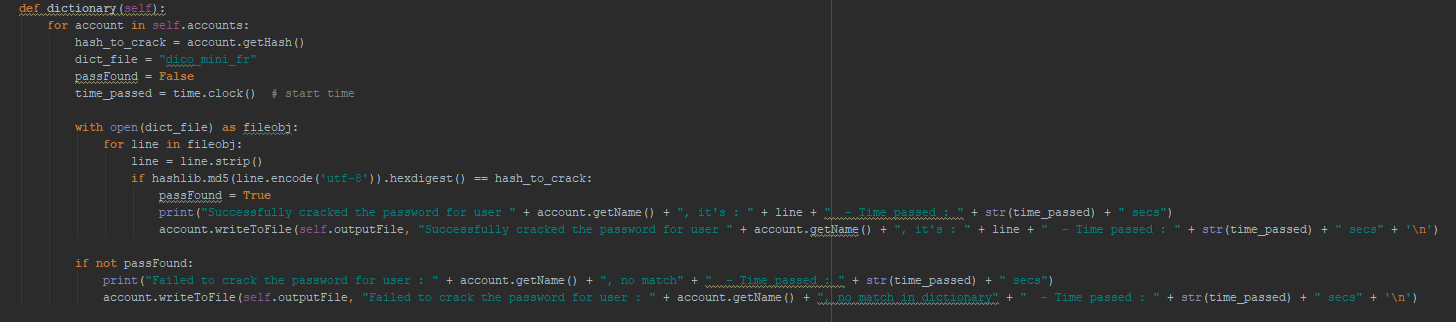


Figure 2 - Fonction dictionary

Cette fonction fait appel à la classe : **Account** qui permet de récupérer le nom de l’utilisateur, son mot de passe hashé et d’écrire dans un fichier texte les résultats de notre tentative de déchiffrement.

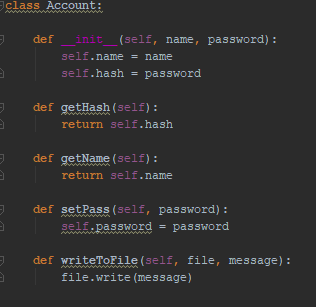


Figure 3 - Classe Account

Dans la fonction dictionary, nous ouvrons le fichier **dico\_mini\_fr** avec l’expression **« with open(dict-file) as fileobj** **»**.

Une fois le fichier ouvert, nous le parcourons ligne par ligne avec l’expression : **« for line in fileobj »**, puis pour chacune de ces lignes, nous encodons son contenu en md5 et le comparons au mot de passe chiffré de chaque utilisateur.

Si la comparaison s’avère vraie, nous affichons dans la console le résultat et l’écrivons dans le fichier de sortie **« outpout.txt »,** sinon nous affichons que le déchiffrage s’est avéré être un échec.

Le temps de traitement pour le déchiffrement de chaque mot de passe est également affiché dans la console et écrit dans le fichier outpout.txt.

Ainsi, grâce à la méthode d’attaque par dictionnaire, nous avons découvert 2 mots de passe dissimulés dans le fichier shadow :

* Utilisateur : ROOT / Mdp : t3\_p@hEN\_v12
* Utilisateur : GISELLE / Mdp : brazil

Le mot de passe de l’utilisateur FRED n’est pas contenu dans le dictionnaire. Les autres mots de passe n’étaient pas extractibles du fichier shadow.

Ainsi, notre programme affiche dans la console :

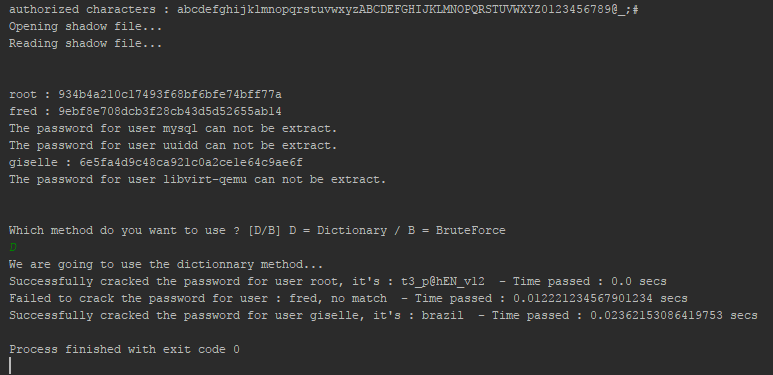


Figure 4 - Programme shadowBreaker - attaque par dictionnaire

Et écrit dans le fichier outpout.txt les informations suivantes :



Figure 5 - fichier outpout.txt - attaque par dictionnaire

Tentative d’attaque par brute force :

Lorsque des mots de passes n’ont pas été trouvé avec la méthode par dictionnaire, la méthode par brute force s’occupe du problème de façon un peu moins subtile mais efficace.

En effet, nous allons tester toute les chaînes de caractères possibles.

Pour ça nous utilisons les constantes de chaines fournies par python :



Puis, pour construire nos différentes possibilités de mots de passe et itérer dessus, nous utilisons la librairie très utile itertools :



## Comparaison des résultats de la méthode BruteForce et Dictionnaire :

La méthode BruteForce nous a permis de découvrir trois mots de passes extraits du fichier shadow, ce qui était le maximum, puisque les trois autres ne s’avéraient pas être extractibles.

En revanche, la méthode d’attaque par dictionnaire ne nous a permis d’en déchiffrer que deux sur les trois possibles. Néanmoins, elle s’est avérée bien plus rapide, car l’exécution du script n’as pris qu’une seconde contre plusieurs heures.

Ainsi, l’attaque par BruteForce est nettement plus efficace si l’on prête attentions au nombre de résultats, mais bien moins en termes de performances et d’optimisation du temps.

## Avis, remarques, interrogations

Concernant notre développement, nous avons rencontré des difficultés quant à l’optimisation de notre BruteForce, nous avons au fil de la phase de développement, gagné peu de temps d’exécution ; néanmoins les résultats étaient présents et notre attaque a fonctionnée.

L’attaque par dictionnaire s’est avérée relativement simple et étonnement efficace en termes de temps, nous le supposions, mais pas autant.

Ce TP fu très instructif, soulevant de grandes questions d’optimisation et de temps de traitement, en espérant pouvoir gagner en temps d’exécution par le futur.

Nous vous joignons notre code source contenant les fichiers suivants :

* shadow.txt
* dico\_mini\_fr.txt
* outpout.txt
* shadowBreaker.py
* README.txt