Cela fonctionne remarquablement bien car, sans contraintes particulières, nous sommes tous tentés de choisir des mots simples, des noms, des prénoms, des petites phrases, et que les possibilités sont alors relativement peu nombreuses.

Cette utilisation non uniforme des possibilités équivaut à une réduction de l'espace des possibilités, ce qui diminue le nombre moyen d'essais pour casser un mot de passe.

Voici les 25 premiers éléments d'un de ces dictionnaires de mots de passe. Elle est tirée d'une base de 5 millions de mots de passe ayant fuité en 2017 et repérés par SplashData:

1. 123456; 2. Password; 3. 12345678; 4. qwerty; 5. 12345; 6. 123456789; 7. letmein; 8. 1234567; 9. football; 10. iloveyou; 11. admin; 12. welcome; 13. monkey; 14. login; 15. abc123; 16. starwars; 17. 123123; 18. dragon; 19. password; 20. master; 21. hello; 22. freedom; 23. whatever; 24. qazwsx; 25. trustno1.

Remarquez le «password» en seconde position et le sympathique «iloveyou» en position 10. Bien sûr, des telles listes changent selon le pays où elles sont collectées, les sites concernés et au cours du temps.

Pour les mots de passe de 4 chiffres (par exemple le code PIN des cartes SIM des téléphones), les résultats sont encore plus étonnants. En 2013, le site DataGenetics, s'appuyant sur une collecte de 3,4 millions de mots de passe, chacun comportant 4 chiffres, a mesuré que la suite de 4 chiffres la plus utilisée était 1234, qui représentait 11% des choix, suivie par 1111 et 0000 avec des scores de 6% et 2%.

Le mot de passe de 4 chiffres le moins utilisé était 8068. Méfiez-vous quand même, ce n'est peut-être plus vrai maintenant que ces résultats ont été publiés. Le choix 8068 n'était présent que 25 fois parmi les 3,4 millions de suites de 4 chiffres de la base, ce qui est beaucoup moins que les 340 utilisations qu'on aurait trouvées si chaque combinaison de 4 chiffres avait été utilisée avec la même fréquence.

Les 20 premières séries de 4 chiffres sont: 1234; 1111; 0000; 1212; 7777; 1004; 2000; 4444; 2222; 6969; 9999; 3333; 5555; 6666; 1122; 1313; 8888; 4321; 2001; 1010.

Même sans dictionnaire de mots de passe, l'utilisation des différences entre fréquences d'usage des lettres (ou des doubles lettres) dans une langue permet d'organiser efficacement une attaque. Certaines méthodes d'attaque prennent aussi en compte que, pour faciliter la mémorisation, on choisit des mots ayant une certaine structure comme A1=B2=C3, AwX2AwX2, O00.Ili. (que j'ai longtemps utilisé), ou alors obtenus en combinant plusieurs mots simples comme password123 ou johnABC0000. En exploitant de telles régularités, on détermine un ordre d'attaque des mots de passe qui accélère la recherche. La conclusion est simple: dans un espace donné, il faut vraiment choisir au hasard son mot de passe. Certains logiciels le font pour vous et vous proposent donc un mot de passe au hasard. Il faut cependant être conscient qu'un logiciel qui engendre un mot de passe au hasard risque d'utiliser, délibérément ou non, un mauvais générateur pseudoaléatoire, auquel cas ce qu'il propose sera imparfait.

Informés de tout cela, les sites internet convenablement conçus analysent les mots de passe proposés au moment de leur création et refusent ceux qu'il serait trop facile de retrouver. C'est agaçant, mais c'est pour votre bien!

TESTER LES MOTS DE PASSE...

Un outil permet de vérifier si l'un de vos mots de passe n'a pas déjà été piraté. Il utilise une base de plus de 500 millions de mots de passe obtenue en regroupant des listes dévoilées à la suite d'attaques diverses:

https://haveibeenpwned.com/Passwords

J'ai essayé e=mc2e=mc2 que j'aimais bien et pensais sûr, et j'ai obtenu la réponse déconcertante: «This password has been seen 110 times before.» Quelques autres essais montrent qu'il est difficile de proposer un mot de passe facile à mémoriser et que la base ne connaît pas. Par exemple, aaaaaa est apparu 353071 fois; a1b2c3d4, 105134 fois; abcdcba, 353 fois; abczyx: 158 fois; acegi, 111 fois; chirac, 600 fois; sarkozy, 680 fois; hollande, 832 fois; macron, 279 fois.

Notons qu'être original reste possible. Voici cinq exemples de mots de passe que le site n'a pas dans sa liste: eyahaled (mon nom à l'envers); bizzzzard; meaudepace; modeuxpass; abcdef2018. Maintenant que je les ai essayés, je me demande si la base ne pourrait pas les ajouter lors de sa prochaine mise à jour et je ne les utiliserai donc pas!

Le NIST (National Institute of Standards and Technology), aux États-Unis, a récemment publié un avis où il recommande la méthode des dictionnaires pour filtrer le mot de passe choisi par un utilisateur qui est en train d'en créer.

... ET SAVOIR LES CONSERVER

Parmi les règles qu'un bon concepteur de serveur internet doit absolument appliquer, il y a celle-ci: ne pas garder dans les mémoires de l'ordinateur qui fait fonctionner le site internet la liste en clair des couples [identifiant d'utilisateur, mot de passe].

La raison est évidente: un pirate pourrait accéder à l'ordinateur contenant cette liste, soit parce que le site est mal protégé, soit parce qu'une faille profonde dans le système, ou dans la puce au cœur de la machine, aura été exploitée, alors qu'elle est inconnue de tous (faille Zero-day)... sauf de l'attaquant. >

BIBLIOGRAPHIE

Wikipedia, **Mot de passe** et **Rainbow table**, entrées consultées en juin 2018.

G. Avoine *et al.*, **How to handle rainbow tables with external memory**, dans J. Pieprzyk et S. Suriadi (éd.), *Information Security and Privacy - ACISP 2017*, pp. 306-323, Springer, Cham, 2017.

P. A. Grassi *et al.*, **Digital Identity Guidelines**, NIST Special Publication 800-63-3, 2017 (https://doi.org/10.6028/NIST.SP.800-63-3).

G. Avoine *et al.*, **Characterization and improvement of time-memory trade-off based on perfect tables**, *ACM Transactions on Information and System Security*, vol. 11(4), article 17, 2008.

P. Oechslin, **Making a faster cryptanalytic time-memory trade-off**, dans D. Boneh (éd.), *Advances in Cryptology - CRYPTO 2003*, pp. 617-630, Springer, 2003.

M. E. Hellman, **A cryptanalytic time-memory trade-off**, *IEEE Transactions on Information Theory*, vol. IT-26, n° 4, pp. 401-406, 1980.

3 LES TABLES ARC-EN-CIEL

Vous êtes un pirate informatique et vous cherchez le moyen d'exploiter une ou plusieurs données que vous vous êtes procurées. Ces données sont du type [identifiant d'utilisateur, h(mot de passe)], où h est une fonction de hachage connue et fixée *(voir l'encadré 2)*. Le mot de passe appartient à l'espace obtenu avec 12 lettres minuscules, ce qui correspond à 56 bits d'information et à $26^{12} = 9{,}54 \times 10^{16}$ mots de passe possibles.

Méthode 1. Vous faites défiler tous les mots de passe M possibles. Vous calculez l'empreinte h(M) de chacun en regardant si c'est la bonne. Vous n'avez pas besoin d'un important volume de mémoire, car, à chaque nouvel essai, vous effacez les résultats précédents, sauf bien sûr le point de repère qui vous permet de savoir où vous en êtes de votre énumération. En revanche, il vous faut beaucoup de temps pour faire défiler tous les mots de passe possibles. Si votre ordinateur peut mener un milliard d'essais par seconde, il vous faut $26^{12}/(10^9 \times 3\,600 \times 24) = 1\,104$ jours, soit environ 3 ans. Ce n'est pas impossible à envisager et, en utilisant un réseau d'ordinateurs d'un millier de machines, une journée suffira. Cependant, il n'est pas envisageable de reprendre un tel calcul à chaque fois que vous découvrirez une nouvelle donnée de la même catégorie.

Méthode 2. Vous vous dites : « Je vais calculer les empreintes de tous les mots de passe, ce qui me prendra du temps, mais je vais mémoriser ces empreintes dans une grande table. Je n'aurai ensuite qu'à regarder cette table pour savoir quel mot de passe a été utilisé, et cela me servira pour tout nouveau mot de passe de la même catégorie dont je réussirai à connaître l'empreinte. »

Il faudra $(9{,}54 \times 10^{16}) \times (12 + 32)$ octets de mémoire, car il faut 12 octets pour le mot de passe et 32 pour l'empreinte si elle comporte 256 bits comme pour la fonction SHA256. Cela fait $4{,}2 \times 10^{18}$ octets, soit 4,2 millions de disques durs d'une capacité de 1 téraoctet !

La mémoire requise est trop grande. Cette méthode n'est pas plus envisageable que la première. La méthode 1 exige trop de calcul, la méthode 2 trop de mémoire. Ça coince dans les deux cas : le calcul est trop long pour chaque nouveau problème, ou le précalcul avec mémorisation des résultats trop volumineux pour conserver les calculs faits.

Pourrait-on trouver une méthode intermédiaire qui exigerait moins de calculs que la méthode 1 pour chaque nouvelle donnée, mais peut-être un peu plus de mémoire, et qui exigerait moins de mémoire que la méthode 2, mais peut-être un peu plus de calculs ? Oui, c'est possible. L'idée a été proposée par l'Américain Martin Hellman en 1980, puis perfectionnée en 2003 par Philippe Oechslin, de l'École polytechnique fédérale de Lausanne, et plus récemment encore par Gildas Avoine, de l'Insa de Rennes.

Voici comment procéder : on se dote d'abord d'une fonction C qui, à partir de l'empreinte h(M) du mot de passe M produit un nouveau mot de passe $C(h(\text{M}))$. Les empreintes sont des nombres écrits en binaire, et les mots de passe des nombres écrits dans une base de numération ayant autant de chiffres qu'il y a de symboles autorisés (disons K). Cette fonction C, dite de conversion, sera par exemple une conversion de la base 2 à la base K. Le précalcul crée des tables de données dénommées *tables arc-en-ciel*, sans doute parce qu'elles sont comme un ensemble de lignes parallèles. Pour une donnée de cette table, on part d'un mot de passe quelconque M_0, on calcule son empreinte $h(\text{M}_0)$, puis un nouveau mot de passe possible $C(h(\text{M}_0)) = \text{M}_1$, et on recommence à partir de M_1, et ainsi de suite. Sans rien mémoriser d'autre que M_0, on calcule ainsi la suite M_1, M_2, ... jusqu'à ce que $h(\text{M}_n)$ commence par 20 zéros, ce qui ne se produit qu'une fois sur 1 000 000 environ (car ce que produit une fonction de hachage est assimilable à un tirage aléatoire uniforme et que 2^{20} vaut à peu près 1 000 000). On mémorise alors le couple $[\text{M}_0, h(\text{M}_n)]$ dans la table.

On calcule un très grand nombre de couples de ce type. Si, lors d'un calcul, on tombe sur le même $h(\text{M}_n)$ final, on n'ajoute rien, sauf si le calcul a été plus long (en nombre d'étapes), auquel cas on garde le nouveau couple et on efface l'ancien.

Chaque couple $[\text{M}_0, h(\text{M}_n)]$ représente la série de mots de passe M_0, M_1, ..., M_k et leurs empreintes, mais sans les mémoriser. Si l'on veut ne laisser aucun vide dans le tableau calculé, le temps de calcul sera plus important que celui nécessaire pour faire dérouler tous les mots de passe de l'espace qu'on explore. C'est envisageable avec un réseau d'ordinateurs, mais on peut ne mener le calcul de la table que partiellement : la base constituée ne résoudra alors qu'une partie des mots de passe dont on connaîtra l'empreinte. Dans le cas d'un calcul complet, l'espace mémoire pour stocker (indirectement) tous les couples calculés est, en ordre de grandeur, un million de fois plus petit que celui de la méthode 2 envisagée plus haut. Il est donc de moins de 4 disques durs de 1 téraoctet. Cette fois, c'est facile.

Voyons maintenant comment cette donnée enregistrée dans les disques permet de retrouver chaque mot de passe de l'espace envisagé en quelques secondes. Nous supposerons pour

$$\text{M}_0 \xrightarrow{h} h(\text{M}_0) \xrightarrow{C} \text{M}_1 \xrightarrow{h} h(\text{M}_1) \xrightarrow{C} \text{M}_2 \xrightarrow{h} \dots \xrightarrow{C} \text{M}_n$$

$$\text{N}_0 \xrightarrow{h} h(\text{N}_0) \xrightarrow{C} \text{N}_1 \xrightarrow{h} h(\text{N}_1) \xrightarrow{C} \text{N}_2 \xrightarrow{h} \dots \xrightarrow{C} \text{N}_p$$

....

$$\text{Q}_0 \xrightarrow{h} h(\text{Q}_0) \xrightarrow{C} \text{Q}_1 \xrightarrow{h} h(\text{Q}_1) \xrightarrow{C} \text{Q}_2 \xrightarrow{h} \dots \xrightarrow{C} \text{Q}_r$$

En connaissant le début de chaque ligne et la fin (ce sont les seules choses qu'on mémorise lors du précalcul), le pirate peut retrouver tout mot de passe dont il connaît l'empreinte : on part de l'empreinte connue qui est quelque part dans le tableau, et on applique C et h plusieurs fois, ce qui amène en bout de ligne. Cela permet de repérer la ligne où se trouve le mot de passe.
Connaissant la ligne où se trouve le mot de passe, on repart du début de la ligne (qui est connu) et, en quelques applications de h et C, on retombe sur l'empreinte qu'on connaît, ce qui indique le mot de passe recherché.
Beaucoup de calculs auront été nécessaires pour établir la première et la dernière colonne de la table. Mais ensuite, en ne mémorisant que deux colonnes, et moyennant un calcul (en gros le parcours d'une ligne), on retrouve tout mot de passe à partir de son empreinte.