Implémentation de la technique dite de

« Time-Memory Trade-Off »

dans le décryptage de mots de passe Windows (SAM LM).

Inspiré du document de Philippe Oeschlin http://lasecwww.epfl.ch/pub/lasec/doc/Oech03.pdf

du Laboratoire de Sécurité et de Cryptographie (LASEC) de Lausanne.

Par Jérôme ATHIAS

Jerome@ATHIAS.fr

http://www.athias.fr/jerome/DOC/

Dernière mise à jour : 26/06/2004

Avant propos :

En 1980, Martin Hellman décrit une technique de « Time-Memory Trade-Off » qui réduit le temps de décryptage basée sur l'utilisation de données pré calculées chargées en mémoire.
Cette méthode fut ensuite implémentée par Rivest avant 1982 en introduisant des séparateurs qui réduisent considérablement le nombre de boucles d'analyse de la mémoire pendant le décryptage.

Depuis, cette technique a été très largement étudiée, mais aucune optimisation publiée.
Philippe Oeschlin propose une nouvelle approche pour le calcul des données qui réduit par 2 le nombre de calculs préalables pour le décryptage.
Sa méthode n'utilisant pas de séparateurs particuliers, ce qui réduit la longueur des chaînes et, de facto, le nombre de calculs.

Il a ainsi pu décrypter 99.9% de tous les mots de passes Windows alphanumérique (2^37) en 13.6s en utilisant 1.4Go de données (2 CDS) alors que cela prend 101s avec la technique « traditionnelle ».
L'accélération peut être plus conséquente en fonctions des paramètres utilisés.

Introduction :

Le décryptage exhaustif requière beaucoup de puissance de calcul et beaucoup de temps. Quand la même attaque est menée plusieurs fois, il est possible de l'exécuter complètement préalablement et de stocker les résultats en mémoire. Une fois ce pré calcul réalisé, il devient alors possible de faire un décryptage quasi instantané. Ceci requiert beaucoup de mémoire.

Dans un système comme Windows, où les mots de passe sont cryptés en partant du mot de passe en clair comme clé et d'une chaîne de caractère fixe, qui donne un mot de passe crypté (password hash), on peut implémenter la technique de Time-Memory Trade-Off en générant des « tables de correspondances » pré calculées entre le mot de passe en clair et le mot de passe cryptés.

Quoiqu'il en soit cette technique n'est pas garantie et s'inscrit dans le domaine des probabilités, le résultat étant fonction de la mémoire et du temps dont on dispose pour réaliser le décryptage.

Méthode originale :

Se référer au document de Philippe Oeschlin.

Références :

Mots de passe locaux de Windows NT - Christophe GRENIER

Application : :

« En quelques mots, RainbowCrack est un crackeur de hash. Alors que les bruteforces traditionnaux test tous les mots de passe possibles 1 par 1, RainbowCrack procède autrement. Il calcule à l'avance toutes les combinaisons mot de passe en clair - mot de passe crypté et les enregistre dans des fichiers appelés "rainbow tables". Cela peut prendre beaucoup de temps pour pré calculer les tables, mais une fois terminé, vous pourrez cracker le mot de passe crypté en quelques secondes s'il se trouve dans vos tables. »

Traduction personnelle, de Zhu Shuanglei.

" In short, the RainbowCrack tool is a hash cracker. While a traditional brute force cracker try all possible plaintexts one by one in cracking time, RainbowCrack works in another way. It precompute all possible plaintext - ciphertext pairs in advance and store them in the file so called "rainbow table". It may take a long time to precompute the tables, but once the one time precomputation is finished, you will always be able to crack the ciphertext covered by the rainbow tables in seconds .", Zhu Shuanglei.

Calculs préliminaires

Différents paramètres entrent en jeu dans la configuration des calculs préliminaires.

En fonction des chances de réussite recherchées, il faudra ajuster la *charset* (ensemble des caractères utilisés), la place disque disponible...

Le *key space*, ou ensemble des possibilités, représente le nombre de mots de passe possibles en fonction du *charset* utilisé.

Ainsi si l'on considère toutes les lettres de l'alphabet :

charset alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

keyspace = 26^1 + 26^2 + 26^3 + 26^4 + 26^5 + 26^6 + 26^7 = 8353082582

==> Il y a 8 353 082 582 combinaisons possibles.

Pour obtenir une estimation des paramètres à utiliser afin d'obtenir un résultat représentatif dans un délai raisonnable, je programmai un petit utilitaire.

Nous générerons des rainbow tables permettant de cracker des mots de passe composés des caractères :

"ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/€ "

NB : la casse des mots de passe est "indifférente"; elle n'entre pas en jeu ici; majuscules et minuscules seront retrouvées

En accord avec Zhu Shuanglei, ces paramètres ne sont pas les meilleurs dans le sens où ils mettent en jeu l'utilisation d'un espace disque final conséquent, ce qui ralentit le processus de cracking. D'autres paramètres permettraient d'obtenir le même pourcentage de chance de réussite avec un espace disque plus réduit, mais nécessitent un temps de pré calcul plus important.

C'est là la finalité recherchée qui dicte l'adaptation des paramètres à utiliser pour l'optimisation vers l'obtention de cette finalité. :-)

Pour réaliser le calcul des tables, j'utilisai plusieurs ordinateurs personnels :

Portable P3-750Mhz/256 RAM (SD133) :

==> 100 000 chaînes/15 min 3 s

==> 1 table/5 jours

P4 1.7Ghz/256 RAM (SD133) :

==> 100 000 chaînes/7 min 51 s

==> 1 table/ 2,6 jours

Athlon XP 1800~1.5Ghz/1Go RAM (SD133) :

==> 100 000 chaînes/6 min 47 s

==> 1 table/ 2,36 jours

P4 2.8C HT~2.8-3Ghz/1.5Go RAM (DDR400) :

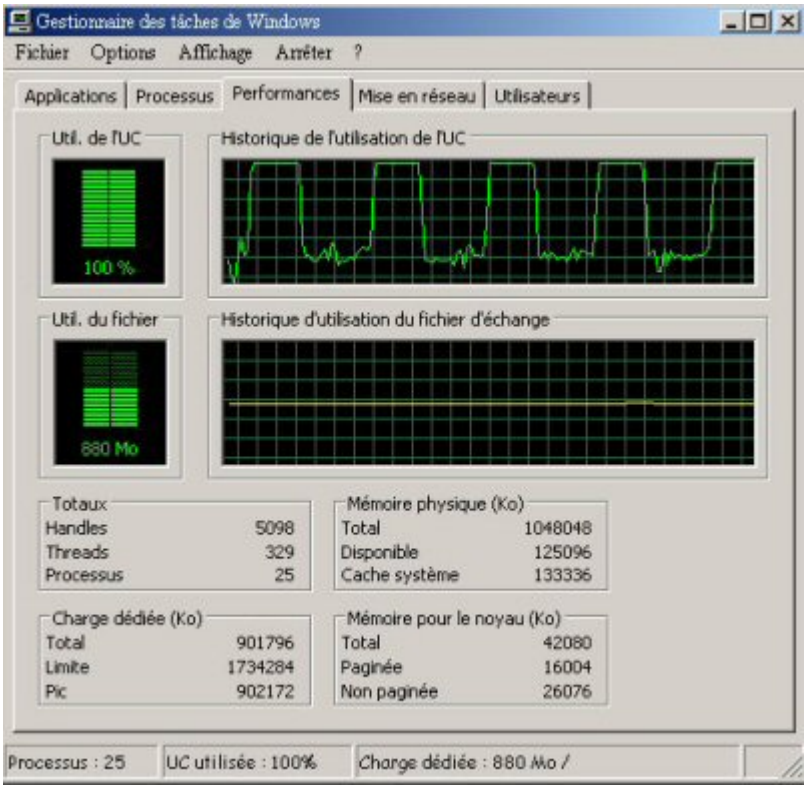
==> 100 000 chaînes/4~5 min (suivant l'overclocking)

==> 1 table/ 1,72 jours

Tri

Les tables doivent être triées pour maximiser les performances.

LE crack :



Comme vous pouvez le constater, le processus de crack se déroule bien avec rc :-)

Quelques résultats :

Crack de 10 mots de passe alphanumériques en 42 secondes

P4 2.8C / 1.5Go DDR 400 / HD : 80Go Maxtor ATA133 2Mo cache

lm_all_40_3200x50000000_bla.rt:
800000000 octets lus, temps d'accès disque: 17.17 s
verifying the file...
recherche de 2 hashes...
Temps d'analyse: 13.38 s

lm_all_41_3200x50000000_bla.rt:
800000000 octets lus, temps d'accès disque: 15.29 s
verifying the file...
recherche de 2 hashes...
Temps d'analyse: 13.37 s

AMD 1800XP/1Go SDRAM133 / HD : 80 Go Maxtor ATA133 2Mo cache

reading lm_all_10_3200x50000000_bla.rt ...
800000000 octets lus, temps d'accès disque: 22.71 s
verifying the file ...
recherche de 2 hashes ...
Temps d'analyse: 13.62 s

reading lm_all_11_3200x50000000_bla.rt ...
800000000 octets lus, temps d'accès disque: 20.98 s
verifying the file ...
recherche de 2 hashes ...
Temps d'analyse: 13.63 s

AMD 1800XP/1Go SDRAM133 / HD : 80 Go Maxtor ATA133 2Mo cache (avec seulement quelques tables)

Statistiques :

Chaînes déchiffrées: 2 of 2 (100.00%)
Temps d'accès disque: 722.08 s
Temps d'analyse: 430.33 s
Nombre de chaînes testées :: 335372946
Nombre de fausses alarmes: 2071
Nombre de fausses alarmes step: 2185376

Utilisateur Mot de passe

Administrator R4$T5%Y6^U7&

P4 2.8C HT oc~3.0Ghz /512 DDR 400 (swapping)-1.5/ HD : 80 Go HITACHI S-ATA 8Mo cache

lm_alpha-numeric_0_2400x40000000_bla.rt:
238522368 octets lus, temps d'accès disque: 8.72 s
verifying the file...
recherche de 10 hashes...
L'équivalent de 7655f495252646f5 is RRZQLTE
L'équivalent de 6cf46db4e8ca6ac1 is ELGVKUR
L'équivalent de 40eeedcc9c5f8e36 is WOXSUCH
L'équivalent de 2668dc65b5b938bf is CEJJMML
Temps d'analyse: 20.50 s
238522368 octets lus, temps d'accès disque: 6.20 s
recherche de 6 hashes...
L'équivalent de 563355e04cb3025a is SENHRGE
L'équivalent de afba921022b081de is ZYXTOL
Temps d'analyse: 1.47 s
162955264 octets lus, temps d'accès disque: 4.38 s
recherche de 4 hashes...
Temps d'analyse: 0.83 s

…

Statistiques :

Chaînes déchiffrées: 10 of 10 (100.00%)
Temps d'accès disque: 52.91 s
Temps d'analyse: 33.61 s
Nombre de chaînes testées : 32041990
Nombre de fausses alarmes: 9260
Nombre de chaînes testées à cause des fausses alarmes : 8099307

Résultats

563355e04cb3025a SENHRGE hex:53454e48524745
7655f495252646f5 RRZQLTE hex:52525a514c5445
6cf46db4e8ca6ac1 ELGVKUR hex:454c47564b5552
40eeedcc9c5f8e36 WOXSUCH hex:574f5853554348
60707a26021168d7 QGFBCCH hex:51474642434348
afba921022b081de ZYXTOL hex:5a5958544f4c
9a67ea388309e406 IHSZGRD hex:4948535a475244
2668dc65b5b938bf CEJJMML hex:43454a4a4d4d4c
16b58f3a714d5484 NGJKSWA hex:4e474a4b535741
dcd799d90644e30c LAWPNJ hex:4c4157504e4a

Il est amusant (et inquiétant?) de voir que l'on peut facilement déchiffrer des mots de passe de comptes auto attribués par Windows :

ASPNET [>x87r9E8001&7 hex:5b3e7838372394538303031263 7
ASPNET %oCIA$M6m/)O1P hex:256f434941244d366d2f294f3150

Meilleures salutations à Philippe Oechslin et aux étudiants du LASEC, à Zhu Shuanglei et à tout ceux qui s'en ressentiront comme destinataires ;-)

Ne soyez jamais effrayés d'essayer quelque chose de nouveau. Rappelez-vous que des amateurs bâtirent une arche, des professionnels produisirent le Titanic.