

Cassage et durcissement des mots de passe

Seconde partie : Unix

Denis Ducamp / Hervé Schauer Consultants

Novembre 2002

Le premier article a abordé un certain nombre de problèmes au sujet des mots de passe et leurs applications dans le milieu Windows. Ce second article présente cela dans le milieu Unix. À chaque fois, les aspects système, applications et réseau sont décrits. **Introduction**

Le présent article est composé de 5 parties :

1. La localisation des empreintes sous Unix : pour savoir où l'attention des administrateurs doit se porter.
2. Les différents algorithmes de chiffrement sous Unix : comment, sur les systèmes et les réseaux Unix, les mots de passe sont chiffrés.
3. Les logiciels de cassage contre Unix : transformer en protection les meilleurs logiciels, avant que les pirates ne les emploient pour vous attaquer.
4. La protection des mots de passe sous Unix : comment protéger ses mots de passe aux niveaux système, applications et réseau.
5. Le durcissement des mots de passe sous Unix : comment améliorer la qualité des mots de passe choisis par l'utilisateur.

Ce second article spécifique à Unix ne redonne pas les notions génériques qui ont déjà été abordées. Pour prendre connaissance des notions de "chiffrement et stockage d'un mot de passe", de "méthodes de cassage" et de "règles de constitution de bons mots de passe", consultez le premier article.

1 La localisation des empreintes sous Unix

Une empreinte est le résultat du hachage d'un mot de passe, donc une transformation par un algorithme non-réversible. Cette opération enregistre dans le système une "image" du mot de passe et empêche quiconque d'accéder au mot de passe en clair.

La localisation des empreintes dépend fortement du contexte : les empreintes systèmes et les empreintes applicatives.

1.1 La localisation des empreintes systèmes sous Unix

Historiquement, les mots de passe sous Unix sont enregistrés dans le fichier */etc/passwd*. Ce fichier devant être en lecture pour tout le monde, les empreintes ont été déplacées dans des fichiers accessibles qu'aux administrateurs, appelés les *shadow passwords*. Suivant le type d'Unix le fichier diffère :

- Système V : */etc/shadow* (Linux, Solaris, etc.)
- Systèmes BSD : */etc/master.passwd* (NetBSD, FreeBSD, OpenBSD, BSDi, etc.)
- AIX : */etc/security/passwd*
- etc.

L'idée importante est qu'aucun utilisateur n'a accès aux empreintes des autres et que seuls des processus privilégiés puissent y accéder.

Pour Owl, une alternative au système *shadow* a été créée. Owl est une distribution Linux à la sécurité améliorée avec comme principale approche la revue proactive de code source pour plusieurs types de vulnérabilités logicielles, voir le site <http://www.openwall.com/Owl/fr/> pour plus d'informations. Il découpe le fichier des mots de passe en un fichier par utilisateur dans une arborescence dédiée. La racine de cette arborescence est restreinte au groupe shadow. Celle-ci contient un répertoire par utilisateur, et restreint à celui-ci et au groupe auth. Chaque répertoire contient un fichier appartenant à l'utilisateur et en lecture pour le groupe auth.

La commande `passwd`, comme toutes celles authentifiant des utilisateurs, n'a plus besoin d'être SUID root, mais seulement SGID shadow. Un utilisateur obtenant les droits de ce groupe ne pourra que contourner la politique de mots de passe qui lui est appliquée.

Pour plus d'informations, consulter les diapositives 19 à 25 de la présentations disponible sur

<http://www.openwall.com/presentations/core02-owl-html+images/> Le slide 21 montre les détails de l'arborescence mentionnée ci-dessus.

1.2 La localisation des empreintes applicatives sous Unix

De nombreuses applications possèdent leurs propres bases d'authentification, chacune ayant son système particulier.

apache est certainement l'application la plus courante qui effectue de l'authentification à partir de sa base propre. Les comptes sont enregistrés dans un fichier généralement appelé *.htpasswd*. En fait ce fichier peut avoir n'importe quel nom, mais la documentation donnant ce nom comme exemple, quasiment tous les administrateurs l'utilisent. Ces fichiers, qui peuvent se trouver n'importe où, contiennent des empreintes calculées à partir des algorithmes DES, MD5 ou SHA, ou des mots de passe en clair si le support a été explicitement activé à la compilation.

Certaines applications peuvent avoir besoin de leur base à eux car le protocole d'authentification est incompatible avec les empreintes systèmes. Un exemple est les serveurs POP3 avec support du protocole d'authentification *apop* (voir la partie sur les algorithmes réseaux) qui nécessite que le mot de passe puisse être récupéré en clair par le serveur. Suivant le démon, soit la base est centralisée dans un répertoire système tel que `/etc` (patch *apop* de Dug Song pour *popa3d*), soit le mot de passe est enregistré en clair dans un fichier de l'arborescence de l'utilisateur, tel que `~/apop/secret` (*cucipop*). Un intérêt est que l'utilisateur possède deux mots de passe différents, l'un pour le système et l'autre pour l'accès au courrier, ce dernier en cas de vol ne pouvant être utilisé pour se connecter interactivement.

D'autres applications peuvent avoir besoin de leur base à eux afin que les utilisateurs applicatifs n'aient pas à être créés au niveau système. De nombreux scripts cgi de discussion sur le web, tels que *wwwboard*, *discus board*, etc. possèdent leurs propres fichiers contenant des empreintes généralement calculées en DES. Des exemples de noms de fichiers sont *admin.txt*, *passwd.txt* ou *users.txt*. Ces fichiers se trouvent souvent dans le même répertoire que le script et comme ils doivent être lisibles par le script ils sont donc récupérables via le serveur web par quiconque pouvant utiliser l'application.

2 Les différents algorithmes de chiffrement sous Unix

L'algorithme utilisé dépend fortement du contexte, suivant que l'authentification se passe au niveau système ou au niveau réseau.

2.1 Les différents algorithmes de chiffrement sous Unix

Trois algorithmes sont utilisés pour enregistrer de façon non-réversible les mots de passe des utilisateurs déclarés au niveau du système : DES, MD5 et BlowFish

2.1.1 DES

Le mot de passe est tronqué à 8 caractères. S'il n'est pas assez long alors il est complété avec des caractères nuls. Puis il est utilisé comme clé de chiffrement DES à 56 bits pour chiffrer une chaîne d'octets nuls. Pour générer la clé seuls les 7 bits de poids faible de chaque caractère sont conservés. La fonction DES est appelée 25 fois afin de ralentir les programmes de cassage de mots de passe.

Un diversifiant est également utilisé afin que deux utilisateurs ayant le même mot de passe n'aient pas les mêmes empreintes. Cette graine de 12 bits est utilisée dans la fonction de chiffrement DES pour modifier des tables internes de cette fonction. Chaque mot de passe peut donc être haché de 4096 (2^{12}) façons possibles afin d'empêcher de ne hacher qu'une seule fois un mot de passe candidat pour attaquer simultanément plusieurs comptes.

En revanche, il est aujourd'hui possible de concevoir des dictionnaires pré-calculés en enregistrant les 4096 hachés différents pour chaque mot, chacun nécessitant un peu plus de 32 Ko de mémoire de masse, soit 3 Go pour 100 000 mots. De plus lorsqu'un système possède plusieurs centaines d'utilisateurs alors des collisions arrivent, c'est-à-dire qu'une graine sert plusieurs fois. À partir de 4097 comptes, il est alors impossible d'empêcher cela, limitant cet algorithme aux systèmes d'une cinquantaine de comptes.

La longueur de l'empreinte est de $64 + 12 = 76$ bits, soit 13 octets dans la base d'authentification, chaque groupe de 6 bits étant remplacé par un caractère parmi les 64 suivants : ./0-9A-Za-z

La longueur de mot de passe prise en compte étant de 8 caractères, le nombre de combinaisons réellement utilisées par des utilisateurs non éduqués est trop faible. Une extension HP/UX permet d'obtenir des mots de passe de 16 caractères, en fait deux mots de passe de 8 caractères chacun. Pour calculer la seconde moitié, ce sont les 12 derniers bits de l'empreinte de la première partie qui sont utilisés comme graine. La sécurité n'est pas forcément améliorée, en effet dans le cas d'un mot de passe de dix caractères, l'attaquant obtient très rapidement les deux derniers et peut s'en servir pour optimiser l'attaque sur les huit premiers, en cherchant par exemple dans un dictionnaire les mots de dix caractères se terminant par ces deux là. L'empreinte dans la base d'authentification est de 24 caractères.

La fonction de chiffrement DES ayant été très étudiée durant les deux dernières décennies, les mises en oeuvre sont de plus en plus efficaces, même sur des processeurs personnels. Une extension BSDi permet d'avoir non plus 25, mais 725 appels par défaut, le nombre d'appels pouvant être spécifié, celui-ci devant être impair. De plus la taille de la graine est portée de 12 à 24 bits. L'empreinte dans la base des comptes est de 20 caractères avec le premier égal à '_', suivi de 4 octets pour le nombre d'itérations et de 4 octets pour la graine. Cet algorithme permet finalement les mots de passe de très grande longueur, techniquement illimitée. L'attaque décrite contre l'extension HP/UX n'est pas applicable ici puisqu'il est impossible à partir de l'empreinte d'analyser une partie du mot de passe pour en optimiser l'attaque de l'autre. Cette extension BSDi est également disponible sur tous les systèmes BSD libres grâce à la bibliothèque FreeSec.

2.1.2 MD5

Le but premier de cet algorithme est d'avoir une fonction de hachage de mots de passe sans utilisation de fonction de chiffrement. L'algorithme MD5 est fondé sur la fonction de hachage MD5. Le calcul s'effectue par 1000 appels de la fonction MD5 entrelacés avec des transformations simples telles que des décalages. La longueur des mots de passe

est illimitée. La taille de la graine est comprise entre 6 et 48 bits, empêchant le calcul de dictionnaires pré-calculés et permettant d'éviter les collisions de graines.

La longueur de l'empreinte est de 27 à 34 caractères dans la base d'authentification, les premiers caractères étant '\$1\$', suivi de 1 à 8 caractères de graine et du caractère '\$'.

Son principal défaut est le nombre fixe d'itérations, qui est déjà trop bas pour les matériels professionnels actuellement disponibles.

Cet algorithme est disponible sur tous les systèmes BSD, les systèmes Linux (libc5 et glibc2) et généralement sur tous les systèmes ayant le support de PAM.

2.1.3 BlowFish

Le but premier de cet algorithme est non seulement d'avoir une fonction de hachage de mots de passe aujourd'hui impossible à optimiser sur des processeurs spécialisés, mais surtout d'avoir une fonction qui va pouvoir s'adapter au cours du temps.

La faiblesse principale des algorithmes DES et MD5 est que le coût maximal de calcul d'une empreinte est fixe, c'est-à-dire que le nombre d'instructions à exécuter est limité. Puisque la puissance des processeurs augmente, elle est multipliée par deux tous les un an et demi d'après la loi de Moore, le nombre d'instructions exécutées par seconde augmente et donc le temps d'un calcul diminue. Cet algorithme est paramétrable, comme l'extension BSDi à l'algorithme DES, afin de spécifier le nombre d'itérations des fonctions sous-jacentes utilisées.

Une autre faiblesse de ces algorithmes est qu'ils sont de plus en plus étudiés et donc que le nombre d'instructions à exécuter par calcul diminue. L'utilisation d'instructions SIMD (Single Instruction Multiple Data, comme MMX ou SSE) qui appliquent à plusieurs données la même instruction en permettant de réaliser plusieurs calculs dans le même temps qu'un seul précédemment. Il est de plus possible de mettre en oeuvre la fonction DES sur des puces spécialisées extrêmement rapides. L'Electronic Frontier Foundation a publié un livre intitulé "Cracking DES" dans lequel ils décrivent la programmation d'une puce de chiffrement DES. En modifiant ce programme il est théoriquement possible de lui permettre de casser des empreintes DES en des temps records. Ce livre est librement accessible sur <http://www.eff.org/descracker/>.

Le calcul s'effectue par 64 appels de la fonction BlowFish entrelacés avec des transformations simples pour chiffrer la chaîne "OrpheanBeholderScryDoubt" de 192 bits. La longueur des mots de passe est limitée à 55 caractères, ce qui n'est pas une sérieuse limitation. La taille de la graine est 128 bits. Le nombre d'appels à la fonction BlowFish est paramétrable de 2^4 à 2^{31} itérations.

La longueur de l'empreinte est de 60 caractères dans la base d'authentification, les premiers caractères étant '\$2a\$', suivi de 2 caractères de paramétrage, du caractère '\$' et de 128 bits (21 caractères plus deux bits) de graine.

Cet algorithme est disponible sur OpenBSD, les versions récentes de FreeBSD et sur les systèmes Linux sous Glibc 2.1.3 avec le patch crypt_blowfish de **Solar Designer** tel que Owl. Ce patch est disponible sur

<http://www.openwall.com/crypt/>.

Le document *A Future-Adaptable Password Scheme* de Niels Provos et David Mazières accessible sur

http://www.usenix.org/events/usenix99/provos/provos_html/ décrit les défauts des différents algorithmes et comment celui-ci a été mis au point.

Un autre document, historique celui-ci puisque datant du 3 avril 1978, décrit les problèmes de sécurité des mots de passe : <http://plan9.bell-labs.com/7thEdMan/vol2/password> à consulter avec une commande du genre `goff -t -e -ms -T ascii password | less`.

2.1.4 Exemples

Voici quelques exemples d'empreintes sur un système FreeBSD 4.7-RELEASE-p1, celui-ci supportant les algorithmes décrits. perl utilisé ci-dessous ne fait qu'utiliser la fonction crypt du système :

- DES


```
$ perl -e 'print crypt("toto", "06")."\n"'
```

```
06TshBHR3MB3.
```
- BSDi DES


```
$ perl -e 'print crypt("toto", "_J9..0666")."\n"'
```

```
_J9..0666oiDpX6qdOao
```
- MD5


```
$ perl -e 'print crypt("toto", "\$1\$06\$")."\n"'
```

```
$1$06$ghvXl7nTv00cpFJB4WKTol
```
- BlowFish


```
$ perl -e 'print crypt("toto", "\$2a\$06\$8Gg1PZ9J5wYaQKd0FCQNMO")."\n"'
```

```
$2a$06$8Gg1PZ9J5wYaQKd0FCQNMOqHb3.yv.8aBrAf.f0vaN4BvED0rF9..
```

Il est à noter que les caractères '\ ' présent devant chaque signe '\$' est nécessaire afin que ce dernier ne soit pas interprété de façon spéciale par perl.

2.2 Les différents algorithmes de chiffrement sur un Réseau Unix

L'algorithme utilisé dépend fortement du protocole applicatif utilisé mais aussi de son niveau.

Les commandes telnet, ftp, rlogin, pop, http, etc, majoritaires dans les réseaux locaux, comme sur l'Internet, envoient simplement le mot de passe en clair, ce qui en fait l'algorithme le plus présent. Dans de nombreux cas, le mot de passe est juste "brouillé" comme avec HTTP, c'est-à-dire qu'il est codé de telle façon que quelqu'un écoutant le réseau ne voie pas le mot de passe en clair, mais n'importe quel programme pourrait le décoder très rapidement. L'inconvénient principal est que le mot de passe peut être capturé sur les réseaux publics par des personnes pouvant ensuite usurper l'identité volée.

La première façon d'empêcher ce vol est d'utiliser le protocole de défi / réponse. Ceci permet au client, via un échange de données avec le serveur, de lui prouver qu'il connaît le mot de passe sans l'envoyer en clair sur le réseau. Pour cela, à la connexion, le serveur envoie un défi au client. Le client utilise ce défi et le mot de passe entré par l'utilisateur pour calculer une réponse qu'il retourne au serveur. De son côté, le serveur a effectué la même opération avec le défi expédié au client et le mot de passe contenu dans la base des utilisateurs. Il compare alors les résultats et considère que le mot de passe saisi par l'utilisateur est correct lorsque les résultats sont identiques.

Deux exemples sont les protocoles *apop* et *HTTP Digest*. Dans *apop* (RFC 1939), la réponse est le MD5 de la concaténation du défi et du mot de passe. Dans *HTTP Digest* (RFC 2617), la réponse est le MD5 de la concaténation de plusieurs chaînes dont le défi, le mot de passe et une partie de l'URL demandée.

Il est à noter que l'utilisation du protocole défi / réponse, s'il permet de protéger le mot de passe sur le réseau qui est l'endroit de moindre confiance, déporte le problème sur le serveur où le mot de passe doit être enregistré en clair, ou en un équivalent. En cas de piratage du serveur le pirate pourra utiliser la base d'authentification dérobée pour usurper l'identité de tous les comptes.

D'autres solutions pour protéger les échanges réseaux sont le chiffrement de la session applicative, dans laquelle le mot de passe est envoyé en clair, et l'authentification forte par un système de clé publique / clé privée.

3 Les logiciels de cassage contre Unix

De nombreux logiciels de cassage de mots de passe sont utilisés contre les empreintes Unix. Voici les plus courants ainsi que les plus originaux.

3.1 Crack

Crack est le père des logiciels modernes de cassage. Sa dernière version, la 5.0a, date de décembre 1996 mais devrait compiler sans problème sur tous les Unix modernes. Ce logiciel Open-Source pour Unix a été écrit par Alec Muffett et est accessible sur le site <http://www.users.dircon.co.uk/~crypto/>.

Crack ne possède que les fonctions de hachage de mots de passe en DES. Il est capable d'utiliser la fonction crypt(3) du système pour casser tous les algorithmes supportés par le système tels que MD5.

Il contient depuis très longtemps toutes les fonctions de génération de mots de passe aujourd'hui classiques. Tout d'abord Crack utilise le contenu du fichier de mots de passe pour générer des mots de passe potentiels, comme le login mais aussi le nom de l'utilisateur et toutes ses données associées. Ensuite il lit les dictionnaires fournis par l'utilisateur. Un langage de description de modifications permet de définir les transformations des mots de passe potentiels avant hachage. Le fichier de configuration en standard contient de nombreuses règles préconfigurées et suffisantes dans de nombreux cas.

Depuis la version 5.0, les dictionnaires utilisés doivent être compressés dans un format assez simple, permettant de réduire la taille de ces fichiers de façon assez significative.

3.2 John the Ripper

John the Ripper est actuellement le logiciel de cassage le plus évolué. La dernière versions stable, la 1.6, date de décembre 1998. Ce logiciel Open-Source pour Unix et Windows est écrit et maintenu par Solar Designer, et accessible sur le site <http://www.openwall.com/john/>.

John met en oeuvre les algorithmes DES, BSDi DES, MD5, BlowFish et LanMan, tous ceux-ci de façon extrêmement optimisée en assembleur sur les processeurs x86, sparc, alpha, ppc, pa-risc et mips 32 et 64 bits.

Il contient toutes les fonctions de génération de mots de passe contenues dans Crack avec en plus d'autres possibilités. La première est l'écriture dans le fichier de configuration de fonctions de filtrage ou de génération de mots de passe. Le langage utilisé est une version simplifiée du C. Ceci permet d'accélérer les calculs en filtrant les mots de passe à hacher, ou en les générant directement tels que souhaités.

La seconde possibilité est la fonction de génération par "force brute intelligente", permettant réellement d'accélérer la performance d'un cassage par force brute. Le système utilisé est basé sur la construction de statistiques à propos de la constitution des mots de passe déjà cassés. Celles-ci sont utilisées par l'option *incremental* pour générer des mots de passe qui ressemblent à ceux déjà obtenus, puis pour s'en éloigner au fur et à mesure.

Le fichier de configuration en standard contient plus de règles préconfigurées que Crack, ainsi que quelques fonctions de génération et de filtrage prédéfinies.

La version en cours de développement, la 1.6.32-dev lors de l'écriture de cet article, est non seulement stable mais aussi bien plus rapide. Pour information, certains algorithmes sur certains processeurs sont programmés avec des instructions SIMD (Single Instruction Multiple Data) afin d'effectuer plusieurs hachages simultanément. Par exemple l'algorithme DES sur un processeur Intel de base passe de 120000 hachages par seconde à 270000 grâce aux instruc-

tions MMX, mais aussi à l'optimisation de la mise en oeuvre DES. Il en est de même pour l'algorithme MD5 sur plusieurs processeurs RISC.

3.3 QCrack

QCrack est le seul logiciel qui effectue actuellement du cassage par dictionnaires pré-calculés contre des empreintes DES. La dernière version, la 1.02, date de janvier 1997. Ce logiciel Open-Source pour Unix a été écrit par The Crypt Keeper. Il ne possède plus de site officiel, mais il est accessible à plusieurs endroits.

Afin de fonctionner, il est nécessaire de générer le dictionnaire pré-calculé. Pour cela l'utilisateur fournit une liste de mots au logiciel qinit, livré avec, qui va hacher chacun d'eux des 4096 façons possibles. Dans la pratique, en n'enregistrant qu'une image (les huit premiers bits) des hachés une économie de place disque est réalisée. Cette place gagnée (4Ko par mot de passe au lieu de 32Ko) est au détriment d'un temps de cassage ensuite plus important puisque statistiquement un mot de passe sur 256 devra être haché pour vérifier s'il s'agit du bon mot de passe, mais cela permet de façon indéniable d'accélérer les temps de cassage lors de mots de passe généralement choisis par des utilisateurs non formés.

3.4 Autres logiciels de cassage contre Unix

De nombreux logiciels existent pour casser des empreintes Unix. Généralement ceux-ci utilisent la fonction crypt du système et ne possèdent aucune fonctionnalité de génération de mots de passe.

La liste suivante est loin d'être exhaustive.

De très nombreux casseurs DES existent, aussi bien écrits en C qu'en perl et souvent n'utilisant que la fonction crypt() du système.

Les bases LDAP, telles que Netscape Directory, stockent généralement leurs mots de passe en SHA1, mais peuvent aussi le faire en DES. L'empreinte SHA1 est reconnaissable à ce qu'elle commence par {SHA} (et non pas par SHA1 ;-) et est suivie de 160 bits codés en base 64. Le calcul est simple puisqu'il suffit d'appeler une fonction SHA1 pour obtenir l'empreinte. Ainsi le mot de passe Bonjour1 a pour empreinte {SHA}8B6PG6HVfprncZul2F3mqsG1p2w= :

```
$ echo -n Bonjour1 | openssl sha1 -binary | openssl base64
8B6PG6HVfprncZul2F3mqsG1p2w=
```

Le principal avantage de cet algorithme est sa grande vitesse d'exécution, ceci permettant qu'un serveur authentifie plusieurs milliers d'utilisateurs par seconde tout en pouvant effectuer d'autres tâches simultanément. En revanche, le manque de graine est une grande vulnérabilité puisque un casseur peut alors en ne chiffrant qu'une seule fois chaque mot de passe potentiel tester tous les utilisateurs de la base.

Une version avec graine, nommée Salted SHA, permet d'éviter cela sans rendre significativement plus longs les calculs. Il suffit simplement de concaténer la graine au mot de passe et d'en calculer l'empreinte SHA1. La graine est alors concaténée au résultat, le tout étant codé en base 64.

L'authentification web la plus répandue est nommée "Basic HTTP". Elle ne consiste qu'au codage en base 64 de la concaténation du login et du mot de passe avec le caractère ':' entre les deux. Le décodage se fait très simplement via des commandes comme openssl et mimencode :

```
$ echo 'ZHVjYW1wO1BldG10Q3VyaWVleDStKQ==' | mimencode -u
```

4 La protection des mots de passe sous Unix

La méthode de protection des mots de passe utilisée dépend fortement du contexte, suivant que les empreintes sont sauvegardées au niveau système, réseau ou applicatif.

4.1 La protection des mots de passe sous Unix

La protection la plus simple est de cacher les mots de passe aux utilisateurs, cette technique étant appelée les "*shadow passwords*". Les mots de passe sont déplacés du fichier **/etc/passwd** vers le fichier **/etc/shadow** (ou équivalent), ce dernier ne devant être lisible que par les administrateurs. La commande *pwconv* permet d'effectuer ce déplacement, *pwunconv* pouvant l'inverser.

Les applications devant authentifier des utilisateurs doivent avoir été compilées pour supporter cette fonction, mais aujourd'hui toutes les applications modernes supportent cette fonctionnalité qui est souvent activée par défaut.

Sur les systèmes avec PAM, il est nécessaire d'ajouter dans le fichier **/etc/pam.d/passwd** le terme *shadow* à la fin de la ligne suivante :

```
password required /lib/security/pam_pwdb.so
```

Une autre méthode de protection est de changer l'algorithme de hachage utilisé pour en choisir un plus résistant que le DES au cassage comme MD5 et BlowFish. Sur les systèmes Linux sans PAM ajouter **MD5_CRYPT_ENAB yes** dans le fichier **/etc/login.defs**

Sur les systèmes avec PAM, il est nécessaire d'ajouter dans le fichier **/etc/pam.d/passwd** le terme *md5* à la fin de la ligne suivante :

```
password required /lib/security/pam_pwdb.so
```

Sous OpenBSD le fichier **/etc/passwd.conf** doit être modifié pour choisir l'algorithme et le configurer :

- *localcipher=md5* pour faire du MD5
- *localcipher=blowfish,x* pour faire du BlowFish avec 2^x itérations, x devant être compris entre 4 et 31.

Sous FreeBSD le fichier **/etc/login.conf** doit être modifié pour choisir l'algorithme :

- *passwd_format=md5* pour faire du MD5
- *passwd_format=blf* pour faire du BlowFish

4.2 Pourquoi protéger les mots de passe sur un réseau Unix

La première raison est que les protocoles les plus utilisés sont vulnérables à l'écoute passive, l'attaquant pouvant collectionner très facilement les mots de passe circulant en clair sur son réseau local. Malgré une légende urbaine persistante, les commutateurs n'ont jamais empêché quiconque de renifler le réseau, il faut juste effectuer de l'écoute active.

Plutôt que d'utiliser un programme comme tcpdump ou snort, des programmes permettent de constituer directement une base d'authentifications. Les programmes les plus communs sont :

- *linsniff666.c* capture le début des connexions ftp, telnet, pop2, pop3, imap2 et login.
- *dsniff* <<http://naughty.monkey.org/~dugsong/dsniff/>> par Dug Song <dugsong@monkey.org> capture les mots de passe en clair (ou obscurcis) dans plus de 30 protocoles :
 FTP, Telnet, SMTP, HTTP, POP, poppass, NNTP, IMAP, SNMP, LDAP,
 Rlogin, RIP, OSPF, NFS, YP/NIS, SOCKS, X11, CVS, IRC, AIM, ICQ, Napster,
 PostgreSQL, Meeting Maker, Citrix ICA, Symantec, NAI Sniffer, Microsoft

SMB, Oracle SQL*Net, Sybase and Microsoft SQL auth info.

4.3 La protection de telnet

Pour protéger ce protocole deux méthodes sont possibles : soit en ne protégeant que la phase d'authentification, soit en chiffrant toute la session.

La façon la plus simple de protéger l'authentification est d'utiliser des mots de passe jetables, aussi dits non rejouables. Le but est que même si le mot de passe que quelqu'un vient d'utiliser est capturé, celui-ci ne peut plus être valable à nouveau. L'utilisateur doit donc avoir une liste de mots de passe générée par l'administrateur du système, ou un petit programme approprié (il en existe pour palm-pilot par exemple). Même si une longue suite de mots de passe est capturée, il n'est pas possible de deviner le prochain à utiliser.

Les deux systèmes principaux sont *S/KEY One-Time Password System* (RFC1760) et *One Time Passwords in Everything* (OPIE - RFC1938) - <<http://inner.net/opie>>.

Ces systèmes ne demandent qu'un support au niveau du serveur, tout client telnet étant compatible. Il est également possible de les utiliser avec ftp, ssh et d'autres protocoles.

Un autre système est une authentification par mots de passe sécurisée en utilisant une technique de preuve sans apport de connaissance et un protocole d'échange de clés asymétriques. Le principal système existant est *Stanford Remote Authentication Project* (SRP) - <<http://srp.stanford.edu/>>.

Le problème majeur est qu'il est nécessaire de modifier à la fois les serveurs et les clients. Ceci empêche de se connecter depuis tout système non compatible.

Ces méthodes ont le défaut de ne protéger que la phase d'authentification. Or lorsqu'un administrateur se connecte à un serveur alors le mot de passe de root passe en clair dans la session et il est possible pour un pirate d'injecter des paquets dans la connexion pour faire exécuter des commandes avec l'identité de la personne attaquée. Pour contrer cela, la session complète peut être chiffrée.

La méthode la plus simple est d'encapsuler la session telnet dans ssl, soit en utilisant un wrapper ssl (voir stunnel ci-dessous) soit en utilisant des clients et serveurs patchés.

La méthode la plus utilisée aujourd'hui est d'utiliser *ssh*, un remplaçant sécurisé de rlogin et rsh avec chiffrement et authentification forts. *ssh* est à la fois un programme et un protocole créés par **Tatu Ylönen**. Deux versions de protocoles incompatibles existent : 1.5 et 2, correspondant respectivement aux versions 1.2.x et 2.x du programme accessible sur <<ftp://ftp.ssh.com/pub/ssh/>>. Une version commerciale existe et est diffusée par la société ssh.com de Tatu Ylönen.

Des problèmes de licences associées à ces différentes versions ont amené l'équipe du **projet OpenBSD** à développer *OpenSSH* à partir de la version 1.2.12 (dernière version libre -dans le sens BSD- de Tatu Ylönen). Le support des protocoles 1.3, 1.5 et 2.0 ont été rajoutés et la version OpenBSD est portée sous les autres Unix dont Linux, Solaris, AIX, IRIX, HP/UX, FreeBSD et NetBSD, mais aussi sous Windows avec l'environnement cygwin. Ce programme utilise la bibliothèque libcrypto de *OpenSSL* comme moteur de chiffrement.

D'autres mises en oeuvre existent, le site de *OpenSSH* listant les principales, comme *putty* pour Windows. Il faut savoir qu'en France seule la version nommée *ssf* de **Bernard Perrot** est officiellement autorisée. Celle-ci correspond à la version 1.2.27 de Tatu Ylönen (avec de nombreux bogues corrigés) pour laquelle la taille de l'espace de clé est limitée à 2^{128} . Son usage est libre (les utilisateurs n'ont aucune démarche à effectuer), *ssf* ayant fait l'objet d'une déclaration auprès du SCSSI.

ssh permet de plus d'encapsuler des connexions TCP et gère nativement l'encapsulation des dépôts d'affichages X11.

C'est pour tout cela que ssh est de plus en plus utilisé, non seulement par des administrateurs qui veulent pouvoir effectuer leurs tâches à distances, mais aussi par les utilisateurs de base qui sont sensibles à la confidentialité de leurs données. Pour les utilisateurs avancés, un VPN bon marché peut être mis en place en encapsulant PPP, voir la FAQ de SSH <<http://www.employees.org/~satch/ssh/faq/ssh-faq-5.html#ss5.4>>.

Bien sûr tout ceci doit être correctement utilisé, les attaques actives contre les utilisateurs distraits permettent d'intercepter les connexions chiffrées, ssl et ssh, et obtenir les données en clair, dont le mot de passe utilisé lors de l'authentification.

4.4 La protection d'autres protocoles : POP3 / HTTP

Pour ces deux protocoles, une version défi / réponse existe. APOP est le protocole d'authentification pour POP3, Digest celle de HTTP. Non seulement ceux-ci ne sont pas mis en oeuvre dans tous les clients ni dans tous les serveurs, mais en plus ils nécessitent que le mot de passe soit enregistré en clair (ou en un équivalent) sur le serveur.

Dans certains cas APOP peut être intéressant, cela permettant d'avoir deux authentifications différentes pour récupérer son courrier et pour se connecter interactivement.

Une meilleure méthode est d'encapsuler la session dans ssl : quasiment tous les clients et serveurs HTTP savent utiliser ce mode de chiffrement, ainsi que tous les clients majeurs POP3.

4.5 La protection des mots de passe côté serveur

L'encapsulation ssl est une bonne façon de protéger son authentification, en plus de garantir une certaine confidentialité sur le réseau. Afin de rajouter le support de ssl aux serveurs qui n'ont pas été conçus pour, il est possible d'utiliser un wrapper comme *stunnel* : <<http://stunnel.mirt.net/>> (ou <<http://www.stunnel.org/>>). *stunnel* écoute sur un port, déchiffre la communication ssl reçue et renvoie les données vers le port du serveur. Le client se connecte alors sur le port de *stunnel* et communique en ssl, le trafic sur le réseau étant alors protégé par la couche ssl.

Les protocoles classiques ainsi "sécurisables" sans modification du démon sont POP-2, POP-3, IMAP, NNTP et HTTP. Même si SMTP-TLS n'est pas une simple encapsulation de SMTP dans SSL, comme c'est le cas pour les autres protocoles, *stunnel* est capable de gérer le début de la connexion en clair jusqu'à l'établissement de la communication ssl.

En fait, *stunnel* est également utilisable côté client, *stunnel* écoutant sur un port un trafic en clair et le renvoyant chiffré vers un serveur ssl. Cette fonctionnalité est notamment utilisé dans le cas de clients de messagerie ne supportant pas ssl de façon native.

ssl offrant des options de gestion de certificats, *stunnel* est capable de vérifier le certificat présenté par son interlocuteur. Ceci est utilisé notamment pour restreindre au maximum les accès d'utilisateurs distants à un serveur de messagerie : un pirate n'ayant pas de certificat valide, il ne pourra accéder au serveur protégé.

Finalement il est possible, comme avec *ssh*, de mettre en place des VPN en encapsulant PPP, voir les exemples sur le site non officiel de *stunnel* <<http://www.stunnel.org/examples/pppvpn.html>>.

4.6 La protection des mots de passe applicatifs sous Unix

L'application la plus utilisée est *apache*. Les bases d'utilisateurs sont souvent appelées **.htpasswd** comme dans les exemples de la documentation, mais peuvent porter n'importe quel nom. Comme chaque utilisateur peut décider de restreindre une arborescence en demandant à s'authentifier, ceux-ci placent souvent leur base dans le répertoire à

protéger, ceci faisant qu'elle est accessible via HTTP. Pire que cela, l'arborescence web est souvent accessible via FTP anonyme ou d'autres protocoles plus utilisés localement comme NFS ou SMB.

Dans la configuration par défaut d'*apache*, l'accès est maintenant interdite aux fichiers commençant par **.ht**.

Pour les systèmes de discussion web qui enregistrent la base des utilisateurs dans le répertoire des cgi, comme *www-board* et *discus board*, il est important de ne pas les utiliser à moins d'être capable au niveau du serveur web de bloquer l'accès à ces fichiers sensibles.

Sur les serveurs Netscape, la base LDAP est stockée dans un fichier nommé **id2entry.dbb**. Ce fichier qui n'est pas dans l'arborescence exportée par HTTP, ne doit en aucun cas être accessible via un autre moyen de partage de fichiers.

4.7 Exemples d'accès à des fichiers de mots de passe

Il est relativement facile de trouver de telles bases d'utilisateurs. Si vous en cherchez une précisément vous ne l'obtiendrez certainement pas, mais si vous en cherchez vous en trouverez à coup sûr.

Ci dessous quelques fichiers **.htpasswd** trouvés via *ftpsearch* (lorsque ce système fonctionnait encore bien). Parmi ceux-là, certains étaient mis à jours plusieurs fois par semaine avec des comptes ajoutés ou supprimés et des mots de passe mis à jour.

```
25.8K 2000 Jan 20 ftp.xxxxxxxxxxxx.edu /pub/.../hnpeople/.htpasswd
68.3K 2000 Jan 20 ftp.xxxx.hu /customers/.../cv/.htpasswd
 6.4K 2000 Jan 17 ftp.xxxx.hu /customers/.../employers/.htpasswd
 44 2000 Jan 11 ftp.xxxxxxxxxxxx.dk /projects/.../.htpasswd
 59 1999 Dec 16 ftp.xxxxxxxxxxxx.com /Products/protected/.htpasswd
 39 1999 Dec 14 ftp.xxxxxx.net /showcase/.../.web/.htpasswd
 23 1999 Dec 14 ftp.xxxxxxxxxxxx.edu /coe/.../prot-dir/.htpasswd
 41 1999 Nov 9 ftp.xxxxxxxxxxxx.edu /depts/.../PRIVATEadm/.htpasswd
 0 1999 Nov 3 ftp.xxxxxxxxxxxx.edu /depts/sun2-xttys/.htpasswd
 40 1999 Oct 26 ftp.xxxxxxxxxxxx.edu /student/.../securedadmin/.htpasswd
 19 1999 Oct 20 ftp.xxxxxx.ch /irc/.../243/.htpasswd
 25 1999 Oct 18 ftp.xxxxxxxxxxxx.edu /student/.../password/.htpasswd
1.0K 1999 Oct 15 ftp.xx.pt /disk2/.../_pwprotect/.htpasswd
1.0K 1999 Oct 15 ftp.xxxxxx.no /.11/.../_pwprotect/.htpasswd
1.0K 1999 Oct 15 ftp.xxxxxxxx.pt /.3/.../_pwprotect/.htpasswd
1.0K 1999 Oct 15 ftp.xxxxxxxx.pt /.d9/.../_pwprotect/.htpasswd
 40 1999 Oct 14 ftp.xxxxxxxxxxxx.edu /student/.../directoradmin/.htpasswd
 41 1999 Oct 6 ftp.xxxxxxxxxxxx.edu /student/.../secureadmin/.htpasswd
 21 1999 Oct 4 ftp.xxxxxxxxxxxx.edu /depts/.../protimadmin/.htpasswd
 25 1999 Sep 29 ftp.xxxxxxxxxxxx.edu /coe/.../ftp/.htpasswd
```

De même via *altavista*, des fichiers **admin.txt**, **passwd.txt** et **users.txt** étaient référencés.

1. Index of /~j9706248/discus_admin
URL: www.xxxxxxxxxxxx.uk/~j9706248/discus_admin/
2. Index of /Tools/discus_admin
URL: www.xxxx.org/Tools/discus_admin/
3. Index of /~ycstweb/discus_admin_9787
URL: www.xxxx.com/~ycstweb/discus_admin_9787/

4. Index of /chode/discuss/admin/
URL: www.xxxxxxx.com/chode/discuss/admin/
5. Index of /Discus/discus2_30
URL: xxxxxxxx.com/Discus/discus2_30/
6. Index of /spring/discus_admin
URL: www.xxxxxxxx.net/spring/discus_admin/
7. Index of /~linyi/discus_admin_145129241/
URL: xxxxxxxxxxxx.sg/~linyi/discus_admin_145129241/
8. Index of /discus_admin_229269155
URL: www.xxxxxxxxxxxx.com/discus_admin_229269155/
9. Index of /kit/discus_admin
URL: users.xxxxxx.net/kit/discus_admin/
10. Index of /discus_admin_04251121
URL: www.xxxxxxxxxxxxxx.net/discus_admin_04251121/

Si ces comptes ne donnent pas forcément accès à des informations intéressantes, comme les utilisateurs ont la fâcheuse manie d'utiliser le même mot de passe pour des choses de sensibilités différentes, ces clés peuvent aboutir à des Saint-Graal avec beaucoup de chance et pas mal de patience...

5 Le durcissement des mots de passe sous Unix

Comme vu dans la première partie, le durcissement des mots de passe des utilisateurs est la fonctionnalité qui n'accepte un nouveau mot de passe qu'après s'être assuré qu'il suit un certain nombre de règles. Ce durcissement est donc toujours lié au système. Cela assure que les mots de passe choisis ne sont pas trop simples et qu'ils résistent plus longtemps contre les programmes de cassage de mots de passe.

5.1 Pourquoi durcir les mots de passe ?

Avant de voir comment durcir les mots de passe sous Unix, voici quelques statistiques sur des mots de passe cassés, initialement chiffrés en DES, ainsi que sur les temps de cassage maximaux pour les différents types d'algorithmes afin de vous convaincre de l'utilité du durcissement des mots de passe.

Sur un sous-ensemble des comptes que j'ai pu casser, voici comment sont constitués ces mots de passe cassés :

```
2215 (100.00 %) Mots de passe crackés

1015 ( 45.82 %) Un mot en minuscules
 441 ( 19.91 %) Un mot en minuscules suivi du chiffre '1'
 209 (  9.44 %) Un mot en minuscules suivi d'un chiffre autre que '1'
   2 (  0.09 %) Un mot en minuscules suivi d'une majuscule
  40 (  1.81 %) Un mot en minuscules suivi d'un autre caractère

 38 (  1.72 %) Le chiffre '1' suivi d'un mot en minuscules
 12 (  0.54 %) Un chiffre autre que '1' suivi d'un mot en minuscules
  7 (  0.32 %) Un autre caractère suivi d'un mot en minuscules
```

191 (8.62 %) Plusieurs chiffres suivis d'un mot en minuscules
 5 (0.23 %) Un mot en minuscules suivi de plusieurs chiffres
 12 (0.54 %) minuscule(s) chiffre(s) minuscule(s)
 2 (0.09 %) chiffre(s) minuscule(s) chiffre(s)
 57 (2.57 %) nombre
 9 (0.41 %) minuscules et chiffres

 88 (3.97 %) Une majuscule suivie de minuscule(s)
 22 (0.99 %) Une majuscule suivie de minuscule(s) et du chiffre '1'
 11 (0.50 %) Une majuscule suivie de minuscule(s) et d'un chiffre
 1 (0.05 %) Une majuscule suivie de minuscule(s) et de chiffres
 1 (0.05 %) Une majuscule suivie de minuscule(s) et d'une majuscule
 8 (0.36 %) Une majuscule suivie de minuscule(s) et d'un caractère

 24 (1.08 %) Un mot en majuscules
 1 (0.05 %) Un mot en majuscules suivi du chiffre '1'
 2 (0.09 %) Un mot en majuscules suivi d'un autre chiffre
 1 (0.05 %) Un mot en majuscules suivi de plusieurs chiffres
 0 (0.00 %) Un mot en majuscules suivi d'un autre caractère
 1 (0.05 %) Une suite de majuscule(s) et de minuscule(s)
 0 (0.00 %) Une suite de majuscule(s) et de chiffre(s)
 1 (0.05 %) Une suite de majuscule(s), minuscule(s) et chiffre(s)

 14 (0.63 %) Toute autre combinaison de caractères

Ce qui montre que seulement trois règles suffisent à décrire plus des trois quarts des mots de passe cassés.

En fait il s'agit là de la version optimiste de ces statistiques où les comptes laissés à l'entière convenance des utilisateurs suivent plus facilement les statistiques suivantes (limitées aux règles représentant plus de 1%) :

(69.68 %) Un mot en minuscules
 (13.84 %) nombre
 (4.27 %) Une majuscule suivi de minuscule(s)
 (3.52 %) Plusieurs chiffres suivis d'un mot en minuscules
 (3.14 %) Un mot en majuscules
 (1.95 %) Un mot en minuscules suivi du chiffre '1'
 (1.57 %) Un mot en minuscules suivi d'un chiffre autre que '1'

La longueur des mots de passe cassés est aussi intéressante :

0 0.32 %
 1 0.09 %
 2 0.81 %
 3 4.56 %
 4 7.04 %
 5 10.47 %
 6 34.22 %
 7 19.68 %
 8 22.89 %

Ici 60% des mots de passe cassés ne font que 6 caractères.

Sur les comptes sans restriction mentionnés ci-dessus, en fait la longueur minimale est fixée à quatre caractères, les longueurs observées deviennent :

0	0.00 %
1	0.03 %
2	0.03 %
3	0.14 %
4	25.97 %
5	22.50 %
6	27.33 %
7	12.46 %
8	10.59 %

Soit les trois quarts des mots de passe cassés ne font que 6 caractères.

En étudiant les mots de passe cassés, 17% sont directement déductibles du login et un pourcentage de 29% est atteint avec en plus les informations associées à l'utilisateur.

Les dernières statistiques sont relatives aux temps de cassage qui sont de plus en plus courts (temps obtenus avec john-1.6.32 sur un AMD Athlon(tm) XP 1600+ à 1400 MHz) :

- DES (517184 c/s) :
 - chiffrement DES de 6 caractères alphanumériques : 1h10
 - chiffrement DES de 7 caractères en minuscules : 4h20
 - chiffrement DES de 7 caractères alphanumériques : 1,75 jours
 - chiffrement DES de 8 caractères en minuscules : 4,67 jours
 - chiffrement DES de 8 caractères alphanumériques : 2 mois
- LANMAN (3070336 c/s) :
 - chiffrement LANMAN de 7 caractères alphabétiques : 0h45
 - chiffrement LANMAN de 7 caractères alphanumériques : 7h00
 - chiffrement LANMAN de 7 caractères (69 caractères) : 28 jours

Le nombre de tests possibles pour les autres chiffrements sont :

- BSDI DES : 17152 c/s
- FreeBSD MD5 : 4048 c/s
- OpenBSD BlowFish : 239 c/s

En imaginant de construire une machine fondée sur celle de l'EFF, effectuant 4,5E9 chiffrements par seconde, la totalité des combinaisons des 95 caractères possibles sur une longueur de 8 caractères ne prendrait que 17,5 jours...

En conclusion il est facile de dire que les mots de passe choisis par les utilisateurs non avertis sont trop simples, ceci étant particulièrement grave quand il suffit d'un seul mot de passe pour accéder à un système.

5.2 Le durcissement des mots de passe sous Unix

La méthode classique sous Unix de durcir les mots de passe d'un système est de remplacer la commande *passwd* afin que chaque nouveau mot de passe suive des règles strictes. Un tel programme de remplacement est *npasswd* de **Clyde Hoover** : <http://www.utexas.edu/cc/unix/software/npasswd/>.

npasswd effectue sur les nouveaux mots de passe de nombreux tests à partir de règles de transformations et de dictionnaires avant de les accepter.

Une autre méthode, utilisable seulement sur les systèmes avec PAM, est d'utiliser un module de vérification de la force des nouveaux mots de passe. Deux existent : *CrackLib* et *pam_passwdqc*.

CrackLib est fondé sur la version 2.7 de la bibliothèque de même nom par **Alec Muffett** (auteur de Crack). Ce module n'accepte un nouveau mot de passe seulement si celui-ci n'est pas cassable par Crack.

pam_passwdqc de **Solar Designer** est un projet **Openwall** : <<http://www.openwall.com/passwdqc/>>. Ce module n'est pas fondé sur un programme de cassage de mots de passe comme pourrait le faire croire le fait que **Solar Designer** soit aussi l'auteur de *John the Ripper*. Ce module définit en fait les tailles et complexités acceptables pour différents types de mots de passe et de passe-phrases.

Une des caractéristiques intéressantes de *pam_passwdqc* est qu'il est possible de définir combien de mots doivent composer une passe-phrase et que certaines transformations ne comptent pas dans le calcul de la complexité comme la mise en majuscule de l'initiale d'un mot ou le simple ajout d'un chiffre après un mot.

pam_passwdqc est supporté sur les systèmes Linux, FreeBSD, Solaris et HP/UX sur lesquels PAM est utilisable pour changer de mot de passe.

Voici une petite anecdote pour finir de vous convaincre de mettre en place du durcissement. Un administrateur chargé de tester la force des mots de passe de plusieurs systèmes, profite d'être au courant pour changer le sien par un qu'il pensait plus costaud en prenant les premières syllabes de deux mots n'ayant qu'un lointain rapport et en y mettant le chiffre 1 entre elles. Après plusieurs jours de calculs, à l'époque Crack n'était encore qu'en version 4, son mot de passe n'est toujours pas trouvé. En arrivant au bureau un matin, il déverrouille son économiseur d'écran et tombe nez à nez avec son mot de passe... Crack avait réussi cet exploit simplement en prenant un mot d'origine est-européenne et en remplaçant la lettre 'i' par le chiffre '1'.

La morale de cette histoire est simplement que chaque méthode est nécessaire mais insuffisante, celles-ci étant complémentaires.

5.3 Quelques règles de constitution

Tout mot de passe doit suivre ces quelques règles simples de constitution ainsi décrites, chacune étant nécessaire mais non suffisante :

1. La longueur du mot de passe doit être suffisante : non chiffré en DES il doit contenir au moins 6 à 7 caractères, ceux chiffrés en DES doivent en contenir 8.
2. La constitution doit être suffisamment complexe : il doit utiliser au moins un caractère de chacune des catégories suivantes :
 - les lettres minuscules
 - les lettres majuscules
 - les chiffres
 - les autres caractères (ponctuation...)

Les caractères de contrôle n'étant pas portables ils ne peuvent être utilisés que dans des cas restreints.

3. Le mot de passe ne doit jamais faire référence à une information liée à l'utilisateur, à l'installation du système, à l'entreprise, à l'organisation, etc.
4. Il ne doit pas être un simple mot référencé dans un dictionnaire (français, anglais, technique ou autre) ni une de leurs transformations plus ou moins complexes décrites dans les programmes de cassage de mots de passe.

Quelques méthodes simples d'obtention de bons mots de passe faciles à retenir peuvent être :

- combiner deux mots existants en introduisant des chiffres et/ou des caractères de ponctuation,
- utiliser des mots écrits en phonétique,
- utiliser les premières lettres de vers, phrases, expressions, adresses, etc.

La combinaison de mots sans rapports avec l'introduction d'autres caractères permet d'obtenir des passe-phrases d'un bon niveau de sécurité et relativement faciles à retenir avec un peu d'entraînement. Le problème majeur étant d'arriver à taper la vingtaine de caractères qui les composent sans se tromper.

Une dernière chose importante est de ne pas utiliser le même mot de passe pour protéger des accès de sensibilités différentes. Ainsi le mot de passe qui vous sert à accéder à hotmail ne doit pas être le même que celui qui protège vos messages au travail, ni celui qui vous permet de consulter votre compte en banque. Une fois que vous vous êtes constitué une demie douzaine de bons mots de passe et passe-phrases, chacun peut être associé à un niveau de confidentialité différent.

Denis Ducamp

Denis.Ducamp@groar.org - <<http://www.groar.org>>

Denis.Ducamp@hsc.fr - <<http://www.hsc.fr>>

Denis Ducamp est consultant en sécurité informatique chez Hervé Schauer Consultants et spécialisé dans la sécurité systèmes et réseaux. Cet article ainsi que sa première partie ont été écrits à partir d'une présentation nommée "*crackage et durcissement des mots de passe*" développée chez HSC et publiquement accessible sur <<http://www.hsc.fr/ressources/presentations/mdp2/>>.

\$Id : part2.sgml,v 1.14 2002/12/03 07 :05 :24 ducamp Exp \$