Sécurité des systèmes Unix

Mathieu Blanc

CEA

Sommaire

- **Introduction à la sécurité système**
- **Installation d'un système minimal**
- **Sécurité du système**
- **Sécurité des accès réseau**
- **Sécurité des applications**
- **Surveillance du système**

Introduction

- **La sécurité est un processus, pas un état permanent**

- **Manipulations régulières :**
  - Installer un système minimal taillé suivant les besoins
    - limite le nombre de failles et de logiciels à gérer
  - Conserver ses systèmes à jour
    - de nouvelles failles découvertes à chaque instant
  - Surveiller les fichiers de logs
    - rechercher les événements suspects ou incompréhensibles
  - Contrôler son propre système
    - contrôle d'intégrité des fichiers sensibles (prog, config, ...)
  - Attaquer son propre système
    - vérifier ses propres vulnérabilités, locales et distantes

Installation d'un système minimal

Installation d'un système minimal

- **Objectifs**
  - Planning de l'installation
  - Pourquoi minimiser le système ?
- **Noyau**
  - Etre ou ne pas être modulaire
  - Le choix des pilotes
- **Partitions**
  - Le bon partitionnement conditionne la vie du système
- **Applications**
  - Bien choisir les packages

Planning de l'installation

- **Roadmap**
  - Objectif
    - construire un système sécurisé selon ses besoins
  - Avant l'installation
    - choisir une distribution, vérifier l'intégrité
  - Pendant l'installation
    - partitions, small is beautiful
  - Après l'installation
    - configuration, du système au réseau

Planning de l'installation

- **Choix d'une distribution Linux**
  - orientés RPM : RedHat, Mandrake, Suse, Fedora
    - pour : supportés par des entreprises
    - contre: rpm, patches "maisons" non standards
  - debian
    - pour : très stable, documentation complète, apt
    - contre : un peu vieillot, interfaces d'admin
  - Slackware
    - pour : fichiers de conf. non modifiés, up-to-date
    - contre : système de package trop simple, interfaces d'admin
  - orientés sources : Linux From Scratch, Gentoo
    - pour : construits selon ses besoins, optimisés, minimalistes
    - contre : longs à construire et configurer
  - Autres
    - construits pour un usage spécifique (Immunix, owl, ...)

Planning de l'installation

- **Choix d'un système *BSD**
  - FreeBSD : orienté performances
    - pour : très performant, sécurisation forte possible
    - contre : tout à faire « à la main » : mise à jour, sécurisation, le moins d'architectures supportées
  - OpenBSD : orienté sécurité
    - pour : audit et amélioration du code source des applications, tout est chrooté, pile non exécutable
    - contre : moins performant, peu orienté station de travail
  - NetBSD : le générique
    - pour : le nombre de plateformes supportées, le système de package
    - contre : pas aussi sûr que OpenBSD, pas aussi performant que FreeBSD

Planning de l'installation

- **Préparation de l'installation & mise à jour**
  - Déroulement idéal
    - Récupérer toutes les mises à jour disponibles
    - Installer la machine hors réseau
      - présence de failles tant que l'OS n'est pas à jour
  - Vérifier l'intégrité
    - à vérifier après chaque téléchargement, sur chaque CD
    - ne fournit *aucune garantie* quant à l'origine
      - recalcul des MD5 par le pirate
  - Vérifier les signatures numériques
    - garantie l'origine, l'intégrité, etc...
      - un patch ne garantit pas la résolution des problèmes ;-)

Pourquoi minimiser le système ?

- **Pourquoi ne pas faire une installation classique, et n'utiliser que les outils dont on a besoin ?**
  - occupation inutile de l'espace disque
  - plus d'applications : plus de mises à jours à effectuer (patches de sécurité notamment)
  - plus d'applications : plus de failles potentielles
  - plus d'applications : probabilité plus forte pour un intrus de trouver des outils lui facilitant la tâche
  - un outil suspect sera plus facilement repéré s'il n'est pas noyé dans la masse

Pourquoi minimiser le système ?

- **Limiter les *outils* installés et les *services* actifs au strict *nécessaire***
- **Procéder de même, lorsque cela est possible, pour les fonctionnalités du *noyau***
- **De façon générale, supprimer le superflu permet :**
  - de limiter les failles potentielles
  - de repérer plus facilement un élément suspect
- **Cela s'applique aux fonctionnalités du noyau :**
  - la gestion de certains périphériques ou systèmes de fichiers peut s'avérer plus dangereuse qu'utile (USB par exemple)
  - un noyau monolithique sans support des modules permettra d'éviter le chargement de modules piégés
  - le noyau est la base du système, il est primordial que cette base soit saine

Pourquoi minimiser le système ?

- **Dans le cas des services :**
  - ouvrir des services inutiles peut permettre à un pirate :
    - d'obtenir des informations (OS fingerprint, uptime...)
    - de pénétrer plus facilement le système
    - de le faire tomber avec un DOS (exemple de chargen et echo en UDP)
  - un port ouvert mais non autorisé sera également mieux détecté (ICMP admin prohibited par exemple)
  - laisser ces services actifs mais en interdire l'utilisation via du filtrage de paquets est une alternative, moins propre que l'absence du service
- **Distinction entre service public et service d'administration**
  - il est impératif que le monde extérieur n'ait pas accès aux services d'administration
    - n'écouter que sur l'interface adéquate
    - filtrage de paquets
    - restrictions applicatives

Noyau : modularité ?

- **Quelques éléments ne sont pas forcément les bienvenus pour un noyau que l'on souhaite sûr.**
  - La gestion des modules peut s'avérer dangereuse, pour un serveur on favorisera un noyau monolithique
    - Noyau plus volumineux
  - Connaître son système permet de savoir ce dont il n'a pas besoin :
    - inutile de supporter tous les systèmes de fichiers, toutes les cartes graphiques, tous les protocoles réseau
    - de même, on peut sciemment inhiber certains périphériques ou certaines fonctions en ne les gérant pas dans le noyau :
      - USB : DiskOnKey, claviers, lecteurs/graveurs externes
      - ports série, parallèle...
      - communications réseau : ppp, 802.11, infra-rouge...
  - Le bon sens est notre ami
    - un serveur aura rarement besoin de pilote pour sa carte son...
    - idem pour les fonctionnalités expérimentales !

Noyau : les pilotes

- **Le minimum vital pour un noyau...**
  - Sont indispensables pour le bon fonctionnement du système (simple bon sens) :
    - le support des périphériques à utiliser (disques SCSI, cartes réseau...)
    - le support des systèmes de fichiers utilisés (ext3 ? ramfs ?)
    - les protocoles réseau usuels
  - Certains systèmes d'exploitation offrent des fonctionnalités axées sécurité qui peuvent s'avérer utiles, voire nécessaires :
    - un pare-feu (netfilter, pf...)
    - des algorithmes de cryptographie (utilsation pour IPsec, chiffrement de partitions)
    - une implémentation d'IPsec
    - une gestion des politiques de sécurité
    - des protections système spécifiques (pile non exécutable d'OpenBSD et grsecurity, contraintes sur certains fichiers)

Applications nécessaires

- **Le kit de survie doit contenir :**
  - Les commandes UNIX de base (compilées en statique si possible)
    - gestion des processus : ps, top, kill...
    - simples outils texte : cat, more...
    - manipulation de fichiers : cp, rm, mv, ls...
  - Un éditeur de texte, vi le plus souvent
  - Un interpréteur de commande (shell) : sh, bash, csh, zsh...
- **L'administration de la machine a aussi ses nécessités :**
  - outils de configuration/contrôle réseau : ifconfig, netstat, route...
  - outils de configuration du système : gestion du système de fichiers, des utilisateurs, shutdown...
  - serveur sshd si l'accès par le réseau est autorisé
  - outils d'administration de sécurité, en fonction de l'installation :
    - paramétrage du pare-feu (iptables, pfctl, ipfw...)
    - configuration des politiques de sécurité
    - configuration IPsec

Applications nécessaires

- **N'oublions pas le principal !**
  - S'il s'agit d'un serveur, une ou des applications spécifiques seront installées
  - Même si les précautions évoquées jusqu'ici pourront s'appliquer au niveau de sa configuration, on entre dans le domaine de la sécurité au niveau applicatif plus que système.
- **Quelques exemples simples :**
  - serveur web (typiquement, Apache)
    - architecture modulaire : choix rigoureux des plugins installés
    - accès aux répertoires, autorisation de lister les contenus...
  - serveur FTP
    - limitation des commandes autorisées
    - choix d'un serveur sécurisé plutôt que d'un serveur versant dans la surenchère de fonctionnalités
- **La sécurisation de ces démons fait intervenir d'autres concepts abordés plus loin.**

Applications superflues

- **Laissons de côté le confort moderne pour plus de sécurité**
  - Certains éléments sont souvent nécessaires lors de l'installation et de la configuration initiale de la machine, mais représentent un réel danger en production :
    - compilateurs et outils associés : cc, gcc, make
    - debuggers : gdb, valgrind
    - désassembleurs, outils d'analyse de binaire : objdump, fenris
- **L'installation de X se justifie rarement sur un serveur (à moins bien sûr d'être nécessitée par un service fourni)**
- **De manière générale, éviter les outils qui permettraient à un intrus de recueillir des informations variées si leur présence n'est pas totalement justifiée :**
  - sniffers
  - lsof, netcat...

Applications superflues

- **Des cas difficiles à trancher...**
  - Certains outils occasionnellement utiles à un administrateur pourront l'être encore plus pour un intrus
    - Webmin
    - En général, les applications d'administration centralisée
  - Les langages de scripts (Perl, Python...)
    - pratiques pour réaliser des tâches courantes (scripts de l'administrateur)
    - adaptés également pour des petits programmes à utilisation ponctuelle
    - mais véritables boîtes à outils aux possibilités immenses (Perl a été qualifié de Unix's Swiss Army Chainsaw !)
    - certains langages permettent d'obtenir des exécutables compilés : à privilégier
- **Il n'y a pas de « recettes » absolues**
  - une bonne connaissance de son système, un zest de bon sens et beaucoup de paranoïa permettent d'obtenir un degré de sécurité satisfaisant.

Installation

- **Une fois qu'il a été décidé ce qui doit ou non être présent sur le bastion, place à l'installation.**
- **Le procédé sera très dépendant de :**
  - l'OS choisi
  - la distribution adoptée
  - ces choix peuvent découler de contraintes liées au type de machine utilisé
    - contraintes matérielles : l'OS voulu permet-il la pleine exploitation de la machine (RAID, SCSI, multi-processeurs...) ?
    - contraintes d'installation : la distribution permet-elle une installation sur une machine avec des disques SCSI ?
    - contraintes contractuelles : souhaite-t-on utiliser une application certifiée pour un OS ou une distribution ?

Installation

- **Il est assez improbable d'avoir dès l'installation tous les packages voulus, et seulement eux. On peut considérer trois familles d'OS ou de distribution :**
  - les systèmes sobres, permettant de faire une installation minimale, et d'ajouter par la suite les packages que l'on souhaite (OpenBSD)
  - les systèmes lourds, installant de nombreux outils qui devront être supprimés par la suite (Redhat, SuSe, Mandrake)
  - les systèmes intermédiaires, ou il est possible de choisir précisément ses packages dès l'installation du système (Debian, FreeBSD)
- **Parfois il est possible de choisir des méta-packages : groupes de packages groupés par thèmes (jeux, applications réseau, etc).**
- **Selon le type d'installation choisi, on devra ensuite épurer le système ou au contraire le compléter (ce qui est préférable : on sait précisément ce que l'on a).**

Sécurité du système

Sécurité du système

- **Authentification sous Unix**
  - PAM : authentification sur mesure
  - OTP : Les mots de passe jetables
- **Sécuriser les daemons**
  - `chroot`
  - Séparation de privilèges
- **Comptes captifs**
- **Shells restreints**

PAM : Pluggable Authentication Modules

- **Mécanisme d'authentification centralisé**
  - Les applications « PAMifiées » délèguent leur authentification
  - Les librairies PAM contrôlent cette authentification
  - Le paramétrage des librairies PAM est effectué et contrôlé par l'administrateur pour chaque application
  - Deux types de configuration :
    - `/etc/pam.conf` (fichier unique) contient plusieurs lignes par application
    - `/etc/pam.d` (dossier) contient un fichier de configuration pour chaque application utilisant les PAM
- **Intégration des PAM suivant la distribution**
  - Support des PAM
    - Redhat, Mandriva, Debian, FreeBSD
  - Pas de support des PAM
    - Slackware, OpenBSD

PAM : Pluggable Authentication Modules

- **4 catégories gérées**
  - **Authentification** : vérification de l'identité (couple identifiant/authentifiant)
  - **Compte** : vérification des informations de compte (mot de passe expiré, appartenance à un groupe)
  - **Mot de passe** : mis à jour du mot de passe (obliger l'utilisateur à changer de mot de passe après expiration)
  - **Session** : préparation de l'utilisation du compte (tracer la connexion, monter des répertoires utilisateurs)
- **4 contrôles de réussite**
  - **requisite** : tous ces modules DOIVENT réussir
  - **required** : au moins un de ces modules doit réussir
  - **sufficient** : la réussite de ce module suffit à valider la pile de modules
    - résultat ignoré si module “required” a échoué avant
  - **optional** : résultat considéré seulement s’il n’y a pas d’autre module dans la pile

PAM : Pluggable Authentication Modules

- **Exemple : login**

```
auth       required     /lib/security/pam_securetty.so
auth       required     /lib/security/pam_stack.so service=system-auth
  auth       required     /lib/security/pam_env.so
  auth       sufficient   /lib/security/pam_unix.so likeauth nullok
  auth       required     /lib/security/pam_deny.so
auth       required     /lib/security/pam_nologin.so

account    required     /lib/security/pam_stack.so service=system-auth
  account    required     /lib/security/pam_unix.so

password   required     /lib/security/pam_stack.so service=system-auth
  password   required     /lib/security/pam_cracklib.so retry=3
  password   sufficient   /lib/security/pam_unix.so nullok md5 shadow use_authtok
  password   required     /lib/security/pam_deny.so

session    required     /lib/security/pam_stack.so service=system-auth
  session    required     /lib/security/pam_limits.so
  session    required     /lib/security/pam_unix.so
session    optional     /lib/security/pam_console.so
```

Les mots de passe à usage unique (OTP)

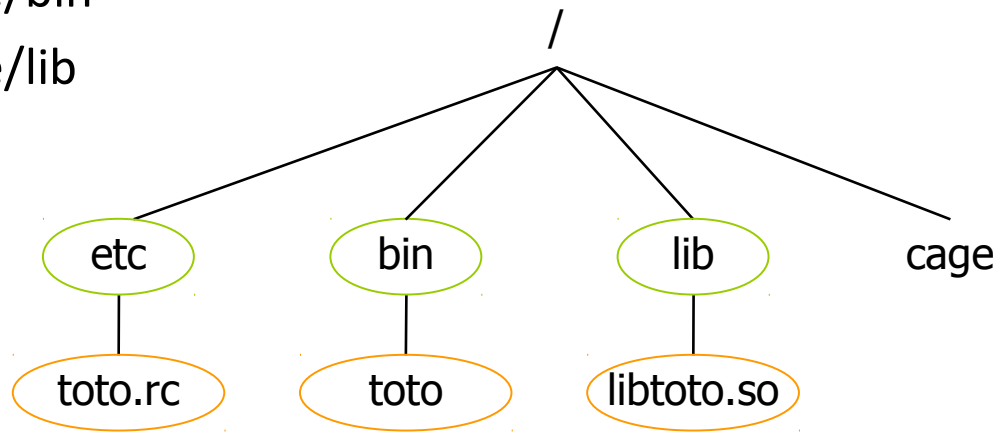
- **Le mécanisme OPIE (One-time Passwords In Everything) est inclus dans FreeBSD et OpenBSD**
  - Il est supporté par login, ftpd et su
  - Un module PAM le prend également en charge
- **L'algorithme S/KEY est utilisé pour générer les mots de passe à usage unique**
  - Lors de la phase de login, l'utilisateur reçoit un challenge
  - Il doit recopier ce challenge dans un calculatrice
  - Celle-ci fournit la réponse au challenge
  - L'utilisateur peut s'authentifier avec cette réponse
- **Les commandes concernant l'usage de OPIE :**
  - opiepasswd
    - Initialise OPIE pour un utilisateur avec le mdp fourni
  - opiekey
    - Calcule les réponse aux challenges OPIE
  - opieinfo
    - Affiche le numéro de séquence et la graine courantes dans OPIE
    - Permet de générer une liste de futures réponses OPIE

Sécuriser les daemons : mise en cage

- **S'il est capable d'exploiter une faille d'un service, un pirate va chercher à exécuter d'autres commandes sur la machine.**
- **Une parade consiste à limiter l'environnement du daemon.**
- **Le principe de la cage**
  - dissimuler une partie du système au programme
  - ne lui laisser voir que ce dont il a besoin pour s'exécuter correctement
    - fichiers de configuration
    - bibliothèques
    - autres exécutables, scripts, etc
    - fichiers spéciaux (pipes, sockets...)
    - fichiers de log
- **En pratique**
  - on fait passer un répertoire du système pour la racine (/) de celui-ci, l'application ne pouvant (en théorie) pas voir en dehors de ce répertoire

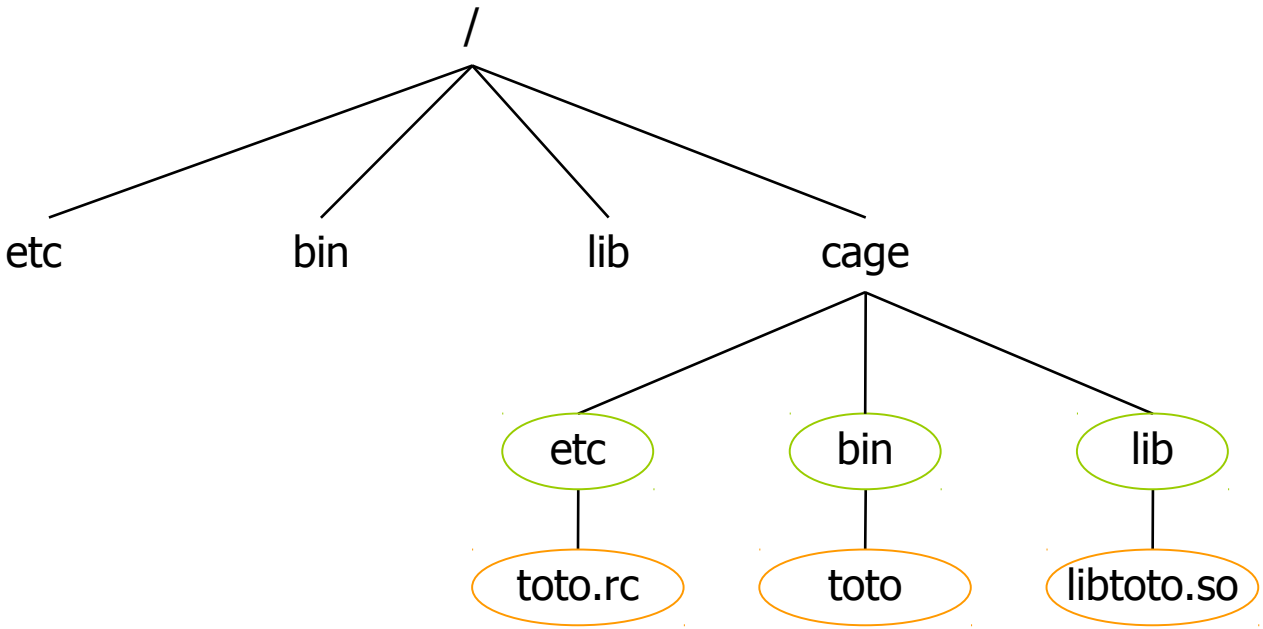
Construction d'une cage

- **Exemple de mise en cage avec chroot**
  - Fichier exécutable : /bin/toto
  - Configuration : /etc/toto.rc
  - Bibliothèque : /lib/libtoto.so
  - Nouvelle racine : /cage
- **On reproduit l'arborescence sous /cage**
  - mkdir /cage/etc
  - mkdir /cage/bin
  - mkdir /cage/lib



Construction d'une cage

- **On copie les fichiers du daemon :**
  - cp /bin/toto /cage/bin/toto
  - cp /etc/toto.rc /cage/etc/toto.rc
  - cp /lib/toto.so /cage/lib/libtoto.so



Construction d'une cage

- **On lance ensuite la commande**
  - `chroot /cage /bin/toto [options]`
- **Comment déterminer les fichiers nécessaires au programme et devant se trouver dans la cage ?**
  - on peut distinguer 2 types de fichiers :
    - les bibliothèques dynamiques
    - le reste
  - les bibliothèques dynamiques chargées par un exécutable sont données par `ldd`
    - on copie ces bibliothèques dans l'arborescence de la cage
    - mais il est également possible d'éviter le problème en recompilant l'exécutable statiquement (édition de liens)
  - les divers fichiers accédés par le programme lors de son exécution peuvent être déterminés avec un outil tel que `strace`
- **Certains daemons ont des options de lancement pour exécuter automatiquement un appel à chroot.**

Sécuriser les daemons : séparation des privilèges

- **Problématique : un programme ayant besoin des droits root à un moment de son exécution doit-il pour autant les conserver tout au long de son exécution ?**
  - Cas typiques : serveurs sur un port inférieur à 1024
- **La séparation de privilèges propose une solution.**
  - un ou des utilisateurs spécifiques sont créés pour l'application
  - celle-ci est lancée avec les droits root
  - les opérations nécessitant ces droits sont effectuées en tant que root
  - les autres opérations sont effectuées avec les droits de l'utilisateur spécifique (changement d'uid/gid)
  - pour cela, utilisation d'un fork :
    - exécution du processus père avec les droits privilégiés
    - exécution du processus fils avec les droits restreints
- **Exemples : sshd, vsftpd, popa3d, qmail**

Compte captif

- **Dans le cas de la séparation de privilèges, on crée des utilisateurs spécifiques, qui ne peuvent pas se connecter.**
- **Il existe une autre catégorie d'utilisateurs spécifiques, voués à l'exécution d'une tâche unique. Ce sont les *comptes captifs* :**
  - leur shell n'est pas un interpréteur de commande
    - attention à ce que l'application choisie ne permette pas l'exécution de commandes non contrôlées !
      - par exemple, more permet de lancer des commandes
  - lorsqu'ils se connectent, l'application définie comme shell s'exécute
  - l'exécution terminée, l'utilisateur est déconnecté, sans avoir eu la main sur le système
- **Cette fonctionnalité peut être utilisée pour sécuriser le système bien qu'historiquement ce ne soit pas ce qui a motivé sa création.**
- **Quelques exemples :**
  - date::60000:100:Run the date program:/tmp:/sbin/date
  - uptime::60001:100:Run the uptime program:/tmp:/usr/ucb/uptime
  - finger::60002:100:Run the finger program:/tmp:/usr/ucb/finger
  - sync::60003:100:Run the sync program:/tmp:/sbin/sync

Shell restreint

- **On peut limiter les fonctionnalités de l'interpréteur de commande de certains utilisateurs (comptes ouverts ou utilisateurs spécifiques).**
- **Pour cela, un shell restreint permet notamment :**
  - de restreindre un utilisateur au répertoire courant (HOME)
  - de l'empêcher de modifier son PATH
  - de l'empêcher d'exécuter des commandes ne se trouvant pas dans son PATH
  - de l'empêcher de modifier des variables d'environnement
  - de limiter les redirections d'entrée/sortie (>, >>)
- **Les variables d'environnement sont fixées par un fichier de configuration (typiquement .profile) au login, et ne peuvent ensuite être modifiées**
- **Des shells tels que bash, zsh ou ksh proposent un mode restreint**

Sécurité des accès réseau

Sécurité des accès réseau

- **Gestion des services**
  - tcp_wrappers, inetd, même combat

Gestion des services

- **Afin de se prémunir contre les attaques réseau, on peut procéder par étapes** :
  - de quels services a-t-on besoin ?
    - élimination des autres
  - pour chaque service nécessaire :
    - sur quelle(s) interface(s) doit-il écouter ?
    - sur quel(s) port(s) (standards ou non) ?
    - comment est lancé le serveur ?
      - au démarrage, en standalone ?
      - manuellement ?
      - via inetd/xinetd ?
  - qui peut accéder aux différents services ?
    - autorisations dans la configuration du service
    - règles avec les tcpwrappers
    - configuration d'un pare-feu
    - accès depuis un VPN (Virtual Private Network)

Gestion des services

- **Les services peuvent être lancés par inetd, le « super serveur »**
  - plutôt que de tourner en permanence sur le système, ils sont invoqués seulement lorsque nécessaire, par inetd. Ceci permet d'alléger le système.
- **Il écoute sur les ports qui sont spécifiés dans `/etc/inetd.conf`**
  - Lorsqu'il reçoit une requête, il la transmet à l'application dédiée.
- **Exemple de configuration :**

```
chargen     dgram  udp   wait     root     internal
discard     stream tcp   nowait   root     internal
telnet      stream tcp   nowait   telnetd.telnetd /usr/sbin/tcpd /usr/sbin/in.telnetd
```

  - syntaxe : port, type de socket, protocole, wait/nowait, utilisateur(.groupe), commande et arguments
    - le mot-clef *internal* désigne un service fourni nativement par inetd
    - l'exemple de telnet utilise ici le tcpwrapper tcpd, qui va permettre d'introduire une notion de contrôle d'accès au service telnet

Gestion des services

- **Les tcpwrappers permettent une ébauche de sécurisation des services lancés via inetd, notamment :**
  - en utilisant du double-reverse lookup sur les adresses distantes
  - en implémentant des ACL (access control lists) basées sur les IP des clients et les services sollicités
  - en ayant recours au protocole ident
  - en permettant de loguer les divers événements
- **Configuration des ACL : `/etc/hosts.{allow, deny}`**
  - avant de répondre à une requête, on regarde si le client est autorisé dans /etc/hosts.allow
  - s'il n'y est pas, on regarde s'il est interdit dans /etc/hosts.deny
  - s'il n'y apparaît pas non plus, la requête est traitée
- **Une syntaxe riche permet de peaufiner la gestion des différents services.**

Gestion des services

- **Exemple de configuration :**

```
# /etc/hosts.allow:
# Autorise telnetd seulement depuis trusted-hosts.org
# L'interdire depuis le reste du réseau
# Autoriser finger pour les machines du réseau local
# Exécuter false_fingerd pour les autres machines
# Tout interdire depuis evil-guys.net
# Autoriser tout le reste
#
telnetd : trusted-hosts.org : allow
telnetd : all : deny
in.fingerd : LOCAL : allow
in.fingerd : all : twist /usr/local/bin/false_fingerd
all : evil-guys.net : deny
all : all : allow
```

Gestion des services

- **Si le principe est similaire à celui d'inetd (super serveur permettant d'alléger la charge système), xinetd a l'avantage de proposer plus de fonctionnalités :**
  - only_from / no_access : adresses de machines ou réseaux pouvant ou non utiliser le service
  - access_times : heures ouvrables du service
  - configuration assez fine des logs possible
  - bind / interface : choix de l'interface sur laquelle on écoute
  - paramètres liés à la charge
    - » limite de connexions
    - » temps durant lequel le service est interrompu quand le maximum est atteint
    - » rlimit : limiter l'utilisation des ressources système
    - » nice : priorité du processus serveur
- **Xinetd est donc à préférer à inetd !**

Gestion des services

- **Ces premières étapes sont malheureusement insuffisantes :**
  - authentification basée uniquement sur des adresses IP
    - vulnérable à l'IP spoofing
      - » exploitable de façon triviale avec UDP
      - » attaques également possibles avec TCP
    - sur un LAN, facilement contournable avec de l'ARP spoofing
  - outils permettant de tromper ce genre de protection : hunt, dsniff, arp-sk...
- **Il faut donc considérer ces protections comme une première étape, et ne pas se contenter d'une « impression de sécurité ».**
  - Le cours d'architecture réseau abordera plus largement ces problèmes, notamment au sujet des pare-feu.

Sécurisation des applications

Sécurisation des applications

- **Les menaces sur les applications**
  - Buffer overflow
- **Protection des applications**
  - Jail
  - Cloisonnement dans le noyau
    - grsecurity
    - SELinux
    - BSD securelevel

Les menaces sur les applications

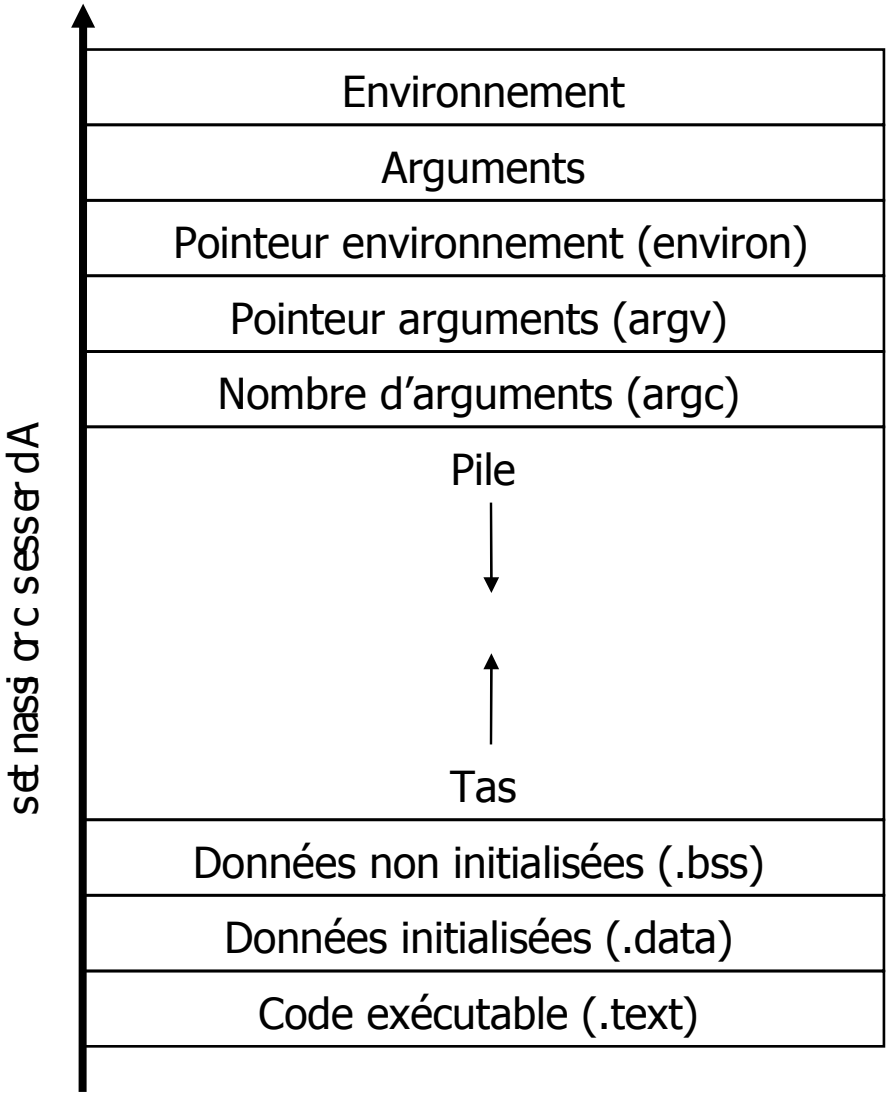
- **Quelles sont les principales causes des vulnérabilités publiées chaque jour ?**
  - principalement des erreurs de programmation
    - mauvais contrôle des données fournies par un utilisateur
    - oubli de cas particuliers
  - aussi bien dans des **applications type serveur** (wu-ftpd, bind, sendmail...) que dans des **applications web** (diverses architectures basées sur php, sur des bases de données, etc...)
- **Les possibilités d'exploitation sont alors multiples**
  - DOS (Déni de Service) : plantage du serveur par exemple
    - atteinte à la disponibilité
  - accès à des données normalement inaccessibles
    - atteinte à la confidentialité
  - exécution de code sur la machine
    - viol de l'intégrité du système

Buffer Overflows

- **Les attaques de type buffer overflow reposent sur la démarche suivante :**
  - profiter du manque de précautions dans la programmation d'une application
    - pas ou peu de contrôle des données passées par l'utilisateur
    - utilisation de fonctions dangereuses
      - strcpy, sprintf, gets...
  - passer ainsi une chaîne de caractères spéciale à l'application
    - contenant du code exécutable
    - permettant de dérouter le flot d’exécution vers ce code
  - faire ainsi exécuter le code voulu par l'application (avec, notamment, les droits de l'application en question)

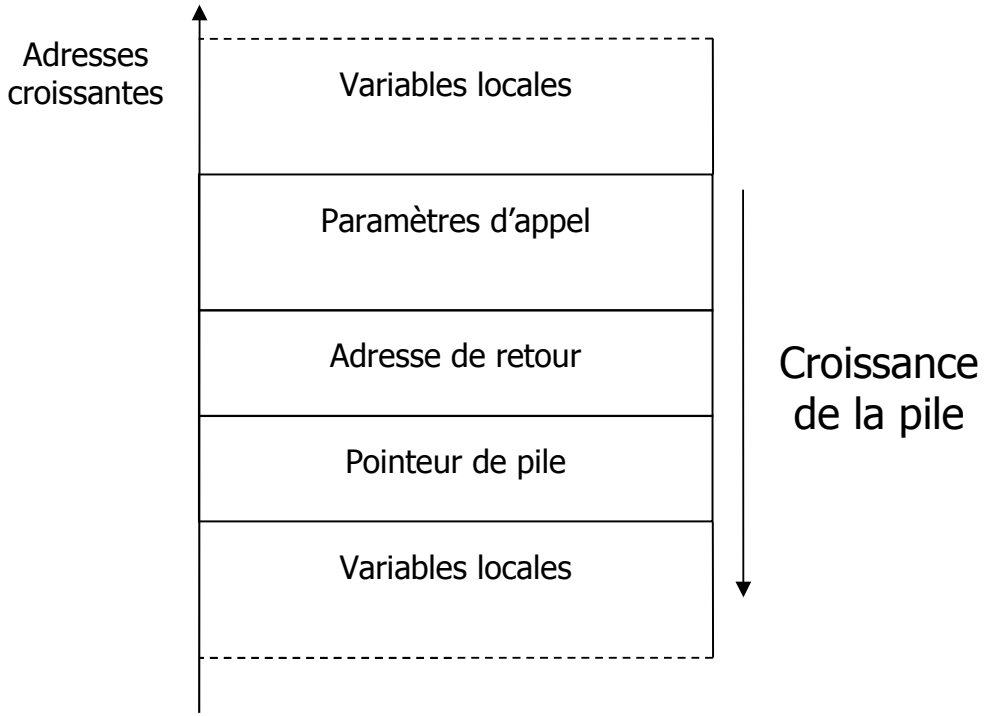
Buffer Overflows

- **Représentation simplifiée de la mémoire virtuelle d'un processus**



Buffer Overflows

- **Structure de la pile**



Buffer Overflows

- **Vulnérabilités liées à la gesion des buffers**
  - code :

```
int main(int argc, char **argv) {
char b1[7] = ''tintin'';
char b2[4] = ''abc'';

strcpy(b2, argv[1]);
printf(''b1 : %s\nb2 : %s\n'', b1, b2);
return 0;
}
```

  - en mémoire, dans la pile : exécution de ''./a.out milou''
    - appel de `strcpy()`

\0	\0	n	i
t	n	i	t
\0	c	b	a

`strcpy(b2, ''milou'');` →

\0	\0	n	i
t	n	\0	u
o	l	i	m

    - Appel de printf()
      - b1 : u
      - b2 : milou

Buffer Overflows

- **La possibilité d'écraser des données dans la pile permet dans certains cas de faire exécuter du code arbitraire au programme vulnérable :**
  - on utilise un shellcode
    - appelé ainsi car il permet usuellement d'obtenir un shell, il consiste en une suite d'instructions assembleur passée sous forme de chaîne de caractères
  - le but du jeu est de modifier, en écrasant sa valeur dans la pile, la valeur de l'adresse de la prochaine instruction à exécuter
  - l'idée est de la faire pointer vers les instructions passées au programme
- **Ici il s'agit d'exploiter simplement des données de la pile, mais il existe des variantes (heap overflow : exploitation du tas ; return-into-libc...)**
- **L'intérêt de faire exécuter du code personnalisé à une telle application se manifeste quand celle-ci est Set-UID (notamment root)**

Sécurité des applications

- **Les types d'attaque que nous avons évoqués découlent de techniques de programmation trop permissives ou oublieuses des problématiques de sécurité.**
  - une programmation rigoureuse et consciente de ce genre de problèmes permet d'éviter une première vague de vunérabilités.
- **Mais on ne peut avoir une totale confiance dans les applications que l'on utilise, d'où le besoin de protections supplémentaires au niveau système.**
- **Intéressons-nous alors à des solutions qui, déployées sur le système, permettent de limiter ou d'empêcher l'exploitation des failles qui ne manqueront pas de se manifester dans nos applications.**

jail

- **La commande jail fait partie du système FreeBSD. Elle étend le concept de chroot.**
  - Création de compartiments étanches dans le système (prisons)
    - Même noyau, mais séparation du système de fichier, de l'accès au réseau, de la liste des processus…
  - Deux utilisations
    - Confinement d'une application (chroot renforcé)
    - Machine virtuelle
      - Limitée : pas d'émulation de matériel ou de chargement de noyau
- **Invocation**

```
jail <path> <hostname> <ip-number> <command>
```

Cloisonnement dans le noyau

- **Pour lutter contre la majorité des buffer overflows**
  - pile, et tas, non exécutables : les zones mémoire accessibles en écriture ne sont pas exécutables
  - adresses aléatoires des fonctions des bibliothèques dynamiques : gêne les attaques de type return-into-libc
  - protections au moment de la compilation : empêcher l'écrasement de certaines informations-clé de la pile (stack guard / stack shield)
  - réimplémentation des fonctions dangereuses de la libc (libsafe)
- **Pour limiter les possibilités d'exploitation d'une faille**
  - chroot permet de restreindre ce qui est perçu, et donc accessible, par l'application vulnérable
  - MAC (Mandatory Access Control)
    - définition de politiques de sécurité pour le contrôle d'accès
    - contrôle d'accès obligatoire par un mécanisme central
    - peut favoriser une plus grande granularité au niveau de la gestion des différents accès (par exemple, contrôler les appels systèmes autorisés pour une application donnée...)

grsecurity

- **Le patch grsecurity pour le noyau Linux propose une série de fonctionnalités orientées sécurité.**
- **On distingue les catégories suivantes :**
  - RBAC
  - Process-based ACL
  - `chroot()` restrictions
  - PaX
  - Auditing
  - Randomization
  - autres : restrictions sur les tubes nommés (FIFO), les sockets, /proc...
- **On peut choisir celles à activer au moment de la compilation du noyau.**

grsecurity

- **Pour se protéger contre les buffer overflows, grsecurity inclut le patch PaX qui implémente :**
  - la non-exécutabilité de la pile et du tas
  - adresses mémoire aléatoires
    - pile
    - tas
    - bibliothèques dynamiques
  - empêcher la récupération d'adresses via /proc
- **Ce type de protection contre les buffer overflows est le plus intéressant :**
  - intégré au système, il ne nécessite pas de recompilation des exécutables
  - il couvre différents types d’attaques
    - stack overflow
    - heap overflow
    - return-into-libc

grsecurity

- **Deux modes d'apprentissage**
  - Mode complet
    - gradm -F -L learning.logs
    - gradm -F -L learning.logs -O policy.new
  - Par rôle
    - Restriction à un ou plusieurs rôles (marqués par l'option l)
    - gradm -E -L learning.logs
    - gradm -L learning.logs -O policy.new
- **Définition des rôles**

```
role admin sA
subject / rvka
   / rwcdmlxi

role default G
role_transitions admin
subject /
   /           r
   /opt                    rx
   /home                   rwxcd
   /mnt                    rw
[…]
```

  - Résolution du rôle : utilisateur -> groupe -> default

grsecurity

- **Confinement d'un processus (ici sshd)**

```
subject /usr/sbin/sshd dpo
   /            h
   /bin/bash            x
   /dev                 h
   /dev/log             rw
   /dev/random          r
   /dev/urandom         r
   /dev/null            rw
   /dev/ptmx            rw
   /dev/pts             rw
   /dev/tty             rw
   /dev/tty?            rw
   /etc                 r
   /etc/grsech
   /home
   /lib                 rx
   /root
   /proc                r
   /proc/kcore          h
   /proc/sys            h
```

```
/usr/lib  rx
/usr/share/zoneinfo r
/var/log
/var/mail
/var/log/lastlog   rw
/var/log/wtmp
  w
/var/run/sshd
/var/run/utmp
  rw

-CAP_ALL
+CAP_CHOWN
+CAP_SETGID
+CAP_SETUID
+CAP_SYS_CHROOT
+CAP_SYS_RESOURCE
+CAP_SYS_TTY_CONFIG
```

SELinux

- **Security-Enhanced Linux est un mécanisme de contrôle d'accès inclus au noyau Linux depuis la branche 2.6**
- **Il permet de réaliser du confinement d'applications**
- **Ses principes de conception sont les suivants**
  - Architecture Flask
    - Séparation des modules « Décision » et « Application »
      - Langage de configuration indépendant de l'architecture
    - Labellisation des sujets et objets
      - `utilisateur:role:type:niveaux`
  - Modèles implémentés
    - Type Enforcement
    - RBAC
    - SELinux UID
    - MLS
  - Audit
    - Configurable par permission individuelle

- **Détails d'implémentation**
  - Intégration dans le code des appels système
    - LSM
  - Security Server en espace noyau
    - Responsable des décisions d'accès
    - Contient la politique de sécurité
  - Modification des commandes standard
    - Librairie d'appels à SELinux : libselinux
    - Intégration dans les packages fileutils, cron, login, sshd…
  - Nouvelles commandes d'administration de SELinux
    - Statistiques : sestatus, seinfo
    - Edition : apol, seaudit, sediff (Tresys)
    - Chargement : checkpolicy (compile et charge la politique)
    - Labels des fichiers : setfiles positionne les labels statiques du système de fichiers (attributs étendus)

- **Exemples : politique de sécurité pour honeyd**
  - Labels des fichiers

```
# Le binaire honeyd
/usr/local/bin/honeyd   system_u:object_r:honeyd_exec_t

# Les fichiers de configuration de honeyd
/usr/local/share/honeyd                 system_u:object_r:honeyd_conf_t
/usr/local/share/honeyd/nmap.assoc      system_u:object_r:honeyd_conf_t
/usr/local/share/honeyd/xprobe2.conf    system_u:object_r:honeyd_conf_t
/usr/local/share/honeyd/nmap.prints     system_u:object_r:honeyd_conf_t
# Le fichier de paramétrage de honeyd que nous utilisions
# sachant qu'il faut développer le sien
/usr/local/share/honeyd/config.sample   system_u:object_r:honeyd_conf_t

# Les scripts lancés par honeyd
/usr/local/share/honeyd/scripts(/.*)?
   system_u:object_r:honeyd_script_exec_t

# Les logs des scripts test.sh et web.sh vont dans /tmp/log
/tmp/log    system_u:object_r:honeyd_script_log_t
# ...
```

  - Ensuite les labels sont appliqués à l'aide de la commande setfiles

Règles de Type Enforcement

```
# Declarons honeyd comme daemon réseau
daemon_domain(honeyd)
can_network(honeyd_t)

# Type declarations pour honeyd
type honeyd_conf_t, file_type, sysadmfile;
type honeyd_script_exec_t, file_type, sysadmfile, exec_type;
type honeyd_script_log_t, file_type, sysadmfile;
type honeyd_script_t, domain, privlog;

# Honeyd peut être exécuté par init ou l'admin
role system_r types honeyd_t;
role sysadm_r types honeyd_t;

# Quand sysadm lance honeyd, il passe dans le domaine honeyd_t
domain_auto_trans(sysadm_t, honeyd_exec_t, honeyd_t)

# Autorisation à honeyd de lire les fichiers de configuration
# Meme si ces fichiers étaient en écriture (rw-), honeyd ne pourrait
# les modifier, à cause du MAC assuré grâce à ces lignes de politiques
allow honeyd_t honeyd_conf_t:dir { search };
allow honeyd_t honeyd_conf_t:file { getattr read };

# /dev/urandom utilisé par honeyd pour générer des nombres aléatoires
allow honeyd_t random_device_t:chr_file { read };

# Honeyd a besoin du réseau
# La capability net_raw est nécessaire (sniffer réseau à base de libpcap)
allow honeyd_t honeyd_t:capability { net_raw };
# reste du réseau :
allow honeyd_t honeyd_t:packet_socket { bind create getopt ioctl read setopt write };
allow honeyd_t honeyd_t:rawip_socket { sendto write create getopt setopt };
# ...
```

BSD securelevel

- **Le noyau FreeBSD définit 5 niveaux de sécurité**
- **Le niveau courant du système est normalement décidé par init**
  - Il peut être placé plus haut par root
  - Il ne peut jamais être abaissé
- **Caractéristiques des 5 niveaux :**
  - -1 : Permanently insecure mode
    - Le système reste au niveau 0 en permanence
  - 0 : Insecure mode
    - Attributs immutable ou append-only modifiables, ouverture des devices en lecture ou écriture
  - 1 : Secure mode
    - Attributs immutable ou append-only non modifiables, écriture interdite sur les devices de disques montés, mem ou kmem, chargement ou retrait de modules noyau impossibles
  - 2 : Highly secure mode
    - 1 + écriture interdire sur tous les devices disques, commande newfs interdite en mode multi-utilisateur, ajustement de l'heure réduit à un maximum de une seconde
  - 3 : Network secure mode
    - 2 + vérouillage des règles de pare-feu
- **Le lancement d'un jail ne modifie pas le niveau du système hôte. Le système placé en jail peut avoir un niveau supérieur.**

Surveillance du système

Surveillance du système

- **Gestion des logs**
  - Mais qu'est-ce qui s'est passé ?...
    - syslog
    - accounting
  - Trop de logs tue l'admin
- **Outils divers**
  - netstat
  - lsof

Les journaux d'évènements

- **Types de surveillances**
  - journaux d'évènements
    - collecte des évènements du système (noyau, démons, ...)
    - gestion des fichiers de logs
    - extraction des informations
  - comptabilité (accounting)
    - enregistrement des ressources utilisées par un processus

Gestion des logs

- **Syslogd est le démon Unix implémentant la gestion des logs système et noyau.**
  - Secondé par klogd pour le traitement des logs du noyau
- **Il permet la gestion des logs**
  - locaux : utilisation d'un socket Unix (/dev/log)
  - distants : écoute sur le port UDP 514
- **Les messages de logs sont classés**
  - selon une famille (facility) : auth, daemon, kern, syslog...
  - puis selon un niveau d'importance (priority) : debug, warning, alert, panic...
- **Chaque message envoyé au démon contient la date et le nom de l'hôte l'ayant émis.**
- **Ensuite, le fichier de configuration /etc/syslog.conf permet de déterminer, selon le couple facility/priority, ce qu'il adviendra du message :**
  - envoyer dans un fichier (lequel)
  - afficher en console
  - ignorer...

Gestion des logs

- **Pour tester une configuration du démon syslog ou lui envoyer des messages depuis des scripts**
  - logger
    - on spécifie une priorité et éventuellement une famille
    - on envoie alors un message au démon
    - celui-ci le traite alors en fonction de ces paramètres, selon les indications du fichier de configuration
- **Pour faciliter l'exploitation simple des logs, on peut mentionner :**
  - logrotate (Linux), newsyslog (FreeBSD)
    - permet l'archivage des logs en renommant et compressant les anciens logs de façon automatique
  - lastlog, wtmp, utmp
    - la commande lastlog et les deux fichiers wtmp et utmp (commandes who, ac) permettent de voir qui est actuellement logué sur le système, et quand un utilisateur s'est connecté pour la dernière fois

Gestion des logs

- **L'analyse des logs doit idéalement être effectuée régulièrement et avec attention.**
- **Néanmoins, il s'agit d'un travail fastidieux, d'où l'existence d'outils permettant de faciliter cette exploitation.**
- **Les fonctionnalités couramment proposées sont :**
  - condenser/résumer les logs
  - trier les logs par catégories
  - mettre en évidence certains logs
    - visuellement, via un système de consultation adapté
    - dans un rapport qui ne présente que les logs intéressants
  - effectuer un traitement périodique des logs
    - résumé journalier envoyé par mail à l'administrateur, par exemple
  - effectuer un traitement sur une durée déterminée
  - exécuter une commande donnée lorsqu'un type de log est rencontré
- **Ceci ne doit pas empêcher une consultation et une conservation traditionnelles des logs.**

Gestion des logs

- **Les fonctionnalités précédentes sont implémentées de façon plus ou moins exhaustive dans quelques outils.**
- **Bien souvent, ces outils sont codés en Perl ou dans un langage de script adapté au travail optimisé sur des chaînes de caractères avec utilisation d'expressions régulières.**
- **On peut citer :**
  - swatch
    - l'un des plus anciens outils de ce type, assez (trop) simple
  - logsurfer
    - continuité de swatch
    - beaucoup plus de foncitonnalités
      - gestion de contextes
      - règles dynamiques
      - gestion des rotations de fichiers
  - logcheck
    - envoi par mail des lignes de logs jugées intéressantes
    - configuration assez fine possible

Comptabilité (accounting)

- **Il peut être intéressant de ne pas se contenter de loguer les débuts et fins de sessions des utilisateurs, mais aussi les commandes exécutées**
  - dans le cas où l'utilisateur paie pour utiliser le système, en fonction des ressources qu'il consomme
  - dans le cas où l'on veut surveiller l'activité des utilisateurs, les programmes qu'ils utilisent...
- **Un support de cette fonctionnalité est nécessaire au niveau noyau.**
- **Les informations obtenues sont du type**
  - utilisateur
  - commande
  - conditions d'exécution, de terminaison
  - date de fin, temps
- **Les données sont sauvegardées dans un fichier**
  - L'accès au journal doit être restreint
- **Les outils associés**
  - `accton` : active/désactive l'accounting
    - `accton /var/account/acct`
  - `lastcomm` : informations sur une commande passée
    - `lastcomm [user | tty | command ]`
  - `sa` : résumé de l'accountig

Outils divers

- **Contrôle des paramètres noyau : `sysctl()`**
  - Exemples concernant la pile IP
    - contrôlés via le /proc : net.ipv4.conf.all.*
    - arp_filter, arp_announce, arp_ignore, ...
    - log_martians, rp_filter, echo_ignore_broadcast, ...
    - tcp_syncookies, synack_retries, syn_retries, ...
  - ou sous forme de clés dans /etc/sysctl.conf
    - sysctl -w <clé=val>: changer une valeur de clé
    - sysctl -p [fichier] : lit un fichier de conf
    - sysctl -a : affiche toutes les clés

Outils divers

- **Surveiller ses connexions**
  - `netstat` : affiche les information relatives au réseau
    - -r : table de routage
    - -au | -at : connexions en cours et serveurs en écoute
    - -l : sockets en écoute
    - -p : processus associé (non standard)
  - `lsof` (list open files) : indique les fichiers
    - -i [[port:]TCP|UDP] (-i 22:TCP) : sockets utilisées
    - -p PID : fichiers associés à un processus
  - `fuser` : indique les PIDs utilisant des fichiers/socket
    - -n domaine : tcp, udp (-n tcp 22)

Références

Références

- **Sécurité système sous Unix :**
  - Practical Unix & Internet Security
    - Garfinkel & Spafford, chez O'Reilly
- **Comprendre les vulnérabilités**
  - Smashing the stack, for fun and profit
    - Aleph1
  - série d'articles sur les débordements de buffer, les bugs de format…
    - www.security-labs.org
  - séparation de privilèges
    - www.hsc.fr/ressources/presentations/privsep/index.html.en
- **Protections**
  - la page de grsecurity
    - www.grsecurity.net
  - SELinux
    - nsa.gov/selinux/
  - le handbook FreeBSD
    - www.freebsd.org
  - les ressources sur le site OpenBSD
    - www.openbsd.org