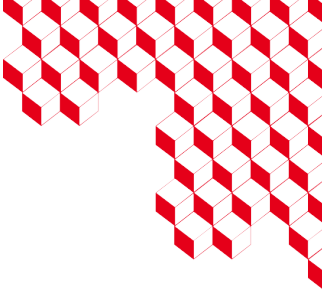


Sécurité des Systèmes Linux

Sécurisation intrinsèque du système

Mathieu Blanc

Commissariat à l'Énergie Atomique et
aux Énergies Alternatives

Sommaire

Préambule à la sécurité système

Sécurité système

Sécurité des applications

Installation d’un système minimal

Sommaire

Préambule à la sécurité système

Sécurité système

Sécurité des applications

Installation d'un système minimal

Sécurité du matériel

- Environnement du système et étapes préparant le démarrage du système d'exploitation, notamment
  - Piégeage des composants
  - Séquence de boot
  - Rayonnement électromagnétique
- Aspects périphériques non techniques
  - Corbeilles, tableaux blancs, imprimantes partagées
  - Risques d'incendie, électriques, inondations
- Objectifs
  - Meilleure évaluation de la cohérence de la sécurité

Sécurité du matériel

- Pourquoi placer les mécanismes de sécurité dans les couches inférieures du système
  - Augmenter l'assurance grâce à la simplicité des mécanismes
  - Réduire la dégradation de performance grâce à l'intégration

- Une vulnérabilité dans les couches inférieures peut annuler les efforts de sécurisation des couches supérieures (effet « court-circuit »)
  - Vol de disques
  - Parallélisme avec les couches réseaux (attaques ARP)
  - Parallélisme avec la frontière Applications/système (serveur Web/sécurité système)

Sécurité du matériel

- Attaques par les composants matériels
  - Attaques sur le bus Firewire (2005)
    - R/W sur la mémoire vive
    - Déverrouillage de session, passage de processus en UID 0
  - Attaques sur la SMRAM
    - Emplacement idéal pour un rootkit discret
    - Détection quasi-impossible
  - Attaques sur le bus PCI
    - Espionnage d'autres composants, injection de données
    - R/W sur la RAM (encore !)
  - Nouvelles technologies pour l'administration des machines
    - Exemple : Intel AMT
    - Possibilité d'installer un système entièrement à distance

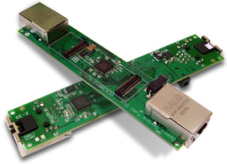
Sécurité du matériel

- Sécurité du boot
  - BIOS : contrôle des périphériques de démarrage
  - OS Loader : verrouiller la configuration du système, sinon
    - `linux init=/bin/bash`
    - Windows (F8) : mode debug
  - Chiffrement de disque
    - mais... copie de la RAM à chaud
    - mais... piégeage du loader avec un keylogger
  - Puces TPM
    - possibilité de stocker une empreinte de la séquence de démarrage, du BIOS au noyau
    - mais... complexité de la mise à jour du système
    - mais... attaques matérielles possibles

Sécurité du matériel

- Méthodes à 2€ (environ)
  - Keylogger hardware

  - Module de capture sur VGA, DVI ou HDMI
  - PC miniature
  - Attaques dites « du personnel d'entretien »



Sécurité du matériel

- Rayonnements compromettants
  - Effet « TEMPEST »

    Tout matériel ou système qui traite ou transmet, sous forme électrique, des informations est sensible à des perturbations électromagnétiques temporaires. Ces perturbations, qualifiées de signaux parasites, sont provoquées par les variations du régime électrique, dans les différents circuits du matériel considéré durant son fonctionnement.

    - Ondes électromagnétiques
    - Courant de conduction le long de canalisations ou de câbles
  - Capture à distance de frappes clavier, d'image sur un écran
  - Contre-mesures : cage de Faraday, brouilleurs

Sommaire

Préambule à la sécurité système

Sécurité système

Sécurité des applications

Installation d'un système minimal

Sécurité du système

- Authentification sous Unix
  - PAM : authentification sur mesure
  - OTP : mots de passe jetables
- Autorisation
  - dans les applications type `login` ou `ssh`
- Audit
  - Surveillance du système
  - Framework d'audit

Identification et Authentification

- Identification
  - Savoir qui est qui, pour déterminer les accès autorisés
  - Notion de login ou `username`
- Authentification
  - Prouver qu'on est qui on prétend être
  - Notion de secret partagé ou `password`

Un peu de théorie

- On fait de l'authentification depuis longtemps
  - Techniques fondamentalement basées sur la présence physique des personnes :
    - Témoignage d'un gardien
    - Détention d'un badge
    - Signature
  - Repose essentiellement sur la dissuasion
    - Contournement simple techniquement mais sévèrement réprimé
- L'authentification électronique a été conçue pour s'abstraire des limites des moyens classiques :
  - Économie de surveillance
  - Possibilité d'accès distants
  - Disponibilité
- Le problème repose alors sur la notion de « preuve électronique »

Un peu de théorie

- Les protocoles d'authentification sont les méthodes utilisées pour mettre en place un contrôle d'accès logique aux systèmes d'informations (ici systèmes Unix)
- Ces protocoles sont d'autant plus critiques que les utilisateurs aiment les systèmes « Single Sign On »
- Le contrôle d'accès se décompose en deux étapes :
  - L'identification: « Mon nom est Bond, James Bond »
  - L'authentification: « Voici mon code d'accès au MI5 »
- Plusieurs approches pour l'authentification sont possibles en fonction de ce que j'apporte pour m'authentifier :
  - Quelque chose que je connais
  - Quelque chose que je possède
  - Quelque chose que j'incarne

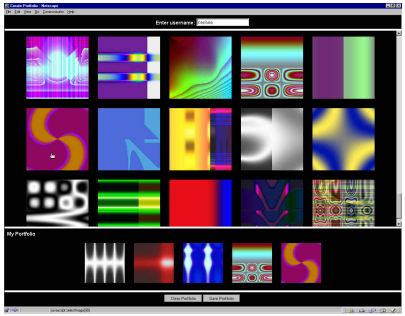
Un peu de théorie

- Quelque chose que je connais
  - C'est l'approche la plus fréquemment rencontrée
  - L'utilisation d'un mot (phrase) de passe est l'implémentation la plus courante de cette approche

- On peut mentionner quelques alternatives connues
  - L'implémentation par association
    - Le système connaît préalablement des associations et vous en soumet une ou plusieurs pour validation
    - +: Cela augmente la quantité d'information qu'un attaquant doit connaître pour usurper une identité
    - -: Nécessite un arrangement compliqué entre utilisateurs et système
  - L'implémentation par « challenge-response »
    - Le système pose une question à laquelle seul l'agent identifié est sensé pouvoir répondre
    - +/- : idem
    - Schéma aussi utilisé dans l'approche (« ce que je possède »)

Un peu de théorie

- Usenix, Security Symposium 2000, Denver (CO), « Déjà-vu »
  - Rachna Dhamija
  - Problème du facteur humain dans l'authentification
  - Idée: facile à retenir, difficile à transmettre
  - Utilisation en production ?



Un peu de théorie

- Quelque chose que je possède
  - Autre moyen de résoudre le problème de la transmission / duplication de l'authentifiant
  - Après l'identification d'un agent par le système, un moyen de valider cette identité est de vérifier la possession par cet agent d'un dispositif que seule la personne identifiée possède
  - Cela implique que
    - Système et agent se soient mis d'accord sur le dispositif à posséder
    - Le système peut facilement et sûrement reconnaître ce dispositif
    - L'agent ne le perd / donne pas

  - Différente implémentation de cette approche sont possibles:
    - Type « clef » : la possession du dispositif suffit (secureID)
    - Type « carte à puce » : possession + mot de passe (code PIN)
    - Type « calculette » : possession + challenge/response

  - Certains de ces mécanismes posent des problèmes de synchro
  - Résout partiellement certains pbs liés à l'approche par mot de passe (divulgation, trivialité, certaines attaques réseaux ... pas toutes)

Un peu de théorie

- Quelque chose que j'incarne
  - Cette approche consiste à vérifier l'identité prétendue par une caractéristique inhérente de l'agent identifié
    - Chaque être humain possède plusieurs caractéristiques uniques : Empreintes digitales, vocales, rétiniennes, forme de la main, ADN ...
  - Résout certains problèmes des approches précédentes (oubli, dons, divulgation)
  - Rarement utilisé (coût, difficulté de mise en œuvre), uniquement dans des environnements où l'usurpation d'identité est très critique ou dans le cadre d'accès physiques (AdP)
  - Pose des problèmes juridiques et culturels
  - Répond plus au problème d'identification que d'authentification (il est difficile de changer d'empreinte rétinienne ...)

Authentification forte

- Définition : AF = 2F
- 2 des 3 facteurs sont nécessaires
  - Je connais
  - Je possède
  - Je suis / j'incarne
- Approche privilégiée : 1 + 2
  - 3 : peu sûr
- Exemples :
  - Accès des postes nomades
    - SSH + PAM + RADIUS + OTP
  - Accès RNIS pour astreinte
    - Serveur RAS + OTP
  - Kerberos + Carte à Puce (PKINIT)

PAM : Pluggable Authentication Modules

- Mécanisme d'authentification centralisé
  - Les applications « PAMifiées » délèguent leur authentification
  - Les bibliothèques PAM contrôlent cette authentification
  - Le paramétrage des bibliothèques 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
    - 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
  - Lors de la phase de login, l'utilisateur reçoit un challenge
  - Il doit recopier ce challenge dans une 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

Autorisation

- Objectifs de l'autorisation
  - L'authentification conditionne l'accès sur la preuve d'identité
  - L'autorisation conditionne l'accès sur le besoin
    - Besoin par rapport à un rôle particulier :
      utilisateur, administrateur, client, fournisseur ...
    - Besoin par rapport à une tâche particulière :
      indexation, sauvegarde, exécution planifiée ...

- Mécanismes d'autorisation
  - Pas de mécanisme unifié sur les systèmes Unix
  - Traitement spécifique à chaque application
  - En général sur les programmes de type `login` ou `ssh`

- Le terme autorisation est quasi-équivalent à celui de contrôle d'accès

Autorisation

- Exemples de programme incluant du contrôle d'accès
  - Modules d'accès PAM (catégorie session)
    - `pam_access`, `pam_time`
    - utilisé essentiellement par `login`
  - `su`
    - configuration spécifique `/etc/sudoers`
  - `sshd`
    - {*Allow*|*Deny*}{*Groups*|*Users*}
  - `apache`
    - Directives *Allow*/*Deny*

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, daemons, ...)
    - gestion des fichiers de logs
    - extraction des informations
  - comptabilité (accounting)
    - enregistrement des ressources utilisées par un processus

Gestion des logs

- `syslogd` est le daemon Unix implémentant la gestion des logs système et noyau (secondé par `klogd` pour les logs 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` ...
  - selon un niveau d'importance (`priority`) : `debug`, `warning`, `alert`, `panic` ...
- Chaque message contient la date et le nom de l'hôte l'ayant émis
- Puis le fichier de configuration `/etc/syslog.conf` permet de déterminer, selon le couple `facility` / `priority`, si le message doit être envoyé dans un fichier (lequel), affiché en console, ignoré ...

Gestion des logs

- Pour tester une configuration du daemon `syslog` ou lui envoyer des messages depuis des scripts
  - `logger`
    - on spécifie une priorité et éventuellement une famille
    - on envoie un message au daemon
    - 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` / `utmp` (commandes `who`, `ac`) permettent de voir qui est actuellement loggé 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 fonctionnalités : 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 de type : utilisateur, commande, conditions d'exécution, de terminaison, date de fin, temps, ...
- Les données sont sauvegardées dans un fichier (accès à restreindre)
- Les outils associés
  - `accton` : active / désactive l'accounting
  - `lastcomm [user|tty|command]` : infos sur une commande passée
  - `sa` : résumé de l'accounting

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 configuration
    - `sysctl -a` : affiche toutes les clés

Outils divers

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

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, 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`

Sommaire

Préambule à la sécurité système

Sécurité système

Sécurité des applications

Installation d'un système minimal

Sécurisation des applications

- Les menaces sur les applications
  - *buffer overflow*
- Protection des applications
  - `chroot`, `jail`
  - Cloisonnement dans le noyau
    - `grsecurity`
    - `SELinux`

Les menaces sur les applications

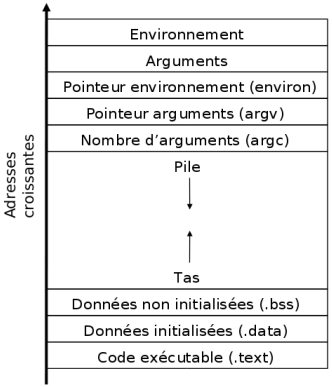
- Quelles sont les principales causes des vulnérabilités publiées chaque jour ?
  - principalement des erreurs de programmation
    - mauvais contrôle des données fournies par un utilisateur
    - oubli de cas particuliers
  - aussi bien dans des applications type serveur (`wu-ftpd`, `bind`, `sendmail`, ...) que dans des applications web (diverses architectures basées sur `php`, sur des bases de données, ...)
- 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
    - violation de l'intégrité du système

Buffer Overflows

- Les attaques de type *buffer overflow* reposent sur le principe suivant :
  - profiter du manque de précautions dans la programmation d'une appli
    - 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)

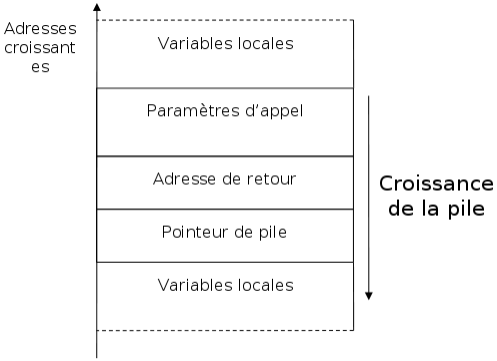
Buffer Overflows

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



Buffer Overflows

- Structure de la pile :

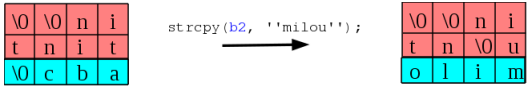


Buffer Overflows

- Vulnérabilités liées à la gestion 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`



    - appel de `printf`

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 vulnérabilités.
- 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.

Sécuriser les daemons : mise en cage

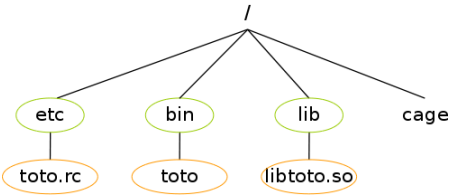
- S'il est capable d'exploiter une faille d'un service, un pirate va chercher à exécuter d'autres commandes sur la machine.
- Une parade consiste à limiter l'environnement du daemon.

- 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
    - 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

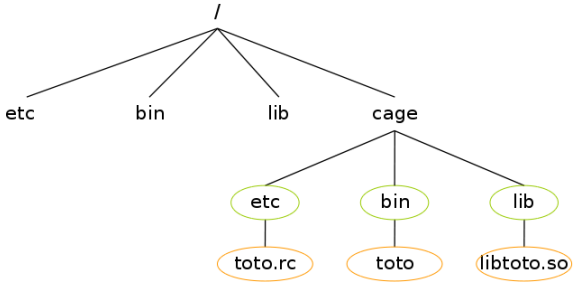
Construction d'une cage

- 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 deux types de fichiers :
    - les bibliothèques dynamiques
    - le reste
  - les libs dynamiques chargées par un binaire 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`.

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 usuel.
    attention à ce que l'application choisie ne permette pas l'exécution de commandes non contrôlées ! (comme `more` par exemple)
  - 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
```

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 absentes de 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.

Sommaire

Préambule à la sécurité système

Sécurité système

Sécurité des applications

Installation d’un système minimal

Installation d’un système minimal

- Objectifs
  - Planning de l’installation
  - Pourquoi minimiser le système ?
- Noyau
  - Être 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 packages 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 packages
    - contre : pas aussi sûr que OpenBSD, pas aussi performant que FreeBSD

Planning de l'installation

- Préparation de l'installation et mises 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 hashs MD5 par le pirate
  - Vérifier les signatures numériques
    - garantit l'origine, l'intégrité, ...
      mais 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, si 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 des 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, ou pour le réseau : ppp, 802.11, infrarouge, bluetooth, ...
  - 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 :
    - 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 (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, 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 daemons 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 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 d'administration)
    - 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
- Il n'y a pas de « recette » absolue
  - une bonne connaissance de son système, un zeste de bon sens et beaucoup de paranoïa permettent d'obtenir un degré de sécurité relativement satisfaisant.

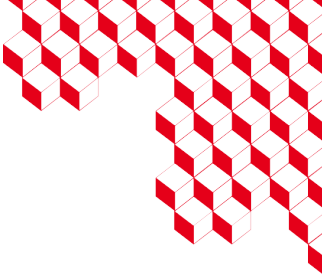
Installation

- 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
    - 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 distributions :
  - 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, où 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 ce qu'on a).



Vous avez des questions ?

mathieu.blanc@cea.fr