

# Sécurité physique

**Mathieu Blanc**

**Pierre Lalet**

**CEA**

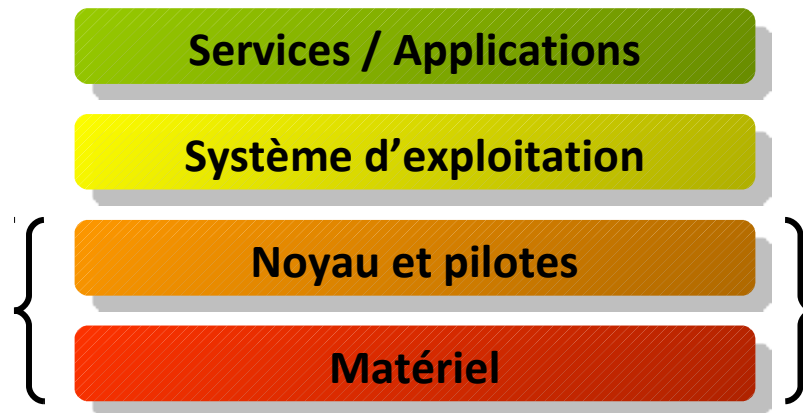
- **Sécurité physique**
- **Séquence de démarrage**
- **Identification et élimination des rootkits**

# Sécurité physique



- **La sécurité « matérielle » par opposition à la sécurité « système »**
  - Tout ce qui se passe (ou peut se passer) avant que le système d'exploitation soit complètement opérationnel:
  - Inclut la séquence de boot: « boot strapping »
- **On ne s'intéresse pas aux aspects périphériques (mais importants) de la sécurité matérielle (paper-board, corbeille, imprimantes partagées ...), ni aux aspects sécurité physique au sens contrôle d'accès ou « protection contre les sinistres » (incendie, inondation ...)**
- **Objectifs:**
  - Sensibilisation
  - Meilleure évaluation de la cohérence des mesures de sécurité

- **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 rendre inutiles 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)

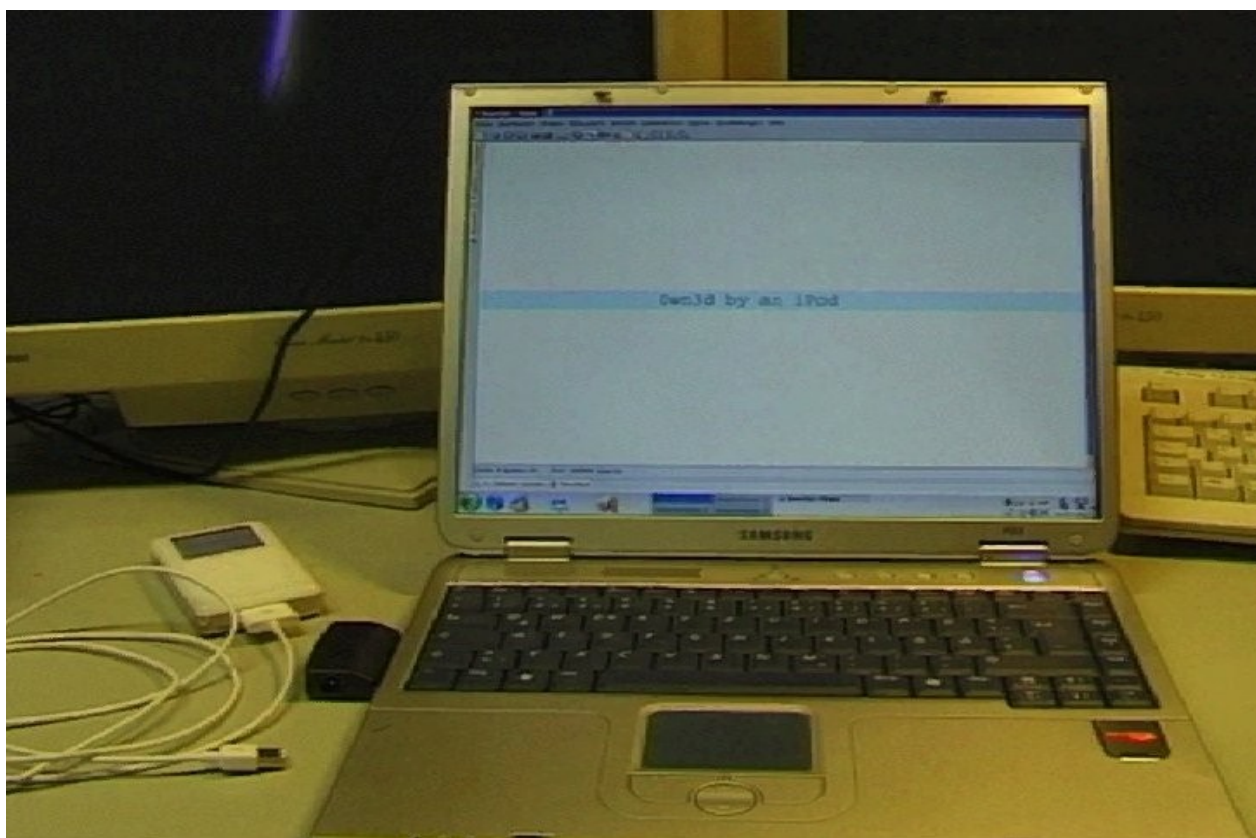


- **Vol de disque dur**
  - **Contre-mesures « physiques »**
    - Contrôle d'accès physique (prévention)
    - Cadenas (prévention, dissuasion)
    - Scellés (dissuasion)
  - **Contre-mesures « architecturales »**
    - Pas de données sur les stations de travail (voire terminaux passifs)
    - L'accès physique aux serveurs (salle machine) doit être contrôlé (ne serait-ce que pour le réseau...)
  - **Contre-mesures « logiques »**
    - Chiffrement de données
    - Chiffrement de fichiers vs chiffrement de FS
    - Chiffrement de FS : Utilisation de la fonctionnalité **crypto-loop** du noyau 2.6
      - DM-crypt (Device Mapper Crypto)



- **Attaque de la procédure de boot**
  - Boot en single ou sur un autre périphérique
- **Contre-mesures : protection de la procédure de boot**
  - Éviter les multi-boot
  - Prédéfinir la séquence de boot (on ne boot pas sur un périphérique externe, sur certains PC : F12 non blocable)
  - Verrouiller le BIOS
    - En Général, plusieurs niveau de verrouillage

- **Attaque via port FireWire (Maximillian Dornseif)**
  - <http://md.hudora.de/presentations/firewire/2005-firewire-cansecwest.pdf>
- **USB ?**





- **Installation de nouveau matériel: risques induits**

- De nouveau type de périphériques d'entrée/sortie se développent :
  - Il faut valider leur conformité avec votre politique de sécurité
    - Ex: port USB ou Firewire
  - Écoute du clavier



Flash memory size	Price
512K	99\$
1Meg	149\$
2Meg	199\$

- Fermez votre bureau à clef et surveillez la femme de ménage 😊

- **Points spécifiques à Linux sur PC:**

- **Gestion des consoles virtuelles**

- Linux permet d'utiliser plusieurs consoles virtuelles.
- Init après avoir traversé chaque « runlevel », « fork » un getty pour chaque console virtuelle (6 par défaut), selon les instructions contenue dans /etc/inittab
- Le verrouillage de ces consoles peut poser un problème aux utilisateurs
- Contre-mesure:
  - La commande vlock (Virtual Console lock program) avec l'option « -a »
  - Supprimer ces consoles virtuelles

- **Gestion du serveur X et interruption Ctrl+Alt+Del**

- Le verrouillage de l'écran peut être désactivé
- Contre-mesure:
  - Supprimer les consoles virtuelles et inhiber la séquence de touche Ctrl+Alt+Del
  - Exec startx

- **Rayonnements compromettants (TEMPEST)**
  - Réf. : « Computer Security Basics », O'reilly; DCSSI, ...
  - 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.
  - Généralement les **signaux parasites** se manifestent sous la forme :
    - d'ondes électromagnétiques qui se propagent par rayonnement dans l'espace environnant ;
    - de courants de conduction qui se propagent le long des différents conducteurs reliés au matériel concerné.
  - Parmi l'ensemble des signaux parasites générés durant le fonctionnement des matériels, il en existe **qui sont représentatifs des informations traitées**. Leur capture et leur analyse peuvent permettre la restitution des informations. Ces signaux sont ainsi qualifiés de *signaux parasites compromettants*.
  - Contre-mesure: Cage de Faraday, zonage électromagnétique et de couplage (voir document du DCSSI)

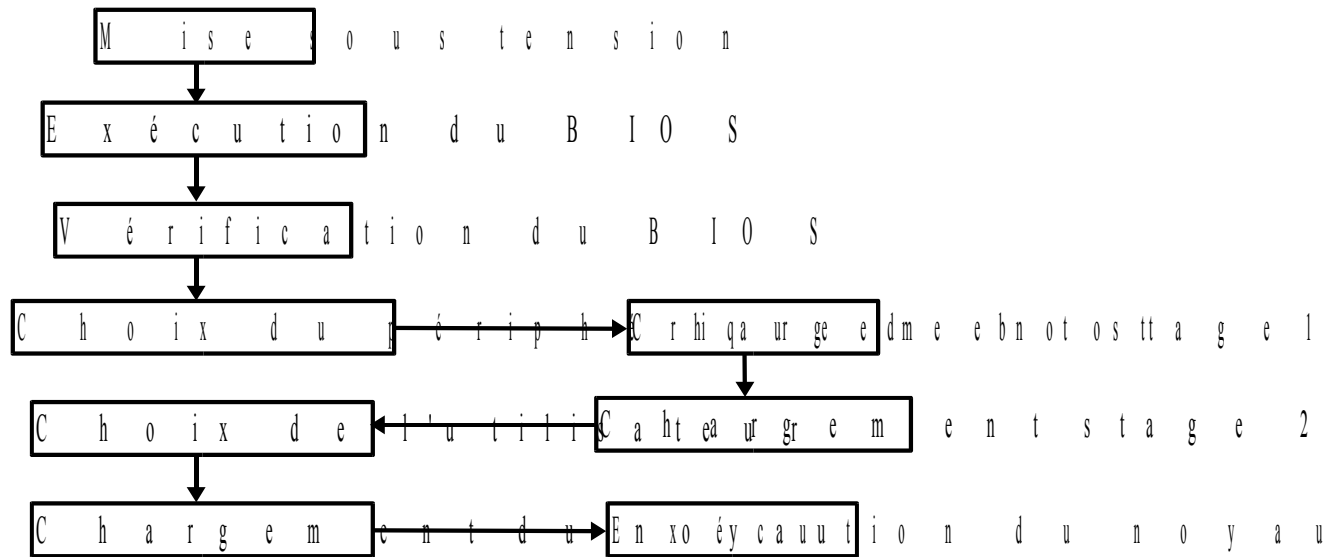
- **« *If a bad guy has unrestricted physical access to your computer, it's not your computer anymore* »**
- Importance des couches inférieures (principe qu'on retrouve dans d'autres domaines de la sécurité informatique).
- Bien connaître son matériel (maîtriser les configurations matérielles de votre parc ...)
- Étudier les conséquences des évolutions.
- L'accès physique à vos serveurs doit être réservé aux administrateurs.
- Tester, patcher.
- Penser à ces problèmes pour évaluer globalement la cohérence/pertinence d'une solution de sécurité.
- Axe de réflexion : Qu'en est-il de l'accès physique à une prise réseau?

# Le processus de boot

# Le processus de boot

- **Le boot, une pièce en 5 actes**

- hardware : alimentation et tests des composants
- firmware (BIOS) : détection des composants
- bootloader (ou autre) : préparation pour l'OS
- noyau : chargement des composants clés de l'OS
- utilisateur : démarrage des services, interfaces, ...





- **Et le courant fut ...**
  - état indéterminé de tous les composants matériels
    - processeurs, bus, MMU, périphériques, ...
  - chargement en mémoire du BIOS écrit en ROM
    - ROM: Read Only Memory
- **Risques**
  - accès physique à un ordinateur
    - vol, destruction, ajout de “mouchards”, ajout de matériel
  - backdoor matérielle
    - générateur aléatoire biaisés
  - signaux électro-magnétiques compromettants
    - effet *Tempest*

- **Déroulement du BIOS**

- POST (Power On Self Test): init. composants essentiels
  - Processeur(s), mémoire, bus, ...
- initialisation extensions
  - Cartes AGP, PCI, bus IDE, SCSI...
- recherche le loader sur périphériques de stockage
  - disque dur, disquette, CDROM, clé USB...
- chargement du boot loader en mémoire
  - Situé en MBR ou en début de partition
  - Taillé pour le MBR (un secteur du disque)
    - 446 octets
- flux d'exécution transféré au boot loader
  - Tentative de démarrage du système d'exploitation





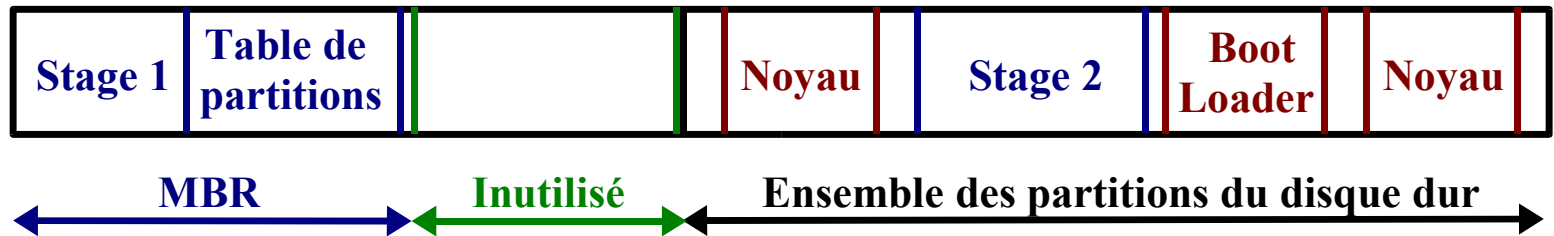
- **Modifications des paramètres**
  - booter sur un “autre” périphérique, et donc un autre OS
    - clé USB, disquette, CD-ROM, réseau ...
  - accès au(x) périphérique(s) natif(s)
    - `mount -t ext2 /dev/hda1 /mnt/hackme`
  - altérations de fichiers sensibles
    - `chroot /mnt/hackme useradd -u 0 -g 0 r00t`
- **Protection par mot de passe**
  - objectif : empêcher l'accès au BIOS
  - problèmes : protection du mot de passe
    - accès physique et reset de la carte mère (pile/jumper)
    - mots de passe “universels” (ex: AWARD 4.50 : AWARD\_SW)
    - cracker : `cmospwd` (C. Grenier)



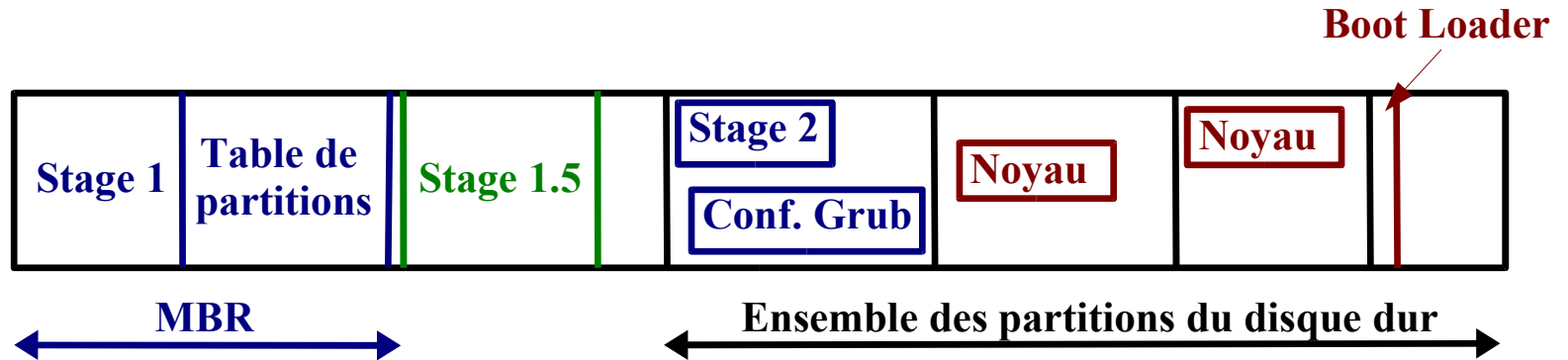
- **Objectifs**
  - Chargement d'un noyau
    - Charger le noyau d'un OS en mémoire
    - Transférer le flot d'exécution au noyau
  - Choix d'un système d'exploitation
    - Plusieurs noyaux d'un même OS
    - Plusieurs OS sur une même machine
    - Passage de différents paramètres au noyau chargé
  - Emplacements
    - Disquette : secteur de boot
    - Disque dur : MBR
    - CDRom : émulation disquette ou disque dur
- **En bref, le boot loader offre une souplesse supplémentaire au chargement d'un OS**



- **Déroutement**
  - Le code exécutable placé en MBR est appelé « stage 1 »
  - Le stage 1 n'est pas suffisant pour lancer l'OS
  - Son rôle est donc de charger un « stage 2 » plus volumineux et de lui passer le contrôle de d'exécution
    - Affichage du menu
    - Chargement noyau et options (notamment initrd)
    - Transfert exécution au noyau
- **Linux**
  - LILO (Linux Loader)
    - Fonctionnement simple
    - Non interactif
  - GRUB (Grand Unified Bootloader)
    - Lecture de plusieurs systèmes de fichiers
      - ext2, FAT, JFS, Minix, ReiserFS, XFS (grub 0.97)
    - Prompt interactif
- **Windows**
  - Boot loader par défaut
    - Repère la partition active et lui transfère le contrôle



- **A l'installation de LILO :**
  - Stage 2 contient les références statiques aux noyaux et autres boot loaders vers lesquels il peut pointer
    - Secteurs du disque
    - `/etc/lilo.conf`
  - Stage 1 connaît l'emplacement statique du stage 2
- **Si l'emplacement du stage 2 est modifié, LILO ne fonctionne plus**
  - Affichage typique : LI
- **Si un des emplacements des noyaux est modifié, celui-ci ne peut plus être chargé en mémoire par LILO**
- **Donc, pour toute modification (noyau, fichier de configuration) :**
  - LILO doit être ré-installé (stage 2 à modifier, qui implique la modification du stage 1)



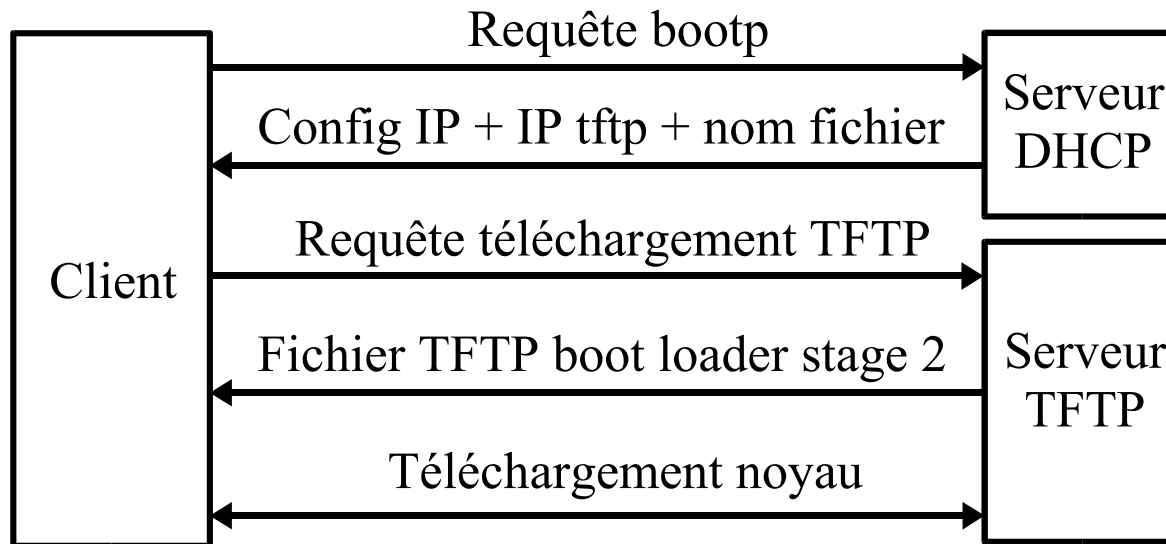
- **A l'installation de Grub, les stages 1, 1.5 et 2 sont copiés**
  - Le stage 1 est installé pour charger le stage 1.5
  - Le stage 1.5 permet de lire un système de fichiers spécifique
  - Il repère le fichier stage2 et le charge
  - Le stage 2 va lire le fichier de configuration Grub
    - Fichier ou lien symbolique
    - `/boot/grub/menu.lst`
- **Les noyaux sont vus comme des fichiers**
  - en cas de modification, pas besoin de ré-installer Grub
- **Idem pour les boot loaders qui sont vu comme des partitions**
- **Idem pour le fichier de configuration de Grub**
- **GRUB peut également remplacer le loader FreeBSD**



- **Injection de paramètres**
  - démarrage en “mode sans échec” (single mode)
    - LILO : linux single
  - surcharge du root device
    - LILO : linux root=/dev/cdrom
  - surcharge du processus init
    - LILO : linux init=/bin/sh
- **Protection**
  - Empêcher l’affichage du prompt
    - LILO : ajouter delay=0, retirer prompt
  - Protéger la modification des entrées du menu
    - GRUB : mot de passe pour la ligne de commande
  - Protéger l’OS par mot de passe
    - LILO : directive password
      - mdp en clair dans /etc/lilo.conf ET /boot/map
      - théoriquement possible de mettre du md5
    - GRUB : password --md5 <hash>

# Boot réseau (PXE)

- **PC possédant une carte réseau compatible PXE**
  - Capable de booter par le réseau
- **Une carte réseau PXE est sélectionnable par le BIOS comme périphérique de boot**
  - Le BIOS de la carte mère va gérer le boot réseau
- **Sécurité du boot réseau**
  - Serveur DHCP + TFTP pirate
    - Protocoles sur UDP (hijacking plus facile)





- **Le boot sur CD-ROM**
  - Systèmes de fichiers sur CD-ROM : ISO9660
  - Plusieurs extensions
    - Joliet : Noms longs type FAT
    - Rock-ridge : Extensions Unix (protections, propriétaire, ...)
    - El-Torito : CD-ROM bootable
- **Intérêts**
  - Permet l'installation de l'OS
  - Permet le boot sans utiliser le disque dur
    - Pour réparer un système endommagé
    - Pour sauvegarder un système
    - Pour upgrader un système
  - ATTENTION : lancement possible d'un autre OS si le BIOS est mal protégé
  - Permet l'utilisation de systèmes sécurisés
    - Le système de fichiers est en mémoire RAM et/ou sur le CD-ROM, sans modification possible



- **setup()**
  - `arch/i386/boot/setup.S`
  - réinitialise les périphériques à la sauce Linux
    - mémoire, carte vidéo, souris, APM/ACPI, ...
  - configuration des interruptions et segmentation
    - Interrupt Descriptor Table (IDT)
    - Global Descriptor Table (GDT)
  - passage du processeur du mode réel au mode protégé
    - adressage de 20 bits (1Mo) à 32 bits (4Go)
  - flux d'exécution transféré à `startup_32()`

- **startup\_32()**
  - décompression du noyau
    - `arch/i386/boot/compressed/head.S`
    - Uncompressing Linux.....
  - positionnement de l'image décompressée en mémoire
    - 2ème fonction `startup_32()` en `0x00100000`
  - saut à `startup_32()` (`arch/i386/kernel/head.S`)
    - début d'exécution du vrai noyau
  - initialisation des registres de segmentation et de l'IDT
    - segments partagés pour les "processus" d'un même niveau
    - handler spécifique : `ignore_int()`
  - configuration de la pile en mode noyau
    - création d'un espace mémoire fonctionnel et utilisable
  - flux d'exécution transféré à `start_kernel()` (`init/main.c`)

# Initialisation du noyau

---

- **start\_kernel()**
  - processus 0 (Adam) (init/main.c)
  - thread noyau
    - exécute une seule fonction
    - travaille uniquement en kernel space
  - affiche la bannière (/proc/version)
  - parse les options du noyau
  - initialise toutes les structures de données du noyau
    - traps, (soft)IRQs, time, console, mémoire, multiples caches
  - initialise le thread noyau init()
    - kernel\_thread(init, NULL, CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND);
  - se met en attente (arch/i386/kernel/process.c:cpu\_idle())

- **init()**
  - `init/main.c`
  - initialise d'autres "composants"
    - `mtrr`, `apm/acpi`, `pci`, `isapnp`, `socket`
  - démarre d'autres kernel threads
    - gestion des interruptions (`[keventd]`, `[ksoftirq_CPUX]`)
    - gestion du swap (`[kswapd]`)
    - gestion du cache disque (`[bdflush]`, `[kupdate]`)
    - autres (`[kapmd]`/`[kacpid]`, `[kjournald]`, `[khubd]`, ...)
  - prépare le root FS (`init/do_mount.c:prepare_namespace()`)
  - libère la mémoire noyau inutilisée (`arch/i386/mm/init.c:free_initmem()`)
  - exécute le processus utilisateur `init()`
    - d'où le `init` et non `[init]`
    - appelle `init=arg`, `/sbin/init`, `/etc/init`, `/bin/init`, `/bin/sh`

- **Accéder à l'espace noyau**
  - exploitation d'une faille de programmation
    - ptrace, mremap, ...
  - module : fonctionnalité permettant de charger du code en espace noyau
    - utilisé pour des drivers ... et des root-kits
  - devices spécifiques représentant la mémoire du noyau
    - /proc/kcore, /dev/mem, /dev/kmem

# Le démarrage en espace utilisateur



- **init**
  - premier processus, père de tous les autres
  - crée les autres processus à partir d'un fichier de conf.
    - méthode : fork() + exec()
  - runlevel : ensemble de commandes à exécuter
    - 0 : halt - 1 : single user - 2 : multiuser sans NFS
    - 3 : multiuser - 4 : inutilisé - 5 : X11 - 6 : reboot
  - pour chaque terminal autorisé (tty), getty lance la commande login
    - configuration de la vitesse, des caractères de commande, ...
  - single mode pour réparation
    - shell fourni par la commande /sbin/sulogin
- **Ensuite, 2 comportements possibles**
  - System V
  - BSD

# Le démarrage en espace utilisateur

---

- **Pour les SYSV**

- Fichier de configuration : `/etc/inittab`
- Ce fichier de configuration définit des niveaux d'exécution (run-levels)
- Les niveaux d'exécution sont :
  - 0 : Pour arrêter le système
  - 1 ou s : Pour passer en mode single user
  - 2 : Mode multi-utilisateurs (exemple : sans réseau)
  - 3 : Mode multi-utilisateurs (exemple : avec réseau)
  - 4 : Mode multi-utilisateurs (exemple : avec NIS)
  - 5 : Mode multi-utilisateurs (exemple : avec X11)
  - 6 : Pour rebooter le système
- Le fichier inittab définit le run-level par défaut à utiliser au boot
- Le fichier inittab a une syntaxe assez simple : pour un run-level, on définit les scripts à exécuter

# Le démarrage en espace utilisateur

---

- **Pour les SYSV**

- Le plus souvent, inittab lance le même script avec, en paramètre, le numéro du run-level (ou le script rcn)
- Ce script est /etc/init.d/rc
- Ce script va exécuter les scripts présents dans /etc/rcn.d où n est le run-level
- Le répertoire /etc/rcn.d contient deux types de fichiers (liens)
  - Knn\* : scripts d'arrêt (Kill)
  - Snn\* : scripts de démarrage (Start)
- Le nombre nn permet d'ordonner le séquençement de ce qui sera lancé
- Les scripts Knn\* et Snn\* sont des liens (souples ou durs selon les Unix) vers les vrais scripts dans /etc/init.d
- Ceci permet, en supprimant le lien, de ne plus lancer le script pour le run-level donné, mais de ne pas effacer le script (idem pour rajouter un nouveau script, il suffit de créer un lien)
  - Un script Knn\* est lancé par le script rc avec le paramètre stop
  - Pour Snn\*, le script rc utilise le paramètre start



# Le démarrage en espace utilisateur

---

- **Pour les BSD**

- Fichier de configuration : `/etc/ttys`
- `init` exécute le script `/etc/rc`
- Le script `/etc/rc` va exécuter d'autres scripts dans `/etc/rc.d`
- Le script `/etc/rc` et les scripts lancés utilise le fichier `/etc/rc.conf` qui regroupe les variables définissant le comportement des scripts
- Exemple :
  - La variable `smtp_enable` définira le lancement d'un MTA
  - Si cette variable est à YES, le script correspondant sera lancé
  - Si cette variable n'est pas à YES, le script ne sera pas lancé
- Intêrets du boot BSD
  - Le comportement du démarrage est totalement paramétrable grâce au fichier `/etc/rc.conf`
  - Les différents scripts `rc.*` réalisent des actions de base nécessaires au démarrage

# Le démarrage en espace utilisateur

---

- **Pour les BSD**

- Les différentes actions des scripts rc.\* sont :
  - Vérifier les systèmes de fichiers
  - Monter les systèmes de fichiers
  - Lancer les différents démons et services
  - Lancer les bannières de connexion
  - Configurer le réseau
  - Lancer l'interface graphique, ...
- Les systèmes BSD possèdent deux niveaux de fonctionnement :
  - Single user et multi-user
- Le niveau multi-user est le mode de fonctionnement normal tel que défini plus haut
- Le niveau single user est choisi par l'utilisateur lors du boot (option -s)
- Cette option est passée par le noyau à init qui lance un shell au lieu de /etc/rc
- A la sortie de ce shell, init lancera /etc/rc et le système démarrera en mode multi-user

# Les rootkits

# Le problème des rootkits

---



- **Les rootkits remettent en cause la sécurité physique**
  - Ils dissimulent l'activité du pirate
    - Camouflage de fichiers, de sockets, de processus
  - Ils fournissent un moyen d'accès au pirate
    - Aussi appelé « backdoor » ou cheval de Troie
- **Ils modifient le système de façon difficilement détectable**
  - Techniques de camouflage très avancées
  - Nécessité d'adapter les techniques de détection
- **Ils remettent en cause la confiance**
  - dans le système d'exploitation
  - dans les outils de sécurité

# Le problème de la confiance

---

- **Reflections on trusting trust (Thompson 1984)**
  - ajout d'un backdoor dans /bin/login
    - accès root pour tous les systèmes avec ce binaire
  - les sources login.c sont présentes sur le système
    - tout le monde peut remarquer la backdoor dans les sources
    - Thompson remet le login.c propre
  - l'admin peut recompiler login.c => /bin/login "propre"
    - Thompson modifie le compilateur C : s'il compile login.c, ajout de la backdoor
  - les sources du compilateur sont présentes sur le système
    - tout le monde peut remarquer la backdoor dans les sources
    - Thompson remet le compilo.c propre
  - compilateur C écrit ... en C
    - le binaire du compilo reconnaît ses propres sources, et ajoute sa backdoor pour login.c



- **Systeme compromis : l'intrus peut**
  - utiliser les ressources
    - mémoire, disque, bande passante, ...
  - récupérer des données
    - keylogger, locate mp3 avi visa
  - rester invisible sur le système
    - modifier le comportement du système pour se cacher
- **Systeme compromis : l'admin peut**
  - détecter les fichiers/processus modifiés
  - restaurer l'intégrité du système ...
- **Mais peut-on encore faire confiance au système ?**



- **Intrus : modifier les binaires**
  - changer le comportement normal des commandes
    - `ps` pour cacher les processus de l'intrus
    - `netstat` pour cacher les connexions de l'intrus
- **Du côté de l'admin**
  - si une seule commande est oubliée, l'intrus est vu
  - détection fondée sur des fonctions de hashage
    - base de référence des hash sur un support read-only

```
$ md5sum ~/lrk5/ifconfig 086394958255553f6f38684dad97869e
$ md5sum `which ifconfig`f06cf5241da897237245114045368267
```
- **De l'importance de créer une base d'empreintes...**
  - sauf si le prog. de vérification est compromis !!!

- **Intrus : modifier les bibliothèques**

- une bib. dynamique affecte plusieurs programmes
- moins de modifications, plus de discrétion

```
$ ldd `which uptime` `which ps`  
/usr/bin/uptime:  
libproc.so.2.0.7 => /lib/libproc.so.2.0.7 (0x40025000)  
libc.so.6 => /lib/libc.so.6 (0x40032000)  
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)  
/bin/ps:  
libproc.so.2.0.7 => /lib/libproc.so.2.0.7 (0x40025000)  
libc.so.6 => /lib/libc.so.6 (0x40032000)  
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

- **Du côté de l'admin**

- emergency kit contenant des binaires statiques





- **Intrus : modifier le noyau**
  - difficile de patcher tous les binaires et bib. dyn.
  - attaquer la seule ressource partagée : le noyau
  
- **L'espace noyau, un nouvel El Dorado**
  - nouvelle génération de root-kits
  - intrus plus puissant que root/admin
  - contrôle complet sur le user space
  - sniffer avant le firewall
  - ajouter des kernel threads invisibles



- **Autres magouilles**
  - patcher les headers des fonctions
    - `jmp <fake fct>`
  - injection de kernel/user thread
    - héritage des droits du processus
- **Du côté de l'admin**
  - outils de détection : agissent en userland
    - AIDE, Tripwire, chkrootkit, rk hunter, ...
  - nécessité de vérifier l'intégrité en espace noyau
    - vérification d'adresses, d'instructions, ...
  - ou de ruser ...
    - mesure du temps, vérifier `/proc/<pid>`



- **Le rootkit modifie l'OS**
  - Impossible d'éliminer un rootkit en passant par l'OS
  - Boot sur un autre système et nettoyage du disque dur
  - Réinstallation complète du système
- **Si le rootkit est lié à un virus**
  - Principe du virus : infection de l'ensemble des fichiers exécutables du système
  - Implique le nettoyage de tous les fichiers
- **Si le rootkit est lié à un ver**
  - Principe des vers : infection d'un maximum de machines
  - Implique le nettoyage de toutes les machines
- **Le pire des cas : réinstallation complète de l'ensemble du parc de machines (serveurs et stations)**

# Conclusion

# Conclusion

---



- **La sécurité physique est une étape obligatoire**
  - Nécessaire pour garantir la sécurité de l'OS
  - Toute faiblesse dans la sécurité physique annule la sécurité des couches supérieures (maillon faible)
  - Accès physique à un serveur = accès root
- **Le processus de démarrage doit être sécurisé**
  - Démarrage sur un autre périphérique
  - Démarrage sur un autre OS
  - Dans le cas du multi-boot, tous les systèmes doivent garantir une sécurité équivalente
- **Les rootkits représentent un grand challenge**
  - Rootkit = accès physique à la machine
  - Difficiles à détecter
  - Difficiles à éliminer