Sécurité Linux

Frédéric RAYNAL
fred(at)security-labs.org

Damien AUMAITRE
damien(at)security-labs.org

Overview

- Historique
- Unix, qu'est-ce que c'est ?
- Du big bang à la matrice
- Banalités essentielles

Première partie I

Historique

Unix, un peu d'histoire

- **Avant** : les Bell Labs abandonnent MULTICS
- **1969** : UNICS (UNiplexed Information & Computing Service)
  - Ken Thompson (Bell Labs) veut jouer sur son PDP-7
  - Il recrée un OS en assembleur à partir de MULTICS
    - multi-tâches : exécution *simultanée* de plusieurs programmes
    - multi-utilisateurs : plusieurs utilisateurs en même temps
  - Portage de l'assembleur vers le B
- **1972-1974** : Unix TSS
  - Dennis Richie crée le langage C
  - Portage de l'OS du B vers le C

Apparition du premier OS *portable*

Unix, encore un peu d'histoire

- **Naissance des BSD**
  - Ken Thompson installe un Unix Version 6 à Berkeley
  - Bill Joy & Chuck Haley s'amusent avec Unix Version 6
  - 1BSD puis 2BSD sur PDP-11 (financements DARPA)
- **Jusqu'en 1984**
  - Développement jusqu'à Unix Version 7
  - Vente aux universités à un prix modique
- **1984 : démantèlement d'AT&T**
  - Premier Unix commercial : System III, puis System V
  - AT&T vend Unix à Novell en 1993
  - Novell vend Unix à SCO (Santa Cruz Operation) en 1995

Unix, suite de l'histoire

BSD

- Ajout de nombreuses fonctionnalités
  - Mémoire virtuelle & pagination, noms de fichier longs (¿14 char), nouveau FS, gestion des signaux, réseau (TCP/IP), ...
- Ajout de programmes
  - Éditeur (vi), shell (csh), compilateurs LISP et pascal, ...
- 1991 : fondation de BSDi, premier BSD pour Intel
  - Bill Jolitz quitte rapidement BSDi pour poursuivre 386BSD
- sert de base à Sun, DEC, Free/Net/OpenBSD, ...

Sun MicroSystems (Stanford University Network)

- 1980 : Bill Joy est co-fondateur de Sun
- 1983 : première version de SunOS (NFS inclus)

Unix, fin de l'histoire (les vainqueurs)

Linux (Linus Torvald)

- 1984 : R. Stallman crée le projet GNU

  GNU is not Unix

- 1985 : R. Stallman crée la FSF
- 1991 : premier noyau Linux, compromis entre MINIX et Unix commercial
- Complété depuis par de nombreux programmes GNU

Les BSDs

- Système complet (noyau + programmes)
- Fondés à partir de 4.3BSD/Net2, 386BSD, 4.4BSD-Lite
- 1993 : FreeBSD (efficacité) & NetBSD (portabilité)
- 1996 : OpenBSD (sécurité)

Histoires d'Unix : résumé

- 1965 MULTICS
- 1969 UNICS
- 1972-1976 Unix TSS v1 ... v6
- 1977 Unix BSD
- 1979 Unix TSS v7
- 1983 Unix System V
- 1984 SunOS, Projet GNU
- 1985 MINIX, FSF
- 1986 HP-UX
- 1990 IBM AIX
- 1991 noyau Linux, système 386BSD
- 1992 procès perdu d'Unix System Labs (AT&T) vs. BSDi
- 1993 NetBSD, FreeBSD
- 1996 OpenBSD

Les 2 principales souches : System V et BSD

Des différences à tous les niveaux

- Utilisateur : commandes identiques, options différentes
  - ps -ef (SysV) / ps -aux (BSD)
- Programmeur : fournit des API différentes
  - IPC (SysV) / socket réseau (BSD)
- Système : comportement variable
  - Gestion des signaux : SysV reset le handler, BSD non
- Administration : initialisation & commandes différentes
  - init : /etc/inittab (SysV) / /etc/ttys (BSD)
  - /etc/rc.* (dir / SysV) / /etc/rc* (fichiers / BSD)

Unix System V

Historique

- Dernière version en 1990 : SVR4
- Plate-formes : Intel x86 et SPARC (Solaris 2 / SunOS 5)
- Marque la fin de la relation Sun / AT&T
- Qui : Solaris, IBM AIX, SCO OpenServer

Spécificités

- Origine BSD : pile TCP/IP, socket UFS, csh
- Origine SunOS : VFS, NFS, mémoire virtuelle (+ `mmap`, RPC
- Divers : ksh, support POSIX, Aync I/O, modules noyau, SMP

Berkeley Software Distribution (BSD)

Historique

- Premières versions dans les années 70 à base du code d'AT&T
- BSD Net/1 (1989) : recodage de toute la pile réseau pour éviter de payer une licence très chère à AT&T
- BSD Net/2 (1991) : recodage de **tous** les utilitaires contenant du code AT&T (seul le noyau a encore du code sous licence)
- BSD386 (BSDi), fondé sur Net/2, est poursuivi en 1992 par les Unix System Laboratories d'AT&T
  - Ralentissement du développement de 386BSD (libre) par Bill Jolitz
  - Apparition du noyau Linux de Linus Torvalds
- Aboutissement du procès en 1994 : retrait de 3 fichiers, modifications de 70, le tout sur plus de 70000 fichiers
- Création de {Free, Net, Open}BSD, emprunt de code dans Mac OS X et Solaris

Les bâtards : Linux et QNX

GNU/Linux

- Un noyau tout seul
- Des programmes fournis par la FSF
- Développé à partir de rien par des volontaires

QNX

- Système commercial temps-réel complet
- Source du boyau temps réel disponible à des fins non commerciales
- Micro-noyau spécialisé pour les logiciels embarqués
  - Même Cisco le met dans son nouveau IOS-XR

POSIX (Portable Operating System Interface)

Principe de la norme

- Intersection entre les différentes branches d'Unix
  - Permet de garantir l'interopérabilité des programmes POSIX
- Définit les fonctions nécessaires à un Unix
  - Ne précise pas l'implémentation (syscall vs. bibliothèque)

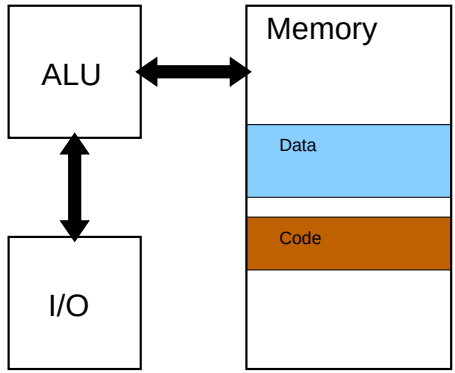
Des fonctions POSIX

- Gestion des processus : `fork()`, `waitpid`, `execve()`, `exit()`
- Gestion des fichiers : `open()`, `close()`, `read()`, `write()`
- FS : `mkdir()`, `unlink()`, `mount()`, `rmdir()`
- Divers : `chdir()`, `chmod()`, `kill()`, `time()`

Architecture de Von Neumann

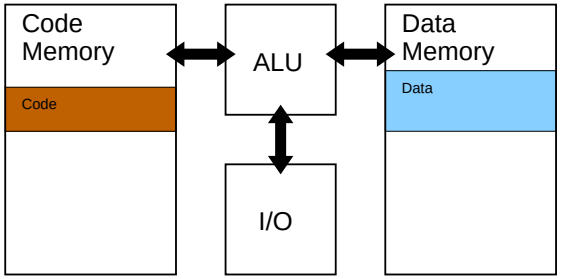
- Premier modèle d'ordinateur
- **Programmable**, les instructions n'ont plus besoin d'être *hardcodées* dans l'ordinateur
- Les instructions et les données partagent la même mémoire

$\Longrightarrow$ Injection de code, redirection de flux d'exécution



Architecture Harvard

- D'autres modèles existent
- Des erreurs dans la manipulation des données peuvent influencer le comportement du programme
- Injection de code impossible
- Ex. : utilisée sur certains Cray

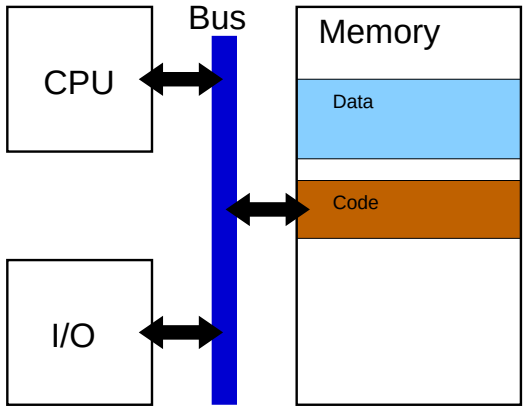


Architecture Von Neuman à base de bus

- Conception actuelle et moderne . . . pensée dans les années 40
- Les I/O peuvent accéder à la mémoire sans passer par le CPU

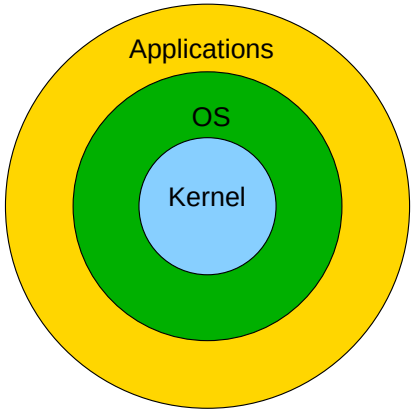
$\Longrightarrow$ Préservation des cycles CPU

$\Longrightarrow$ Opportunités pour du *hardware* malicieux (FireWire, USB . . .)

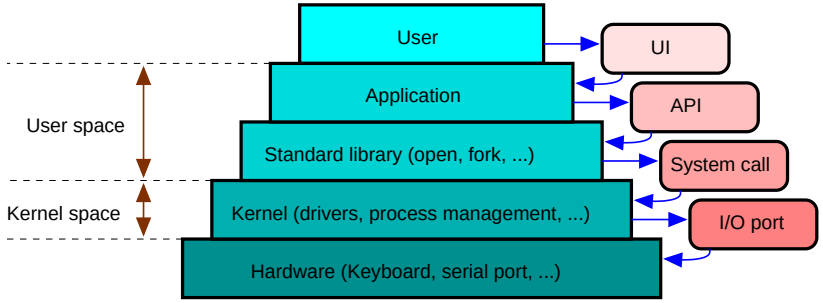


Couches logicielles

- Noyau (*kernel*)
  - Abstraction pour le matériel
  - Ordonne les tâches à exécuter
- Système d'exploitation (*Operating System*)
  - Exécute des services
  - Fournit des bibliothèques standards
  - Fournit les outils pour administrer le système
- Applications
  - Effectuent le travail . . .



Pile logicielle



Deuxième partie II

Unix, qu'est-ce que c'est ?

Le secret d'Unix

3 briques de base très simples mais néanmoins générales

L'intégralité d'Unix repose sur 3 notions :

- Processus : unité élémentaire de gestion des traitements
- Fichier : unité élémentaire de gestion des ressources
- IPC (*Inter Process Communication*) : arbitre pour ressources partagées

Les processus

De l'algorithme au processus

- Algorithme = instructions + données
- Programme = réalisation d'un algorithme
- Processus = exécution d'un programme

Quelques difficultés

- Comment gérer l'accès aux instructions et aux données ?
- Comment choisir quel processus exécuter à un instant donné ?
- Comment partager les ressources du système ?

Commandes sur les processus

Lister et manipuler les processus

- `ps` (process status) : liste les processus
  - Nombreuses options pour voir des paramètres
- `top` : liste tous les processus en temps réel
  - Pratique pour voir les ressources consommées
- `kill` : envoie d'un signal à un processus
  - HUP (hang up/1), QUIT (3), KILL (9), TERM (15), ...
- `strace` : surpervise les appels systèmes d'un processus
  - Debug quand ça plante, surveillance d'un user
- `ltrace` : supervise les appels de fcts d'un processus
  - Quand ça plante

Flot d'exécution (1/2)

Instructions : gérées par un CPU

- Thread : flot d'instructions d'un programme
- Multi-tâche : exécution parallèle de plusieurs threads
- Scheduler : décide du thread à exécuter

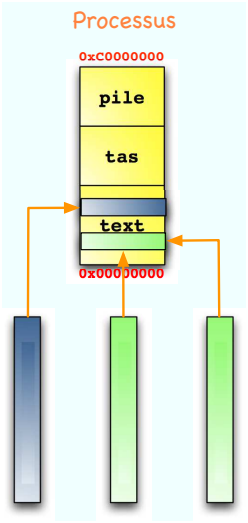
Données : placées en mémoire

- Adressage logique : espace d'adressage identique pour tous les threads, indépendant de la mémoire physique
- Swap : mémoire (lente) placée sur le disque
- Mémoire virtuelle : mécanismes transparents de gestion de la mémoire

Flot d'exécution (2/2)

Processus

- Ensemble de threads + espace d'adressage
- Tout processus est créé par un autre processus
  - Appel système `fork()`
- Tout processus est identifié par un numéro unique
  - PID = Process ID
- Tout processus appartient à un utilisateur et un groupe
  - UID = User ID
  - GID = Group ID

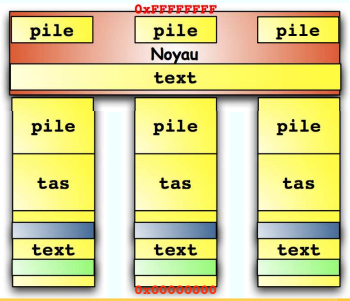


De la mémoire physique à la mémoire virtuelle

Agencement de la mémoire

- Mémoire physique transformée en mémoire virtuelle via pagination (adr. linéaire) puis segmentation (adr. logique)
- Vue logique (adresses de 0x00000000 à 0xFFFFFFFF) indépendante de la réalité physique (RAM disponible)

$\Longrightarrow$ Accès à la mémoire indépendant de la RAM



Les fichiers

Représentation des données

- Tout est fichier (données sur disque, périphériques, ...)
- Non typés : format des données inconnu de l'OS
- Système de fichiers : décrit les "propriétés" des fichiers

Abstraction unique : tout ce qui ne concerne pas l'exécution est manipulé sous forme de fichiers

Opérations sur les fichiers (1/2)

Opérations élémentaires

- interface de programmation unique pour TOUS les fichiers
  - périphériques, fichiers, sockets
- existence/implémantation variable selon le type de fichier

Ouverture	`open()`
Fermeture	`close()`
Lecture	`read()`
Écriture	`write()`
Déplacement de la tête de lecture	`seek()`
Attente d'un évt. externe	`select()`
Informations	`stat()`

Opérations sur les fichiers (2/2)

Modes d'accès

- Mode bloc : accès par blocs de données
  - Existence de la fonction `seek()` : déplacement libre dans le fichier
  - Ex. : fichier ou disque dur
- Mode caractère : accès séquentiel aux données
  - absence de `seek()`
  - Ex. : socket réseau ou imprimante

Ajout d'opérations spécifiques

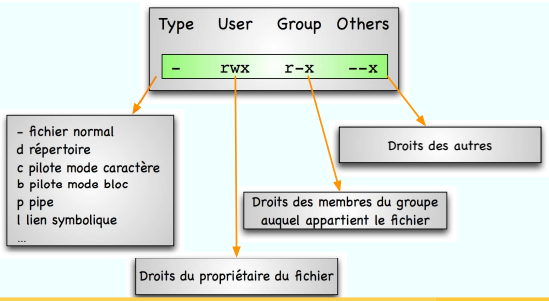
- Sockets : `accept()`, `bind()`, ...
- Répertoires : `chdir()`, `readdir()`, ...
- Drivers : `ioctl()`

Caractéristiques des fichiers

Propriétés

- Tout fichier appartient à un utilisateur et un groupe
- 3 permissions : r=read w=write x=execute
- 3 niveaux de protection existent : UGO

```
>> ls -l /etc/passwd
-rw-r--r-- 1 root wheel 1374 Sep 12 2003 /etc/passwd
```

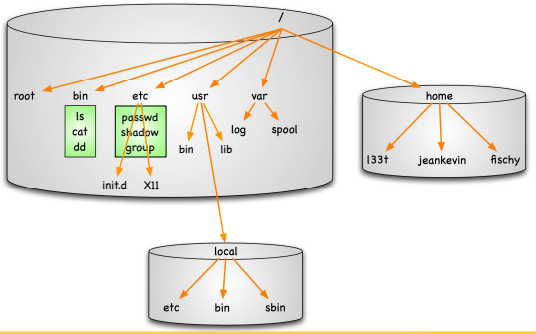


Organisation physique / logique

Fichiers réguliers

- Identifiés par un chemin unique
- Vue logique (chemin) indépendante de la réalité physique (partitions, clusters, etc.)

$\Longrightarrow$ Chemin indépendant des partitions



Notions étendues

Cas spéciaux

- Devices (`/dev`) : caractérisés par un couple (majeur/mineur) pour identifier le pilote associé
- Sockets : n'apparaissent pas dans l'espace de nommage

Systèmes de fichiers

- Espace de nommage unique, mais composé de plusieurs sous-arborescences
- Une arborescence correspond à une partition montée système de fichiers : organisation des données sur un support
  - Ex. : ext2, ext3, UFS, FAT, NTFS, ...
  - Peut être intégralement virtuel : `/proc`

Commandes sur les fichiers

Créer, s'informer, déplacer ou effacer

- `touch` : crée un fichier vide ou modifier les dates
- `stat` : donne toutes les informations sur le fichier
- `ls` : liste les fichiers (cf. les options)
- `cp` : copie une source vers une destination
- `mv` : déplace d'un endroit à un autre
- `rm` : efface un fichier

Fichiers et processus

- `lsof` : liste les fichiers ouverts par un processus
- `fuser` : liste les processus qui utilisent un fichier

Du binaire au processus (1/2)

Exécution d'un programme normal

- Fichier sur le disque, avec un UID/GID

```
bash>> /bin/ls /sbin/root_world_in_30s
-rwxr-xr-x     1 root    wheel    88096 Jun 30 01:00 root_world_in_30s
```

- Commande lancée depuis un shell

```
bash>> /sbin/root_world_in_30s
```

- Le processus shell est fork()é
- Le processus fils est remplacé par le programme désiré

```
execve("/sbin/root_world_in_30s");
```

- Le processus voulu tourne en mémoire

```
>> id
uid=501(raynal) gid=501(raynal) groups=501(raynal), 80(admin)
>> ps aux | grep root_world
raynal 2704 0.0 0.0 1860 302 p1 S+ 1:22AM 0:00.01 root_world_in_30s
```


Du binaire au processus (2/2)

Exécution d'un programme *Set-UID*

- Fichier sur le disque, avec un UID/GID

```
bash>> /bin/ls /sbin/ping
-r-sr-xr-x 1 root wheel 24876 May 27 21:20 ping
```

- Commande lancée depuis un shell

```
bash>> /sbin/ping 127.0.0.1
```

- Le processus shell est fork()é
- Le processus fils est remplacé par le programme désiré

```
execve("/sbin/ping 127.0.0.1");
```

- L'UID du processus fils devient celle du proprio (root)
- Le processus voulu tourne en mémoire

```
>> id
uid=501(raynal) gid=501(raynal) groups=501(raynal), 80(admin)
>> ps aux | grep ping
root 2704 0.0 0.0 18260 304 p1 S+ 1:24AM 0:00.01 /sbin/ping 127.0.0.1
```


Tout Unix en 1 slide

- Système portable, multi-utilisateurs et multi-tâches
- Partages des ressources (mémoire, disque, ...)
- Exécution simultanée de plusieurs programmes par un utilisateu
- Primitives simples pour construire des applications complexes
- Redirection des entrées/sorties

Troisième partie III

Il était une fois ... le retour de la vengeance

Histoire de boot

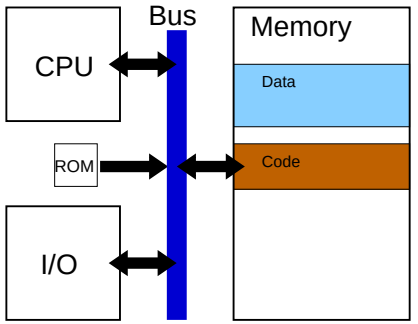
Le boot, une pièce en 5 actes

- 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, ...

Acteurs de Hard

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



Sécurité des acteurs de Hard

Risques

- Accès physique à un ordinateur
  - Vol, destruction, ajout de "mouchards", ajout de matériel
- Backdoor matérielle
  - Générateur aléatoire biaisé, keylogger
- signaux électro-magnétiques compromettants
  - Effet Tempest

Basic Input/Output System

Déroulement du BIOS

- POST (Power On Self Test) : init. composants essentiels
  - processeur(s), mémoire, bus, ...
- Initialisation extensions
  - Carte vidéo, interface IDE, ...
- Recherche le loader sur périphériques de stockage
  - Disque dur, disquette, clé USB, ...
- Chargement du boot loader en mémoire
  - boot loader limité à 512 octets
- Flux d'exécution transféré au boot loader

Sécurité du BIOS

Modifications des paramètres

- Booter sur un "autre" périphérique
  - 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)

Boot Loader (1/3)

Objectif

- Transférer des données stockées sur un support physique en mémoire vive
- Passer le contrôle à ces "données"

Exemples de boot loaders

- Disquette : secteur de boot
  - Copie l'OS de la disquette vers la mémoire
- Disque dur : MBR (Master Boot Record)
  - Charge le 1er secteur de la partition activée (partition loader) qui se comporte comme un secteur de boot
  - Joue directement le rôle d'un secteur de boot

Boot Loader (2/3)

Déroulement

- Demande quoi charger à l'utilisateur
- Chargement du noyau et des données optionnelle
  - initrd, paramètres spécifiques
- Initialisation d'un environnement sain d'exécution
  - Passage du mode réel au mode protégé
- Flux d'exécution transféré au noyau

Problèmes

- Accès au support physique
  - Direct via le BIOS
- Lecture des données du support physique
  - Suppose la capacité de "décoder" le File System (FS)

Boot Loader (3/3)

Taxonomie

- *Spécialisé* : supporte un seul type de périphériques
  - Ex : secteur de boot de la disquette
- *Userland* : nouveau noyau chargé à partir de l'OS natif
  - Attention à ne pas écraser en mémoire le noyau courant
- *FS aware* : "mini-noyau" avec ses drivers pour différents FS et matériels
  - GRUB : GRand Unified Bootloader
    - Stage 1 : recherche des secteurs contenant le noyau
    - Stage 2 : chargement du noyau
- *Indépendant du FS* : utilise des ressources externes pour connaître l'organisation des données
  - LILO : LInux LOader
    - Ne sait lire que le format binaire du noyau Linux
    - Va y chercher le driver du FS

Sécurité du boot loader (1/2)

Injection de paramètres

- Démarrage en "mode sans échec" (single mode)
  - `LILO : linux single`
- Surcharge du root device : boot le noyau natif, mais démarre le système de son choix
  - `LILO : linux root=/dev/cdrom`
- Surcharge du processus init : remplace le premier processus par celui de son choix
  - `LILO : linux init=/bin/sh`

Sécurité du boot loader (2/2)

Empêcher le passage de paramètres

- Réduire le délai d'attente à 0 : marche pas :-(
  - LILO/lilo.conf : ajouter `delay=0`, retirer `prompt/timeout`
  - grub/menu.lst : `timeout 0`
- Demande de mdp pour autoriser les paramètres LILO/lilo.conf : ajouter restricted

Protéger le système par un mot de passe

- Soumettre les paramètres à un mdp
  - LILO/lilo.conf : directive `password`
  - grub/menu.lst : `password --md5 <hash>`
- Protéger les fichiers de configuration

Noyau : loading Linux ... (1/4)

Vers le Moyen-Âge : 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()`

Noyau : chargement (2/4)

Vers la Renaissance : les `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`)

Noyau : Linux version ... (3/4)

Vers les Temps Modernes : `start_kernel()`

- Processus 0 (Adam) (`init/main.c`), thread noyau
  - Exécute une seule fonction
  - Travaille uniquement en espace noyau
- 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()` (Ève)
  - `kernel_thread(init, NULL, CLONE_FS|CLONE_FILES|CLONE_SIGHAND)` ;
- se met en attente (`arch/i386/kernel/process.c :cpu_idle()`)

Noyau : `init()` (4/4)

Vers l'Ère Contemporaine : `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`

Sécurité du noyau

Accéder à l'espace noyau

- Exploitation d'une faille de programmation
  - Drivers, `ptrace`, `mremap`, ...
- Modules : 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`

`user : init`

Description

- 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`

init selon les modes

Déroulement à la System V (linux, Irix, ...)

- configuration dans `/etc/inittab`
- Exécute `/etc/rc.sysinit` : dernières initialisations
  - Port série, horloge, usb, firewire, `/proc`, ...
- Démarre les services associés au runlevel choisi
  - Scripts contenus dans `/etc/rc+.d`
- Ouverture et configuration des terminaux
  - `3 :23 :respawn :/sbin/getty 38400 tty3`
- Exécution de `/bin/login` sur chaque terminal
  - Attend qu'un utilisateur rentre son nom/mot de passe

Déroulement à la BSD (SunOS, *BSD, ...)

- Configuration générale dans `/etc/rc`

Entrez votre mot de passe

Login

- Utilisateur rentre nom/mot de passe
- Calculs à base du mot de passe
- Vérification dans les fichiers associés
  - `etc/passwd` et `/etc/shadow` (SysV) ou `/etc/master.passwd` (BSD)
- `fork()+exec()` d'un shell si les mdp correspondent

Calcul du mot de passe (1/2)

La solution du passé : DES

- Le mdp est tronqué à 8 caractères
- Les 7 premiers bits de ces caractères sont concaténés
  - Création d'une clé de 56 bits
- Une chaîne constante est chiffrée 25 fois par DES
  - En général, il s'agit d'une chaîne de 64 bits à 0
  - Graine de 12 bits pour introduire du désordre dans DES
- Empreinte = chaîne de 13 caractères (76 bits)
  - Les 13 caractères parmi les 64 (2^6) "`./0-9A-Za-z`"
  - 2 premiers = graines (salt - 12 bits) : pour éviter que des utilisateurs avec le même mdp n'aient la même empreinte
  - 11 caractères (64 bits / 6 = 10.6667)

Calcul du mot de passe (2/2)

MD5

- Graine de 6 à 48 bits
- Mot de passe non limité en taille
- Empreinte de 27 à 34 caractères (1000 appels à MD5)
  - `$1$<1 à 8 octets de graine>$<hashé>`

Complexité maximale de DES et MD5 non configurable ⇒ trop sensibles à l'accroissement de puissance des machines

Blowfish

- *n* appels à Blowfish pour chiffrer "OrpheanBeholderScryDoubt"
  - *n* de 2^4 à 2^{31}, 64 par défaut
- Mot de passe limité à 55 caractères, graine de 128 bits
- Empreinte de 60 caractères

Les utilisateurs

Identification

- Un utilisateur est un numéro
  - UID = User ID
- Tout programme lancé par un utilisateur devient un processus tournant sous l'identité de cet utilisateur
  - Sauf pour les programmes Set-UID / Set-GID
- Tout fichier créé par un utilisateur lui appartient
  - Il ne peut pas en changer le propriétaire
  - Il peut changer les permissions associées DAC)

root (UID=0)

- Compte super-utilisateur disposant de tous les pouvoirs
  - Création/suppression d'utilisateur, de processus, de fichiers
- seulement protégé par un (bon) MdP et les mécanismes internes du système
  - Contrôles des permissions, cloisonnement

Quatrième partie IV

Sécurité Unix, question de bon sens

Introduction : pourquoi ?

Linux devient de plus en plus mature

- Bcp de distributions avec différentes philosophies
  - redhat/mandrake/fedora, slackware, debian, gentoo, ...

Linux devient de plus en plus *user friendly*

- Installation, utilisation et administration plus faciles
  - Détection automatique du matériel, bcp d'interfaces, ...

Mais la sécurité n'est pas souvent la priorité . . .

Introduction : processus

La sécurité est un processus, elle dépend du temps

Manipulations régulières

- 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

Roadmap

Objectif

⇒ construire un Linux 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

C'est mon choix

(Trop de) Linux disponibles

- Orientés RPM : RedHat, Mandriva, Suse, Fedora
  - Pour : supportés par des entreprises
  - Contre : rpm, patches "maisons" non standards
- Debian, Ubuntu
  - Pour : très stable, documentation complète, apt
  - Contre : soit un peu vieillot, soit trop tourné vers l'utilisateur
- Slackware
  - Pour : fichiers de conf. non modifiés, up-to-date
  - Contre : gestion des dépendances (?), 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 (knopix, owl, . . .)

Signer ici svp

Installation & mise à jour

- Récupérer toutes les mises à jour disponibles
- Installer la machine hors réseau
  - Présence de failles tant que l'OS n'est pas à jour

Intégrité

- À vérifier après chaque téléchargement, sur chaque CD
- Ne fournit aucune garantie quant à l'origine
  - Recalcule des MD5 par le pirate

Signatures numériques

- Garantie l'origine, l'intégrité, etc...
  - un patch ne garantit pas la résolution des problèmes ;-)

Les partitions

Ne pas multiplier les OS sur une même machine

a /var	a nodev, nosuid, noexec	a • Spoolers (mails, impressions), logs, ... • Prévoir des sous-répertoires/partitions pour certains serveurs
/usr	ro	Binaires (prog/lib), documentations, source (noyau)
/tmp	nodev, nosuid, noexec	Fichiers temporaires, pipes, sockets, . . .
/home	nodev, nosuid	Maison des utilisateurs (espace disque contrôlable à l'aide de quotas
/boot	ro	Emplacement du noyau, sous forme binaire, utilisé pendant le boot
/usr/local	ro	Binaires recompilés par l'admin (optionnel)
/		racine, contient entre autre le répertoire de configuration /etc, et tous ceux absents des partitions précédentes
chroot	nodev, nosuid	Partitions dédiées pour chaque service (en général sous /var ou /home)

Minimisation (1/2)

Moins on installe de choses, moins on est vulnérable !

Installation

- Se restreindre uniquement à ce dont on a besoin
  - Xorg/XFree86 ou gcc sur un serveur ?
- Contrôler les serveurs installés/démarrés
  - pcmcia sur ordinateur de bureau ...
  - `portmap` sans aucun service RPC, ...
  - Lister les serveurs démarrés : `chkconfig – ls` !
  - Services démarrés par wrapper : `(x)inetd`

Minimisation (2/2)

Premières tâches de l'admin

- Gestion des packages : retirer ce qui est en trop
  - lister les packages installés : `rpm -qai` ou `dpkg -l`
  - retirer un package : `rpm -e` ou `dpkg -r`
- Gestion des services : stopper (voire désinstaller) les services inutilisés
  - Combiner `netstat -ntaul` et `ls /etc/rc+.d`
- Gestion des utilisateurs : détruire/désactiver les comptes inutilisés
  - `userdel / deluser` ou `!!` dans le champ mdp de `/etc/shadow`

Installation des patches

Vulnérabilités et patches

- S'inscrire aux listes de diffusion concernant son OS
- À l'annonce d'une vuln., surveiller l'activité associée
  - Faille apache ⇒ monitorer le trafic vers port 80/443
- Vérifier les signatures
- En production, tester les conséquences du patch
  - Je l'ai installé . . . et y'a pu rien qui marche :-(
- Surveiller la sortie des updates
  - Outils "haut niveau" : `urpmi -- apt-get`

Durcissement

Sécurité multi-niveaux

- Espace utilisateur : scripts restrictifs
  - bastille-linux : supporte la plupart des distributions, Mac OS X, HP-UX, . . .
- Espace noyau : patches
  - GRSecurity : durcissement + ACLs
  - PaX : protections des zones mémoire
  - SELinux : contrôle d'accès très fin

Surveiller

Types de surveillances

- Comptabilité (accounting) : enregistrement des ressources utilisées par un processus
- Journalisation : enregistrement des événements du système (noyau, démons, ...)

Audit

- Analyser le comportement du système
  - Ajouter des sources (tripwire/AIDE, chkrootkit, ...)
- À réaliser dès la fin de l'installation
  - comprendre le comportement "normal" du système

Accounting

Les outils

- Support au niveau noyau
  - `CONFIG_BSD_PROCESS_ACCT`
- Sauvegarde des informations dans un fichier
  - Date de création, user, commande, mémoire, ...
  - restreindre les droits d'accès au journal
- outils *userland*
  - `accton` : active/désactive l'accounting
  - `lastcomm` : informations sur une commande passée
  - `sa` : résumé de l'accounting

Logging (1/2)

Outils

- Deux services de journalisation
  - `klogd` : espace noyau (pas de configuration)
  - `syslogd` : appel système en espace utilisateur
- Centralisation des logs "userland" via un démon
  - Configuration dans `/etc/syslog.conf`
- Double classification
  - par niveau de priorité : `emerg, alert, err, warn, info, ...`
  - par "facility" : `auth, cron, daemon, user, mail, lpr, ...`
- Possibilité d'exporter vers une machine distante
- Gestions des logs
  - Intégrité : les entrées ne sont pas modifiées (signature)
  - Disponibilité : historique et sauvegarde des journaux
  - Confidentialité : besoin d'en savoir (mdp dans les logs)

Logging (2/2)

Serveur de logs distant (RFC 3164)

- Clients
  - `/etc/syslog.conf : *.* @ip_serveur`
- Relais de logs
  - Autoriser les connexions vers 514/UDP
  - Option de configuration : ??? (BSD), -h (linux)
- Serveur : collecteur de logs
  - Autoriser les connexions vers 514/UDP
  - option de configuration : `-u` (BSD), `-r` (linux)

Est-ce réellement "secure" ?

- duplication des logs ⇒ plusieurs machines à rooter
- Synchronisation des horloges
- Aucune authentification ⇒ spoof des logs

Surveillance active

Commandes & fichiers

- `/var/log/lastlog` : contient l'heure du dernier login
- `/var/log/utmp` : conserve les informations sur les utilisateurs *présents* sur le système
  - Commandes : `who, w, users`
- `/var/log/wtmp` : conserve les informations sur les utilisateurs *passés* sur le système
  - Commandes : `last, ac`

Touches finales

Empreinte générale du système

- Construire une empreinte du système
  - Fera office de référence en cas de compromission
  - AIDE, Tripwire : calculs de hash sur les fichiers importants

Snapshot

- Prévoir une sauvegarde des "parties sensibles"
- Plus facile pour la reproduction
  - Restauration pour clonage ou post-intrusion

Quelques outils pratiques

Gestion des logs

- syslog-ng : permet l'export des logs chiffrés, authentifiés, et signés !
- logwatch, lire : outils d'analyse de logs

Synchronisation / archivage

- unison : permet de synchroniser 2 répertoires distants
- duplicity : chiffre la sauvegarde avec gpg, l'envoie sur le serveur de backup en ssh

Mon meilleur ennemi

L'utilisateur

- Définit par un login, un UID et un GID
  - Possible d'avoir 2 logins différents avec un même UID
  - Possède un groupe principal et des groupes secondaire

    ```
    >> groups raynal
    raynal, users, wheel
    ```

  - Login == confort, le système ne travaille qu'avec l'UID
- *Discretionnary Access Control* (DAC) : chaque utilisateur est responsable de ses fichiers
  - Contrôle individuel des droits UGO
- UID = 0 : compte privilégié, généralement appelé root
  - Possède presque tous les droits sur le système
  - Possible (recommander) de le désactiver et d'utiliser `sudo` à la place

Fichiers et inodes

Qu'est-ce que c'est

- inode : structure de données propre au système de fichiers qui contient les méta-informations sur le fichier
  - Permissions, attributs, taille, ...
- Certains éléments sont imposé par POSIX pour qualifier un système de fichiers compatible Unix
  - ReiserFS n'a pas d'inodes, mais fournit autrement le même type de service
- Chaque fichier possède exactement un inode
- Inclue également une liste de blocs sur le disque dur, blocs contenant les données du fichier
- Sert surtout pour le noyau à gérer les fichiers
  - Abstraction commune à bcp de systèmes de fichiers (VFS)

Fichiers et permissions (1/2)

```
-rw-r--r-- 1 root wheel 1374 2006-02-23 18:12 /etc/passwd
```

Caractéristiques d'un fichier

- `-rw-r--r--` : les permissions (UGO)
  - set User—Group ID bit (s) : change l'ID du processus
  - *sticky bit (t)* : fichiers manipulables par root ou propriétaire

    `drwxrwxrw` **`t`** `12 root sys 4096 2004-06-28 18:01 tmp`
- 1 : nb de "hard links" vers (fichier), nb de fichiers (dir)
- `root wheel` : UID/GID
- 1374 : taille du fichier
- 2004-02-23 18 :12 : date de la dernière modification
  - Access (dernière lecture) : `read()`, `mmap()`, `execve()`, ...
  - Modify (dernière écriture) : `write()`, `truncate()`, `mknod()`, ...
  - Change (chgt niveau inode) : `chown()`, `link()`, ...
- `/etc/passwd` : nom du fichier

Fichiers et permissions (2/2)

Modification des droits

- `chown` : changement du propriétaire
  - Restreint aux fichiers de l'utilisateur (sauf root)
- `chgrp` : changement du groupe
  - Rrestreint aux groupes de l'utilisateur (sauf root)
- `chmod` : changement des permission UGO
  - Restreint aux fichiers de l'utilisateur (sauf root)
- `mknod` : création de fichiers spéciaux (caractère ou bloc)
  - Réservé à root

Fichiers et attributs

Au-delà de l'inode

- Chaque système de fichiers propose ses propres attributs
- Les attributs influencent le comportement du système

ext2/ext3

- `lsattr` : liste les attributs
- `chattr` : change les attributs
- Quelques exemple
  - `A` : *access time*pas mis à jour (gain de performance)
  - `a` : *append only*, le fichier n'est ouvrable qu'en ajout
  - `i` : *immutable*, le fichier ne peut être supprimé, modifié, hard-linké, etc. (seulement pour root)

Identification avec PAM

Pluggable Authentication Module (PAM)

- Historiquement, saisie d'un mdp, puis vérification par rapport à `/etc/passwd`
- Quelques évolutions : s/DES/MD5, `/etc/shadow`, ...
- Problème : à chaque évolution, il faut reprendre de (très) nombreuses applications
- Solution : déléguer & centraliser la gestion de l'identification
  - Les applications doivent être "PAM compliant"

Principe de PAM

Fonctionnement

- Mécanisme flexible d'authentification des utilisateurs
  - Réglages des horaires, ressources, etc.
- Modules sous forme de bibliothèques dynamiques
  - Chargement "à la demande"
- Ordre de chargement & configuration des modules gérés par l'admin
  - Authentification centralisée & indépendante de l'application

Configuration

Fichiers / répertoire

- `/etc/pam.conf` : fichier centralisant toutes les configuration, peu employé
- `/etc/pam.d/` : répertoire contenant un fichier par service
  - Syntaxe : `type control module args`
  - type : partie de l'authentification concernée
  - control : niveau d'exigence de réussite
  - module : la bibliothèque à utiliser
  - args : les paramètres à passer

Les types dans PAM

4 types de modules pour tout gérer

- `auth` : authentification de l'utilisateur (Unix, SMB, ...)
- `account` : gestion des services (horaires, ressources, ...)
- `session` : choses à faire en début/fin de session (logging, montage de répertoire, ...)
- `password` : update de tokens d'auth (challenge/réponse changement de mot de passe, ...)

Les contrôles dans PAM

4 contrôles pour tout gérer

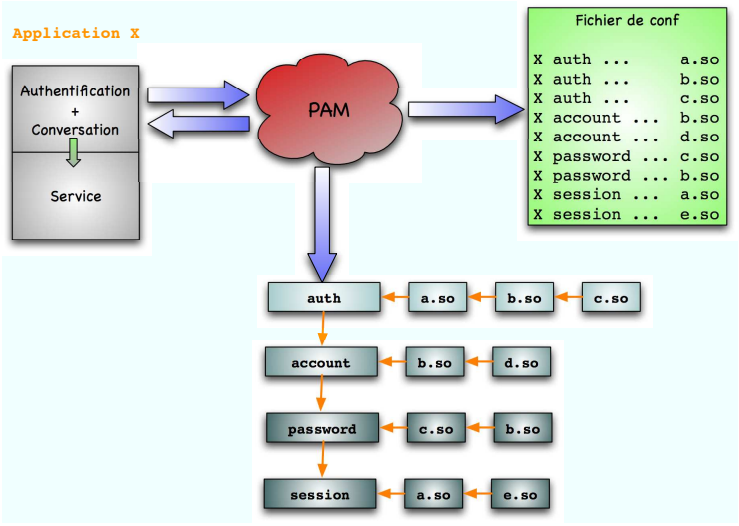
- `requisite` : si échec, on quitte l'appli !
  - L'auth de tous ces modules DOIT réussir
- `required` : si échec, on teste le module suivant
  - L'auth d'au moins un de ces modules doit réussir
- `sufficient` : la réussite de ce module suffit à valider la pile de modules
  - Résultat ignoré si module "required" a échoué avant
- `optional` : résultat considéré seulement s'il n'y a pas d'autre module dans la pile

En concret, ça marche comment ?

Exemple pour `/etc/pam.d/ssh`

```
# accès restreint à root si /etc/nologin présent
auth required pam_nologin.so
# controle et initialise les variables d'env.
auth required pam_env.so
# auth Unix classique
auth required pam_unix.so nullok_secure
# défaut : bloque l'accès aux comptes expirés
account required pam_unix.so
# défaut : quelques logs ...
session required pam_unix.so
# affiche le MotD si auth==succès
session optional pam_motd.so
# "format" du mot de passe
password required pam_unix.so nullok obscure min=4 max=8 md5
```


Ça marche comment ... en dessin



Applications "sensibles"

2 types d'applications "sensibles"

- Celles qui permettent de modifier ses privilèges
  - L'application est une cible
  - Ex : démons, programmes Set-UID root, ...
- Celles qui manipulent des informations "intéressantes"
  - L'application est un vecteur (cible intermédiaire)
  - Ex : contenu d'une bdd, pages web, ...

Configuration générale

Principes de précaution

- Calibrer précisément les privilèges
  - Pas assez : l'appli ne peut tourner correctement
  - Trop : encouragement à attaquer, risque de rebond
- Séparation de privilèges : découper le service en fonction des privilèges requis
  - 1 processus apache en root pour écouter sur le port 80
  - des processus fils tournant sous un autre UID pour servir les pages web

Doit être prévu dès la conception du logiciel

Mesures externes

Utiliser le système pour se protéger

- un compte dédié par application "sensible"
  - Si ce compte est compromis, cloisonnement à ce compte
  - Désactiver le mdp
- `chroot()` : cloisonner l'application
  - Restreindre le contexte d'exécution
  - Pas prévu pour la sécurité ... fuites possibles
  - Une partition dédiée pour chaque service chroot()é

`chroot()` howto

Construire sa prison

- `mkdir` : reconstruire l'arborescence nécessaire
- `ldd` : trouver les bibliothèques dynamiques
- cp : copier les fichiers/programmes utilisés
  - pas de lien en dehors du `chroot()`
- tenter de démarrer le service ... qui va planter
- `strace` : surveiller les erreurs
  - `strace -eopen`
- `mknod` : construire des devices

La pile IP

Paramètres noyau : `sysctl()`

- 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

Filtrage de paquets

Le parefeu sous Linux 2.6

- Firewall stateful
- Hooks à différents points du voyage d'un paquet
  - Hook == fonction de callback
- quand un paquet arrive sur un hook ⇒ évaluation
  - Évaluateur == table IP
- Chaque table constituée de "chaînes"
  - Chaîne : ensemble de règles aboutissant à une décision
- Chaque règle indique :
  - Sa table
  - Sa chaîne
  - Un pattern (motif de reconnaissance pour un paquet)
  - Une cible (target) : la décision

Netfilter

3 tables ... d'où le nom "iptables" ;-)

- filter : filtrage simple par adresse, port, état, ...
- NAT : translation d'adresses
- mangle : modifications des entêtes et champs

2 types de chaînes

- built-in : propres à chaque table
- user chains : chaîne créée pour les besoins de l'admin

Target

- DROP : paquet abandonné, aucun retour
- REJECT : paquet abandonné, prévient l'émetteur
- ACCEPT : paquet accepté
- LOG : paquet enregistré (non terminal)

Énigme

Question

Pourquoi, même si on bloque avec le pare-feu toutes les connexions, peut-on encore sniffer sur le réseau ?

Réponse

Parce que la capture du paquet dans le noyau se fait **avant** qu'il soit transmis au pare-feu

Qui est en écoute ?

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)

Choisir ses services

2 modes de démarrage

- Direct : le serveur est en écoute, et traite les connexions au fur et à mesure
  - Script de démarrage présent dans `/etc/init.d`
- Indirecte : la connexion est traitée par un wrapper puis transmise au serveur une fois la session établie
  - démarrage via `[x]inetd.conf`

Outils

Monitoring

- `iptraf` : permet de surveiller le trafic par port / interface
- `fail2ban` : ajoute automatiquement des règles de fw pour les attaquants qui brute-force des comptes

Cinquième partie V

Patches et autres protections

PaX

- Patch noyau implémentant la non-exécution de pages mémoire ainsi que la randomisation de l'espace d'adressage (ASLR)
- A pour objectif d'empêcher l'exécution de code arbitraire
- Supporte plus de 10 architectures (Alpha, i386, ia64, Mips, Mips64, Parisc, PPC, PPC64, Sparc, Sparc64, x86_64)
- Inclus dans la distribution gentoo-hardened ainsi que dans grsecurity

Historique

- Né en 2000 comme une preuve de concept d'utilisation des TLB pour séparer la lecture/écriture et l'exécution de pages mémoire
- Introduction de l'ASLR en 2001
- VMA mirroring en 2002 (base de SEGMEXEC)

PAGEEXEC

Émuler une sémantique de pages non exécutables sur les processeurs qui ne le supportent pas (en particulier les processeurs x86).

PAGEEXEC

- Les pages à rendre non exécutables sont marquées avec le bit superviseur
- L'utilisation d'une telle page provoque une faute de pages qui est interceptée par le gestionnaire de fautes de pages
- Il vérifie si la faute est due à l'exécution de code
- Dans ce cas, le processus est terminé

Inconvénient : relativement coûteux en temps processeur

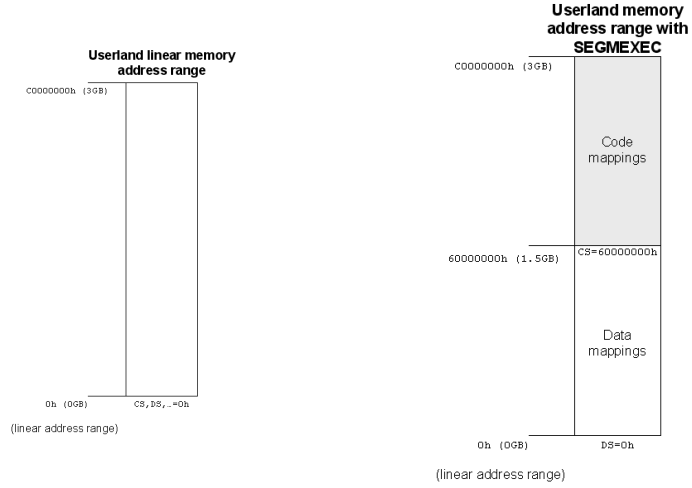
SEGMEXEC

SEGMEXEC

- Utilise la segmentation pour obtenir des pages non exécutables
- Sépare le segment de code utilisateur du segment de données utilisateur
- Utilise le VMA mirroring pour que les pages de code soient aussi disponible dans le segment de données

Inconvénient : complexe à implémenter, réduit l'espace d'adressage à une taille de 1,5 Go

SEGMEXEC



SEGMEXEC

Sans SEGMEXEC :

```
# cat /proc/self/maps
08048000-0804c000 r-xp 00000000 08:03 1603665    /bin/cat
0804c000-0804d000 rw-p 00003000 08:03 1603665    /bin/cat
0804d000-0806e000 rw-p 0804d000 00:00 0          [heap]
b7dd4000-b7dd5000 rw-p b7dd4000 00:00 0
b7dd5000-b7ef8000 r-xp 00000000 08:03 1327889    /lib/libc-2.6.1.so
b7ef8000-b7efa000 r--p 00123000 08:03 1327889    /lib/libc-2.6.1.so
b7efa000-b7efb000 rw-p 00125000 08:03 1327889    /lib/libc-2.6.1.so
b7efb000-b7eff000 rw-p b7efb000 00:00 0
b7f14000-b7f15000 r-xp b7f14000 00:00 0          [vdso]
b7f15000-b7f2f000 r-xp 00000000 08:03 1328128    /lib/ld-2.6.1.so
b7f2f000-b7f30000 r--p 00019000 08:03 1328128    /lib/ld-2.6.1.so
b7f30000-b7f31000 rw-p 0001a000 08:03 1328128    /lib/ld-2.6.1.so
bfcf8000-bfd0f000 rw-p bfcf8000 00:00 0          [stack]
```


SEGMEXEC

Avec SEGMEXEC :

```
# cat /proc/self/maps
110d4000-110d9000 r-xp 00000000 08:01 868139     /bin/cat
110d9000-110da000 r--p 00004000 08:01 868139     /bin/cat
110da000-110db000 rw-p 00005000 08:01 868139     /bin/cat
110db000-11106000 rw-p 110db000 00:00 0          [heap]
4fbae000-4fbaf000 rw-p 4fbae000 00:00 0
4fbaf000-4fcdc000 r-xp 00000000 08:01 822891     /lib/libc-2.6.1.so
4fcdc000-4fcde000 r--p 0012d000 08:01 822891     /lib/libc-2.6.1.so
4fcde000-4fcdf000 rw-p 0012f000 08:01 822891     /lib/libc-2.6.1.so
4fcdf000-4fce3000 rw-p 4fcdf000 00:00 0
4fce6000-4fce7000 r-xp 4fce6000 00:00 0          [vdso]
4fce7000-4fd01000 r-xp 00000000 08:01 822916     /lib/ld-2.6.1.so
4fd01000-4fd02000 r--p 00019000 08:01 822916     /lib/ld-2.6.1.so
4fd02000-4fd03000 rw-p 0001a000 08:01 822916     /lib/ld-2.6.1.so
5d380000-5d396000 rw-p 5d380000 00:00 0          [stack]
710d4000-710d9000 r-xp 00000000 08:01 868139     /bin/cat
afbaf000-afcdc000 r-xp 00000000 08:01 822891     /lib/libc-2.6.1.so
afce6000-afce7000 r-xp 4fce6000 00:00 0          [vdso]
afce7000-afd01000 r-xp 00000000 08:01 822916     /lib/ld-2.6.1.so
```


MPROTECT (1/2)

- Avec PAGEEXEC et SEGMEXEC, la notion de pages non exécutables a un sens
- Il reste à empêcher qu'une page puisse être à la fois exécutable et inscriptible

MPROTECT

- Empêche qu'une zone de mémoire puisse être accessible à la fois en exécution et en écriture
- Exception pour les fichiers ELF avec relocation (entrée DT_TEXTREL dans la table dynamique)

MPROTECT (2/2)

Plus qu'une seule possibilité pour exécuter du code arbitraire :

- Charger dans la mémoire du processus dont il a pris le contrôle un fichier (dans lequel il a injecté du code) à l'aide de `mmap`
- Méthode utilisée pour le chargement des bibliothèques partagées.

`MPROTECT` est incompatible avec les programmes générant du code à la volée (typiquement Java).

Address Space Layout Randomization (ASLR)

Objectif

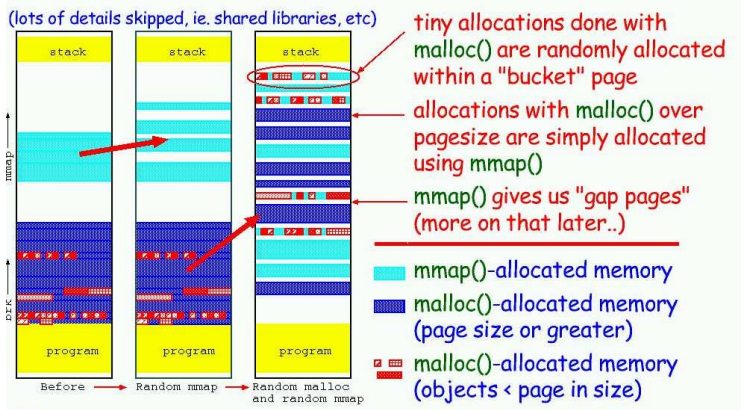
Empêcher toute adresse écrite en dur d'avoir un sens (complique la fiabilisation des exploits).

ASLR

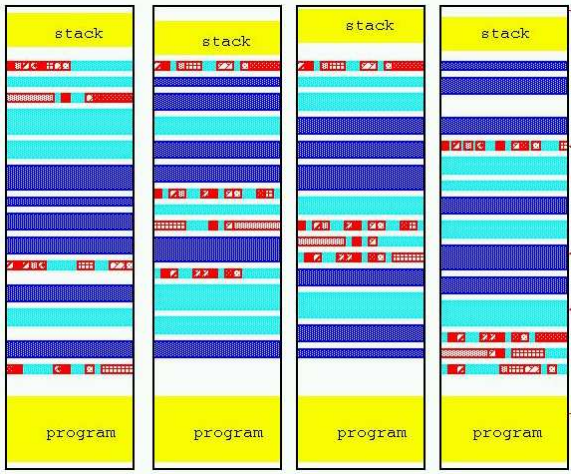
- Randomisation de la pile utilisateur (RANDUSTACK)
- Randomisation des zones mmap()ées (RANDMMAP)
- Randomisation de l'exécutable (RANDEXEC)

- Nécessité d'utiliser des binaires de type ET_DYN pour profiter pleinement de l'ASLR
- Possibilité d'utiliser RANDEXEC (c'est plus une preuve de concept néanmoins)

Address Space Layout Randomization (ASLR)



Address Space Layout Randomization (ASLR)



W∧X

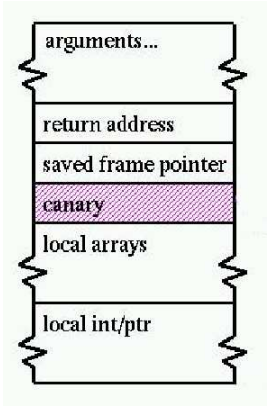
- Apparu en mai 2003 dans OpenBSD 3.3
- Empêche qu'une page soit disponible à la fois en écriture et en exécution
- Dispositif très similaire à PaX, cependant il n'existe pas d'équivalent de la restriction de `mprotect()`

W∧X

Principe

- Utilisation de la segmentation : le segment de code a une limite de 512 Mo, i.e toute adresse supérieure à 0x20000000 n'est pas exécutable
- Assainissement des binaires : le trampoline *sigreturn* n'est plus dans la pile, la GOT et la PLT ainsi que les sections `.ctors` et `.dtors` ne sont plus disponibles en écriture, la section `.rodata` n'est plus exécutable
- Modification du loader pour que les zones de données soient mappées au-dessus des 512 Mo

Propolice/SSP



- Une valeur aléatoire est insérée au prologue de la fonction
- Elle est vérifiée à l'épilogue
- Les variables locales sont réordonnées

Propolice/SSP

- Extension de gcc issue de StackGuard
- Insère un canari à la compilation pour détecter les débordements de tampons
- Réordonne les variables locales pour placer les buffers après les pointeurs de manière à éviter leur corruption
- Copie les pointeurs donnés en argument dans une zone de mémoire située avant les variables locales
- Options du compilateur : `-fstack-protector`, `-fno-stack-protector`, `-fstack-protector-all`, `-fno-stack-protector-all`

Grsec

- Patch du noyau linux
- Renforce la sécurité sur 3 axes :
  - Détection
  - Prévention
  - Confinement
- Inclus PaX
- Contient un système RBAC (Role-Based Access Control)

Historique

- Commencé en février 2001
- Première diffusion pour le noyau Linux 2.4.1
- Fondé sur le port d'Openwall sur Linux 2.4

Audit des appels système

Détection

- Audit (`exec`, `chdir`, `mount`, `umount`, IPC)

```
Nov 19 23:03:35 gentoo-vmware grsec: exec of /sbin/hwclock
                (/sbin/hwclock --adjust --localtime)
                by /bin/bash[bash:1818] uid/euid:0/0 gid/egid:0/0,
                parent /bin/bash[bash:1817] uid/euid:0/0  gid/egid:0/0
```


Logs renforcés

Détection

- Log des attaques : signaux (SIGSEGV, SIGABRT, SIGBUS, SIGKILL), fork ratés, ptrace, changement de temps (stime, settimeofaday), exec dans un chroot, *capabilities* refusées

```
Nov  9 14:54:11 gentoo-vmware grsec: denied resource overstep by requesting 4096
                for RLIMIT_CORE against limit 0 for /usr/lib/paxtest/anonmap[anonmap:6433]
                uid/euid:0/0 gid/egid:0/0, parent /usr/lib/paxtest/anonmap[anonmap:6432]
                uid/euid:0/0 gid/egid:0/0
```


Bloquer les exploits

Prévention

- Utilisation de PaX
- Durcissement de certains appels systèmes : `chroot()`, `ptrace()`, `mmap()`, `link()`, `symlink()`, `sysctl()`
- Randomisation des IPID, RPC XID, ports privilégiés utilisés dans les RPC, PID

Durcir le système

Prévention

- Module netfilter furtif répondant sur les ports libres pour ralentir les scanners de ports
- Amélioration de chroot : rend les appels systèmes conscients du chroot, chdir() automatique, *capabilities* restreintes
- Restriction des fuites d'informations sur l'adressage mémoire et anti-bruteforce

Restreindre les privilèges

Confinement

- TPE (Trusted Path Execution) : les utilisateurs ne peuvent exécuter que des binaires possédés par root dans des répertoires inaccessibles en écriture pour tout le monde
- RBAC (Role Base Access Control) : même root peut être restreint

Sixième partie VI

Unix sécurisés

Gentoo Hardened

- Projet visant à utiliser la distribution Gentoo sur des serveurs hautement sécurisés
- Utilise un noyau patché avec PaX, grsec et SELinux
- Possède une chaîne de compilation PIE/SSP
- Seule distribution Linux proposant une intégration poussée de ces mécanismes de sécurité (Adamantix n'étant plus maintenue)

OpenBSD

- Système d'exploitation issu des BSD4.4
- Né d'un fork de NetBSD en 1995
- A pour motivation principale d'être l'OS le plus sécurisé
- Portable et standard
- Système gratuit avec une licence *très libre*

Fonctionnalités

- Sécurité pro-active (chaque bug est potentiellement exploitable)
- Politique de full-disclosure
- Un processus d'audit de code permanent
- Système complet (Kernel+userland)
- Cryptographie omniprésente
- Supporte 17 plateformes matérielles
- Configuration par défaut bien faite (activation explicite des services par exemple)

Sécurité

- Introduction de nouvelles fonctions : `strlcpy()` et `strlcat()`
- Séparation de privilèges et révocation des droits dès que possible
- Chroot des services
- Utilisation de SSP/ProPolice
- Protection de l'espace d'adressage : W$\wedge$X, segment .rodata, `malloc()` et `mmap()` randomisés

Cryptographie

- IPSEC
- Kerberos
- Générateurs de nombres aléatoires forts, transformations et fonctions de hachage cryptographiques intégrées, support pour du matériel cryptographique
- Utilisation de bcrypt (dérivé de blowfish) pour hacher les passwords
- Swap chiffrée (chaque zone est chiffrée par une clé qui est révoquée dès que la zone est inutilisée)
- Pile tcp/ip randomisée (ISN, timestamps, source ports)

Divers

- Kernel flags (permissions supplémentaires sur les fichiers)
- Securelevels (durcissement du noyau)
- Systrace (proxy pour appels systèmes)
- `rm -P` (équivaut à un `shred`)

Kernel flags

Ils permettent d'ajouter des permissions sur les fichiers. Les flags les plus courants sont :

sappnd : append-only flag. Les fichiers ne peuvent être effacés ou édités. Très utile pour les fichiers de logs.

schg : immutable flag. Les fichiers ne peuvent être modifiés en aucune façon.

Securelevels (1/2)

Le noyau se comporte différemment suivant le securelevel utilisé. Il existe 4 securelevels : -1, 0, 1 et 2 :

-1 : Aucune fonctionnalité supplémentaire est utilisée

0 : Utilisé par le système jusqu'au passage en mode multi-utilisateurs, toujours aucune fonctionnalité ajoutée

Securelevels (2/2)

1 : Niveau par défaut.
- Pas d'écriture dans /dev/mem ou /dev/kmem
- Pas de raw access aux disques durs
- Les flags `schg` et `sappnd` ne peuvent être enlevés
- Les modules noyau ne peuvent ni chargés ni déchargés

2 : Même chose que pour le niveau -1 avec en plus :
- L'horloge système ne peut aller en arrière
- `pfctl` ne peut plus modifier les règles de filtrage
- Les paramètres sysctl du débogueur noyau DDB ne sont pas modifiables

Systrace

- Gestionnaire d'accès d'appels systèmes
- Possibilité de spécifier pour chaque programme les appels systèmes autorisés et leurs utilisations

```
native-bind: sockaddr match "inet-*:53" then permit
native-chdir: filename eq "/" then permit
```


Septième partie VII

Durcissement de services

Chroot

- Appel système (*change root directory*)
- Change la racine du système de fichiers pour les programmes situés dans le chroot
- Si on est root, il est couramment énoncé que l'on peut sortir de la prison
- Quelques méthodes permettant de sortir du chroot ...

Breaking Chroot (1/4)

Avec `mount`

- Si on est root, on peut créer des systèmes de fichiers et les monter dans le chroot
- Avec le pseudo système de fichiers devpts on peut injecter du code dans les terminaux (exemple : un terminal admin)

Breaking Chroot (2/4)

Avec `ptrace`

- En tant qu'utilisateur, il peut s'attacher à n'importe quel processus de même UID que lui
- Il ne faut pas de processus avec le même UID que le processus chrooté en dehors de la prison

Breaking Chroot (3/4)

Avec `fchdir`

- Appel système permettant de changer le répertoire courant pour un descripteur de fichier ouvert sur un autre répertoire
- Si un processus garde un descripteur de fichier ouvert sur un répertoire situé en dehors du chroot, il est possible de sortir de la prison

Breaking Chroot (4/4)

Avec `sysctl`

- Appel système renseignant sur l'état du noyau et permettant d'en modifier la configuration sous réserve d'avoir les droits suffisants
- Exemple : changer le chemin du chargeur de modules noyau par `/bin/sh`

Conclusion

- Bonne idée mais à utiliser correctement
- Réduire les droits du processus
- Ne pas oublier le `chdir`
- Si possible utiliser le patch `grsecurity` qui corrige les attaques précédentes

```
chdir("/foo/bar");
chroot("/foo/bar");
setuid(non zero UID);
```

Présentation et historique

- Serveur web le plus répandu (cf. Netcraft)
- Apache 0.6.2 diffusé en avril 1995
- Apache 1.0 en décembre 1995
- Apache 2.0 en Avril 2002 (réécriture complète)
- Deux versions :
  - 1.3 version stable
  - 2.2 version de développement

Quelques attaques Web classiques

Directory traversal

```
http://www.example.com/cgi-bin/lame.cgi?file=../../../../etc/
```

Null byte

```
http://www.example.com/cgi-bin/lame.cgi?page=../etc/passwd%00
```

Buffer overflows

```
http://www.example.com/cgi-bin/lame.cgi?type=AAAAAAAAAAAAAAAA
```


Configuration d'Apache (1/2)

- Créer un utilisateur spécial (exemple : `apache`)
- Vérifier les droits :
  - S'assurer que seul `root` a la permission d'écrire dans les répertoires d'Apache
  - S'assurer que seul `root` peut lire les logs
- Réduire la surface d'exposition :
  - Retirer les pages de diagnostic
  - Enlever la génération de listing automatique
  - Enlever la signature du serveur :`Servertoken Prod`

Configuration d'Apache (2/2)

- Charger uniquement les modules utilisés
- Limiter les types `Mime`
- Augmenter le niveau de logs
- Désactiver les cgi
- Utiliser le `mod_access`
- Chrooter si possible

Modsecurity

- Développé par Ivan Ristic
- Firewall pour applications web
- Open source
- Projet démarré en 2002
- Support commercial par Thinking Stone en 2004, racheté en 2006 par Breach Security

Utilisation

- IDS/IPS pour le HTTP
- Log du trafic HTTP
- Just-in-time patching
- Durcissement des applications web

Fonctionnalités intéressantes

- Transformation des données (anti-évasion)
- Stateful (sessions, utilisateurs, adresses IP ...)
- Note les anomalies
- Corrélation d'évènements
- Peut rediriger sur un honeypot
- Support du XML

Firewall pour applications web

3 modes de fonctionnement :

- Just-in-time patching
- Positive security
  - Validation des entrées par un modèle
  - Chaque application possède ses règles
  - Les règles doivent être ajustées en permanence
- Negative security
  - Recherche des actions malicieuses
  - Détection par signature
  - Modèle générique avec certains ajustements par application
  - Proche d'un IPS

Just-in-time patching

- L'application est vulnérable à une injection SQL
- Les logins ne doivent contenir que des caractères alphanumériques

```
<LocationMatch "^/app/login.asp$">
  SecRule ARGS:username "!^\w+$" "deny,log"
</LocationMatch>
```


Positive security

- Même chose que le just-in-time patching
- Mais chaque paramètre doit être défini
- Besoin d'une période d'apprentissage

```
<LocationMatch "^/web/bin/auth/auth.dll$">
  SecDefaultAction "log,deny,t:lowercase"
  SecRule REQUEST_METHOD !POST
  SecRule ARGS:destination " URL" "t:urlDecode"
  SecRule ARGS:flags "[0-9]{1,2}"
  SecRule ARGS:username "[0-9a-zA-Z].{256,}"
  SecRule ARGS:password ".{256,}"
  SecRule ARGS:SubmitCreds "!Log.On"
  SecRule ARGS:trusted "!(0|4)"
</LocationMatch>
```


Negative security

- Fonctionne comme un IPS
- Parle nativement le protocole HTTP ainsi que le HTML
- Gère les sessions et les cookies
- Fonctionnalités anti-évasion

Configuration

register_globals

- Désactivé depuis PHP 4.2
- Très dangereux

Exemple :

```
<?
if (isset($admin) == false) {
    die "This page is for the administrator only!";
}
?>
```

Configuration

allow_url_fopen

- Permet de traiter les URLs comme des fichiers
- Activé par défaut !

Exemple :

```
http://www.example.com/view.php?what=index.php
<? include($what) ?>
```


Configuration

enable_dl

- PHP peut charger dynamiquement des modules avec `dl()`
- Il est possible d'écrire un module maison (cf. phrack 62 *"Attacking Apache with builtin Modules in Multihomed Environments"*, *andi@void*)

Configuration

expose_php

- Il est inutile d'afficher la version de PHP utilisée
- Présence d'*easter eggs* :
  - Crédits :
    `http://www.example.com/index.php?=PHPB8B5F2A0-3C92-11d3-A3A9-4`
  - Logo :
    `http://www.example.com/index.php?=PHPE9568F34-D428-11d2-A769-0`

Configuration

disable_functions

- PHP autorise la désactivation de certaines fonctions
- Permet de réduire la surface exposée

Configuration

open_basedir

- Permet de restreindre l'accès au système de fichiers
- Ne remplace pas un chroot
- Peut améliorer la sécurité

Configuration

Options liées aux logs

- Enregistrer tous les messages d'erreurs en utilisant `error_reporting = E_ALL` et `log_errors = On`
- Par défaut les erreurs vont dans les logs d'Apache, il est plus utile de créer un fichier par application
- Mais désactiver l'affichage des erreurs sur les pages web en utilisant `display_errors = Off` et `display_startup_errors = Off`

Configuration

Limites

- Limiter la mémoire que peut consommer un script avec `memory_limit`
- Limiter la taille d'une requête POST avec `post_max_size`
- Limiter le temps d'exécution avec `max_input_time` et `max_execution_time`

Configuration

Uploads

- Désactiver les uploads si besoin (`file_uploads = Off`)
- Limiter la taille avec `upload_max_filesize`
- Créer un répertoire spécifique avec `upload_tmp_dir`

Configuration

Safe mode

Rajoute certaines restrictions pour tenter d'améliorer la sécurité, par exemple :

- Chaque fichier doit avoir le même UID que celui du script qui l'utilise (possibilité d'avoir le test uniquement sur le GID)
- Les variables d'environnement peuvent être protégées en écriture
- Seuls les binaires contenus dans un certain répertoire peuvent être exécutés

Suhosin

- Composé :
  - D'un patch renforçant le cœur de PHP
  - D'une extension pour PHP
- Remplace le patch *hardened-php*
- Appliqué par défault dans OpenBSD
- Signifie Ange gardien en coréen

Fonctionnalités

- Patch :
  - Protection bas-niveau
  - Débordement de tampons
  - Chaînes de format
- Extension :
  - Rajout de fonctionnalités (exemple : fonctions cryptographiques)
  - Protection à l'exécution (exemple : chiffrement des cookies, protection contre les HTTP Response Splitting)
  - Protection des sessions
  - Peut filtrer les entrées
  - Fonctionnalités de logs avancées

Huitième partie VIII

Sécurité de bout en bout

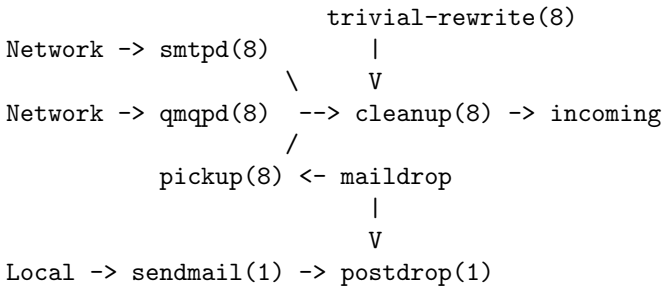
Postfix

- Serveur mail en GPL
- Remplaçant de sendmail
- A pour objectif d'être facile à administrer, configurable et sécurisé
- Compatible avec les programmes écrits pour sendmail
- Développé par Wietse Venema (Coroner's Toolkit, TCP wrapper, SATAN)

Architecture

- Typiquement unix
- Beaucoup de petits programmes indépendants qui n'effectuent qu'une seule tâche
- Un serveur principal chargé de lancer les autres programmes à la demande

Architecture



Sécurité (1/2)

- Moindre privilège : la plupart des composants de Postfix peuvent être chrootés et abandonnent leurs privilèges
- Isolation : l'attaquant doit corrompre plusieurs binaires pour arriver du réseau sur la machine
- Le serveur principal tourne avec un UID bien défini
- Aucun programme composant Postfix utilise le bit `set-uid`

Sécurité (2/2)

- File d'attente :
  - Initialement, utilisation d'un répertoire accessible en écriture à tout le monde pour gérer la file d'attente des messages
  - Finalement, utilisation d'un programme `set-gid`, `postdrop` pour l'ajout de mail
- Confiance : toutes les données extérieures, y compris celles venant des programmes composant Postfix, sont filtrées
- Les données trop grosses sont coupées puis réassemblées

Vsftpd

- Serveur ftp en GPL
- Sécurisé et rapide
- Utilisé par Redhat, OpenBSD, Debian, Suse et beaucoup d'autres
- Développé par Chris Evans, chercheur de vulnérabilités employé par Google

Secure by design

- Toutes les données provenant de l'extérieur sont manipulées par un processus non-privilégié tournant dans une prison `chroot`
- Le code privilégié est le plus petit possible et est utilisé dans un processus séparé, père du processus non-privilégié
- Toutes les requêtes parent-fils sont considérées non-sûres et transitent par une socket
- Le père cherche à limiter ses privilèges au maximum. Après le login, le père calcule les privilèges dont il a besoin et utilise les fonctions `capabilities()` et `chroot()` en ce sens

Secure by implementation

- Utilisation d'une API pour manipuler les buffers centralisant et limitant ainsi le code susceptible de contenir des buffers overflows
- Similaire à la classe `String` en C++
- Toutes les allocations, les calculs de taille et les copies sont réalisés en un seul endroit

Neuvième partie IX

Et pour terminer (enfin)

Conclusion : last but not least

Unix

- Système avec des concepts très simples
- Système hautement modulaire

Linux

- Permet d'adapter la sécurité à ses besoins
  - Poste de travail, serveur, bastion, passerelle, ...
- Sécurité "multi-niveaux" : système, appli, réseau
  - Mécanismes indépendants de sécurisation