Chapitre 2

Virtualisation

I Principes de virtualisation

1. Histoire

C'est en France, dans les laboratoires d'IBM, à Grenoble, que les premiers travaux sur les machines virtuelles font leur apparition. Nous sommes alors en 1960 et Big Blue vient de créer son premier ordinateur avec une base logicielle commune, le *System/360*, qui pour la première fois n'obligera pas à changer tout le matériel (imprimante, carte...) lors de la prochaine évolution du processeur. Cela permettra également de conserver les programmes, sans avoir à en ré-écrire spécifiquement des parties. Cette première évolution sera continuée par la naissance du *CP-40* [16], considérée comme la première machine virtuelle [17]. Le produit se transforma en *VM/CMS*. La suite logique donnée par IBM les conduira vers les *mainframe* qui virtualisent leur système d'exploitation. IBM étant à l'époque un vendeur de matériel, il ne pouvait alors pas faire d'avantage la promotion d'une technologie qui auraît permit au client d'économiser sur ses ventes.

Par la suite, d'autres sociétés comme HP, avec ses *PA-RISC ou IA64*, ou encore Sun avec la série des *E10K/E15K/E20K/E25K* utiliseront la technologie de virtualisation.

Le système commercial le plus connu actuellement se nomme *VMware*, du même nom que la société qui le commercialise. C'est cette société qui popularisera auprès du grand public l'intérêt des machines virtuelles, que ce soit pour des tests ou de la production. Cependant, bien que la société VMware soit leader dans le domaine, elle privilégie l'usage des machine virtuelles en tant que machine virtuelle, ce qui ne représente qu'un des types de la virtualisation [21] [22], comme nous le verrons dans la section II. De nombreux logiciels libres existent aujourd'hui à tous les niveaux logiciels, nous en citeront quelques uns pour nous attarder plus particulièrement sur *Linux Vservers*.

2. Usages

Tout comme nous l'avons vu, l'usage premier n'est pas récent et concernait les grands constructeurs de matériels et de systèmes. C'est cependant avec l'explosion d'Internet que la technologie de virtualisation verra de nouveaux utilisateurs. En effet, les premiers *ordinateurs* personnels grand public, comme ceux d'Atari et Amstrad verront une communauté intéressée de retrouver leurs anciens programmes ou jeux. Il en va de même pour les émulateurs de console de jeux.

Un domaine d'avantage dédié à la recherche, dans le domaine de la sécurité Informatique, utilise beaucoup ce principe de virtualisation. Celui des pots de miel, qui, comme nous le verrons dans le chapitre dédié à

ce sujet, permet aux administrateurs d'émuler tout ou partie d'un système sans devoir compromettre les *hôtes*, à savoir la première couche système.

C'est plus récemment que le monde de la production revient à cette notion, par besoin de rationnalisation des coûts et des usages. Ainsi, Le développement et la mise en production sont facilité par le seul déplacement des machines virtuelles. Des tests peuvent dès lors être effectués sans avoir peur de casser le système hôte. Les nouveaux systèmes sont de simples copies logiciels des systèmes de base. En terme de sécurité, toute attaque peut être limitée à l'hôte virtualisé, rendant de ce fait une attaque dans un environnement. Pour finir, une meilleure gestion de la puissance des processeurs et de la mémoire peut être envisagée.

II Les différentes familles de virtualiseurs

La virtualisation peut se présenter sous différentes formes, que ce soit de manière matérielle, par une émulation de machine complète, le partage de ressources via un isolateur, ou encore des applications virtuelles, tout comme le permet la technologie *java* avec sa fameuse machine virtuelle. Nous présentons dans cette section les différents types de virtualisation qui étaient à notre disposition et ce qui a orienté le choix de la solution.

1. Théorie

Avant de découvrir les différents types de virtualisation qui existent, il est nécessaire d'expliquer les espaces d'exécutions, communément appelés *ring*. Les rings ou anneaux sont des espaces protégés, mis à disposition par le processeur. Le niveau de sécurité s'étend du plus sécurisé, appelé le *Ring0* au plus permissif, tout comme nous le montre la figure 2.1, le plus éloigné du Ring0.

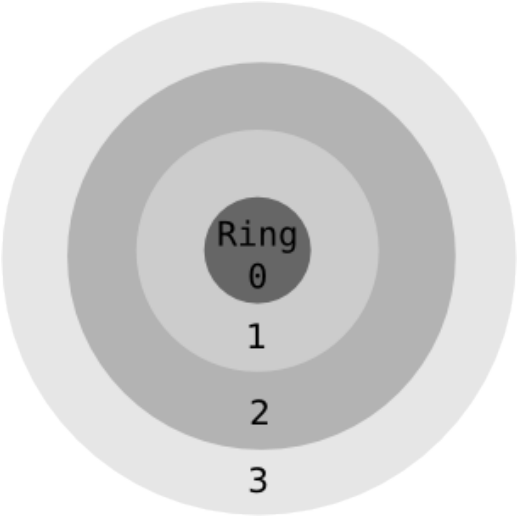

FIG. 2.1 – Concept des anneaux

De ce fait, lorsqu'une application se trouve par exemple dans le *Ring3*, elle doit faire appel à une instruction spéciale pour pouvoir utiliser un périphérique qui se trouverait dans une couche supérieure. Typiquement, les processeurs de type X86 possèdent quatre anneaux. La plupart des systèmes fonctionnant sur ces processeurs n'emploient que deux de ces anneaux, le Ring0 appelé *kernel land* et le Ring3 nommé *user land*.

2. Matériel

La généralisation des logiciels de virtualisation répond à un besoin, mais aussi à une demande. Ces technologies, inclus directement au coeur des processeurs, sont axés sur l'administrabilité et la sécurisation. Pour le premier, c'est la gestion de plusieurs couches logiciel, présenté sur différentes partitions, permettant ainsi, comme au niveau logiciel, d'offrir plusieurs *machines*, de prévoir des tests, des bascules. La sécurisation se porte au niveau des piles mémoires qui peuvent être mieux gérées et mieux contrôlées, indépendemment du système d'exploitation. Les acteurs majeurs dans ce domaine sont *INTEL* avec son *Vanderpool* [9] et *AMD* avec *PACIFICA*. Cette technologie est cependant encore très jeune et a déjà été prise en défaut par le chercheur en sécurité, Joanna Rutkowska [11].

3. Isolateur

L'isolation est issue du procédé de la mise en place d'un niveau de virtualisation du système d'exploitation. Ainsi, le noyau, la mémoire, le CPU peuvent être partagés. L'application lancée dans un *contexte*, également appelé zone d'exécution, ne *connaîtra* que les applications lancées dans le même contexte.

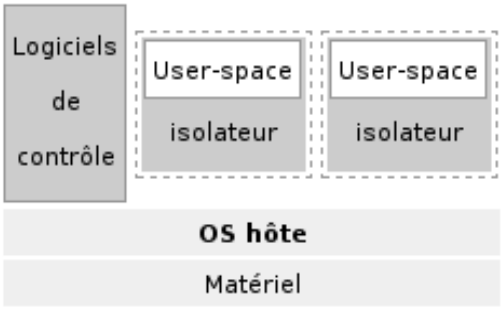

FIG. 2.2 – Architecture d'un isolateur

L'isolateur se présente sous la forme d'un patch pour le noyau et gère le partage de la mémoire, du CPU et s'assure qu'un utilisateur ou une application ne pourra pas *sauter* d'un contexte vers un autre contexte. En ce sens, l'utilitaire *chroot* ne peut pas être considéré comme un isolateur, puisque son action consiste uniquement à changer le chemin de la racine. C'est ce qu'illustre Brad Spengler dans son article [12] sur les différentes façons de briser la *prison de verre* et la solution fournit par son patch destiné au noyau, GrSec [1]. Le schéma 2.2 montre la place de l'isolateur au sein du système, avec l'exécution en *userspace* des applications et la présence d'un logiciel de contrôle. Les produits disponibles sur le marché sont *Linux-Vserver*, *BSD Jails* ou *OpenVZ*.

4. Noyau en espace utilisateur

Il est possible de considérer un noyau comme une application. C'est la solution de virtualisation proposée par exemple par *User Mode Linux* ou encore *Cooperative Linux*. Dès lors, l'accès au matériel ne s'effectue plus que par un dialogue avec le noyau de l'hôte. Cette solution est donc assez restrictive en terme de capacité d'interaction puisque le noyau n'est alors plus utile que pour la gestion des applications qui seront *au-dessus* de lui.

Il est préférable d'utiliser cette solution dans le cadre de développement de nouveaux noyaux, ou alors pour installer un système d'exploitation dans un environnement à droit restreint, où l'utilisateur pourrait installer des applications, mais pas supprimer le système existant.

[1] http://www.grsecurity.org

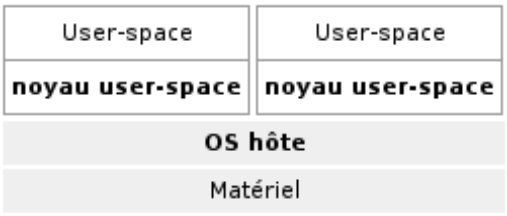

FIG. 2.3 – Architecture dans l'espace utilisateur

5. Machine Virtuelle

La machine virtuelle est sûrement le mode de virtualisation le plus connu, puisque popularisé auprès du grand public par l'intermédiaire de la société *VMware*. L'idée avancée est de reconstituer un environnement complet, y compris matériel et de mettre à disposition une plate-forme complète de virtualisation, capable de supporter différents environnements simultanément.

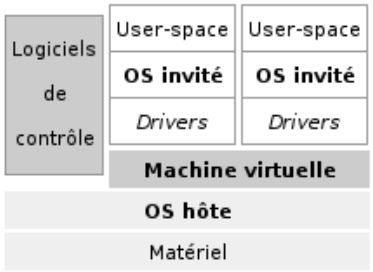

FIG. 2.4 – Architecture d'une machine virtuelle

Bien que cette solution soit plébiscite du public, elle demande néanmoins de grosses configurations matérielles afin de supporter la reconstruction matérielle, en surcouche d'un système d'exploitation, déjà consommateur de ressources. Les logiciels libres connus pour cet usage sont *QEMU* et *VirtualBox*.

6. Hyperviseur

L'hyperviseur est une couche directement posée au-dessus du matériel. C'est un noyau optimisé pour la gestion directe de systèmes d'exploitation. Ainsi, contrairement aux autres méthodes, il n'y a pas de latence dû à un système hôte. Les surcouches sont elles aussi, normalement, optimisées pour fonctionner avec l'hyperviseur. Les résultats sont donc optimaux. Le logiciel *XEN* est à ce jour le seul produit dans ce domaine, bien que de nouveaux concurrents apparaissent.

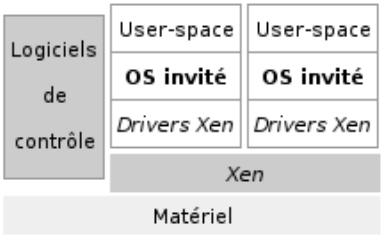

FIG. 2.5 – Architecture d'un hyperviseur

7. Choix de la machine virtuelle

Comme nous l'avons déjà expliqué, notre travail se base sur un choix de logiciels libres. De ce fait, nous pouvions nous porter sur les technologies et logiciels suivants :

- isolateur avec *Linux-Vserver*,
- espace utilisateur avec *User Mode Linux*,
- machine virtuelle avec *VirtualBox* ou *Qemu*,
- hyperviseur avec *Xen*.

Chacune des technologies a ses avantages et ses inconvénients. Au-delà de nos besoins, nous pouvons d'ores et déjà regarder les résultats obtenus par *Quétier et al* dans [1]. Les résultats obtenus pour *VMware* peuvent être considérés identiques pour *VirtualBox*, la technologie étant identique.

TAB. 2.1 – Comparaison des temps de lancement à froid et à chaud de 4 outils de virtualiasation

	Lancement à froid	Lancement à chaud
Vserver (100 VMs)	460s	30s
UML (100 VMs)	1040s	160s
Xen (50 VMs)	860s	860s
VMware (50 VMs)	1400s	810s

Ce premier tableau 7. montre des différences lors de la première initialisation, appelé lancement à froid, puis lors de démarrage alors que les bibliothèques sont déjà en mémoire. L'étude sur *Xen* et *VMware* se limite à 50 instances, limitation de la gestion des IRQ par le matériel. On constate que l'isolateur démarre plus rapidement que les autres outils et gagne 1500% de temps de chargement lorsque ses bibliothèques sont déjà présentes.

Les expériences relatées dans les graphiques 2.6 et 2.7 se fondent sur l'observation de la charge issue de l'utilisation d'une application au sein d'un outil de virtualisation. Le premier graphique 2.6 montre très clairement que la technologie de machine virtuelle utilise énormément le CPU avec une courbe qui tend vers une exponentielle. L'isolateur et l'hyperviseur restent très proche de la charge théorique.

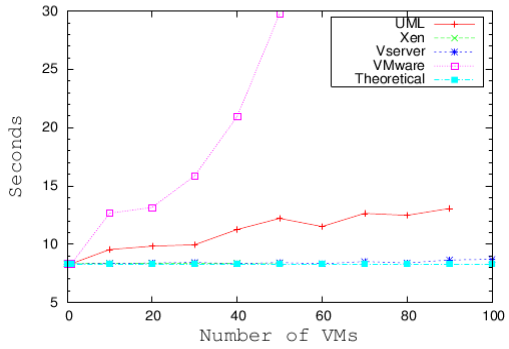

FIG. 2.6 – Charge du CPU en fonction du nombre de VM

Concernant la surcharge mémoire, la figure2.7 montre un comportement *erratique* de l'outil UML, alors que les résultats de l'isolateur et l'hyperviseur restent toujours aussi proche de la théorie. Tout comme lors de l'analyse du CPU, l'application dans la machine virtuelle consomme d'autant de mémoire qu'il y a de machines virtuelles lancées.

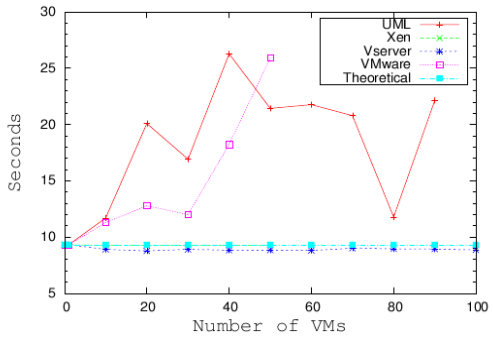

FIG. 2.7 – Charge de la mémoire en fonction du nombre de VM

En tirant parti de ces résultats, mais également des moyens mis à notre disposition, les logiciels Xen et Linux-Vserver semblent convenir à nos besoin. Cependant, nous n'avons pas besoin de noyaux spécifiques pour les différentes machines virtuelles que nous utiliserons, ni d'avoir accès à différents systèmes d'exploitation, puisque nous utiliserons uniquement GNU/Linux. L'usage requiert uniquement une séparation et de la sécurité entre les différentes machines ou contectes. Pour tout cela, notre choix se porte sur Linux-Vserver.

III Linux-Vserver

1. Fonctionnement

Tout serveur se constitue de trois éléments essentiels :
- le matériel,
- le noyau,
- les applications.

Le matériel peut être spécifique selon les besoins de service. Le noyau joue alors le rôle d'interface, de couche d'abstraction, donnant accès à des applications qui n'ont pas besoin de connaître les spécifités du matériel pour pouvoir communiquer avec lui. Certaines applications, quant à elles, nécessitent une instance père unique pour pouvoir fonctionner. De ce fait, tout démarrage d'une autre occurrence de cette entité serait vouée à l'échec. La mise en place de Serveurs Privés Virtuels (*VPS*) permet donc d'isoler la couche applicative en donnant l'accès à un seul et unique noyau qui se charge de la communication avec le matériel.

Cette solution permet à la fois de spécifier des adresses précises et différentes pour chaque VPS, mais également d'avoir un hôte central en mesure de gérer et d'accéder à tous les contextes hébergés.

Pour finir, n'importe quelle distribution GNU/Linux peut être installée, indépendamment du noyau. La liberté d'utilisation est donc conservée et permet de s'adapter aux connaissances de chaque utilisateur.

2. Installation de Linux-Vserver

Linux-Vserver est disponible sur douze architectures matérielles différentes, dont celle qui nous intéresse, la plate-forme x86. Pour pouvoir l'installer, nous avons besoin de deux choses : le patch pour le noyau linux et les outils de gestion de l'isolateur.

Nous allons commencer par la récupération et l'installation du patch. Pour ce faire, il nous suffit de nous connecter au site internet et de récupérer un fichier diff compressé, qui comporte les modifications à apporter au noyau. Nous nous positionnons donc dans le répertoire des sources du noyau pour télécharger le fichier :

```
cd /usr/src/linux
wget http://ftp.linux-vserver.org/pub/kernel/\
vs2.0/patch-2.6.17.14-grsec2.1.9-vs2.0.2.1.diff.bz2
```

Lorsque le téléchargement est terminé, nous pouvons appliquer les modifications au noyau afin d'insérer les routines nécessaires au fonctionnement de Vserver :

```
bzcat ./patch-2.6.17.14-grsec2.1.9-vs2.0.2.1.diff.bz2 | patch -p1
```

Le noyau est à présent prêt à être configuré. Il faut à présent installer les outils qui vont nous permettre de manipuler nos contextes :

```
apt-get install util-vserver vserver-debiantools
```

Le premier paquet contient un ensemble de binaires qui permettent entre autre de démarrer les contextes et d'y entrer. Le deuxième livre le script *newvserver* qui permet la création de contextes.

3. Configuration de Linux-Vserver

Il existe deux étapes de configuration. La première est obligatoire, puisque c'est le choix des options à activer au niveau du noyau. La deuxième concerne l'ajustement des contextes installés. Cette deuxième étape n'intervient que si l'on souhaite étendre les montages, changer les adresses IP...Notre usage ne nécessite pas d'affinement particulier, nous utiliserons donc la configuration par défaut, tout en nous laissant la possibilité d'y revenir ultérieurement.

Voici les options que nous activons au niveau du noyau :

```
#Linux Vserver
CONFIG_VSERVER_PROC_SECURE=y
```

Ceci configure la sécurité *ProcFS* pour cacher dès l'initialisation les entrées non-process pour tous les contextes excepté le principal et spectateur qui est sécurisé par défaut.

```
CONFIG_VSERVER_HARDCPU=y
```

Cette option permet à l'ordonnanceur du panier jeton de suspendre un processus lorsque les jetons d'un contexte sont vide. Les processus appartenant à ce contexte ne seront plus en mesure d'utiliser des ressources CPU jusqu'à ce qu'un nombre de jetons minimum par contexte soit atteint.

```
CONFIG_VSERVER_HARDCPU_IDLE=y
```

Mise en place de la limitation des tranches d'inactivité dans le CPU. De ce fait, le prochain contexte pourra être planifié le plus tôt possible.

```
CONFIG_INOXID_UGID24=y
```

L'utilisation de *Quota* ou de *Disk Limits* nécessite l'option de tagging. Cette configuration ajoute l'information de contexte persistant aux systèmes de fichiers montés avec l'option *tagxid*.

Il ne reste plus qu'à compiler le noyau et l'installer (voir le chapitre 7).

4. Mise en production

Lorsque le nouveau noyau fonctionne, nous pouvons créer nos environnements. Pour cela, il existe deux méthodes. Soit nous utilisons le script *newserver* fournit par *Debian*,

```
newvserver --vsroot /var/lib/vservers/ --hostname vserver1 \
--domain eof.eu.org --ip 192.168.0.10/24 \
--dist etch --mirror http://ftp.fr.debian.org/debian/
```

soit nous utilisons la commande générique *vserver*.

```
vserver vserver2 build  -n vserver2   --hostname vserver2.eof.eu.org
--i eth0:192.168.0.11/24   -m debootstrap -- -d sarge
```

Dans les deux cas, nous construisons ici un environnement vierge, à partir de paquets que nous allons chercher sur Internet, pour former des contextes issus de la distribution GNU/Debian. L'adresse IP est affectée depuis l'initialisation.

Lorsque l'opération est achevée, il ne reste plus qu'à démarrer notre nouveau contexte :

```
vserver vserver1 start
```

Notre vserver est maintenant fonctionnel, nous pouvons désormais le mettre à disposition ou l'utiliser pour héberger des applications.

Si nous souhaitons joindre le nouveau contexte depuis la machine hôte, il suffit *d'entrer* dans celui-ci :

```
vserver vserver1 enter
```

De même, depuis l'hôte, nous pouvons avoir les renseignements sur les contextes qui sont lancés :

```
CTX   PROC    VSZ    RSS  userTIME   sysTIME    UPTIME NAME
0       53 345.7M  96.7M   0m38s80   0m11s83   3h30m05 root server
49153    6  53.1M   6.5M   0m00s70   0m00s20   2m18s89 vserver1
49154    5  45.4M   4.6M   0m00s10   0m00s30   0m05s80 vserver2
```

avec pour chaque entête de colonne les définitions suivantes :

- **CTX** : numéro du contexte. 0 pour le root contexte, 1 pour le contexte de supervision,
- **PROC QTY** : nombre de processus dans chaque contexte,
- **VSZ** : nombre de pages virtuelles de mémoire,
- **RSS** : taille résidente,
- **utime** : temps CPU User-mode accumulé,
- **ctime** : temps CPU Kernel-mode accumulé.

Lors de la mise en place de notre maquette, nous avons utilisé nos contextes pour héberger des serveurs apache, en liaison avec le répartiteur de charge, mais nous avons également testé la mise en place du pot de miel kojoney (voir le chapitre 11, page 71). Pour la mise en place des serveurs DNS, il nous a fallut recompiler les paquets fournis par Debian, car ceux-ci n'avaient pas les *capabilities* activées par défaut, ce qui empêchait le lancement des services serveurs DNS.

Bibliographie

[1] Franck Cappello Benjamin Quetier, Vincent Neri. Selecting a virtualization system for grid/p2p large scale emulation. In *Proc of the Workshop on Experimental Grid testbeds for the assessment of large-scale distributed applications and tools (EXPGRID'06), Paris, France, 19-23 june*, 2006.

[2] Bill Cheswick. In *An Evening with Berferd. In Which a Cracker is Lured, Endured, and Studied*. Management Analytics and Others, 1995. http://all.net/books/berferd/berferd.html.

[3] Fuller et al. Classless inter-domain routing (cidr) : an address assignment and aggregation strategy. In *RFC*. IETF, 1993. http://www.ietf.org/rfc/rfc1519.txt.

[4] Jean-Philippe Pick et al. Corrigé des tp dnssec de l'atelier idsa. In *Atelier IDsA*. IDSA, 2004. http://www.idsa.prd.fr/atelier-idsa/corrige.html.

[5] Rekhter et al. Address allocation for private internets. In *RFC*. IETF, 1996. http://www.ietf.org/rfc/rfc1918.txt.

[6] K. Egevang et P. Francis. The ip network address translator (nat). In *RFC*. IETF, 1994. http://www.ietf.org/rfc/rfc1631.txt.

[7] Andrew Tridgell et Paul Mackerras. The rsync algorithm, Canberra, ACT 0200, Australia. Australian National University, 1999. http://rsync.samba.org.

[8] S. Deering et R. Hinden. Internet protocol, version 6 (ipv6) specification. In *RFC*. RFC Editor, 1998. ftp://ftp.rfc-editor.org/in-notes/rfc2460.txt.

[9] Intel. Intel® vanderpool technology for ia-32 processors (vt-x). Intel, 2005. http://cache-www.intel.com/cd/00/00/19/76/197666_197666.pdf.

[10] Paul Mockapetris. Domain names - implementation and specification. IETF RFC Editor, 1987. ftp://ftp.rfc-editor.org/in-notes/rfc1035.txt.

[11] Joanna Rutkowska. Subverting vista kernel for fun and profit. In *Symposium for Security on Asia Network*. Syscan 06, 2006. http://theinvisiblethings.blogspot.com/2006/06/introducing-blue-pill.html.

[12] Brad Spengler. Chroot(), sécurité illusoire ou illusion de sécurité. In *Misc 09*. Éditions Diamond, 2003. .

[13] C Stoll. In *The Cuckoo's Egg : Tracking a Spy Through the Maze of Computer Espionage*. Pocket Books, 1990.

[14] Andrew Tridgell. Efficient algorithms for sorting and synchronization. In *Ph.d Thesis*. Australian National University, 1999. http://samba.org/~tridge/phd_thesis.pdf.

[15] Wikipedia. Bind. In *Wikipedia, The Free Encyclopedia*. English Wikipedia, 2007. http://en.wikipedia.org/wiki/BIND.

[16] Wikipedia. Histoire du cp-40. In *Wikipedia, The Free Encyclopedia*. English Wikipedia, 2007. http://en.wikipedia.org/wiki/History_of_CP/CMS.

[17] Wikipedia. Machine virtuelle. In *Wikipedia, L'encyclopédie libre*. Wikipedia France, 2007. http://fr.wikipedia.org/wiki/Machine_virtuelle.

[18] Wikipedia. Paul mockapetris. In *Wikipedia, The Free Encyclopedia*. English Wikipedia, 2007. http://en.wikipedia.org/wiki/Paul_Mockapetris.

[19] Wikipedia. Paul vixie. In *Wikipedia, The Free Encyclopedia*. English Wikipedia, 2007. http://en.wikipedia.org/wiki/Paul_Vixie.

[20] Wikipedia. Vinton cerf. In *Wikipedia, The Free Encyclopedia*. English Wikipedia, 2007. http://en.wikipedia.org/wiki/Vinton_Cerf.

[21] Wikipedia. Virtualisation. In *Wikipedia, L'encyclopédie libre*. Wikipedia France, 2007. http://fr.wikipedia.org/wiki/Virtualisation.

[22] Wikipedia. Virtualization. In *Wikipedia, The Free Encyclopedia*. English Wikipedia, 2007. http://en.wikipedia.org/wiki/Virtualization.

[23] Paul Wouters. Dnssec howto. In *Secure DNS, Het beveiligingen van DNS in de praktijk*. C'T Magazine, 2003. http://www.xtdnet.nl/paul/dnssec/.