Nous vivons une époque formidable. Différentes technologies émergent depuis quelques années sur nos UNIX ancestraux afi n d'aboutir au nirvana de l'administration système, ce que dans le jargon marketingo-fl amby il est convenu d'appeler « la convergence » (allumez les projecteurs, les machines à fumée et les lasers). Xen pour l'abstraction matérielle et LDAP pour la centralisation de l'information sont deux des acteurs de la nouvelle manière d'aborder un parc de serveurs...

Note:
Pour la bonne compréhension de cet article, il sera nécessaire d'avoir un minimum de connaissances en systèmes de virtualisation, en particulier Xen, et de posséder quelques connaissances des systèmes BSD, en particulier NetBSD et OpenBSD. Une bonne connaissance de LDAP serait un plus, merci de nous faire parvenir votre CV dans les meilleurs délais.

Matière première

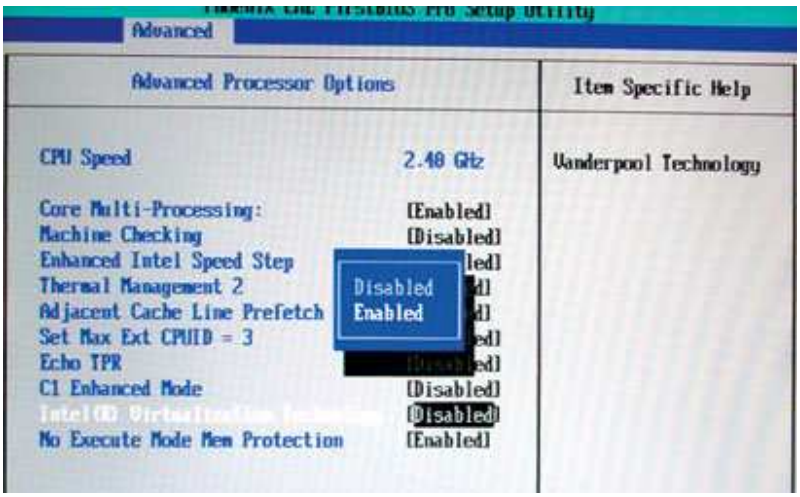
Grâce à l'argent de la dro^W^W^W^Wdivers dons, la secte GCU a obtenu un serveur digne de ce nom, dont voici les specs grossières :

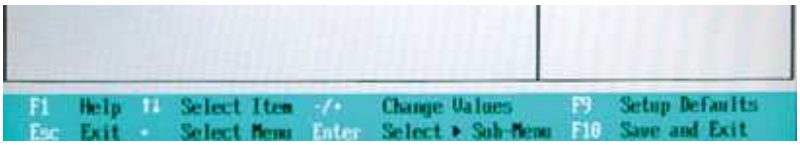
- Intel Core2 Quad core à 2.4 GHz disposant des instructions « Virtualization Technology™ » ;
- 6 gigas de DDR2 ;
- 2x500 Go en RAID1.

L'idée était de profi ter du dimensionnement de ce petit monstre pour monter une plate-forme architecturée autour de Xen. Vous le savez peut-être, chez GCU, on est très BSD. Nous avions donc à l'esprit de monter l'architecture suivante :

- dom0 sous NetBSD ;
- domU shells, destiné à accueillir des accès ssh pour les membres du groupe ayant cotisé à la mirobolante dîme annuelle ;
- domU services, qui hébergera les services classiques hors HTTP ;
- domU gcu où se trouvera le site principal et ses moult plugins ;
- domU www, domaine où nous hébergerons les sites tiers.

Afin de profi ter au maximum de ce nouveau matériel, nous envisagions également, naïfs que nous sommes, de n'avoir que des systèmes de type amd64. Nous verrons que cet espoir s'est envolé. Il est important de noter que si NetBSD dispose d'une version dotée d'un noyau de type domU, ce n'est pas le cas d'OpenBSD, aussi, c'est uniquement grâce au mode HVM (Hardware Virtual Machine) de Xen que nous serons en mesure de mettre en place des domUs ne bénéficiant pas de noyaux modifiés.





Activation de la Virtualization Technology dans le BIOS

Nous avions commencé la confi guration de cette machine en août 2007, et les pré-requis que nous avons mentionnés (tous les domUs en 64 bits) supposaient que :

- le dom0 était lui-même en 64 bits, pas de souci ici ;
- Xen/NetBSD était capable de supporter des architectures domU 64 bits.

Et c'est sur ce dernier point que nous avons dû revoir nos critères :

```
$ grep ^ONLY /usr/pkgsrc/sysutils/xentools3-hvm/Makefile
ONLY_FOR_PLATFORM= Linux-2.[46]*-i386 NetBSD-*-i386
```

Nous devions nous rendre à l'évidence, pas de dom0 64 bits sous NetBSD.
Notre choix se porte alors naturellement sur GNU/Linux, et, plus particulièrement, Debian Etch amd64 que nous savions être une bonne plate-forme pour ce type d'environnement, puisque je l'utilise depuis un moment déjà en milieu professionnel.


La panoplie de CD testés :

La Nativité

Nous installons donc une Debian Etch amd64 en utilisant un CD de type netinst (debian-40r1-amd64-netinst.iso). Aucune particularité n'est à noter, si ce n'est que nous n'allons pas installer des domUs sur des images, comme il est pratique de le faire, mais sur des volumes logiques LVM. Nous disposons de 500 G de disque utile.
Voici donc l'organisation choisie :
~~/boot~~ prend 10 G en ext3. Il est primary sur ~~/dev/sda1.~~
Nous occupons le reste avec ~~/dev/sda5~~, logical :

```
root@zone0:~# vgdisplay
 --- Volume group ---
 VG Name zone0vg
 System ID
 Format lvm2
 Metadata Areas 1
```

```
 Metadata Sequence No 13
 VG Access read/write
 VG Status resizable
 MAX LV 0
 Cur LV 6
 Open LV 2
 Max PV 0
 Cur PV 1
 Act PV 1
 VG Size 456.44 GB
 PE Size 4.00 MB
 Total PE 116849
 Alloc PE / Size 116849 / 456.44 GB
 Free PE / Size 0 / 0
 VG UUID wU1BMg-bjFu-4pmM-1JII-5USVN9Bi-H1c5Tw
```

Il dispose de 5 volumes logiques, soit un par domaine plus le volume de swap

```
root@zone0:~# lvdisplay
 --- Logical volume ---
 LV Name /dev/zone0vg/dom0lv
 VG Name zone0vg
 LV UUID Z53VrA-G5E5-xJfe-rGYy-mUtee0tC-a5gq1b
 LV Write Access read/write
 LV Status available
 # open 1
 LV Size 87.99 GB
 Current LE 22525
 Segments 1
 Allocation inherit
 Read ahead sectors 0
 Block device 254:0

 --- Logical volume ---
 LV Name /dev/zone0vg/gculv
 VG Name zone0vg
 LV UUID X21Efx-9wJs-d6Ee-saaa-0zSP-2puW-f3JylD
 LV Write Access read/write
 LV Status available
 # open 0
 LV Size 97.66 GB
 Current LE 25000
 Segments 1
 Allocation inherit
 Read ahead sectors 0
 Block device 254:1

 --- Logical volume ---
 LV Name /dev/zone0vg/wwwlv
 VG Name zone0vg
 LV UUID A4yzno-Oq7t-mgQ3-VcQGRKS7-dkq1-AHkDvi
 LV Write Access read/write
 LV Status available
 # open 0
 LV Size 97.66 GB
 Current LE 25000
 Segments 1
 Allocation inherit
 Read ahead sectors 0
 Block device 254:2

 --- Logical volume ---
 LV Name /dev/zone0vg/serviceslv
 VG Name zone0vg
 LV UUID 48LAUX-cNSA-J1Tm-Hckp-0glG-sPp3-J9Voju
 LV Write Access read/write
 LV Status available
 # open 0
 LV Size 75.48 GB
 Current LE 19324
 Segments 1
 Allocation inherit
 Read ahead sectors 0
 Block device 254:3

 --- Logical volume ---
 LV Name /dev/zone0vg/swaplv
 VG Name zone0vg
 LV UUID 3VnVUS-CTVi-QZ0Z-dyARKKl4-TX4X-bD2tkU
 LV Write Access read/write
```

```
LV Write Access read/write
LV Status available
# open 1
LV Size 5.86 GB
Current LE 1500
Segments 1
Allocation inherit
Read ahead sectors 0
Block device 254:4

--- Logical volume ---
LV Name /dev/zone0vg/shellslv
VG Name zone0vg
LV UUID hhlOvW-dBh5-NYA1-3Xkc-C5YY-jgnW-mV9tHI
LV Write Access read/write
LV Status available
# open 0
LV Size 91.80 GB
Current LE 23500
Segments 1
Allocation inherit
Read ahead sectors 0
Block device 254:5
```

A l'issue de cette installation, nous disposons d'une machine Debian des plus classiques, si l'on met de côté son indécente rapidité.

Xenification

Passons aux choses sérieuses. Il est temps maintenant de « Xenifi er » notre machine.

Comme nous le disions plus haut, à ce moment de l'installation, nous nourrissons encore l'espoir de faire tourner indiff éremment des domUs 32 ou 64 bits et, par chance, venait d'arriver Xen 3.1 qui permet précisément ce type de folie. C'est donc sur cette dernière version, renommée 3.1, mais qui devait être la 3.0.5, que nous jetons notre dévolu.
Nous savons que nous devrons utiliser les outils suivants :

- brctl afi n de bridger nos domUs aux interfaces physiques ;
- iproute est nécessaire au bon fonctionnement des scripts livrés par Xen ;
- les xen-tools de Debian ne sont pas nécessaires, mais ils pourraient nous simplifi er la vie.
- Cette procédure a été fortement inspirée du fameux «The Perfect Xen 3.1.0 Setup For Debian Etch (i386)» : http://www.howtoforge.com/debian_etch_xen_3.1.

Exécution :

```
# apt-get install bridge-utils iproute xen-tools
```

L'installation du package Xen 3.1 fourni par Xensource est triviale :

```
# cd /tmp
# wget http://bits.xensource.com/oss-xen/release\
/3.1.0/bin.tgz/xen-3.1.0-install-x86_64.tgz
# tar xvzf xen-3.1.0-install-x86_64.tgz
# cd dist/
# ./install.sh
```

Afin de s'assurer que le daemon xend se lance à chaque démarrage :

```
# update-rc.d xend defaults 20 21
# update-rc.d xendomains defaults 21 20
```

Reste à préparer la machinerie noyau/initrd :

```
# depmod 2.6.18-xen
# mkinitramfs -o /boot/initrd.img-2.6.18-xen 2.6.18-xen
```

```
# mkinitramfs -o /boot/initrd.img-2.6.18-xen 2.6.18-xen
```

Et évidemment à confi gurer notre bootloader favori :

```
# /boot/grub/menu.lst
title Xen-3.1 on Debian Etch x86_64 (2.6.18-xen)
root (hd0,0)
kernel /boot/xen-3.1.gz dom0_mem=524288
module /boot/vmlinuz-2.6.18-xen ro \
root=/dev/mapper/zone0vg-dom0lv ro max_loop=64
module /boot/initrd.img-2.6.18-xen
```

Arrêtons-nous un instant sur cette configuration pas si triviale qu'il n'y paraît. L'ordre de lancement semble logique, c'est d'abord l'hyperviseur qui démarre, puis il « appelle » le noyau comme un module tiers. Mais, nous remarquons une directive inconnue à la ligne kernel. La directive ~~dom0_mem~~ nous permet de nous assurer que le dom0 ne prendra jamais plus de 512 Mo de mémoire. Ceci pour laisser tout l'espace disponible (5.5G ici) à nos domUs. La directive ~~max_loop=64~~ nous garantit que nous n'allons pas manquer de loopback devices, massivement utilisés par Xen. Cette erreur est classique et se repère à ce type de message :

```
Error: Device 2049 (vbd) could not be
connected. Backend device not found.
```

Dans notre cas, il est assez peu probable que nous rencontrions de telles limitations du fait que nous utilisons de « vraies » partitions.
Un reboot plus tard, nous contemplons :

```
Linux zone0 2.6.18-xen #1 SMP Fri May 18 16:01:42 BST 2007 x86_64 GNU/Linux
```

Et pour nous assurer que nous sommes bien cloîtrés dans notre domaine 0 :

```
root@zone0:~# xm list
Name ID Mem VCPUs State Time(s)
Domain-0 0 512 4 r----- 110618.7
```

C'est beau, ça sent le frais.
Notre système est fonctionnel, et nous devons maintenant commencer à préparer le terrain pour nos futurs domUs. En premier lieu, faisons en sorte que tout le monde bénéfi cie d'un lien réseau en mettant en place un bridge de la manière la plus classique qui soit... ou presque :

```
# /etc/network/interfaces

auto eth0
iface eth0 inet static
        address 192.168.10.20
        netmask 255.255.255.0
        gateway 192.168.10.254

auto eth1
iface eth1 inet manual

auto xenbr0
iface xenbr0 inet static
        address 192.168.100.254
        netmask 255.255.255.0
        bridge_ports eth1
        bridge_maxwait 0
```

Mais, pourquoi diable bridgeons-nous sur eth1 ? Un peu de pratique : nous ne disposons pour cette machine que d'une seule IP publique. Il est donc hors de question de bridger sur l'interface publique, eth0. Nous verrons par la suite que nous utiliserons une « sorte » de translation de ports pour accéder aux domUs. Une technique classique dans ce genre de cas
consiste à utiliser une interface de type dummy0. Ce n'est pas l'approche que nous avons souhaité mettre en place, mais voici malgré tout son fonctionnement. Il pourra servir à ceux d'entre vous qui ne disposent pas de plusieurs interfaces sur leur machine. Il suffit pour simuler une interface de type dummy (factice), de charger le

machine. Il suffit pour simuler une interface de type dummy (factice), de charger le module dummy, et un ifconfig montrera l'output suivant :

```
dummy0 Link encap:Ethernet HWaddr 76:EB:D7:65:4C:00
        BROADCAST NOARP MTU:1500 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)
```

Reste à utiliser cette interface comme point d'attache du bridge :

```
bridge_ports dummy0
```

Et à l'utiliser comme un bridge classique. Mais, comme nous le disions, nous n'avons pas choisi cette approche. En effet, l'idée de posséder une deuxième interface réseau est des plus alléchantes dans notre contexte, car, comme nous le verrons plus tard, si c'est à cause d'un certain bug que nous n'avons pas choisi Netfi lter pour réaliser la translation de ports,
l'option que nous avons élue, IPVS, nous permettra dans un futur plus ou moins proche de raccorder une autre machine, voire un switch sur lequel seraient branchées plusieurs autres machines, et réaliser ainsi une architecture hautement disponible, de migrer à la volée des domUs d'une machine à l'autre, et j'en passe.
Je vous l'accorde, nous sommes plus dans le registre du fantasme, mais on ne sait jamais, un portefeuille est si vite tomb^W^W^W^W^W^W.
Notre confi guration réseau est prête, notre bridge peut accueillir les machines virtuelles.

La multiplication des petits pains

Rentrons dans le vif (ah-ah) (vous comprendrez cette blague pourrie dans quelques lignes) du sujet. Voici un template de machine virtuelle Xen de type HVM :

```
# /etc/xen/services
kernel = '/usr/lib/xen/boot/hvmloader'
builder = 'hvm'
memory = '1024'
name = 'services'
device_model = '/usr/lib/xen/bin/qemu-dm'
nic=1
vif = [ 'type=ioemu, bridge=xenbr0, model=ne2k_pci' ]
sdl=0
# L'output sera visible sur un
# serveur vnc sur son display 1
vnc=1
vnclisten='192.168.10.20'
vncunused=0
vncdisplay=1
vncpasswd=''
disk = [ 'phy:/dev/mapper/zone0vg-serviceslv,ioemu:
hda,w', 'file:/home/imil/iso/i386cd.iso,hdc:cdrom,r' ]
boot = 'cd'
on_poweroff = 'destroy'
on_reboot = 'restart'
on_crash = 'restart'
acpi=0
apic=0
```

La syntaxe se passe pratiquement de commentaires, mais voyons quelques informations critiques en détail.
Nous voyons que nous ne déclarons qu'une seule carte réseau (~~nic=1~~) et que s'en suivent des précisions sur l'interface réseau virtuelle. On note en particulier que le driver réseau choisi est une carte de type NE2000 PCI. En effet, si rien n'est précisé, la carte par défaut utilisée est une RTL8139, et c'est l'un des problèmes auxquels nous avons été confrontés. Pour une raison que nous ignorons encore, la carte virtuelle en question provoquait, sous des domUs NetBSD et OpenBSD des messages du type :

```
re0: watchdog timeout
```

S'en suivaient d'inévitables lenteurs réseau. Patches, documentations et autres bidouilles autour du mode Duplex de la carte n'ont rien changé. Nous avons donc choisi, entre RTL8139, NE2K et PCNET, la moins problématique, mais pas la plus rapide. A noter que dans des architectures similaires, un domU GNU/Linux accepte sans broncher la carte par défaut.
Vient ensuite la section « affichage ». Ce serveur est destiné à vivre sa vie en salle blanche, loin de tous, sans écran, ni clavier. Aussi, si nous devons intervenir « en console » sur un des domUs, nous le ferons par l'intermédiaire de VNC, un des backends d'affichage disponibles pour Xen (mais aussi pour KVM ou QEMU). C'est la variable vncdisplay qui définit le
port sur lequel écoutera le serveur VNC. Ici, ce domU verra son display exporté sur le port 5901, soit 5900 le port de base de VNC + la variable vncdisplay. Comme il est évident que vous ne laisserez pas ce port ouvert sur l'Internet, il sera nécessaire de recourir au mode forward d'OpenSSH :

```
$ ssh -L 5901:localhost:5901 192.168.10.20
```

Puis de vous connecter simplement à localhost via,
par exemple, vncviewer :

```
$ vncviewer 127.0.0.1:5901
```

A noter, nous avons dû installer le package suivant pour que l'export VNC fonctionne correctement :

```
# apt-get install libsdl1.2debian-all
```

Occupons-nous maintenant des disques qui équiperont notre domU. Deux options possibles :

- un disque physique, déclaré ainsi : ~~phy:/chemin/vers/le/device~~ ;
- une image, déclarée ainsi : ~~file:/chemin/vers/l/image~~.

Remarquons que Xen, en utilisant QEMU comme backend pour l'accès à certains devices comme les disques, supporte les formats suivants :

```
vvfat vpc bochs dmg cloop vmdk qcow cow raw
```

Et pour les versions plus récentes de QEMU :

```
parallels qcow2 vvfat vpc bochs dmg cloop
vmdk qcow cow host_device raw
```

Notez ça dans un coin pour le jour où votre service marketing favori viendra vous demander si Xen est « compatible » avec VMware.
Que va-t-il se passer lors du premier boot de notre domU ? Nous voyons que l'ordre de la séquence de boot, décrit par la variable boot est c, puis d. Notre système sera vierge au premier boot, comme une « vraie » machine. Aussi, il bootera sur l'image ~~iso i386cd.iso~~.

Et nous en venons justement aux images ISO. Voilà probablement l'aspect qui a été le plus coûteux en temps et en illusions lors de cette installation. Nous avons dû grandement revoir nos prétentions à la baisse quant à l'aspect « cutting-edge » de nos domUs, car si les versions i386 des OS que nous avons choisis tournent à merveille, l'histoire est fondamentalement différente pour leurs homologues amd64. Pour NetBSD, le constat fut rapide. La version amd64 de NetBSD 3.1 panique au boot et la version 4.0BETA ne boote pas. Pour OpenBSD, ce fut un peu plus vicieux. Si ce dernier démarre et semble d'une rapidité incontestable, lorsque nous tentons une opération impliquant du forking, au hasard, la compilation d'un port, le load average

de la VM atteint des sommets pour obtenir des temps de compilation de 6 heures pour OpenLDAP par exemple. Après des heures de recherches plus ou moins fructueuses, nous décidons de tenter notre chance avec la version 32 bits d'OpenBSD 4.1, qui, elle, tourne comme une horloge.
Ce sont donc 3 installations, on ne peut plus habituelles, que nous réalisons sur les domUs services, shells et gcu, et je dis bien 3, car le dernier, www, sera une brutale copie bloc à bloc du domU numéro 3 qui possède exactement les mêmes pré-requis et dont le disque mesure au bit près la même taille.
Voici donc l'architecture finale :

- dom0 sous Debian Etch amd64 ;
- domU shells sous OpenBSD 4.1 i386 ;
- domU services sous OpenBSD 4.1 i386 ;
- domU gcu sous NetBSD 3.1 i386 ;
- domU www sous NetBSD 3.1 i386 ;
- soupir...

Afi n d'assurer un minimum de contrôle sur les services accessibles depuis l'Internet, nous plaçons quelques règles de firewalling bien senties qu'il conviendra d'affiner lorsque notre installation sera parfaitement opérationnelle :

```
#!/bin/sh
# /etc/init.d/firewalling.sh
PATH=${PATH}:/sbin:/usr/sbin
# interfaces
WAN=eth0
XEN=xenbr0
# init
iptables -F
iptables -t nat -F
iptables -t mangle -F
# initial rules
iptables -P INPUT ACCEPT # drop at the end
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
# pass all on lo
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
iptables -A FORWARD -i lo -j ACCEPT
iptables -A FORWARD -o lo -j ACCEPT
# replies
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
# rules
# subliminal message: I hate iptables
# ssh to zone0
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
# ssh to services
iptables -A INPUT -p tcp --dport 222 -j ACCEPT
# ssh to shells
iptables -A INPUT -p tcp --dport 2222 -j ACCEPT
# smtp to services
iptables -A INPUT -p tcp --dport 25 -j ACCEPT
# imaps to services
iptables -A INPUT -p tcp --dport 993 -j ACCEPT
# www to gcu
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
# DNS TCP to services
iptables -A INPUT -p tcp --dport 53 -j ACCEPT
# DNS UDP to services
iptables -A INPUT -p udp --dport 53 -j ACCEPT
# drop in all
iptables -A INPUT -i ${WAN} -j DROP
```

Petite astuce, nos divers domUs sont accessibles via une redirection de port, donc sur une seule et même IP. Aussi, pendant la durée de notre installation, nous avons renseigné le fi chier ~~~/.ssh/config~~ avec les informations suivantes afi n que ssh n'imagine pas que votre serveur subit une attaque « man-inthe-middle » :

```
host zone0.gcu-squad.org
UserKnownHostsFile /dev/null
StrictHostKeyChecking no
```

Ceci est une vilaine gruikerie à proscrire absolument en environnement de production, bien évidemment.

Allez, on racke ?

Toutes ces désillusions passées, nous pensions partir racker en salle blanche le week-end suivant, et je ne m'étais pas soucié des problématiques de redirection de ports de l'interface publique aux adresses privées des domUs. Nous avons tous réalisé cette opération des centaines de fois, nous allions préparer cela quelques minutes avant de partir. Une promenade de santé. Après cinq heures de lutte à 20 personnes (3 physiques, le reste sur IRC), à force de lectures, tests, épluchage de mailing lists, forums, nous constatons l'impensable : la redirection de ports via Netfi lter vers nos domUs bogue purement et simplement dans notre cas. De comportements aléatoires en absence de réponse au RST après établissement de la connexion, tout y passe… et rien n'y fait. Un kernel panic suite à un ~~iptables -t nat -F~~ finit de nous convaincre : nous
n'utiliserons pas Netfi lter.
Dans le cadre professionnel, je suis amené à manipuler IPVS (http://www.linuxvirtualserver.org/software/ipvs. html). Un article dans ce même magazine en présentait récemment les bienfaits et, en y repensant, il nous est apparu que ce choix pouvait non seulement régler notre problème, mais aussi nous permettre d'imaginer des architectures autrement plus intéressantes comme nous l'avons souffl é quelques lignes plus haut. Dans l'immédiat, notre load balancing se limiterait à une seule RIP (voir GLMF #97). Testons :

```
# modprobe ip_vs
# apt-get install ipvsadm
# ipvsadm -A -t zone0.gcu-squad.org:2222 -s rr
# ipvsadm -a -t zone0.gcu-squad.org:2222 -r shells:22 -m
```

Ça marche. Nous utilisons ici l'algorithme round-robin (qui cycle sur la même IP) et le mode masquerading d'IPVS. Nous reprendrons nos règles lorsque nous aurons d'autres services que ssh sur nos domUs.

Admin's day

Nos domUs fonctionnels, il était temps de s'attaquer à la fi nalité de toute l'opération, fournir des services sur ces diff érentes machines.
Je suis un grand fan de LDAP. Dans mon parcours professionnel, j'ai été amené à coller du LDAP un peu partout, authentifi cation au sens large, serveurs DNS, FTP, HTTP et plus récemment dans le monde de la VoIP. A mon sens, LDAP a ceci de magique qu'il permet la centralisation cohérente d'un système d'information, et, bien que n'étant pas particulièrement attaché à l'aspect GUI des choses, plusieurs interfaces minimalistes, telles que gq, permettent de très facilement designer un système d'information cohérent, unifié et facile à administrer.
La première étape de ce périple consiste évidemment à installer un serveur LDAP. Naturellement, c'est vers OpenLDAP que notre regard se tourne, et la VM qui hébergera le service sera… « services ».
La FAQ d'OpenBSD spécifi e que, dans la mesure du possible, il est conseillé d'utiliser les packages binaires plutôt que les ports pour installer un logicieltiers.
Pour OpenLDAP, nous avons fait une entorse à la règle. Non que nous soyons accrocs à la compilation (encore que), mais simplement parce que le backend livré par défaut par le package binaire OpenLDAP, ldbm, corrompt inexplicablement la base et fait ainsi crasher le daemon slapd. Aussi, au vu de plusieurs posts sur ce sujet, nous décidons de recompiler le port et d'activer le backend bdb :

```
$ cd /usr/ports/databases/openldap
$ su
```

```
# su
# env FLAVOR=»bdb» SUBPACKAGE=»-server» make install
```

On automatise le lancement du daemon slapd en ajoutant au ~~/etc/rc.local~~ :

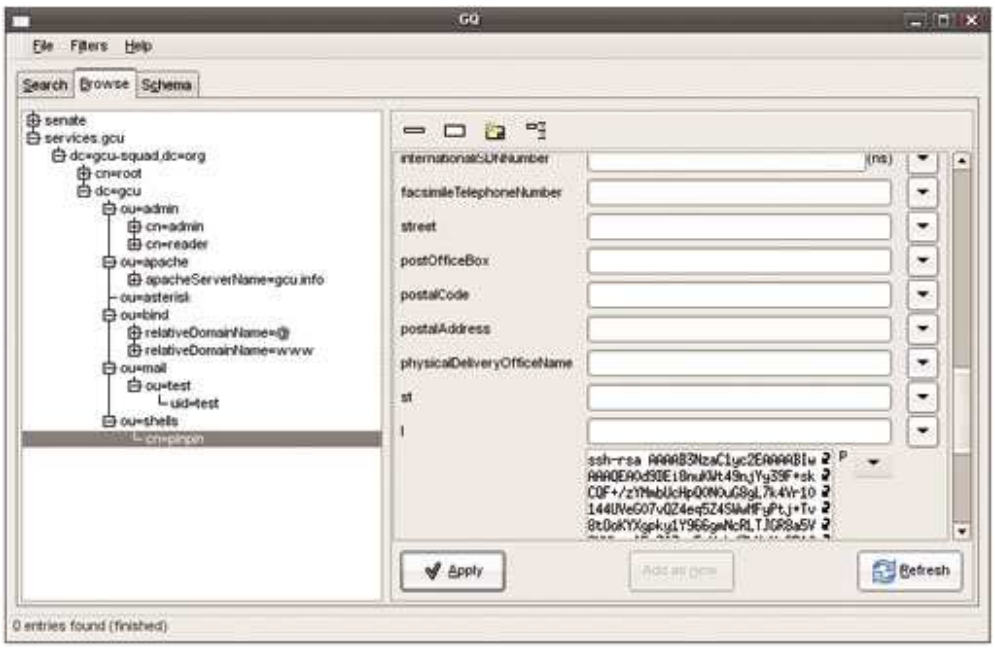
```
# OpenLDAP
if [ -x /usr/local/libexec/slapd -a -r /usr/local/etc/openldap/slapd.conf ]; then
 echo -n ' slapd'; /usr/local/libexec/slapd -u _openldap -f
/usr/local/etc/openldap/slapd.conf
 # l'utilisateur _openldap est ajouté automatiquement par le package
fi
```

Les puristes m'attendent certainement au détour d'une ruelle sombre en voyant que j'ai placé ma configuration OpenLDAP dans ~~/usr/local~~ alors que sous OpenBSD elle est censée se trouver dans ~~/etc~~. Mais, que voulez-vous, j'ai aussi mes petites manies...
Sur cette base LDAP, nous stockerons les informations des services suivants :

- OpenSSH ;
- Exim ;
- POP3S / IMAPS ;
- Bind ;
- Apache.

Ainsi, l'organisation de l'arbre LDAP sera de cette forme :

```
[+] ou=shells
 | |_ cn=pinpin
 |
[+] ou=apache
 | |_[+] apacheServerName=gcu.info
 |
[+] ou=bind
 | |_[+] relativeDomainName=@
 |
[+] ou=mail,dc=gcu,dc=gcu-squad,dc=org
 |_[+] ou=gcu-squad.org
    |__[+] uid=pinpin
```



Dans l'immédiat, notre ~~slapd.conf~~ ressemble à ceci :

```
include /usr/local/etc/openldap/schema/core.schema
```

```
include /usr/local/etc/openldap/schema/core.schema
include /usr/local/etc/openldap/schema/cosine.schema
include /usr/local/etc/openldap/schema/inetorgperson.schema
include /usr/local/etc/openldap/schema/nis.schema
# needed for login_ldap
allow bind_v2
pidfile /var/openldap-data/slapd.pid
argsfile /var/openldap-data/slapd.args
database bdb
suffix «dc=gcu-squad,dc=org»
rootdn «cn=god,dc=gcu-squad,dc=org»
rootpw {SSHA}2xxYeys8nWXp1R9RieG0yYz0Z7blAHGSfM
directory /var/openldap-data
index objectClass eq
access to attr=userPassword
by self write
by anonymous auth
by dn=»cn=god,dc=gcu-squad,dc=org» write
by * none
access to *
by dn=»cn=god,dc=gcu-squad,dc=org» write
by * read
```

Nous ne nous étendrons pas beaucoup plus sur la configuration initiale d'un serveur OpenLDAP. Il serait diffi cile d'être plus succinct que l'offi ciel « OpenLDAP QuickStart guide » : http://www.openldap. org/doc/admin24/quickstart.html. Voici un export LDIF du statut initial de la base :

```
dn: dc=gcu-squad,dc=org
objectClass: dcObject
objectClass: organization
o: GCU-Squad
dc: gcu-squad
dn: cn=god,dc=gcu-squad,dc=org
objectClass: organizationalRole
cn: god
dn: dc=gcu,dc=gcu-squad,dc=org
objectClass: dcObject
objectClass: organization
dc: gcu
o: GCU-Squad
```

Shells

Si GNU/Linux, FreeBSD et NetBSD disposent tous trois de packages de type nss et pam LDAP, ce n'est malheureusement pas le cas d'OpenBSD. Pour mémoire, PAM (Pluggable Authentication Modules) permet de rendre modulaire l'action d'authentifi cation sur un système, et NSS (Name Service Switch) de fournir des callbacks relatifs à la récupération de plusieurs types de données telles que, pour le cas qui nous importe, l'UID et le GID d'un utilisateur. Depuis plusieurs années, de nombreux UNICES possèdent des plugins PAM LDAP, qui permettent donc d'authentifi er un utilisateur selon des informations présentes dans une base LDAP, mais également NSS LDAP, qui y recherche les informations de session. Tous les attributs nécessaires à ces backends sont fournis par le schéma nis. Pour nos UNICES libres, nous devons l'existence de ces modules à Luke Howard, également développeur dans le projet OpenLDAP et auteur de la RFC 2307 « An Approach for Using LDAP as a Network Information Service ». Ouais, y'a des gars comme ça qui vous font sentir tout petit. Mais pour des raisons, peut-être erronées, de sécurité, nous avons statué que la machine qui hébergerait les comptes shell des membres serait un OpenBSD. A nouveau, c'est le spectre de la sécurité qui empêche les plugins nss et pam LDAP de pénétrer dans le bocal du poisson piquant. Mais qu'à cela ne tienne, une solution – un peu tirée par les cheveux, soyons honnêtes – existe : ~~login_ldap~~.

Installons en premier lieu le package en question :

```
# pkg_add -v login_ldap
```

Puis, il s'agit de modifi er le fichier ~~/etc/login.conf~~ afin d'ajouter une nouvelle classe d'authentifi cation, à savoir ~~ldap~~ :

d'authentification, à savoir ldap :

```
ldap:\
:auth=-ldap:\
:x-ldap-server=localhost:\
:x-ldap-server-alt=localhost:\
:x-ldap-port=389:\
:x-ldap-basedn=ou=shells,dc=gcu,dc=gcu-squad,dc=org:\
:x-ldap-uscope=subtree:\
:x-ldap-noreferrals:\
:x-ldap-filter=(&(objectclass=posixAccount)(uid=%u)):
```

La partie la moins sexy est la suivante. OpenBSD ne disposant pas de services de type nss, un utilisateur créé, même s'il est virtuel, devra posséder une ligne le concernant dans /etc/passwd. Nous réalisons cette opération à l'aide d'un simple petit script :

```
#!/bin/sh
BASEDN=»ou=shells,dc=gcu,dc=gcu-squad,dc=org»
PATH=${PATH}:/usr/local/bin:/usr/sbin
for user in `ldapsearch -x -h services -b ${BASEDN} -LLL uid|grep ^uid:|cut -d' , -f 2`
lm_100.indd 37 15/11/2007 15:33:48
Numéro 100 3 8 Décembre 2007
do
echo Creating user $user
useradd -m -d /home/$user -s /bin/sh -L ldap $user
done
```

Nous avons pour l'occasion écrit quelques scripts Perl de plus haut niveau pour gérer l'ajout, la suppression ou la modifi cation d'utilisateurs. Nos utilisateurs créés, reste maintenant à leur permettre de se connecter au domU shells, et ce, en utilisant bien évidemment le système de clés que propose OpenSSH depuis la nuit des temps. Naturellement, c'est dans notre base LDAP que nous stockerons ces clés, et c'est un projet que nous connaissons bien qui permettra d'opérer ce tour de passe-passe : OpenSSH-lpk. OpenSSH-lpk ou OpenSSH LDAP PubKey est un projet initié par un membre de GCU voilà 7 ans (Spoty, si tu nous lis, tu nous manques). Il permet à OpenSSH d'interroger une base LDAP pour connaître la clé publique associée à un utilisateur. Le projet a été repris par Inverse Path (http://www. inversepath.com/) et nous utiliserons le patch destiné à OpenSSH 4.6p1.

```
$ ftp ftp://ftp.fr.openbsd.org/pub/OpenBSD/OpenSSH/portable/openssh-4.6p1.tar.gz
$ tar zxvf openssh-4.6p1.tar.gz
$ cd openssh-4.6p1
$ ftp http://dev.inversepath.com/openssh-lpk/openssh-lpk-4.6p1-0.3.9.patch
$ patch < openssh-lpk-4.6p1-0.3.9.patch
$ ./configure --with-ldap=/usr/local --sysconfdir=/usr/local/etc/ssh-lpk
$ make
$ sudo mkdir -p /usr/local/etc/ssh-lpk
$ sudo cp sshd /usr/local/sbin/sshd-lpk
$ sudo cp sshd_config /usr/local/etc/ssh-lpk
# ssh-keygen -t rsa1 -f /usr/local/etc/ssh-lpk/ssh_host_key -N ''
# ssh-keygen -t dsa -f /usr/local/etc/ssh-lpk/ssh_host_dsa_key -N ''
# ssh-keygen -t rsa -f /usr/local/etc/ssh-lpk/ssh_host_rsa_key -N ''
```

Comme nous le voyons, nous ne polluons pas le basesystem avec notre nouveau binaire « LDAPisé », mais lui préférons un emplacement particulier aux logiciels tiers, /usr/local. Pour démarrer notre nouveau système d'authentification, nous devons copier le schéma LDAP fourni par OpenSSH-lpk sur « services » et l'inclure dans le fi chier slapd.conf :

```
include /usr/local/etc/openldap/schema/core.schema
include /usr/local/etc/openldap/schema/cosine.schema
include /usr/local/etc/openldap/schema/inetorgperson.schema
include /usr/local/etc/openldap/schema/nis.schema
include /usr/local/etc/openldap/schema/openssh-lpk_openldap.schema
```

Reste à compléter la configuration d'OpenSSH-lpk :

```
Port 2222
# [...]
UseLPK yes
LpkLdapConf /etc/openldap/ldap.conf
```

```
LpkForceTLS no
```

Et renseigner le fi chier /etc/openldap/ldap.conf :

```
BASE dc=gcu-squad, dc=org
URI ldap://192.168.100.1
```

Voilà ! Nos utilisateurs peuvent se connecter au service shell sans même s'apercevoir que nous contrôlons l'intégralité de leur compte d'un simple geste... Muahahahaha, nous sommes ignobles.

DNS

Quoi de plus naturel que de servir une zone DNS par LDAP ? Bien sûr que si, pensez-y, une structure en arbre, un root domain, un système de master/slave, des réplications diff érentielles, tout y est ! Et puis, c'est tellement plus sexy d'éditer ses zones à travers un browser LDAP qu'en éditant son fi chier de zone. La discussion n'est même pas envisageable, notre serveur DNS doit lire ses infos dans une base LDAP.

Le serveur DNS Bind propose depuis un certain temps une interface nommée « sdb ». Sur cette interface, de nombreux projets ont proposé divers backends, et, parmi eux, bind-sdb-ldap.

Cette fois, nous pourrons utiliser le source tree d'OpenBSD, bind-sdb-ldap s'appliquant parfaitement à ce dernier :

```
# ftp http://bind9-ldap.bayour.com/bind-sdb-ldap-
1.0.tar.gz
# tar zxvf bind-sdb-ldap-1.0.tar.gz
# cd bind-sdb-ldap-1.0
# cp ldapdb.c /usr/src/usr.sbin/bind/bin/named
# cp ldapdb.h /usr/src/usr.sbin/bind/bin/named/include
```

Puis, comme nous l'indique le fi chier INSTALL de bindsdb-ldap, nous modifi ons le fichier /usr/src/usr. sbin/bind/bin/named/Makefile.in comme suit :

- on ajoute ldapdb.@O@ à la variable DBDRIVER_OBJS ;
- on ajoute ldapdb.c à la variable DBDRIVER_SRCS ;
- on ajoute -I/usr/local/include à la variable DBDRIVER_INCLUDES ;
- on ajoute -L/usr/local/lib -lldap -llber à la variable DBDRIVER_LIBS.

Enfin, nous modifi ons le fi chier source /usr/src/usr. sbin/bind/bin/named/main.c de cette façon :

- on remplace #include "xxdb.h» par #include <ldapdb.h> ;
- on remplace xxdb_init(); par ldapdb_init(); ;
- on remplace xxdb_clear(); par ldapdb_clear();.

Reste à compiler, puis à installer ce bind :

```
# make -f Makefile.bsd-wrapper && make -f Makefile.bsd-wrapper install
```

Nouveau service, nouveau schéma. Sur la VM « services », nous téléchargeons http://bind9-ldap.bayour. com/dnszone-schema.txt, puis, après l'avoir renommé, l'incluons dans notre slapd.conf :

```
include /usr/local/etc/openldap/schema/core.schema
include /usr/local/etc/openldap/schema/cosine.schema
include /usr/local/etc/openldap/schema/inetorgperson.schema
include /usr/local/etc/openldap/schema/nis.schema
include /usr/local/etc/openldap/schema/openssh-lpk_openldap.schema
include /usr/local/etc/openldap/schema/dnszone.schema
```

Créons maintenant une zone afi n de constater la beauté d'une telle interaction en utilisant comme documentation de base http://bind9-ldap.bayour.com/dnszonehowto. html. Voici un extrait LDIF d'un enregistrement basique :

```
dn: relativeDomainName=@, ou=bind, dc=gcu, dc=gcu-squad, dc=org
objectClass: dNSZone
relativeDomainName: @
zoneName: gcu-squad.org
dNSTTL: 3600
dNSClass: IN
sOARecord: ns.gcu-squad.org. hostmaster.gcu-squad.org. 2007091501 3600 1800 604800 86400
nSRecord: ns.gcu-squad.com.
mXRecord: 10 mail.gcu-squad.org.
dn: relativeDomainName=ns, ou=bind, dc=gcu, dc=gcu-squad, dc=org
objectClass: dNSZone
relativeDomainName: ns
zoneName: gcu-squad.org
dNSTTL: 3600
dNSClass: IN
aRecord: 212.10.20.30
```

Je sais pas pour vous, mais moi ça me fi le des frissons partout.

Reste à configurer notre serveur DNS pour qu'il interroge notre base LDAP pour ladite zone : ~~/var/named/etc/named.conf~~ :

```
zone "gcu-squad.org" {
type master;
database "ldap ldap://127.0.0.1/ou=bind,dc=gcu,dc=gcu-squad,dc=org 172800";
};
```

Puis à démarrer ~~bind~~ dans un chroot :

```
named -t /var/named -u named -c /etc/named.conf
```

On assure le démarrage automatique du service à chaque boot en renseignant le fichier ~~/etc/rc.local~~ :

```
# BIND
if [ -x /usr/sbin/named ]; then
echo -n 'named'; /usr/sbin/named -u named -t /var/named -c /etc/named.conf
fi
```

Apache

Il y a quelque temps, j'ai utilisé un module Apache tout à fait pratique : ~~mod-vhost-ldap~~. Ce module permet à Apache de réaliser ses lookups de VirtualHosts dans une base LDAP au lieu d'utiliser des fi chiers plats. Ce module étant relativement âgé maintenant, je n'ai pas eu l'ombre d'un doute quant à sa présence dans ~~pkgsrc~~. Naïf. Je suis naïf. Pas la moindre trace dudit module, pas un seul gentil packageur ne s'est donné la peine de rentrer ce bête module dans l'arbre. Qu'à cela ne tienne, disposant d'un accès sur pkgsrc-wip, le serveur de développement de pkgsrc, je mets la main à la pâte.

Mes premières tentatives m'amènent à comprendre le pourquoi de l'absence de ce module dans ~~pkgsrc~~. En effet, ~~mod-vhost-ldap~~ ne semble plus être maintenu, et seules les versions 2.0 d'Apache, maintenant âgées, permettent sa compilation. Pourtant, un ~~apt-cache search mod-vhost~~ sur une Debian récente me laisse supposer qu'il existe une version à jour de ce plugin. Et, effectivement, il en existe une, maintenue par le projet Debian lui-même.

Muni de ces informations, je me lance dans l'écriture du package ~~pkgsrc~~ (http://pkgsrc-wip.cvs.sourceforge. net/pkgsrc-wip/wip/ap2-vhost-ldap/) :

```
# $NetBSD$
```

```
# $NetBSD$
DISTNAME= mod-vhost-ldap_1.2.0.orig
PKGNAME= ap2-vhost-ldap-1.2.0
CATEGORIES= www
MASTER_SITES= ${MASTER_SITE_DEBIAN:=pool/main/m/mod-vhost-ldap/}
MVL_VERSION= ${PKGNAME:S/ap2-vhost-ldap-//}
MAINTAINER= imil@gcu.info
HOMEPAGE= http://packages.qa.debian.org/m/mod-vhost-ldap.html
COMMENT= Apache 2.2 module LDAP Virtual Hosts support
WRKSRC= ${WRKDIR}/${DISTNAME:S/_/-/:S/.orig//}
NO_CONFIGURE= yes
APACHE_MODULE_NAME= mod_vhost_ldap
do-build:
        cd ${WRKSRC} && \
        ${APXS} -c -n ${APACHE_MODULE_NAME} \
        -I${BUILDLINK_PREFIX.openldap-client}/include \
        -L${BUILDLINK_PREFIX.openldap-client}/lib \
        ${COMPILER_RPATH_FLAG}${BUILDLINK_PREFIX.-client}/lib \
        -DMOD_VHOST_LDAP_VERSION=\\\»mod_vhost_ldap/${MVL_VERSION}\\\» \
        -lldap_r mod_vhost_ldap.c
do-install:
        cd ${WRKSRC} && \
        ${APXS} -i -n ${APACHE_MODULE_NAME} ${APACHE_MODULE_NAME}.la
        ${INSTALL_DATA_DIR} ${PREFIX}/share/examples/mod_vhost_ldap
        ${INSTALL_DATA} ${WRKSRC}/vhost_ldap.conf \
        ${PREFIX}/share/examples/mod_vhost_ldap
        ${INSTALL_DATA} ${WRKSRC}/mod_vhost_ldap.schema \
        ${PREFIX}/share/examples/mod_vhost_ldap
.include «../../www/apache22/buildlink3.mk»
.include «../../databases/openldap-client/buildlink3.mk»
.include «../../mk/bsd.pkg.mk»
```

On notera la petite manipulation relative au nom du package :

```
WRKSRC= ${WRKDIR}/${DISTNAME:S/_/-/:S/.orig//}
```

En effet, les mainteneurs de ce module ne suivent pas particulièrement les standards de nommage. Le fichier à télécharger se nomme ~~mod-vhost-ldap_1.2.0.orig.tar.gz~~, alors que le répertoire créé par la décompression de l'archive est ~~mod-vhost-ldap-1.2.0~~. Je m'étais déjà frotté à ce genre de désordre lors de l'écriture d'un outil présent dans pkgtools, ~~pkg_notify~~, dont le rôle est d'informer un mainteneur de packages qu'une nouvelle version du logiciel est disponible. J'ai éprouvé beaucoup de haine envers quelques développeurs à cette époque... mais je digresse.

On remplit un fi chier ~~DESCR~~ avec un descriptif du logiciel packagé, un fichier ~~PLIST~~ contenant la liste et le chemin relatif des fi chiers installés par le package, puis on génère un checksum à l'aide de cette commande :

```
imil@interceptor:/usr/pkgsrc/wip/ap2-vhost-ldap$ sudo make fetch
=> Fetching mod-vhost-ldap_1.2.0.orig.tar.gz
=> Total size: 8593 bytes
Trying 133.50.218.117...
Requesting http://ftp.jp.debian.org/debian/pool/main/m/mod-vhost-ldap/mod-vhostldap_
1.2.0.orig.tar.gz
100% |***********************************| 8593 14.59 KB/s 00:00 ETA
8593 bytes retrieved in 00:00 (14.57 KB/s)
imil@interceptor:/usr/pkgsrc/wip/ap2-vhost-ldap$ make makesum
```

On vérifie enfi n que ~~pkglint~~ (~~/usr/pkgsrc/pkgtools/pkglint~~) est satisfait de notre package :

```
imil@interceptor:/usr/pkgsrc/wip/ap2-vhost-ldap$ pkglint
looks fine.
```

Et nous pouvons maintenant, après avoir maintes et maintes fois vérifi é que notre package était diff usable, commiter ce dernier sur ~~pkgsrc-wip~~.
A présent, téléchargeons l'arbre WIP (Work-In-Progess) sur notre domU :

```
$ cvs -d:pserver:anonymous@pkgsrc-wip.cvs.sourceforge.net:/cvsroot/pkgsrc-wip login
$ cvs -z3 -d:pserver:anonymous@pkgsrc-wip.cvs.sourceforge.net:/cvsroot/pkgsrc-wip
checkout -P wip
```

```
# mv wip /usr/pkgsrc/
```

Afin de lier APR (Apache Portable Run-time) aux bibliothèques OpenLDAP, cette directive doit être ajoutée au fichier ~~/etc/mk.conf~~ :

```
PKG_OPTIONS.apr-util=ldap
```

De plus, pour installer le script ~~rc.d~~ de démarrage d'Apache dans ~~/etc/rc.d~~, nous ajoutons :

```
PKG_RCD_SCRIPTS=YES
```

Nous pouvons alors lancer l'installation du package ~~ap2-vhost-ldap~~ qui, puisqu'il est muni des dépendances adéquates, entraînera l'installation d'Apache, apr, apr-utils et openldap-client :

```
# cd /usr/pkgsrc/wip/ap2-vhost-ldap && make install clean
```

Puis nous activons le support de ce module dans Apache en ajoutant dans le fichier ~~/usr/pkg/etc/ httpd/httpd.conf~~ :

```
NameVirtualHost *:80
<IfModule mod_vhost_ldap.c>
VhostLDAPEnabled on
Vho stLDAPUrl «ldap://192.168.100.1/ou=apache,dc=gcu,
dc=gcu-squad,dc=org»
# VhostLdapBindDN «cn=admin,dc=localhost»
# VhostLDAPBindPassword «changeme»
</IfModule>
```

Comme à l'accoutumée, il est nécessaire de copier le schéma LDAP, ~~/usr/pkg/share/examples/mod_vhost_ldap/mod_vhost_ldap.schema~~, sur notre machine services et d'ajouter au fichier ~~slapd.conf~~ l'inclusion de ce dernier :

```
include /usr/local/etc/openldap/schema/core.schema
include /usr/local/etc/openldap/schema/cosine.schema
include /usr/local/etc/openldap/schema/inetorgperson.schema
include /usr/local/etc/openldap/schema/nis.schema
include /usr/local/etc/openldap/schema/dnszone.schema
include /usr/local/etc/openldap/schema/mod_vhost_ldap.schema
include /usr/local/etc/openldap/schema/openssh-lpk_openldap.schem
```

Voici un extrait d'export LDIF d'un enregistrement vhost dans notre base LDAP :

```
dn: apacheServerName=gcu.info,ou=apache,dc=gcu,dc=gcu-squad,dc=org
objectClass: top
objectClass: apacheConfig
apacheServerName: gcu.info
apacheDocumentRoot: /home/gcu/www
apacheServerAlias: www.gcu.info
apacheServerAlias: gcu-squad.org
apacheServerAlias: www.gcu-squad.org
apacheServerAdmin: admin@gcu.info
```

Simple, pratique, efficace.

Afin de permettre le lancement du daemon HTTP via son ~~rc.d~~ associé et d'automatiser son démarrage à chaque boot, nous ajoutons :

```
apache=YES
```

au fichier ~~/etc/rc.conf~~, puis nous démarrons simplement Apache :

```
/etc/rc.d/apache start
```

Mail

Le service Mail de notre serveur fait l'objet d'un article à part entière dans ce numéro de GLMF. Nous ne nous attarderons donc pas dessus. Bisous _mat.

Quoi ? C'est prêt ?

Oui ! Enfin ! Après ces moult péripéties, nos services sont fonctionnels. Il ne reste plus donc qu'à les transporter sur l'Internet par le biais de notre nouvel ami IPVS :

```
root@zone0:~# ipvsadm -S
-A -t zone0.gcu-squad.org:smtp -s wrr
-a -t zone0.gcu-squad.org:smtp -r services:smtp -m -w 1
-A -t zone0.gcu-squad.org:domain -s wrr
-a -t zone0.gcu-squad.org:domain -r services:domain -m -w 1
-A -u zone0.gcu-squad.org:domain -s wrr
-a -u zone0.gcu-squad.org:domain -r services:domain -m -w 1
-A -t zone0.gcu-squad.org:www -s wrr
-a -t zone0.gcu-squad.org:www -r gcu:www -m -w 1
-A -t zone0.gcu-squad.org:https -s wrr
-a -t zone0.gcu-squad.org:https -r gcu:https -m -w 1
```

Nous pouvons voir de manière cohérente (Hein Rusty, cohérente !) quelles règles sont chargées. Ces règles sont sauvegardées à l'aide de l'argument ~~save~~ de l'initscript ~~/etc/init.d/ipvsadm~~ qui ne fait rien d'autre qu'exécuter :

```
# ...
IPVSADM=»/sbin/ipvsadm»
IPVSADM_RULES=»/etc/ipvsadm.rules»
# ...
echo -n «Saving IPVS configuration: «
echo «# ipvsadm.rules» > $IPVSADM_RULES
$IPVSADM -S -n >> $IPVSADM_RULES
echo “Done.”
```

~~nmap~~ nous montre bien que ces redirections semblent fonctionner :

```
imil@tatooine:~$ nmap zone0.gcu-squad.org
Starting Nmap 4.20 ( http://insecure.org ) at 2007-11-02 18:49 CET
Interesting ports on zone0.gcu-squad.org (212.85.147.21):
Not shown: 1687 closed ports
PORT STATE SERVICE
22/tcp open ssh
25/tcp filtered smtp
53/tcp open domain
80/tcp open http
113/tcp open auth
443/tcp open https
Nmap finished: 1 IP address (1 host up) scanned in 33.635 seconds
```

On peut ENFIN aller racker.

Ah, une dernière astuce, nous avons constaté que le temps d'attente des taxis couleur ciel est nettement plus élevé que celui des taxis couleur menthe. Bus rouge, nous voici !

La fin ?

Malheureusement, non. A l'issue de cette confi guration pleine de rebondissements, nous nous sommes aperçus que notre bête de course souff rait d'un problème hardware : un des disques montre des signes de faiblesse et nous fonctionnons en mode dégradé depuis quelques semaines. Aussi, est-il inutile de tenter un ping ~~zone0.gcu-squad.org~~ dans les semaines à venir, car notre bijou va bientôt partir chez son créateur afi n de subir quelques opérations chirurgicales. D'autre part, nous avons encore bien des expériences à mener, car ces aléas matériels m'ont privé du dernier jouet que j'avais prévu de mettre en place : un service de VoIP en totale interaction avec notre base LDAP...

Pour finir, j'aimerais profiter de cet article pour transmettre au mag' entier mes sincères félicitations pour cette 100ème, leur souhaiter longue vie, et les remercier pour les milliers (millions ? milliards ?) de lignes publiées, plus instructives et intéressantes les unes que les autres. Dédicace spéciale pour Denis, mais il sait déjà tout le bien que je pense de lui ;)

zone0 dans sa nouvelle demeure

Liens:

- Xen OpenSource : http://xen.org/
- Linux KVM : http://kvm.qumranet.com/kvmwiki
- QEMU (Fabrice Bellard Président) : http://fabrice.bellard.free.fr/qemu/
- NetBSD/xen HOWTO : http://www.netbsd.org/ports/xen/howto.html
- Virtualisation x86 : http://en.wikipedia.org/wiki/X86_virtualization
- LVM HOWTO : http://tldp.org/HOWTO/LVM-HOWTO/
- The Perfect Xen 3.1.0 Setup For Debian Etch (i386) : http://www.howtoforge.com/debian_etch_xen_3.1
- Diverses astuces sur la virtualisation : http://imil.net/wp/?tag=Virtualisation
- IPVS : http://www.linuxvirtualserver.org/software/ipvs.html
- La FAQ OpenBSD : http://www.openbsd.org/faq/fr/
- The NetBSD Guide : http://www.netbsd.org/docs/guide/en/
- OpenLDAP QuickStart : http://www.openldap.org/doc/admin24/quickstart.html
- PADL : http://www.padl.com/
- login-ldap HOWTO en français : http://wiki.gcu.info/doku.php?id=openbsd:login_ldap
- OpenSSH-lpk chez Inverse Path : http://dev.inversepath.com/trac/openssh-lpk
- bind-sdb-ldap : http://bind9-ldap.bayour.com/
- mod-vhost-ldap : http://modvhostldap.alioth.debian.org/
- The pkgsrc guide : http://www.netbsd.org/docs/pkgsrc/