HORS SÉRIE - HORS SÉRIE - HORS SÉRIE - HORS SÉRIE - HORS SÉRIE

GNU LINUX MAGAZINE / FRANCE

France Métro : 6,40€ - DOM 6,95€ - BEL : 7,30€ - LUX : 7,30€ - PORT. CONT. : 7,30€ - CH : 13FS - CAN : 12$ - MAR : 65DH

L 15066 - 30 H - F: 6,40 € - RD

Mai / Juin 2007

HORS SÉRIE N°30

Utilisation et administration avancées de FreeBSD, OpenBSD et NetBSD

VIRTUALISATION

Faites fonctionner vos *BSD dans Xen avec un dom0/hôte GNU/Linux.

CLOISONNEMENT ET PRISON

Utilisez jail, systrace et sysjail pour sécuriser un BSD et mettre les processus en cage.

SUPERVISION

Installez symon sur votre système et découvrez la supervision facile et sûre.

CHIFFREMENT

Chiffrez vos systèmes de fichiers avec CGD/NetBSD ou GELI/FreeBSD.

CAS CONCRET NETBSD

Découvrez à quoi ressemble un VRAI réseau personnel mêlant Wifi, routeur, OSPF, VPN, DMZ...

DÉVELOPPEMENT ET NOTIFICATION

Oubliez select et poll et passez au développement moderne en utilisant kqueue pour vos projets.

IPV6 FACILE

Lancez-vous dans IPv6 en compagnie de NetBSD, DragonFlyBSD et FreeBSD.

CRÉATION DE PAQUETS

Apprenez à créer des paquets et des ports pour OpenBSD et NetBSD.

FREEBSD NOMADE

Réalisez votre propre LiveCD avec un système FreeBSD.

Administration et développement sur systèmes UNIX

Nous avions omis de vous préciser une petite chose concernant le précédent hors-série. Je ne vous cache pas que je m'en veux un peu (mais juste un peu). D'un autre côté, il m'aurait été difficile de vous en parler dans le précédent édito, puisque c'est justement en rapport avec la lecture du magazine... Ah aaahh, voilà qui n'est pas des plus clairs.

Par la lecture du précédent numéro et de celui-ci, vous allez pouvoir accéder à un niveau de conscience supérieur, déjouer les plans des forces du mal (vous savez, les autres-là, pas nous quoi !) et revenir dans votre monde (??). Mais il existe une condition ou plutôt un rituel.

Avant de prendre le GCUmicon^Wmagazine sur le présentoir de votre buraliste adoré, il est impératif que vous prononciez les trois mots sacrés à haute et intelligible voix : « Klaatu ! Verata ! Nicto ! ».

Ce n'est qu'à cette condition que vous pourrez prendre connaissance, sans danger, des formules et incantations mystiques qui se cachent dans les pages qui suivent. Dans le cas contraire, vous risquez de réveiller l'armée des (non)-morts (http://undeadly.org/) et de déclencher une malédiction sans précédent (je vous rappelle que votre inconscience lors de la lecture du numéro précédent a déjà provoqué l'apparition de celui que vous tenez entre vos mains, c'est malin !).

Selon la légende, il paraîtrait même que la lecture du magazine n'est pas la seule action nécessitant d'effectuer le rituel. Ceci concernerait également la relecture, la mise en page, l'impression, etc.

Alors pour plus de sûreté...

« Klaatu ! Verata ! Ni... heu... Nib... Nil... »

« Je reprends.»

« Klaatu ! Verata ! N.... »

« hmmm... »

*« Klaatu ! Verata ! Ni..*kofh* *kofh* *kh*... »*

« VOILÀ ! Je l'ai dit ! »

Comme dirait Ash : « c'est facile, quand il n'y a que trois mots, je retiens ».

Denis Bodor

SOMMAIRE :

USER

SÉCURITÉ

ADMINISTRATION

DÉVELOPPEMENT

Linux Magazine France Hors Série
est édité par **Diamond Editions**
B.P. 20142 - 67603 Sélestat Cedex

Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09

E-mail :
cial@ed-diamond.com

Service commercial :
abo@ed-diamond.com

Site :
www.ed-diamond.com

Directeur de publication :
Arnaud Metzler

PRINTED IN Germany / Imprimé en allemagne / Dépôt légal : 3e Trimestre 1998 / N° ISSN : 1291-78 34 / Commission Paritaire : 09 03 K78 976 / Périodicité : Bimestrielle /
Prix de vente : 6,40 Euros

Rédacteur en chef :
Denis Bodor
Secrétaire de rédaction :
Véronique Wilhelm
Conception graphique :
Kathrin Troeger
Relecture :
Dominique Grosse
Responsable publicité :
Tél. : 03 88 58 02 08
Service abonnement :
Tél. : 03 88 58 02 08
Impression : VPM DRUCK /
www.vpm-druck.de
Distribution France :
(uniquement pour les dépositaires de presse)
MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04
Service des ventes : Distri-médias :
Tél. : 05 61 72 76 24

EMPAQUETTE-MOI LE CODAZ @#!@#!

C'est bien beau d'avoir fait un beau morceau de codaz, mais faut encore le distribuer. Et là, c'est le drame. Le _gens_ veut un clicka-truc, un tar.gz à compiler ça lui parle pas au gens. D'ailleurs, il ne sait pas compiler, faut lui mâcher le boulot. Ça, c'est la vision *aigr* du packaging.

La vision happy-bisounours, c'est qu'il y a un projet qui te tient à cœur, t'as envie d'aider les gentils développeurs à répandre la bonne parole (leur code, si tu suis bien mon raisonnement) et donc tu aimerais le mettre facilement à disposition des gens qui utilisent ton *BSD du bien.

C'est là que l'infrastructure des ports BSD entre en jeu. Et attention, elle va vous sortir le grand jeu. Alors vous savez déjà qu'il y a 3 « flavors » principales de BSD, donc nous sommes allés chercher pour vous les spécialistes en la matière pour chacune des flavors. Ou, du moins, on a essayé. Malheureusement pour les amateurs de la boule rouge, notre gourou préféré est tombé dans une embuscade (du Vim dans un fichier de 16 Go, ça vous tente ?)... Alors seuls Net et Open auront l'honneur de vous dévoiler leurs secrets intimes sous la houlette de talentueux magiciens. Non, ils n'ont pas appris sur le tas la semaine dernière en bricolant un bridge sur une Founera. Jamais, nulle part, avec personne.

GÉNÉRALITÉS

Tout se passe dans une arborescence dans le système de fichiers : `/usr/ports` pour Free et Open et `/usr/pkgsrc` pour Net. Dans les grandes lignes, un port est composé d'un BSD `Makefile` et de 2-3 trucs décrivant le paquet et les sources. Pour une description plus détaillée, se reporter à l'excellent article (quoiqu'un peu *outdated*) de M. Bedis dans le hors-série acte I, que tu peux commander ici (**http://ed-diamond.com/produit.php?produit=462**) si tu ne l'as pas encore. Grosso modo, pour installer un port via les sources, on fait `cd /usr/{ports,pkgsrc}/<category>/<port> && make install clean`. Bien évidemment, on peut utiliser les packages binaires mis à disposition dans les dépôts officiels, mais c'est un peu hors propos ici vu qu'on veut créer un port. De toute façon, un package binaire n'est jamais que le résultat de la compilation d'un port via `make package`.

PORT OPENBSD

AUVERVIOU

Rentrons tout de suite dans le vif du sujet et décortiquons un port existant (j'ai volontairement strippé les répertoires CVS qu'on retrouve dans un *ports-tree*) :

```
landry@renton:~ $ls -FR /usr/ports/archivers/unrar
Makefile distinfo patches/  pkg/

/usr/ports/archivers/unrar/patches:
patch-makefile_unix   patch-rar_hpp         patch-suballoc_cpp

/usr/ports/archivers/unrar/pkg:
DESCR PLIST
```

Comme on peut le voir ci-dessus, un port OpenBSD complet se compose d'un BSD `Makefile`, d'un fichier `distinfo` contenant différentes informations sur l'archive des sources du logiciel porté (taille, sommes md5/sha...), d'éventuels `patches`, d'un fichier `DESCR` décrivant en quelques lignes le logiciel, et enfin d'un fichier `PLIST` listant les fichiers installés par le logiciel.

Regardons tout d'abord comment se passe l'installation du logiciel, pour avoir la succession des différentes étapes :

```
landry@renton:/usr/ports/archivers/unrar/ $sudo make install clean
===>  Checking files for unrar-3.68p0
>> unrarsrc-3.6.8.tar.gz doesn't seem to exist on this system.
>> Fetch www.rarlab.com/rar/unrarsrc-3.6.8.tar.gz.
100% ||   122 KB    00:01
>> Size matches for /usr/ports/distfiles/unrarsrc-3.6.8.tar.gz
```

```
>> Checksum OK for unrarsrc-3.6.8.tar.gz. (sha1)
===>  Verifying specs:  c m stdc++ c m stdc++
===>  found c.40.3 m.2.3 stdc++.42.0
===>  Extracting for unrar-3.68p0
===>  Patching for unrar-3.68p0
===>  Configuring for unrar-3.68p0
===>  Building for unrar-3.68p0
c++ -O2 -pipe  -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -DUNRAR -c rar.cpp
....
strip unrar
===>  Faking installation for unrar-3.68p0
install -c -s -o root -g bin -m 555 /usr/ports/archivers/unrar/w-unrar-
3.68p0/unrar/unrar /usr/ports/archivers/unrar/w-unrar-3.68p0/fake-i386/usr/
local/bin
install -d -o root -g bin -m 755 /usr/ports/archivers/unrar/w-unrar-3.68p0/
fake-i386/usr/local/share/doc/unrar
install -c -o root -g bin -m 444 /usr/ports/archivers/unrar/w-unrar-3.68p0/
unrar/readme.txt /usr/ports/archivers/unrar/w-unrar-3.68p0/unrar/license.
txt  /usr/ports/archivers/unrar/w-unrar-3.68p0/fake-i386/usr/local/share/
doc/unrar
===>  Building package for unrar-3.68p0
Create /usr/ports/packages/i386/all/unrar-3.68p0.tgz
===>  Verifying specs:  c m stdc++
===>  found c.40.3 m.2.3 stdc++.42.0
===>  Installing unrar-3.68p0 from /usr/ports/packages/i386/all/
unrar-3.68p0: complete

===>  Cleaning for unrar-3.68p0
$
```

Quelques commentaires sur les différentes étapes...

- Le rapatriement des sources. C'est le résultat de `make fetch`, qui va vérifier si les sources du logiciel n'existent pas déjà dans `/usr/ports/distfiles` et, le cas échéant, les y téléchargera à partir du `${MASTER_SITES}` (que nous verrons plus tard).

- La vérification de l'intégrité des sources, par `make checksum`. On compare les sommes de contrôle contenues dans le fichier `distinfo` avec celles obtenues avec l'archive des sources présente dans `/usr/ports/distfiles`.

- La vérification des dépendances, par `make depends`. Va éventuellement installer les dépendances manquantes du logiciel, que ce soient des dépendances à l'exécution (`run-depends`), des dépendances à la compilation (`build-depends`) ou des dépendances sur des bibliothèques partagées (`lib-depends`). Il faut noter que les dépendances à l'exécution seront installées juste avant l'installation de notre port. Une dépendance à la compilation sera en général un compilateur, un script particulier (GNU make, `libtool`...), utilisé uniquement pendant la construction du *package*, alors qu'une dépendance sur une bibliothèque sera requise par le logiciel lui-même (p. ex. Gnome dépendra des bibliothèques gtk2).

- La décompression des sources, c'est l'équivalent de `make extract`. L'infrastructure des ports reconnaît la plupart des types d'archives (`.tar.gz`, `bz2`, `.Z`...). À partir de maintenant, tout va se passer dans `${WRKDIR}`, qui est un répertoire de travail situé dans le répertoire du port et nommé `w-${PKGNAME}` où `PKGNAME` est le nom complet de notre logiciel. Les sources sont extraites dans un sous-répertoire de `${WRKDIR}` nommé `${WRKDIST}`. Dans l'exemple pour `unrar`, les sources seront donc dans `/usr/ports/archivers/unrar/w-unrar-3.68p0/unrar`.

- Le *patching* de ces sources via `make patch`, avec les éventuels diffs trouvés dans le répertoire `patches/`. Ici, les patches sont nécessaires pour adapter les sources originales (`vanilla`) du logiciel et le faire fonctionner sur OpenBSD. Généralement, ce sont des patches de sécurité, des modifications de `Makefile`, des `#define` à gauche à droite... Dans un monde idéal, ces patches n'auraient pas lieu d'exister, et toutes les modifs seraient remontées à l'auteur du logiciel qui les aurait appliquées après vérification. Nous verrons que c'est rarement le cas, et que plein de gens codent rarement dans une optique code-portable-à-100%.

- La configuration des sources, faite par `make configure`. Ici, deux grandes écoles, les logiciels qui utilisent les GNU *autotools* et les autres. C'est à cette étape que les `Makefile` des sources seront créés par `./configure` si l'on utilise les autotools. Si l'infrastructure des ports trouve un fichier `scripts/configure`, elle l'exécutera en premier.

- La compilation des sources, équivalent de `make build`. Ici, rien de bien sorcier, on va dans le répertoire de travail, et on appelle `${MAKE_PROGRAM}`, qui sera soit BSD-make, soit GNU-make en fonction de `USE_GMAKE`. On y reviendra plus tard.

- La pseudo-installation, résultant de `make fake`. C'est une petite spécificité de l'infrastructure des ports OpenBSD, une pseudo-arborescence du système est créée par `mtree(8)` dans un sous-répertoire du répertoire de travail nommé `${WRKINST} (fake-${MACHTYPE})`, en utilisant le squelette `${PORTSDIR}/infrastructure/db/fake.mtree`. On va ensuite faire une vraie installation du logiciel dans cette pseudo-arborescence.

- La construction du package binaire proprement dit, par `make package`. Cette étape va créer le package `foo-version.tgz` à partir de la pseudo-arborescence, en y ajoutant un fichier +CONTENTS contenant diverses informations sur le package (la *packing-list*). Le fichier de description du logiciel `pkg/DESCR` est aussi ajouté au package, qui est créé dans `${PACKAGES_REPOSITORY}` (généralement `/usr/ports/packages`). Plus précisément, le package réel sera dans `${PACKAGES_REPOSITORY}/${MACHTYPE}/all`, et des liens symboliques seront créés dans des répertoires `cdrom/` et `ftp/` si la licence du logiciel permet sa libre redistribution.

➧ Enfin, on arrive à l'installation du logiciel sur le système, via `make install`. Cette cible va tout simplement appeler `pkg_add(1)` avec le package que nous venons de créer. Cette commande va tout d'abord vérifier qu'il n'est pas déjà installé, vérifier qu'il ne rentre pas en conflit avec d'autres packages, re-vérifier que les dépendances (d'exécution ET de bibliothèques) sont bien satisfaites, le décompresser effectivement dans `/usr/local` (aussi connu sous le petit nom de `${LOCALBASE}`), et enregistrer les informations connexes au package dans `/var/db/pkg/${FULLPKGNAME}` comme le contenu du package, sa description. Enfin, on enregistre les informations de dépendances directement dans les fichiers `/var/db/pkg/'PACKAGE_DONT_JE_DEPEND'/+REQUIRED_BY`.

➧ Et puis, vu qu'on n'est pas des gros gorets *__hruuuiiikk__*, on va faire un peu de nettoyage avec `make clean`. Par défaut, ça supprimera le répertoire de travail, mais on peut aussi supprimer le package créé via `make clean=package`, ainsi que les sources d'origine avec `make clean=dist`.

Voilà, on a fait le tour dans les grandes lignes de l'infrastructure de *build* d'un port OpenBSD. Sans plus attendre, passons à la pratique, car je sens que vous piaffez d'impatience...

Pour l'exemple, nous allons écrire un port pour un package fournissant une bibliothèque partagée et un port pour un logiciel qui a besoin de cette bibliothèque. Attention, c'est un vrai exemple en direct-live et tout, rien dans les manches, rien dans le chapeau !!

DÉFRICHONS LE TERRAIN

`<mode vie_réelle='ON'>` Bon, disons que j'ai installé un OpenBSD en *desktop-convi-clicka*, avec un `<prosel> joli Xfce4.4 </prosel>` grâce aux ports d'un certain monsieur Maniak. Et imaginons, soyons fous, que j'aie envie de faire un truc bête de gens, comme gérer mes comptes en banque à moi, quelle idée sottte-et-grenue. Un petit coup de googlage, j'apprends que sur les os du bien il existe GnuCash, KmyMoney, et Grisbi. Bon, direct j'élimine GnuCash, un peu trop usine à gaz à mon goût, y'a pas encore la version gtk2 dans les ports, et y'a des dépendances de malade genre tout GNOME, Audiofile et j'en passe... Vu que je connais déjà Grisbi sous minusque (oui, je triche), et que je n'ai généralement pas de libs Qt/KDE installées sur mon système, allez hop, Grisbi est le vainqueur de mon comparatif à-la-louche-totalement-impartial. Houbahop, direction `/usr/ports`.

```
landry@renton:/usr/ports/ $make search name=grisbi
landry@renton:/usr/ports/ $
```

mrrbllbmmrm... bon. Donc va falloir s'y coller. `<mode vie_réelle='OFF'>`

Hepaaa, vous avez vu comment j'ai bien amené mon exemple ? La classe américaine, j'vous dis.

Histoire de se faire un petit espace-à-soi-douillet dans le ports-tree, on va faire ce que nous conseille la doc : se créer un `/usr/ports/mystuff` avec le `chown moi:moi kivabien`. Histoire de ne pas être trop embêté par des histoires de droits, un pti `chmod -R +gw /usr/ports/{distfiles,packages}` permettra d'écrire aux bons endroits (si vous faites partie du groupe `wheel` évidemment) ; et enfin, un petit `echo 'SUDO=/usr/bin/sudo' >> /etc/mk.conf` dira au ports-tree d'utiliser `sudo` quand il le faudra (principalement pour l'installation).

Commençons par le commencement, faire marcher le `make fetch extract`. Un coup d'œil sur la *checklist*, et sur un autre port pris au hasard et on commence avec ça dans son `/usr/ports/mystuff/grisbi/Makefile` :

```
# $OpenBSD$
COMMENT =      "Personal accounting application"
VERSION =      0.5.9
DISTNAME =     grisbi-${VERSION}
CATEGORIES =   productivity
EXTRACT_SUFX = .tar.bz2
MASTER_SITES = ${MASTER_SITE_SOURCEFORGE:=grisbi/}
HOMEPAGE =     http://www.grisbi.org/
MAINTAINER =   Landry Breuil <gaston@gcu.info>

# GPL
PERMIT_PACKAGE_CDROM =   Yes
PERMIT_PACKAGE_FTP =     Yes
PERMIT_DISTFILES_CDROM = Yes
PERMIT_DISTFILES_FTP =   Yes
.include <bsd.port.mk>
```

C'est vraiment l'extra-minimum. `COMMENT` est une courte description de l'appli, `MASTER_SITES` et `DISTNAME` sont utilisés pour savoir où aller chercher les sources (j'ai précisé `EXTRACT_SUFX` car, par défaut, l'infrastructure des ports s'attend à un `.tar.gz`, mais les sources de Grisbi sont disponibles uniquement sous forme de `.tar.bz2`.) Bien renseigner ces champs, l'infrastructure des ports appellera `${FETCH_CMD}` (par défaut `ftp(1)`) avec `${MASTER_SITES}${DISTNAME}${EXTRACT_SUFX}` en argument. NE PAS oublier le / à la fin de `MASTER_SITES` !! Le champ `CATEGORY` est obligatoire aussi, il sert à savoir où va aller notre port. `HOMEPAGE` et `MAINTAINER` ne sont pas obligatoires pour le moment, mais hautement conseillés. Enfin, les `PERMIT_*` sont obligatoires, pour renseigner les conditions de redistribution des sources et des binaires. Ici, Grisbi est sous licence GPL, donc pas de problèmes. Et finalement, `.include <bsd.port.mk>` va charger le reste de l'infrastructure des ports.

Allez, hop, soyons fous, on teste :

```
landry@renton:/usr/ports/mystuff/grisbi/ $make fetch extract
===>  Checking files for grisbi-0.5.9
>> grisbi-0.5.9.tar.bz2 doesn't seem to exist on this system.
>> Fetch downloads.sourceforge.net/grisbi/grisbi-0.5.9.tar.bz2.
```

```
100% ||   979 KB    00:04
>> Checksum file does not exist
===>  Checking files for grisbi-0.5.9
'/usr/ports/distfiles/grisbi-0.5.9.tar.bz2' is up to date.
>> No checksum file.
* Error code 1
```

Erm, oui, évidemment, on a oublié un petit détail, le fichier `distinfo`. Un petit coup de `make makesum`, et hop, c'est réparé.

```
===>  Checking files for grisbi-0.5.9
'/usr/ports/distfiles/grisbi-0.5.9.tar.bz2' is up to date.
>> Checksum OK for grisbi-0.5.9.tar.bz2. (sha1)
===>  grisbi-0.5.9 depends on: bzip2-* - found
===>  Extracting for grisbi-0.5.9
```

Et voilà, il m'a créé le répertoire de travail, et décompressé les sources dans `w-grisbi-0.5.9/grisbi-0.5.9`. Maintenant, il faut préciser comment configurer ces sources. Grisbi utilisant les GNU autotools, il faut en informer l'infrastructure des ports. Tant qu'on y est, vu que c'est un logiciel clicka-convi, on va dire qu'on utilise X.

```
# configure stuff
CONFIGURE_STYLE = gnu
USE_GMAKE =       Yes
USE_X11 =         Yes
```

Hop, un coup de `make configure`, et les ennuis s'approchent à grands pas. Globalement, ça se passe bien, sauf pour ça :

```
checking libofx/libofx.h usability... no
checking libofx/libofx.h presence... no
checking for libofx/libofx.h... no
Libofx header missing. Check your libofx installation
```

Mh, il lui manque une bibliothèque, qui en plus n'existe pas dans le ports-tree. Elle n'est pas _requise_, mais elle permet à Grisbi de supporter le format OFX, qui est un format d'échange ouvert standardisé, beaucoup plus générique que le QIF, format de Quicken (que Grisbi sait lire aussi.) Et puis, ho-comme-de-par-masard, je voulais vous montrer comment écrire un port pour une bibliothèque, ça tombe bien. (*__rires enregistrés__*). Donc, pour l'instant, on laisse Grisbi de coté, et on s'attaque à `libofx` dans `/usr/ports/mystuff/libofx`.

ÇA SE COMPLIQUE, Y'A DES BRANCHES RÉCALCITRANTES

Comme pour le précédent, un `Makefile` de base, puis `make configure` (qui appellera tout seul comme un grand les cibles `fetch` et `extract`) :

```
# $OpenBSD$
COMMENT =      "library to access OFX files"
VERSION =      0.8.3
DISTNAME =     libofx-${VERSION}
CATEGORIES =   textproc
MASTER_SITES = ${MASTER_SITE_SOURCEFORGE:=libofx/}
HOMEPAGE =     http://libofx.sourceforge.net/
MAINTAINER =   Landry Breuil <gaston@gcu.info>

# GPL
PERMIT_PACKAGE_CDROM =   Yes
PERMIT_PACKAGE_FTP =     Yes
PERMIT_DISTFILES_CDROM = Yes
PERMIT_DISTFILES_FTP =   Yes

USE_LIBTOOL=             Yes
CONFIGURE_STYLE=         gnu
.include <bsd.port.mk>
```

Ici, petite spécificité, on a mis `USE_LIBTOOL` pour dire « on va créer une bibliothèque partagée avec GNU libtool ». Hop, `make configure`, et là c'est le drame.

```
checking for ParserEventGenerator.h in /usr/include/OpenSP... no
checking for ParserEventGenerator.h in /usr/local/include/OpenSP... no
checking for ParserEventGenerator.h in /usr/include/sp/generic... no
checking for ParserEventGenerator.h in /usr/local/include/sp/generic... no
checking for ParserEventGeneratorKit.h... no
configure: error: OpenSP includes not found
* Error code 1
```

Bon, ça veut dire quoi tout ça... Il veut OpenSP, qui est un *parseur* SGML, et qui, évidemment, n'est pas dans le ports-tree. Non, on ne va pas écrire un port pour OpenSP (j'ai essayé, il a pas voulu). Alors rusons un peu, il existe déjà un parseur SGML dans le ports-tree dans `textproc/sp`. On va voir si y'a pas moyen d'y mettre un p'tit coup de lime par-là, un p'tit coup de rabot par-ci, et de le faire rentrer au chausse-pied. En fait, on a du bol, ça marche, faut juste *tweaker* un bail le script `configure` de `libofx` pour qu'il teste la présence de `-lsp` au lieu de `-losp`.

```
# on fait un backup de l'original pour la création du patch
landry@renton:/usr/ports/mystuff/libofx/ $cp w-libofx-0.8.3/libofx-0.8.3/
configure{,.orig}
# on remplace avec son éditeur préféré -lsp par -losp.
landry@renton:/usr/ports/mystuff/libofx/ $vi w-libofx-0.8.3/libofx-0.8.3/
configure
...
landry@renton:/usr/ports/mystuff/libofx/ $make update-patches
Processing configure
No patch-* found for configure, creating patch-configure
# on nous propose d'éditer le patch... osef.
landry@renton:/usr/ports/mystuff/libofx/ $cat patches/patch-configure
$OpenBSD$
--- configure.orig      Thu Mar  8 00:02:48 2007
+++ configure   Thu Mar  8 00:07:12 2007
@@ -20639,7 +20639,7 @@ fi
 done
```

```
-OPENSPLIBS="-L$OPENSPLIBPATH -losp"
+OPENSPLIBS="-L$OPENSPLIBPATH -lsp"
 ac_save_LIBS="$LIBS"
 LIBS="$OPENSPLIBS $LIBS"
```

Hop, comme par magie, il nous a fait un patch tout beau !! On tweake un peu le `Makefile` pour passer les bonnes options à configure et corriger 2-3 trucs :

```
# ${LOCALBASE} correspond à /usr/local
# là où tous les ports sont installés sous OpenBSD
# => la source de la vie.
CONFIGURE_ENV=  LDFLAGS="-L${LOCALBASE}/lib -lm"
CONFIGURE_ARGS= ${CONFIGURE_SHARED} \
                --with-opensp-libs=${LOCALBASE}/lib \
                --with-opensp-includes=${LOCALBASE}/include/sp \
                --without-libcurl \
                --disable-doxygen \
                --disable-dot \
                --disable-gengetopt
```

Et hop, on peut faire `make clean patch configure build` ou, tout court, `make build`. Ça compile. Yallah @#!@#!. Maintenant, passons au packaging proprement dit. Pour ça, il faut renseigner `pkg/DESCR` avec 3-4 lignes expliquant le pourquoi du comment du biniou, et faire un petit coup de `make update-plist` pour qu'il crée la pré-packing-list du package. D'ailleurs :

```
landry@renton:/usr/ports/mystuff/libofx/ $make update-plist
===>  Updating plist for libofx-0.8.3
WARNING: unregistered shared lib: ofx (version: 3.1)
/usr/ports/mystuff/libofx/pkg/PLIST is new
/usr/ports/mystuff/libofx/pkg/PFRAG.shared is new
```

Normal, on a oublié de dire dans notre `Makefile` que ce port créait une bibliothèque partagée... `USE_LIBTOOL = Yes` ça suffit pas, on y ajoute ici `SHARED_LIBS = ofx 3.1`. Maintenant, on passe à l'étape « déclaration des papiers à la frontière », *aka* `make clean lib-depends-check`.

```
landry@renton:/usr/ports/mystuff/libofx/ $make clean lib-depends-check
===>  Checking files for libofx-0.8.3
`/usr/ports/distfiles/libofx-0.8.3.tar.gz' is up to date.
>> Checksum OK for libofx-0.8.3.tar.gz. (sha1)
===>  libofx-0.8.3 depends on: libtool-* - found
===>  Extracting for libofx-0.8.3
===>  Patching for libofx-0.8.3
===>  Configuring for libofx-0.8.3
.....
===>  Building for libofx-0.8.3
.....
===>  Faking installation for libofx-0.8.3
.....
===>  Building package for libofx-0.8.3
.....
/usr/ports/packages/i386/all/libofx-0.8.3.tgz:
Missing system lib: c.40 (/usr/local/bin/ofxdump)
Missing system lib: m.2 (/usr/local/bin/ofxdump)
Missing system lib: stdc++.42 (/usr/local/bin/ofxdump)
        WANTLIB += c m stdc++
```

Normal, un petit warning sur la fin, je n'ai déclaré aucune dépendance comme un gros sagouin. Même pas de `LIB_DEPENDS`. Théo, Deanna, Miod et Marc risquent de venir me lyncher en personne (*__kh0f__*), donc je corrige ça fissa :

```
# dependencies
SHARED_LIBS = ofx 3.1
WANTLIB =     c m stdc++
LIB_DEPENDS = ::textproc/sp
```

Et hop youp-la-boom, on peut faire `make install`, et avoir la joie de faire un `pkg_info libofx` et de voir son nom dans la ligne `Maintainer`. Enfin, c'est aussi hautement conseillé de faire un certain nombre de cycles `make clean && make update-patches && make lib-depends-check` pour bien triple-vérifier que y'a plus de trucs qui dépassent. Ceintures, bretelles et combinaison de décontamination sont les 3 mamelles du pantalon bien attaché. Relisez 172 fois la doc aussi.

ENCORE UN PETIT COUP DE TONDEUSE

Revenons à nos moutons, maintenant que nous avons réglé le cas de la branche qui dépassait. Hophop, `cd ../grisbi`. On a installé `textproc/libofx`, tentons maintenant de le faire détecter par le `./configure` de Grisbi.

```
# configure stuff
CONFIGURE_ENV = LDFLAGS="-L${LOCALBASE}/lib -lofx -lsp" \
   CPPFLAGS="-I${LOCALBASE}/include"
```

Ici, je triche un peu pour le forcer à trouver `libofx`, et pour des problèmes de résolution de symboles je rajoute `-lsp`, car la bibliothèque fournie par les ports étant statique, la résolution ne peut se faire toute seule... C'est en tâtonnant qu'on devient *khof* non rien. Hopla, du coup, `make clean build` passe comme une lettre à la poste. On remplit `pkg/DESCR`, on génère la packing-list...

```
landry@renton:/usr/ports/mystuff/grisbi/ $make update-plist
===>  Faking installation for grisbi-0.5.9
.....
===>  Updating plist for grisbi-0.5.9
/usr/ports/mystuff/grisbi/pkg/PLIST is new
```

Et on passe au nettoyage, avec un coup de `make lib-depends-check`... Vu qu'on n'a rien renseigné comme dépendances, il se plaint. Évidemment. Jamais content.

```
[22:54:23] landry@renton:/usr/ports/mystuff/grisbi/ $make lib-depends-
check
===>  Building package for grisbi-0.5.9
Create /usr/ports/packages/i386/all/grisbi-0.5.9.tgz
Link to /usr/ports/packages/i386/ftp/grisbi-0.5.9.tgz
Link to /usr/ports/packages/i386/cdrom/grisbi-0.5.9.tgz

/usr/ports/packages/i386/all/grisbi-0.5.9.tgz:
Missing system lib: X11.9 (/usr/local/bin/grisbi)
Missing system lib: Xext.9 (/usr/local/bin/grisbi)
Missing system lib: Xrender.4 (/usr/local/bin/grisbi)
Missing: atk-1.0.1011 (/usr/local/bin/grisbi): NOT REACHABLE
Missing system lib: c.40 (/usr/local/bin/grisbi)
Missing: cairo.5 (/usr/local/bin/grisbi): NOT REACHABLE
Missing system lib: fontconfig.3 (/usr/local/bin/grisbi)
Missing system lib: freetype.13 (/usr/local/bin/grisbi)
Missing: gdk-x11-2.0.802 (/usr/local/bin/grisbi): NOT REACHABLE
Missing: gdk_pixbuf-2.0.802 (/usr/local/bin/grisbi): NOT REACHABLE
Missing: glib-2.0.1000 (/usr/local/bin/grisbi): NOT REACHABLE
Missing: glitz.2 (/usr/local/bin/grisbi): NOT REACHABLE
Missing: gmodule-2.0.1000 (/usr/local/bin/grisbi): NOT REACHABLE
Missing: gobject-2.0.1000 (/usr/local/bin/grisbi): NOT REACHABLE
Missing: gtk-x11-2.0.802 (/usr/local/bin/grisbi): NOT REACHABLE
Missing: iconv.4 (/usr/local/bin/grisbi): NOT REACHABLE
Missing: intl.3 (/usr/local/bin/grisbi): NOT REACHABLE
Missing system lib: m.2 (/usr/local/bin/grisbi)
Missing: ofx.3 (/usr/local/bin/grisbi): NOT REACHABLE
Missing: pango-1.0.1200 (/usr/local/bin/grisbi): NOT REACHABLE
Missing: pangocairo-1.0.1200 (/usr/local/bin/grisbi): NOT REACHABLE
Missing: pangoft2-1.0.1200 (/usr/local/bin/grisbi): NOT REACHABLE
Missing: png.5 (/usr/local/bin/grisbi): NOT REACHABLE
Missing system lib: stdc++.42 (/usr/local/bin/grisbi)
Missing: xml2.9 (/usr/local/bin/grisbi): NOT REACHABLE
Missing system lib: z.4 (/usr/local/bin/grisbi)
       WANTLIB += X11 Xext Xrender c fontconfig freetype m stdc++ z
```

Il en manque un peu, on va faire ça en deux fois... On rajoute la ligne `WANTLIB` à notre `Makefile`, et on reteste un `make clean lib-depends-check`.

```
===>  grisbi-0.5.9 depends on: gettext->=0.14.6 - found
===>  grisbi-0.5.9 depends on: gmake-* - found
===>  grisbi-0.5.9 depends on: bzip2-* - found
===>  grisbi-0.5.9 depends on: gettext->=0.10.38 - found
===>  grisbi-0.5.9 depends on: libiconv-* - found
===>  Verifying specs: intl.>=3 iconv.>=4 intl.>=3 iconv.>=4 X11 Xext
Xrender c fontconfig freetype m stdc++ z X11 Xext Xrender c fontconfig
freetype m stdc++ z
===>  found intl.3.0 iconv.4.0 X11.9.0 Xext.9.0 Xrender.4.1 c.40.3
fontconfig.3.0 freetype.13.1 m.2.3 stdc++.42.0 z.4.1
===>  Extracting for grisbi-0.5.9
...
```

C'est déjà beaucoup mieux, il résout les dépendances vers les bibliothèques du *basesystem*. Au passage, j'ai rajouté `MODULES = devel/gettext`, car le logiciel se base sur `gettext` pour la gestion des langues... Le mot-clé `MODULES` sert à « activer » un des modules de l'infrastructure des ports situés dans `/usr/ports/infrastructure/mk`.

Bon, par contre, tous ces `NOT REACHABLE`, ça jure... Ce sont les dépendances envers des bibliothèques installées par le ports-tree que l'on a « oublié » ***khof***. Allez hop, on cherche un peu dans son arbre, on bricole, on scotche, on lime, on meule... Et on nettoie ça. Après quelques incantations vaudou et en lisant soigneusement `bsd.port.mk(5)`, j'ai fini par trouver `library-specs(7)` qui m'a expliqué la syntaxe d'une déclaration de dépendance.

```
# dependencies
MODULES = devel/gettext

WANTLIB = X11 Xext Xrender c fontconfig freetype m stdc++ z \
   atk-1.0 cairo glib-2.0 glitz gmodule-2.0 gobject-2.0 \
   pango-1.0 pangocairo-1.0 pangoft2-1.0 png

LIB_DEPENDS = xml2.>=9::textproc/libxml \
   gdk-x11-2.0,gdk_pixbuf-2.0,gtk-x11-2.0::x11/gtk+2 \
   ofx.>=3::devel/libofx
```

Je suis sûr que vous allez vous poser la question héé pourquoi y'a truc dans `LIB_DEPENDS` et machin dans `WANTLIB` ?? Ils sont cons chez OpenBSD, ça fait double emploi !!. Je me la suis posée aussi. En fait, `LIB_DEPENDS` est plutôt pensé pour les dépendances « directes », ici `libofx/gtk2/libxml2`, et `WANTLIB` pour les dépendances « indirectes », *aka.* les dépendances des dépendances... Pas bête, non ?

Hop, on reteste...

```
landry@renton:/usr/ports/mystuff/grisbi/ $make lib-depends-check

===>  Checking files for grisbi-0.5.9
`/usr/ports/distfiles/grisbi-0.5.9.tar.bz2' is up to date.
>> Checksum OK for grisbi-0.5.9.tar.bz2. (sha1)
===>  Verifying specs: atk-1.0 gmodule-2.0 gobject-2.0 glib-2.0 pango-1.0
pangocairo-1.0 pangoft2-1.0 gdk-x11-2.0 gdk_pixbuf-2.0 gtk-x11-2.0 cairo
glitz png xml2 ofx intl.>=3 iconv.>=4 Xext Xrender c fontconfig freetype m
stdc++ z
===>  found atk-1.0.1011.3 gmodule-2.0.1000.3 gobject-2.0.1000.3 glib-
2.0.1000.3 pango-1.0.1200.3 pangocairo-1.0.1200.3 pangoft2-1.0.1200.3 gdk-
x11-2.0.802.1 gdk_pixbuf-2.0.802.1 gtk-x11-2.0.802.1 cairo.5.0 glitz.2.0
png.5.1 xml2.9.3 ofx.3.1 intl.3.0 iconv.4.0 X11.9.0 Xext.9.0 Xrender.4.1
c.40.3 fontconfig.3.0 freetype.13.1 m.2.3 stdc++.42.0 z.4.1
===>  Extracting for grisbi-0.5.9
===>  Patching for grisbi-0.5.9
===>  Configuring for grisbi-0.5.9
...
===>  Building package for grisbi-0.5.9
Create /usr/ports/packages/i386/all/grisbi-0.5.9.tgz
Link to /usr/ports/packages/i386/ftp/grisbi-0.5.9.tgz
Link to /usr/ports/packages/i386/cdrom/grisbi-0.5.9.tgz

/usr/ports/packages/i386/all/grisbi-0.5.9.tgz:
```

Il ne dit rien...Victoire !!!! Bon évidemment, j'avais toutes les dépendances nécessaires installées, mais vous avez compris le principe. Reste plus qu'à faire `make install`,

relire encore 172 fois la doc, faire un cycle de `make clean && make lib-depends-check && make install`... Faut pas se louper, sinon on est grillé à vie après.

/* FIXME: TODO !! */

Bon, évidemment, tout ceci était mon interprétation personnelle de la doc, la doc est belle, espie@ est grand, la doc fait revenir l'être aimé, la doc rend riche, la doc rajeunit ton bide, il _faut_ lire la doc. Ça va, j'insiste pas trop ? Parmi les choses qu'il resterait à aborder, il y a les FLAVORS, les MULTI-PACKAGES, la création d'un port n'utilisant pas les autotools, l'installation d'un *daemon*, etc. Pour des infos sur tout ça, il faudra écouter Radio Trouville FM –clin d'œil entendu–.

REAL MEN DON'T DO BACKUP, THEY UPLOAD THEIR STUFF ON THE NET AND LET OTHERS MIRROR IT

Voilà, j'ai fait le tour de ce que je connaissais/maîtrisais, à vous la joie de l'écriture des ports OpenBSD... Une fois que votre port est prêt, reste plus qu'à en faire profiter la communauté en envoyant une annonce sur ports@openbsd.org avec un .tar.gz de votre port complet, et surtout appliquer les éventuelles corrections qui seraient remontées par des testeurs (au passage, merci à Antoine Jacoutot pour avoir jeté un œil à mes ports, les avoir nettoyés ET *commités*, et enfin avoir bien voulu répondre à mes questions. Comme quoi, chez Open, ils mangent pas les petits n'enfants). Communiquez aussi avec les auteurs du logiciel, remontez-leur d'éventuels bugs qui seraient ressortis de votre étape de *porting*, bref, soyez actif, mettez à jour votre port pour les nouvelles versions, etc. !! Si vous persévérez, peut-être qu'un jour, votre petit bout de contribution infime trouvera aussi son chemin dans le ports-tree officiel !

–gaston–

LINQSES

- man (5) bsd.port.mk
  http://www.openbsd.org/cgi-bin/man.cgi?query=bsd.port.mk
- man (7) ports
  http://www.openbsd.org/cgi-bin/man.cgi?query=ports
- Écrire un port
  http://www.openbsd.org/fr/porting.html
- Tester un port
  http://www.openbsd.org/fr/porttest.html
- Checklist
  http://www.openbsd.org/fr/checklist.html
- Archive de ports@openbsd.org
  http://marc.info/?l=openbsd-ports
- Le port de libofx
  http://ports.openbsd.nu/devel/libofx
- Le port de grisbi
  http://ports.openbsd.nu/productivity/grisbi

PKGSRC NETBSD

INTRODUCTION

Un bon exemple vaut mieux qu'un long discours !

Je vais donc vous décrire comment porter un logiciel dans pkgsrc en procédant par l'exemple.

Tout le monde connaît l'excellent vlock, qui permet de *locker* un terminal. Il n'est pas présent dans pkgsrc, nous allons donc le porter !

CONSTRUCTION DU NID

C'est la première étape !

Vous allez tout d'abord installer url2pkg :

```
# cd /usr/pkgsrc/pkgtools/url2pkg
# make install clean
```

Cet outil va nous permettre de construire la base de notre package pkgsrc. On va tout d'abord se créer un petit nid, qui accueillera notre jeune pousse.

```
# mkdir -p /usr/pkgsrc/wip/vlock
```

Maintenant que le nid est prêt, on va y mettre les meubles :

```
# cd /usr/pkgsrc/wip/vlock
# url2pkg http://www.bedis.eu/NetBSD/vlock-1.3.tar.gz
```

NOTE

Il est recommandé d'utiliser le *repository* des sources officielles. Seulement, pour vlock, y'en a pas, alors système D...

Voyons un peu nos meubles :

```
# ls -l
total 10
-rw-r--r--  1 root  wheel    0 Mar 10 01:22 DESCR
-rw-r--r--  1 root  wheel  213 Mar 10 01:23 Makefile
```

Pour ça, sauvegardez le fichier courant et remplacez par ce contenu :

```
# vlock makefile

CFLAGS += -DUSE_PAM
LDFLAGS = -lpam

OBJS = vlock.o signals.o help.o terminal.o input.o

all: vlock

vlock.man: vlock.1
	groff -man -Tascii vlock.1 > vlock.man

vlock: $(OBJS)
	cc $(OBJS) $(LDFLAGS) -o vlock

clean:
	rm -f ${OBJS} vlock vlock.core
```

et un dernier `make` pour vérifier que tout se passe bien :

```
$ make
cc -O2 -DUSE_PAM  -c vlock.c
cc -O2 -DUSE_PAM  -c signals.c
cc -O2 -DUSE_PAM  -c help.c
cc -O2 -DUSE_PAM  -c terminal.c
cc -O2 -DUSE_PAM  -c input.c
cc vlock.o signals.o help.o terminal.o input.o -lpam -o vlock
```

\o/ Ça compile !!!!

La chambre du petit

Bon, on va mettre à jour notre package.

Dans le répertoire du package, créez un répertoire `patches`.

```
# mkdir /usr/pkgsrc/wip/vlock/patches
```

Depuis le répertoire où l'on a travaillé notre bébé, préparez le papier peint :

```
diff -bu vlock.c.orig vlock.c > /usr/pkgsrc/wip/vlock/patches/patch-aa
diff -bu signals.c.orig signals.c > /usr/pkgsrc/wip/vlock/patches/patch-ab
diff -bu terminal.c.orig terminal.c > /usr/pkgsrc/wip/vlock/patches/patch-ac
diff -bu input.c.orig input.c > /usr/pkgsrc/wip/vlock/patches/patch-ad
diff -bu Makefile.orig Makefile > /usr/pkgsrc/wip/vlock/patches/patch-ae
```

Et on fait les finitions. Exécutez cette commande dans le répertoire du package :

```
# make makepatchsum
```

Ca va permettre de mettre à jour le fichier `distinfo` avec les hash des patches :

```
$ cat distinfo
$NetBSD$

SHA1 (vlock-1.3.tar.gz) = 5da5f905bbba1c8dbcae0513668c46c908a1089e
RMD160 (vlock-1.3.tar.gz) = 434ec6613476efb9386597ac44857d993fb70d47
Size (vlock-1.3.tar.gz) = 17188 bytes
SHA1 (patch-aa) = f827d27a07d17cb92bbb57a40d0a729d56da097a
SHA1 (patch-ab) = 58811ac2eab370a1113710d469ec9162fb5846b5
SHA1 (patch-ac) = 84c2b134db608f241f18b9ea890040f5ad49c3c7
SHA1 (patch-ad) = 6c3207a72ba91795e60d30920e2f17c76c58fb6d
SHA1 (patch-ae) = ad7ae98a10f5c8d53fa6599ce90e3aa89575fead
```

RENTRÉE DES CLASSES

C'est pas le tout de faire des patches, encore faut-il que notre package s'installe proprement.

On va modifier légèrement le `Makefile` généré par `url2pkg` en rajoutant la phase d'installation :

```
do-install:
	${INSTALL_PROGRAM} -c -s -o root -g wheel -m 4555 ${WRKDIR}/
${DISTNAME}/vlock ${PREFIX}/bin
	${INSTALL_MAN} ${WRKSRC}/vlock.1 ${PREFIX}/${PKGMANDIR}/man1
```

Maintenant, on teste l'installation. Exécuter directement dans le répertoire du package :

```
$ sudo make install
=> Required installed package digest>=20010302: digest-20060826 found
===> Skipping vulnerability checks.
WARNING: No /usr/pkgsrc/distfiles/pkg-vulnerabilities file found.
WARNING: To fix, install the pkgsrc/security/audit-packages
WARNING: package and run: ``/usr/pkg/sbin/download-vulnerability-list''.
=> Fetching vlock-1.3.tar.gz
=> Total size: 17188 bytes
Requesting http://www.bedis.eu/NetBSD/vlock-1.3.tar.gz
100% |*****************************************************| 17188
84.76 KB/s    00:00 ETA
17188 bytes retrieved in 00:00 (84.69 KB/s)
=> Checksum SHA1 OK for vlock-1.3.tar.gz
=> Checksum RMD160 OK for vlock-1.3.tar.gz
===> Installing dependencies for vlock-1.3
===> Overriding tools for vlock-1.3
===> Extracting for vlock-1.3
===> Patching for vlock-1.3
=> Applying pkgsrc patches for vlock-1.3
===> Creating toolchain wrappers for vlock-1.3
===> Configuring for vlock-1.3
===> Building for vlock-1.3
cc -O2 -DUSE_PAM  -c vlock.c
cc -O2 -DUSE_PAM  -c signals.c
cc -O2 -DUSE_PAM  -c help.c
cc -O2 -DUSE_PAM  -c terminal.c
cc -O2 -DUSE_PAM  -c input.c
```

```
cc vlock.o signals.o help.o terminal.o input.o -lpam -o vlock
=> Unwrapping files-to-be-installed.
===> Installing for vlock-1.3
/usr/bin/install -c -s -o root -g wheel -m 555 -c -s -o root -g wheel -m
4555 /usr/pkgsrc/wip/vlock/work/vlock-1.3/vlock /usr/pkg/bin
/usr/bin/install -c -o root -g wheel -m 444 /usr/pkgsrc/wip/vlock/work/
vlock-1.3/vlock.1 /usr/pkg/man/man1
=> Automatic manual page handling
=> Registering installation for vlock-1.3
```

Le premier jour d'école du petit s'est bien déroulé. Il est dans la classe, avec ses potes.

```
$ pkg_info vlock
Information for vlock-1.3:

Comment:
Terminal lock

Description:

Homepage:
http://www.bedis.eu/NetBSD/
```

Bon, si on utilise `pkg_delete` pour supprimer notre `vlock`, les fichiers ne seront pas supprimés. Il va falloir améliorer le `Makefile`.

LES PREMIERS COURS

Bon, c'est bien, on a vu par l'exemple comment créer un package dans `pkgsrc`. Un exemple sans la théorie, c'est comme un bon fromage sans le vin. Il y a un manque !

Je vais donc m'efforcer de combler ce manque ici.

Présentation de pkgsrc

Pour utiliser un logiciel sur un Unix, il faut récupérer les sources, installer des dépendances si besoin, le compiler, etc. Ce n'est pas une mince affaire !

Sous NetBSD, toutes ces opérations sont gérées dans `pkgsrc`. `pkgsrc` n'est autre qu'un arbre dans lequel on retrouve nos logiciels classés en catégories. Pour chacun de ces logiciels, il existe des directives de compilation, des patches, etc.

Chaque logiciel dans l'arborescence de `pkgsrc` est appelé « package ».

On peut bien sûr s'affranchir de la compilation en utilisant les versions binaires des packages.

Description d'un package

Un package est donc un logiciel « porté » dans NetBSD. On a vu qu'il devait permettre à ce logiciel de s'installer et, pour cela, il doit répondre à certaines questions :

- Mais quel est donc ce logiciel porté ? -> un fichier `DESCR` contient un petit blabla sur le logiciel.
- Comment récupérer les sources, compiler et gérer les dépendances ? -> un fichier `Makefile` (et fait bien d'autres choses encore).
- Un logiciel, c'est bien, mais quels sont les fichiers installés ? -> tout est dans le fichier `PLIST`
- HAHA, facile de récupérer les sources, mais qui me dit qu'elles n'ont pas été corrompues ? -> le fichier `distinfo` contient les hashes des sources et des patches (pour plus de garantie !!!).
- Ben et si le logiciel est écrit pour un autre OS du bien, mais qu'il ne compile pas ? -> le répertoire patches est là pour ça !

Ces fichiers peuvent être complétés par ceux ci-dessous, qui sont OPTIONNELS :

- `INSTALL` : appelé par `pkg_add` à l'installation du package ;
- `DEINSTALL` : appelé par `pkg_delete` à la désinstallation du package ;
- `MESSAGE` : le contenu de ce fichier sera affiché après l'installation du package ;
- `README` : purement informatif ;
- `TODO` : idem `README`.

Vous pouvez utiliser `pkgtools/pkglint` pour être sûr que votre package est bien conforme au cahier des charges de `pkgsrc`. L'exemple de `vlock` étudié plus tôt comporte quelques non-conformités (fichiers `PLIST` et `DESCR` vides, notamment), mais comme vous l'avez vu, ça ne l'empêche pas de se compiler et de s'installer.

Bref, `pkglint` est un plus pour rendre un travail propre et soigné !

Le Makefile

C'est le fichier le plus important du package.

Building, installation and creation of a binary package are all controlled by the package's Makefile. The Makefile describes various things about a package, for example from where to get it, how to configure, build, and install it.

C'est grâce à ce fichier que `pkgsrc` saura comment compiler, installer et créer un package binaire. Le `Makefile` décrit divers paramètres du package.

Voici les éléments que doit obligatoirement contenir ce fichier :

- `DISTNAME` : c'est le *basename* du package qui sera téléchargé sur le site du package.

- PKGNAME : c'est le nom du package utilisé dans pkgsrc. A fournir uniquement si DISTNAME n'est pas un nom acceptable dans pkgsrc (doublon ou autre...).
- CATEGORIES : liste de catégories auxquelles le package correspond.
- MASTER_SITES : site où récupérer les sources.
- MAINTAINER : qui gère le package dans pkgsrc ?
- HOMEPAGE : site web originel.
- COMMENT : une ligne de commentaire qui décrit brièvement le package.
- PATCHFILES : patches additionnels à récupérer sur PATCH_SITES.
- PATCH_SITES : où chercher les PATCHFILES s'ils ne sont pas trouvés localement.

Mais on peut aussi avoir besoin de choses comme ça :

- CONFLICTS : liste de packages avec lesquels notre package peut entrer en conflit ;
- GNU_CONFIGURE : est-ce que ce package utilise le configure mode GNU ?
- CONFIGURE_ARGS : arguments à passer lors du configure ;
- EXTRACT_SUFX : va permettre de savoir comment décompresser l'archive ;-p
- MAKE_FILE : permet d'indiquer un autre nom que le Makefile classique... ;
- DEPENDS : package requis pour l'installation de notre package.

distinfo

Ce fichier contient les *checksums* de tous les fichiers nécessaires à la construction du package : fichiers des sources et des patches.

On peut regénérer ce fichier avec la commande :

```
# make makedistinfo
```

Les patches

Comme nous l'avons vu précédemment, la plupart des packages ne compilent pas, une fois sortis du four !

Afin de pouvoir utiliser lesdits packages sur NetBSD, il faudra les compiler en utilisant les fichiers dans le répertoire patches/.

Dans ce répertoire, les noms de fichiers doivent être de la forme patch-XX où X est une lettre minuscule. Chaque patch doit être au format diff -bu et ne s'appliquer qu'à un seul fichier source.

Comment générer les patches simplement ?

Placez-vous dans le répertoire de votre package dans l'arbre pkgsrc. Vous devez avoir vos sources dans un répertoire work.

Comme nous l'avons fait pour tout notre exemple, faire une copie de chaque fichier original que vous modifierez lors du portage du package sur NetBSD.

```
# cp monsource.c monsource.c.orig
```

Pour gagner du temps, on peut utiliser pkgvi pour modifier le fichier : il va lui-même créer le .orig et ouvrir un beau vi sur le fichier à éditer, puis va créer le patch dans work/.newpatches.

Pour profiter de la magie de pkgvi, installez le package pkgtools/pkgdiff.

(Merci à Gui²)

Maintenant, il suffit d'exécuter la commande suivante :

```
/usr/pkgsrc/wip/vlock$ sudo mkpatches -d ./patches/ -v
patch-aa -> /usr/pkgsrc/wip/vlock/work/vlock-1.3/Makefile
patch-ab -> /usr/pkgsrc/wip/vlock/work/vlock-1.3/input.c
patch-ac -> /usr/pkgsrc/wip/vlock/work/vlock-1.3/signals.c
patch-ad -> /usr/pkgsrc/wip/vlock/work/vlock-1.3/terminal.c
patch-ae -> /usr/pkgsrc/wip/vlock/work/vlock-1.3/vlock.c
```

C'est *automagically*. Les patches sont générés automatiquement, et placés dans le répertoire adéquat.

Ne pas oublier un petit make distinfo.

Description

Un fichier DESCR doit aussi être renseigné. Il est là pour décrire le logiciel de manière détaillée.

PLIST

Ce fichier contient la liste de tous les binaires, fichiers de configuration, manuels, etc. installés par notre package. Encore une fois, on peut le générer facilement :

```
/usr/pkgsrc/wip/vlock$ sudo make install
=> Required installed package digest>=20010302: digest-20060826 found
===> Skipping vulnerability checks.
WARNING: No /usr/pkgsrc/distfiles/pkg-vulnerabilities file found.
WARNING: To fix, install the pkgsrc/security/audit-packages
WARNING: package and run: ``/usr/pkg/sbin/download-vulnerability-list
=> Checksum SHA1 OK for vlock-1.3.tar.gz
```

```
=> Checksum RMD160 OK for vlock-1.3.tar.gz
===> Installing for vlock-1.3
/usr/bin/install -c -s -o root -g wheel -m 555 -c -s -o root -g wheel
-m 4555 /usr/pkgsrc/wip/vlock/work/vlock-1.3/vlock /usr/pkg/bin
/usr/bin/install -c -o root -g wheel -m 444 /usr/pkgsrc/wip/vlock/
work/vlock-1.3/vlock.1 /usr/pkg/man/man1
=> Automatic manual page handling
=> Registering installation for vlock-1.3

/usr/pkgsrc/wip/vlock$ sudo make print-PLIST
@comment $NetBSD$
bin/vlock
man/man1/vlock.1
```

Pour générer ce fichier, redirigez la sortie du `make print-PLIST` vers `PLIST` !

Il y aurait beaucoup à dire sur `PLIST`. Pour plus d'info, lisez ceci : **http://www.netbsd.org/Documentation/pkgsrc/plist.html**.

Fichiers optionnels

Notre package peut contenir quelques fichiers optionnels pour réaliser différentes tâches :

- `INSTALL` : script *shell* exécuté à l'installation du package (peut servir pour faire un peu de configuration post-installation).
- `DEINSTALL` : script exécuté avant et après que les fichiers soient supprimés. Utile pour supprimer d'éventuels fichiers qui n'ont pas été installés par le package, mais qui lui sont utiles.
- `MESSAGE` : le contenu de ce fichier est affiché une fois que le package est installé. Il permet de donner quelques dernières informations à l'utilisateur.

MAIS IL EST OÙ LE PAQUET ????

La dernière étape, une fois que le logiciel porté compile et s'installe proprement, c'est de créer le package. Pour ça, simplement :

```
/usr/pkgsrc/wip/vlock$ sudo make package
=> Required installed package digest>=20010302: digest-20060826 found
===> Skipping vulnerability checks.
WARNING: No /usr/pkgsrc/distfiles/pkg-vulnerabilities file found.
WARNING: To fix, install the pkgsrc/security/audit-packages
WARNING: package and run: ``/usr/pkg/sbin/download-vulnerability-list''.
=> Checksum SHA1 OK for vlock-1.3.tar.gz
=> Checksum RMD160 OK for vlock-1.3.tar.gz
===> Building binary package for vlock-1.3
Creating package /usr/pkgsrc/packages/All/vlock-1.3.tgz
Using SrcDir value of /usr/pkg
```

Maintenant, on peut installer le package binaire grâce à `pkg_add`. À partir de là, vous savez comment faire...

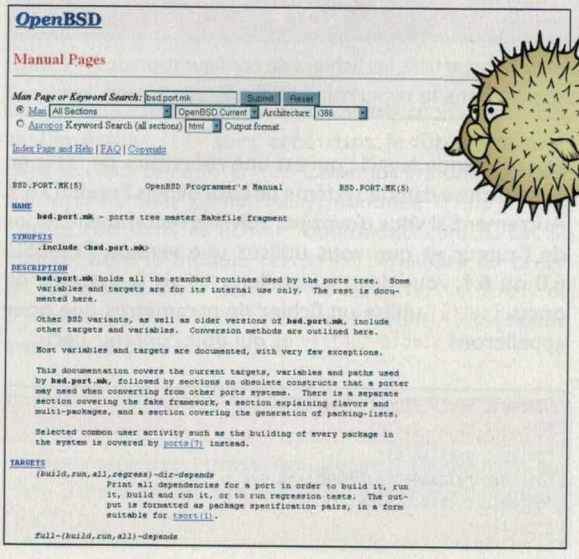
LIENS

Quelques exemples de Makefile :

- **http://imil.net/NetBSD/Makefile**
- **http://imil.net/NetBSD/gcc-pkg-Makefile**

La documentation sur pkgsrc :

- **http://www.netbsd.org/Documentation/pkgsrc/**
- **http://www.netbsd.org/Documentation/software/packages.html**



RÉALISER SON PROPRE LIVECD AVEC UN SYSTÈME FREEBSD

Le LiveCD est un système qui ne nécessite aucune installation. C'est l'outil idéal pour démarrer un ordinateur pour y faire des tests de compatibilité pour une future installation, effectuer des tâches d'administration sans modification du système d'exploitation installé ou même faire du client léger en utilisant un client RDP.

Pour la création de ce LiveCD, nous allons utiliser les scripts mis à disposition par le projet FreeSBIE. L'ensemble des étapes à suivre pour la création du LiveCD se fera sur un sytème FreeBSD 6.2 (qui devra être installé au préalable) et des sources FreeSBIE.

INTRODUCTION À FREESBIE

FreeSBIE est le fruit de l'effort du BUG (*BSD user group*) italien GUFI. Ce projet met à disposition un LiveCD et une « boîte à outils » permettant de créer son propre LiveCD. Suite à des limitations dues à la conception, en 2005 au SOC, a été proposée une réécriture complète de l'ensemble des scripts, ce qui a donné au final FreeSBIE 2. Cette dernière version est très proche sur le mode fonctionnement à la compilation du système FreeBSD en utilisant l'outil `make` et aussi la possibilité de créer ses propres *plugins*. Bien entendu, FreeSBIE est sous licence BSD.

PRÉ-REQUIS

Avant de s'attaquer à la création du LiveCD, il faut, dans un premier temps, récupérer les sources FreeBSD qui contiennent l'ensemble des commandes du système de base et du noyau. L'ensemble des sources est disponible. La méthode que nous allons utiliser pour télécharger les sources est CVSUP.

NOTE

Dans la suite de cet article, les fichiers de configuration de FreeSBIE seront stockés dans le répertoire de l'utilisateur *root*.

La commande `csup(1)` qui est une réécriture de `cvsup` en C est incluse dans le système de base depuis FreeBSD 6.2. Autrement, si vous n'avez pas suivi les recommandations de l'auteur et que vous utilisez une version FreeBSD 6.0 ou 6.1, vous devrez installer `csup` via le système de ports. `Csup(1)` utilise un fichier de paramètres que nous appellerons `stable-supfile` et qui doit contenir ceci :

```
*default host=cvsup.fr.FreeBSD.org
*default base=/var/db
*default prefix=/usr
*default release=cvs tag=RELENG_6_2
*default delete use-rel-suffix
src-all
```

Pour plus d'informations sur ces paramètres, veuillez consulter la page de manuel `csup(1)`. Ce qu'il faut retenir de ce fichier de configuration est le tag=RELENG_6_2 qui correspond à la version des fichiers de FreeBSD 6.2-RELEASE.

Maintenant que le fichier de configuration est prêt, la récupération des sources se fait avec la commande suivante :

```
csup stable-supfile
[beaucoup de lignes]
```

A la fin de cette commande, vous devriez avoir les sources dans le répertoire `/usr/src`.

Il ne reste plus qu'à récupérer FreeSBIE 2. Contrairement à la méthode utilisée pour récupérer les sources de FreeBSD à l'aide de `csup(1)`, nous allons utiliser `cvs(1)`. La commande `cvs(1)` est disponible dans le système de base et ne nécessite aucun fichier de configuration. Avant de continuer, un répertoire de travail est nécessaire :

```
mkdir ~/livecd
```

Puis, pour récupérer les sources :

```
cd ~/livecd
cvs -d :pserver:anonymous@cvs.freesbie.org:/cvs login
[pas de mot passe, tapez juste entrée]
cvs -z3 -d :pserver:anonymous@cvs.freesbie.org:/cvs co -P
freesbie2
```

À la suite de ces commandes, le répertoire `freesbie2` sera créé et contiendra l'ensemble des fichiers nécessaires.

Pour effectuer une mise à jour, vous devrez vous placer dans le répertoire `~/livecd/freesbie2` et exécuter cette commande :

```
cvs -z3 update -dP
```

Maintenant que nous avons ce qu'il nous faut, passons à la configuration de FreeSBIE 2.

ARCHITECTURE

Avant d'aller plus loin dans la configuration, il convient de décrire brièvement l'architecture.

FreeSBIE est composé de trois répertoires principaux :

- `conf/` contient les fichiers de configuration de FreeSBIE et du *kernel* (i386 ou amd64) ;
- `extra/` contient des plugins comme `adduser` qui permet d'ajouter un utilisateur `freesbie` ;
- `scripts/` contient les différents scripts *shell* permettant de créer le LiveCD (comme effectuer une compilation du système de base ou créer la future image ISO à graver de notre LiveCD).

Comme nous l'avons décrit dans l'introduction, FreeSBIE 2 utilise la commande `make(1)`. Cette commande lit le fichier `Makefile`, qui est le pivot du système. Ce fichier décrit le fonctionnement, ce qui doit être exécuté. Par la suite, nous utiliserons des cibles pour la création du LiveCD.

Il y a deux types de cibles, principales (que nous utiliserons) et indirectes. Les cibles principales lancent les cibles indirectes. Plus tard, nous utiliserons la cible principale `iso` qui lancera plusieurs cibles indirectes dont `buildworld`, `buildkernel`, `installworld`, `installkernel` et bien d'autres. On retrouve des cibles similaires à celles utilisées par FreeBSD pour la compilation et recompilation.

Voici un récapitulatif des cibles principales et de leur utilisation :

- `iso` : création d'un fichier ISO à graver ;
- `flash` : installation de FreeBSD sur une clef USB ;
- `img` : création d'un fichier IMG à copier sur une clef USB ;
- `pkgselect` : permet de sélectionner les paquets à copier avec une interface type `curse` en `dialog(1)` ;
- `clean` : suppression des fichiers de travail FreeSBIE ;
- `cleandir` : suppression du répertoire du futur système (par défaut : `/usr/local/freesbie-*`).

CONFIGURATION

FreeSBIE utilise un fichier de configuration qui est `freesbie2/conf/freesbie.conf`. Il doit être créé. Toutes les options par défaut sont dans `freesbie2/conf/freesbie.default.conf` (qu'il ne faut surtout pas modifier !).

La syntaxe de ce fichier est la suivante : NOM_OPTION=valeur.

Voici une liste et une description des options dont nous allons avoir besoin par la suite :

- `SRCDIR` : répertoire contenant les sources de FreeBSD (`/usr/src`) ;
- `PORTSDIR` : répertoire contenant l'arbre des ports FreeBSD (`/usr/ports`, qu'il faudra au préalable récupérer : voir l'article « Gestion avancée des ports dans FreeBSD » paru dans le hors-série n° 29) ;
- `MAKE_CONF` : chemin vers le fichier `make.conf`, utilisé pour activer ou désactiver certaines fonctions du système de base, ce fichier est également utilisé pour la compilation des ports ;
- `EXTRA` : permet d'activer les plugins à utiliser.

Voici une liste des plugins dont nous allons avoir besoin :

- `customroot` : permet d'ajouter des fichiers dans le futur système, par défaut les fichiers situés dans `freesbie2/extra/customroot` ;
- `customscripts` : permet d'exécuter des scripts shell (ayant l'extension `.sh`) situés dans `freesbie2/extra/customscripts` dans l'environnement du futur système (à l'aide de `chroot(8)`), nous l'utiliserons pour modifier les mots de passe ;
- `installports` : permet d'installer des ports dans le futur système, nécessite l'option `INSTALL_PORTS` (voir ci-dessous) ;
- `l10n` : permet de configurer la localisation du système comme le clavier en français ;
- `rootmfs` : permet de créer un espace en mémoire pour `/root` ;
- `rtcmfs` : permet de créer un espace en mémoire pour `/etc` et `/usr/local/etc` ;
- `varmfs` : permet de créer un espace en mémoire pour `/var` ;
- `adduser` : permet de créer un utilisateur `freesbie` et de créer un espace mémoire pour `/home`.

Option pour le plugin `installports` :

- `INSTALL_PORTS` : contient la liste des ports à installer sous la forme suivante « catégorie/nom ».

Fonctionnement du plugin `l10n` :

- Un fichier `l10n` sera créé dans le futur système, dans le répertoire `/etc/rc.d`. Ce script lira une variable d'environnement kernel (*kenv*) nommée `freesbie.lang`, configurée dans le fichier */boot/loader.conf*. C'est là qu'intervient le module `customroot` ! Il suffira de créer un répertoire `boot/` dans `freesbie2/extra/customroot` et d'y créer le fichier `loader.conf` avec `freesbie.lang=fr` pour avoir le système en français.

! ATTENTION

Il faut respecter l'ordre des plugins ! L'exécution se fera dans le même ordre que celui listé dans l'option EXTRA.

CRÉATION DE NOTRE LIVECD

À partir de maintenant, nous avons de solides bases pour faire notre propre LiveCD aux petits oignons.

Voici un fichier freesbie2/conf/freesbie.conf :

```
SRCDIR=/usr/src
PORTSDIR=/usr/ports
MAKE_CONF="/root/livecd/freesbie2/conf/make.conf"
EXTRA="customroot installports adduser customscripts l10n rootmfs etcmfs
varmfs"
INSTALL_PORTS="sysutils/screen irc/irssi"
```

Pour le bon fonctionnement du plugin l10n, nous devons créer le fichier loader.conf :

```
mkdir freesbie2/extra/customroot/boot
echo "freesbie.lang=\"fr\"" >> freesbie2/extra/customroot/boot/loader.conf
```

Nous aurons aussi besoin d'un fichier resolv.conf pour le plugin installports. Ce plugin utilise la fonction chroot et, s'il n'y pas ce fichier, rien ne pourra être téléchargé.

```
mkdir freesbie2/extra/customroot/etc
cp /etc/resolv.conf freesbie2/extra/customroot/etc
```

Grâce au plugin customscripts, nous allons enlever le fichier resolv.conf, car il est inutile sur le LiveCD et ne conviendra pas dans la plupart des situations. Créer un fichier resolv.sh dans freesbie2/extra/customscripts avec ceci :

```
#!/bin/sh
rm /etc/resolv.conf
```

Créons également un script permettant de modifier le mot de passe pour root et l'utilisateur freesbie, nous l'appellerons passwrd.sh :

```
#!/bin/sh
echo h4ck3r | pw usermod -h 0 -n root
echo livecd | pw usermod -h 0 -n freesbie
```

Créer aussi un fichier make.conf dans ~/livecd/freesbie2/conf (celui existant ne correspond pas à nos besoins).

```
echo "BATCH=YES" > freesbie2/conf/make.conf
```

Ce paramètre permet d'utiliser les options par défaut pour l'installation d'un port sans devoir choisir avec une interface Curses d'activer ou de désactiver une option. Si par la suite une option est nécessaire, il suffira de l'ajouter dans make.conf.

Tout est configuré pour créer un LiveCD avec le système de base complet et avec screen et irssi installés.

Lancement de la création du LiveCD :

```
cd ~/livecd/freesbie2
make iso
[beaucoup de lignes]
#### Building bootable ISO image for i386 ####
Saving mtree structure...
Running mkisofs...
ISO created:
-rw-r--r--  1 root  wheel   106M Jan  3 16:45 /usr/obj/FreeSBIE.iso
```

Suivant votre machine, cette étape peut prendre un certain temps. Le statut de chaque cible est stocké dans le répertoire /usr/obj/root/livecd/freesbie2/.tmp_cible, par exemple pour voir le statut de la cible installworld en direct :

```
tail -f /usr/obj/root/livecd/freesbie2/.tmp_buildworld
```

Dès qu'elle sera terminée, le fichier .tmp_buildworld sera renommé en .done_buildworld et ainsi de suite pour chaque cible.

TEST

Avant de graver l'ISO et gâcher un CD-R après s'être souvenu de rajouter l'application obligatoire, nous allons utiliser le logiciel Gemu (disponible dans les ports emulators/qemu) pour la tester. La commande suivante permettra de lancer une machine virtuelle et de démarrer à partir de l'ISO :

```
qemu -boot d -cdrom /usr/obj/FreeSBIE.iso
```

TUNING

Sachant que le fichier de configuration freesbie.conf sera lu par un script shell, il devient très simple de rajouter des fonctionnalités conviviales comme la création de plusieurs ISO différentes suivant un paramètre en ligne de commande. Voici un exemple concret d'un fichier freesbie.conf avec la création de deux ISO possibles :

```
#options globales
SRCDIR=/usr/src
PORTSDIR=/usr/ports
MAKE_CONF="/root/livecd/freesbie2/conf/make.conf"


case ${PROFILS:-} in
        base)
                echo "Création de l'ISO base"
                EXTRA="customroot installports adduser
                customscripts l10n rootmfs etcmfs varmfs"
                INSTALL_PORTS="sysutils/screen irc/irssi"
                CUSTOMSCRIPTS="/root/livecd/base/customscripts"
                CUSTOMROOT="/root/livecd/base/customroot"
                ISOPATH=/usr/obj/FreeSBIE-base.iso
                BASEDIR=/usr/local/freesbie-base-fs
                CLONEDIR=/usr/local/freesbie-base-clone
        ;;
        xorg)
```

```
                echo "Création de l'ISO xorg"
                EXTRA="customroot installports adduser
                customscripts l10n xconfig rootmfs etcmfs varmfs"
                INSTALL_PORTS="sysutils/screen irc/irssi x11/xorg"
                CUSTOMSCRIPTS="/root/livecd/xorg/customscripts"
                CUSTOMROOT="/root/livecd/xorg/customroot"
                ISOPATH=/usr/obj/FreeSBIE-xorg.iso
                BASEDIR=/usr/local/freesbie-xorg-fs
                CLONEDIR=/usr/local/freesbie-xorg-clone
        ;;
esac
```

Dans cet exemple, nous avons deux ISO possibles, la première, `base`, ne contient rien de plus que celle expliquée précédemment. Par contre, la deuxième, `xorg`, va nous permettre d'avoir une interface graphique. Nous allons même faire en sorte que le démarrage soit automatique.

Vous avez certainement remarqué l'apparition de nouvelles options :

- Le plugin `xconfig` dans la liste EXTRA va générer un fichier `xorg.conf` automatiquement au lancement du LiveCD ;
- Le paramètre `ISOPATH` est explicite et contient le chemin où sera créée l'image ISO ainsi que son nom ;
- `BASEDIR` et `CLONEDIR` contiennent respectivement le chemin où s'installeront les programmes et le répertoire temporaire utilisé durant la création d'une image (ISO ou IMG).

Pour que le démarrage de Xorg se fasse tout seul, il faut qu'un utilisateur se connecte automatiquement, par exemple `freesbie`. Pour effectuer cela, nous allons modifier les fichiers `ttys` et `gettytab`. Dans un premier temps, copiez ces fichiers dans `~/livecd/xorg/customroot/etc` :

```
mkdir -p ~/livecd/xorg/customroot/etc
cp /usr/src/etc/etc.i386/ttys ~/livecd/xorg/customroot/etc
cp /usr/src/etc/gettytab ~/livecd/xorg/customroot/etc
```

Modifiez la ligne commençant par `ttyv0` du fichier `ttys` en :

```
ttyv0   "/usr/libexec/getty freesbie"   cons25  on  secure
```

Rajouter ce bout de code à la fin du fichier `gettytab` :

```
freesbie:\
        :al=freesbie:ht:np:sp#115200:
```

Il ne reste plus qu'à modifier le fichier `.cshrc` qui sera lu à la connexion :

```
mkdir -p ~/livecd/xorg/customroot/usr/home/freesbie
cp /usr/src/share/skel/dot.cshrc ~/livecd/xorg/customroot/usr/home/
freesbie/.cshrc
```

et à rajouter ce bout de code à la fin, pour que l'interface graphique soit lancée au démarrage :

```
if (( "$?DISPLAY" == "0" ) && ( "$?SSH_CLIENT" == "0" )) then
        startx
endif
```

Si vous désirez avoir le réseau à partir d'un serveur DHCP et un serveur SSH démarré, rien de plus simple. Il suffit de créer un fichier `rc.conf` dans `~/livecd/xorg/customroot/etc` avec ceci :

```
hostname="FreeSBIE.LiveCD"
ifconfig_DEFAULT="DHCP"
background_dhclient="YES"
sshd_enable="YES"
```

FreeSBIE 2 peut récupérer les variables d'environnement grâce à `env(1)`.

Maintenant que tout est prêt, nous pouvons lancer la création de l'ISO xorg :

```
env PROFILS=xorg make iso
```

TIPS

- Pour éviter de recompiler à chaque fois le système de base et le kernel, il y a les options `NO_BUILDWORLD=YES` et `NO_BUILDKERNEL=YES` ;
- Pour « nettoyer » les fichiers de statut `.tmp*` et `.done*` situés dans `/usr/obj/root/livecd/freesbie2/` :

```
make clean
```

Pour information, ces fichiers sont utilisés lorsqu'il y a eu un problème à une étape : en relançant la création de l'ISO, la dernière cible (en fonction du dernier fichier `.tmp_*`) sera relancée. Donc, en supprimant tous ces fichiers, on repart de zéro.

- Pour « nettoyer » la préparation de l'ISO, c'est-à-dire les répertoires `BASEDIR` et `CLONEDIR` :

```
make cleandir
```

CONCLUSION

J'espère que cette introduction à FreeSBIE vous aura donné envie de découvrir ce projet. Toutes les fonctionnalités n'y sont pas décrites. Je vous invite donc à jeter un œil sur le site officiel. Sachez qu'il est très simple d'y contribuer. Avec de simples connaissances en shell, on peut créer ses propres plugins. Je l'ai moi-même fait en créant les plugins `customscripts` et `pf`.

LIENS

- Site officiel FreeSBIE : **http://www.freesbie.org**
- Site officiel FreeBSD : **http://www.freebsd.org**

NETBSD SANS DISQUE (OU LA MAGIE DES LUTINS QUI COURENT TRÈS VITE DANS LES FILS)

MAIS QU'EST-CE QUE TU FAIS !! Inconsciente que tu es, je sais bien que ce portable est vieux, qu'il n'a que 64 Megs de RAM (oui, c'est une constante chez moi), que sa Cirrus Logic ne doit pas être équipée de plus d'1 Meg... JE SAIS qu'il n'a pas de disque dur, mais regarde, là, sur le côté, une prise de la vie, le connecteur merveilleux : un rj45. Tu jettes pas cette box.

DES ANNÉES DE FRUSTRATION

Ami lecteur, tu es comme moi, tu as dans ta cave, tes armoires, ton sous-sol ou autre endroit propice à stocker « des trucs qui vont servir un jour, j'te jure », un stock hallucinant de vieux matériel informatique. Et depuis bien des années, d'abord ta mère, puis maintenant ta femme et peut être tes enfants ricanent devant ton entêtement à vouloir garder ces reliques d'un ancien temps qui n'ont même pas de cartes 3d – ***ricane****ricane** fait ta petite famille –. Ami lecteur, l'heure de la vengeance a sonné. Bientôt, les moqueurs d'hier vont se transformer en adorateurs d'aujourd'hui, car grâce à toi, têtu mais conquérant, ton petit monde ne va plus rater une seconde de Derrick : de la salle de bains à la penderie, il y aura un *browser*, un lecteur de médias et, finalement, à peu près tout ce qu'ils veulent à part KillKill3d – Murder Edition.

Moins conflictuel, un autre scénario se prête parfaitement à notre expérience. Vous êtes jeune, vous êtes dans le coup, vous avez sans doute déjà manipulé des environnements de virtualisation. Et quel est l'élément le moins portable et le plus contraignant dans un environnement virtualisé ? Le disque. Cet article propose une solution intermédiaire, portable et radicale : pas de disque. Vous pourrez, sans le moindre souci, tester votre environnement virtualisé sous Xen, Qemu, VirtualBox, Parallels (sale) ou encore VMware (très sale) sans vous demander si `qemu-img` réalisera bien la conversion.

On vous le dit : IP, c'est la vie.

Une dernière note d'introduction. Le serveur utilisé pour cet article est une machine ayant les caractéristiques suivantes :

- AMD Sempron 2800+ ;
- 1 Go de RAM ;
- Disque ATA classique de 80Go.

PLAN DE BATAILLE

Vous le savez certainement, la plupart de nos machines, depuis 1999, sont capables de *booter* grâce à une méthode maintenant largement répandue, le boot PXE. Il est assez probable que vos machines à recycler soient capables de réaliser ce qui est également appelé un « Netboot » ou encore « LAN boot » dans certains BIOS. Si ce n'est pas le cas, ne passez pas encore à l'article suivant, nous verrons comme il est aisé de réaliser cette opération à l'aide d'un autre projet libre. Même constat concernant l'approche virtuelle.

Voici donc notre cible :

- faire démarrer notre vieille machine sur le réseau grâce à un Netboot ;
- faire charger à notre machine un noyau sur le réseau grâce à un serveur TFTP ou NFS ;
- monter les partitions usuelles et les utiliser comme un disque local ;
- boire.

Pour mener ces opérations à bien, nous aurons besoin :

- d'un serveur DHCP ;
- d'un serveur TFTP ;
- d'un serveur NFS ;
- des « sets » NetBSD ;
- d'une bouteille de Gin.

Et pour vous montrer que, non, GCU n'est pas un groupuscule extrémiste obtus (la négation s'applique au mot « obtus »), nos divers serveurs seront configurés et démarrés sur une machine munie d'une Debian GNU/Linux. OOooOOOoooooOOooh ! Bon ok, la démarche est strictement identique avec un BSD moderne.

FORMEZ LES BATAILLONS

Ami lecteur, je te l'ai déjà dit, tu es comme moi, tu aimes voir des choses, nous allons donc tester chaque étape de notre périple plutôt que nous contenter d'un test final qui, de toutes façons, foirerait lamentablement et te ferait *troubleshooter* chaque étape de cet article.

Étape I, installation des outils

Afin de réaliser notre premier boot, nous allons installer nos trois serveurs :

```
# apt-get install dhcp3-server tftpd nfs-kernel-server netkit-inetd
```

NOTE

Sous NetBSD, FreeBSD et OpenBSD, ces outils sont disponibles dans le *basesystem*. Il suffit donc de les configurer et d'explicitement demander leur exécution dans **/etc/rc.conf**.

Étape II, configuration basique

Préparons tout d'abord le confortable foyer du futur client. Sur le serveur, exécutons les commandes suivantes :

```
# mkdir -p /home/netbsd/root/dev
# mkdir /home/netbsd/root/swap
# mkdir /home/netbsd/home
# mkdir /home/netbsd/usr
```

Créons dans la foulée le *device* `console` :

```
# mknod /home/netbsd/root/dev/console c 0 0
```

Éditons les quelques fichiers de configuration nécessaires :

- /etc/inetd.conf

Le serveur TFTP est démarré par Inetd, le SuperServeur :

```
# /tftpboot sera le root directory du serveur TFTP
tftp    dgram   udp     wait    nobody  /usr/sbin/tcpd  /usr/sbin/
in.tftpd /tftpboot
```

- /etc/dhcp3/dhcpd.conf

```
ddns-update-style interim; # on sous-traite le Dynamic DNS chez
ManPower
ignore client-updates; # ça va pas être possible avec vos baskets
monsieur


# on traite les ips du subnet 192.168.201.0/24
subnet 192.168.201.0 netmask 255.255.255.0 {
        # on possède un DNS à cette adresse
        option domain-name-servers 192.168.201.254;
        # ainsi qu'une passerelle
        option routers 192.168.201.254;
        # déclaration de la machine à faire démarrer en réseau
        host pxehost {
                # qui sera identifiée par son adresse MAC
                hardware ethernet 00:0e:32:e4:80:23;
                # on attribue à cette machine une IP fixe sur le range
défini plus haut
                fixed-address 192.168.201.100;

                # l'âme du PXE boot est ici, l'emplacement de ce
fichier est relatif au root du serveur TFTP
                filename "pxeboot_ia32.bin";

                # l'IP du "prochain" serveur, comprendre TFTP ou NFS
                next-server 192.168.201.10;
                # le PATH vers le futur filesystem
                option root-path "/home/netbsd/root";
        }

}
```

- /etc/exports, liste des exports NFS

```
/home/netbsd/root pxehost(rw,no_root_squash)
/home/netbsd/swap pxehost(rw,no_root_squash)
/home/netbsd/usr pxehost(rw,no_root_squash)
/home/netbsd/home pxehost(rw,no_root_squash)
```

- /etc/hosts, on y ajoute la correspondance entre le host et l'IP du client :

```
192.168.201.100    pxehost
```

Si vous avez suivi le GLMF Hors-série spécial BSD Acte I, la partie DHCP devrait vous rappeler quelque chose.

Nous devrions maintenant être en mesure de démarrer/redémarrer nos divers serveurs :

```
# /etc/init.d/inetd restart
# /etc/init.d/dhcp3-server restart
# /etc/init.d/nfs-kernel-server restart
```

Afin de s'assurer que notre serveur NFS est bien présent et exporte effectivement le futur *filesystem* de notre NetBSD *diskless*, exécutez :

```
showmount -e
```

Et vous devriez observer une sortie de type :

```
/home/netbsd/usr  pxehost
/home/netbsd/home pxehost
/home/netbsd/swap pxehost
/home/netbsd/root pxehost
```

Notre système hôte dispose maintenant de l'outillage adéquat. Nous allons donc le peupler petit à petit. Et tout d'abord, permettons-lui de démarrer. Pour cela, nous avons besoin :

- d'un bootcode PXE;
- d'un noyau.

Pour obtenir ces deux fichiers, nous allons télécharger les sets `kern` et `base` :

```
# mkdir /home/netbsd/pkg
# cd /home/netbsd/pkg
# wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-
4/200703090002Z/i386/binary/sets/kern-GENERIC.tgz
# wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-
4/200703090002Z/i386/binary/sets/base.tgz
# # oui, on va monter du 4BETA, et quoi, ON A DU POIL OU
BIEN ?!@#
```

Extraire les fichiers utiles :

```
# tar zxvfp ./kern-GENERIC.tgz
# tar -zf ./base.tgz -x ./usr/mdec/pxeboot_ia32.bin
```

Puis les copier au bon endroit :

```
# cp ./usr/mdec/pxeboot_ia32.bin /tftpboot
# cp netbsd /tftpboot
# cp netbsd /home/netbsd/root
```

« Ah tiens, y'a une typo ! ». Non, y'a pas de typo. Nous copions volontairement le noyau dans le *root* du serveur TFTP ET dans le root du futur filesystem, car, comme chez nous on aime bien avoir le choix, on pourra charger notre noyau via NFS ou TFTP. C'est gratos, c'est pour moi.

EVERYTHING IS NOT PROCEEDING AS I HAD FORESEEN

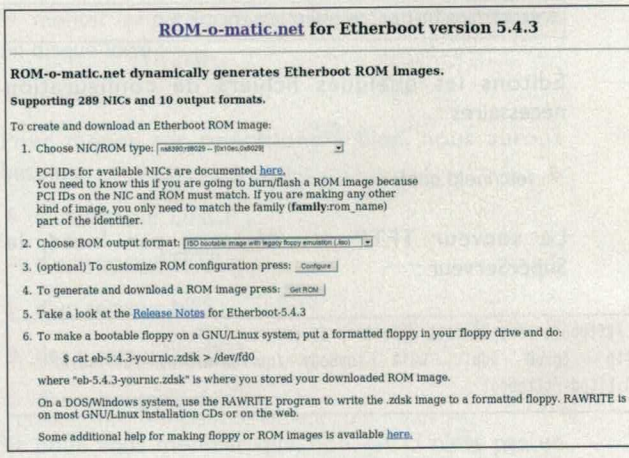
Que nous voilà pleins d'émotion, nous allons faire démarrer notre brouette, éteinte depuis des années, probablement amputée de son disque à l'incroyable capacité de 10Go, fièrement munie de ses 64Megas Octets de RAM EDO qui vous avaient probablement coûté un bras à l'époque. Alors que le ventilateur crache quelques kilos de poussière, vous vous ruez sur le BIOS – non sans avoir lutté pour trouver un clavier AT –, et vous êtes pris de panique : pas l'ombre d'un menu « LAN », « PXE » ou encore « Netboot ». Rien. C'est en rouvrant la bête que vous vous souvenez que les trames Ethernet sont acheminées dans cette machine par... une NE2000 (PCI, s'il vous plaît).

Essuie tes larmes, ami lecteur. La communauté est grande, la communauté est bonne, et il existe une solution tout à fait élégante à ton problème. Et toi qui ricane, toi qui imagine que ton Xen 3.0.3 fraîchement *apt-get'é* ne souffrira pas des mêmes déboires, ne fais pas le fier, car tu vas devoir suivre strictement la même procédure. En réalité, et croyez bien que cela me désole, il n'y a guère que Qemu (dans sa dernière version, merci CMoi), VirtualBox et VMware qui sauront initier un boot réseau sans astuce.

La solution absolue se nomme « Etherboot » [**http://www.etherboot.org/wiki/index.php**]. Ce projet permet ni plus ni moins de générer, sous toutes sortes de formats, des disques de boot réseau. Pour ne rien gâcher, bien que la procédure pour créer ces images soit des plus aisées, le projet propose également, sur le site **rom-o-matic.net**, de générer automatiquement l'image dont vous avez besoin. Il ne reste qu'à la télécharger. Ils sont forts ces étudiants communistes.

Comme il est assez probable que le lecteur de disquettes qui équipait votre Pentium 100 ne doit plus servir que de cache dans votre tour ATA ou votre transportable de 15Kg, nous partirons du postulat que la noble machine possède un « lecteurdecédérom », achat qui vous a probablement valu de vous nourrir aux pâtes et aux œufs pendant plusieurs mois.

Repérez donc sur **rom-o-matic** l'image correspondant à votre carte réseau. Par exemple, dans notre cas, `ns8390:rtl8029`.



Pour une machine physique, gravez cette image à l'aide de votre logiciel de gravure favori.

NOTE

Si, EN PLUS, votre vieille machine ne possédait pas de lecteur CD-ROM, récupérez une image au format zdsk et suivez la procédure décrite sur la page de *download* de l'image pour générer une disquette de Netboot.

Pour une machine virtuelle, sauvegardez simplement l'image `/quelque/part` puis, par exemple, pour un domaine Xen,

ajoutez l'entrée suivante dans le fichier de configuration du domaine :

```
disk = [ 'file:/quelque/part/eb-5.4.3-ns8390.liso,ioemu:hdc:
cdrom,r' ]
boot = 'd'
```

SÉQUENCE ÉMOTION

ÇA Y EST ! Enfin, enfin, nous allons démarrer celle que nous finissions par appeler amicalement « pourriture de veau gavé au Tranxen ». L'image Etherboot ou l'interface PXE native doivent avoir pris la main, et vous devriez voir un écran de ce type :

```
640 KB Base Memory
262144 KB Extended Memory

SYSLINUX 2.08 2003-12-12  Copyright (C) 1994-2003 H. Peter Anvin
Etherboot ISO boot image generated by genliso
Loading ns8390.zli....Ready.
Etherboot 5.4.3 (GPL) http://etherboot.org
Drivers: NE2000/PCI   Images: NBI ELF PXE   Exports: PXE
Protocols: DHCP TFTP
Relocating _text from: [00010220,0001ad90) to [0fef5490,0ff00000)
Boot from (N)etwork or (Q)uit? _
```

Et là, j'ai envie de dire, pardon aux plus prudes, mais : c'est la teuf !

Soit en appuyant sur [N], soit en attendant quelques secondes, les premiers signes de vie de notre système distant devraient apparaître :

```
262144 KB Extended Memory

SYSLINUX 2.08 2003-12-12  Copyright (C) 1994-2003 H. Peter Anvin
Etherboot ISO boot image generated by genliso
Loading ns8390.zli....Ready.
Etherboot 5.4.3 (GPL) http://etherboot.org
Drivers: NE2000/PCI   Images: NBI ELF PXE   Exports: PXE
Protocols: DHCP TFTP
Relocating _text from: [00010220,0001ad90) to [0fef5490,0ff00000)
Boot from (N)etwork or (Q)uit?

Probing pci nic...
[rtl80291
NE2000 base 0x1080, addr 00:0C:29:C8:D9:3C
Searching for server (DHCP).....
Me: 192.168.10.100, DHCP: 192.168.10.10, TFTP: 192.168.10.10, Gateway 192.168.10
.1
Loading 192.168.10.10:pxeboot_ia32.bin ...(PXE).....done


>> NetBSD/i386 PXE Boot, Revision 1.1
>> (builds@bl.netbsd.org, Mon Mar  5 04:34:12 UTC 2007)
>> Memory: 633/260053 k
Press return to boot now, any other key for boot menu
Starting in 1
```

Et si tout se passe « bien », voici ce sur quoi nous devrions nous arrêter :

```
timecounter: Timecounter "clockinterrupt" frequency 100 Hz quality 0
Kernelized RAIDframe activated
atapibus0 at atabus0: 2 targets
cd0 at atapibus0 drive 1: <PRL CD-ROM (1), - 31415B265, FWR1000> cdrom removable
cd0: 32-bit data port
cd0: drive supports PIO mode 4
cd0(piixide0:0:1): using PIO mode 4
boot device: ne2
root on ne2
nfs_boot: trying DHCP/BOOTP
nfs_boot: DHCP next-server: 192.168.10.10
nfs_boot: my_addr=192.168.10.100
nfs_boot: my_mask=255.255.255.0
nfs_boot: gateway=192.168.10.1
root on 192.168.10.10:/home/netbsd/root
root file system type: nfs
exec /sbin/init: error 2
init: trying /sbin/oinit
exec /sbin/oinit: error 2
init: trying /sbin/init.bak
exec /sbin/init.bak: error 2
init: not found
panic: no init
Stopped in pid 1.1 (init) at    netbsd:cpu_Debugger+0x4:        popl    %ebp
db>
```

J'entends d'ici M. La Quenelle se gausser : « <kernel> OOO111!! OMG OMG!!!1! ROTFL NO INIT OOL \\rainbow ». Et il aura bien raison, car, effectivement, ni init, ni rien d'autre. Notre système est pour le moment totalement vierge.

PROMISED LAND

Maintenant que nous avons constaté notre victoire sur le démarrage réseau, nous pouvons éteindre Germaine pour nous concentrer sur notre serveur PXE/NFS. C'est à grand renfort de tar(1) et autres vi que nous allons monter un NetBSD fonctionnel de toutes pièces. Au début de cet article, nous avons préparé l'arborescence nécessaire pour accueillir l'ensemble du *userland*. Il nous reste à la peupler. Pour ce faire, nous allons tout d'abord télécharger l'ensemble des sets (nous devrions déjà posséder la base) :

```
# pwd
/home/netbsd/pkg
# wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-4/200703090002Z/
i386/binary/sets/comp.tgz
# wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-4/200703090002Z/
i386/binary/sets/etc.tgz
# wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-4/200703090002Z/
i386/binary/sets/games.tgz
# wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-4/200703090002Z/
i386/binary/sets/man.tgz
# wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-4/200703090002Z/
i386/binary/sets/misc.tgz
# wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-4/200703090002Z/
i386/binary/sets/text.tgz
# wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-4/200703090002Z/
i386/binary/sets/xbase.tgz
# wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-4/200703090002Z/
i386/binary/sets/xcomp.tgz
# wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-4/200703090002Z/
i386/binary/sets/xetc.tgz
# wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-4/200703090002Z/
i386/binary/sets/xfont.tgz
# wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-4/200703090002Z/
i386/binary/sets/xserver.tgz
```

Puis, décompressez le tout en nous assurant de conserver les bons droits :

```
# cd ../root
# for file in ../pkg/*tgz; do tar zxvfp $file;done
```

Puisque nous avons choisi de créer une partition usr, nous déplaçons le contenu de root/usr dans cette dernière :

```
# mv usr/* ../usr
```

Il nous reste à rendre tout ceci opérationnel. Tout d'abord, en créant du swap : 128 mégas devraient suffire :

```
# pwd
/home/netbsd
# dd if=/dev/zero of=./swap bs=4k count=32k
```

Puis, en créant une /etc/fstab :

```
pxeserver:/home/netbsd/swap   none  swap  sw,nfsmntpt=/swap
pxeserver:/home/netbsd/root   /     nfs   rw 0 0
pxeserver:/home/netbsd/usr    /usr  nfs   rw 0 0
pxeserver:/home/netbsd/home   /home nfs   rw 0 0
```

Ainsi qu'un /etc/hosts :

```
192.168.201.254    pxeserver
192.168.201.100    pxehost
```

On configure l'interface réseau en créant un fichier /etc/ifconfig.<interface>, par exemple /etc/ifconfig.pcn0 :

```
inet pxehost netmask 255.255.255.0 broadcast 192.168.201.255
```

Et évidemment, on complète le fichier /etc/rc.conf :

```
# cette variable est initialement à NO, le système ne
bootera pas si elle n'est pas placée à YES
rc_configured=YES

hostname="pxehost"
defaultroute="192.168.201.100"
nfs_client=YES
# rc ne doit pas reconfigurer le réseau, hérité du boot
auto_ifconfig=NO
net_interfaces=""
```

ET C'EST QUI TON PAPA ? HEIN ? C'EST QUI ?

Nous voici prêts pour le second et dernier boot à l'issue duquel notre système sera parfaitement opérationnel. Démarrez votre machine, virtuelle ou non, en mode *single user*. Pour cela, au décompte du bootloader, appuyez sur une touche, puis, à l'invite, tapez :

```
> boot netbsd -s
```

NOTE

Il semble qu'avec certains logiciels de virtualisation, Qemu, entre autres, le décompte soit « un peu » rapide pour avoir le temps de frapper une touche. Si tel était votre cas, remettez la variable rc_configured à NO dans le fichier rc.conf.

Vous devriez, après chargement du noyau, vous trouver devant une invite de ce type :

```
pcppi0 at isa0 port 0x61
pcppi0: children must have an explicit unit
midi0 at pcppi0: PC speaker (CPU-intensive output)
sysbeep0 at pcppi0
isapnp0 at isa0 port 0x279: ISA Plug 'n Play device support
npx0 at isa0 port 0xf0-0xff
pcppi0: attached to attimer0
isapnp0: no ISA Plug 'n Play devices found
timecounter: Timecounter "clockinterrupt" frequency 100 Hz quality 0
Kernelized RAIDframe activated
atapibus0 at atabus0: 2 targets
cd0 at atapibus0 drive 1: <QEMU CD-ROM (1), - 314158265, FWR1000> cdrom removable
cd0: 32-bit data port
cd0: drive supports PIO mode 4
cd0(piixide0:0:1): using PIO mode 4
boot device: ne2
root on ne2
nfs_boot: trying DHCP/BOOTP
nfs_boot: DHCP next-server: 192.168.10.10
nfs_boot: my_addr=192.168.10.100
nfs_boot: my_mask=255.255.255.0
nfs_boot: gateway=192.168.10.1
root on 192.168.10.10:/home/netbsd/root
root file system type: nfs
Enter pathname of shell or RETURN for /bin/sh:
```

Après avoir appuyé sur [entrée], vous devriez être en possession d'un csh tout ce qu'il y a de moins convivial. Ce dernier devrait suffire le temps de créer l'ensemble des devices :

```
# cd /dev
# /bin/sh ./MAKEDEV all
```

Et maintenant, ^D !

Si tout s'est correctement déroulé, vous devriez admirer une mire de *login*.

VENGEAAAAAAAAAAANCE !

Vous voici en possession de plusieurs dizaines de consoles NetBSD potentielles, dont probablement aucun des composants ne posera de problème de compatibilité, puisque fort ancien. Évidemment, il est temps d'épater la galerie, d'en mettre plein les mirettes, DE FAIRE SE PROSTERN... Il est temps de montrer des trucs que le GENS a envie de voir, et, pour cela, vous n'avez besoin de rien de plus. Rien de plus que cet UNIX muni d'un serveur X et d'un client ssh dont je copie ici un extrait du *man* :

```
     -X      Enables X11 forwarding.  This can also be specified on
             a per-host basis in a configuration file.

             X11 forwarding should be enabled with caution.  Users
             the ability to bypass file permissions on the remote h
             (for the user's X authorization database) can access t
             local X11 display through the forwarded connection.  A
             attacker may then be able to perform activities such a
             keystroke monitoring.

             For this reason, X11 forwarding is subjected to X11 SE
             extension restrictions by default.  Please refer to th
             -Y option and the ForwardX11Trusted directive in ssh_c
             for more information.
```

Ce qui signifie qu'en activant l'option `X11Forwarding` sur un serveur depuis lequel vous voudrez exporter des sessions Gnome/KDE/Xfce/whatever, vous n'aurez, sur le terminal NetBSD diskless, besoin que d'un « simple » serveur X11. J'sais pas pour vous, mais moi ça me fait des trucs.

Une autre technique bien connue consistera à utiliser la variable d'environnement `$DISPLAY` et la commande `xhost`, démonstration :

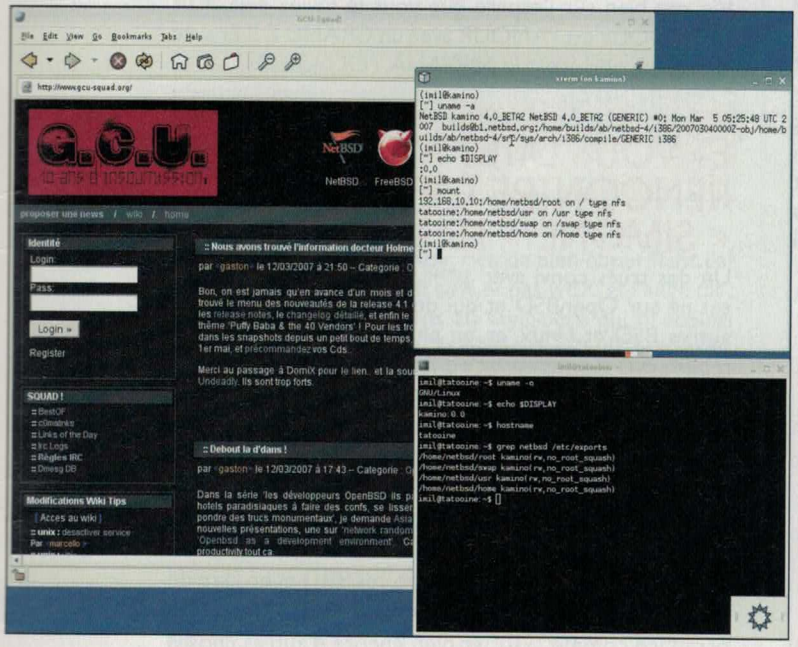
- sur la machine dont vous voulez exporter les applications (inutile d'être sous X) :

```
DISPLAY=pxehost:0.0; export DISPLAY
```

- sur le client NetBSD diskless (X doit être lancé) :

```
xhost +pxeserver
```

Voici pour finir l'illustration de cette méthode, un export complet d'environnement depuis ma station tatooine vers l'environnement NetBSD/NFS kamino.



Sans déconner, c'est quand même plus *groovy* d'aller lire les derniers titres de Slashdot que de feuilleter *Closer* non ?

RÉFÉRENCES :

- http://www.netbsd.org/Documentation/network/netboot/
- http://ltsp.org/

SIMONE SURVEILLE TES BABASSES, TU PEUX DORMIR TRANQUILLE

Comme le titre l'indique, vous l'avez compris, nous allons causer *monitoring*. Attention, ça c'est du buzzword qui fait frétiller les managers : ils aiment avoir des graphes, des chiffres, des trucs en vert qui clignotent, des indicateurs, des alertes, et tout ce genre de mots-clés qui justifient leur paie indécente tout en leur faisant récupérer tous les lauriers de votre travail. Le manager est fourbe, le manager est un vil suppôt de Satan. Pour se consoler, on peut toujours se dire que le service informatique lui refile exprès un laptop mista-certified-100%-bloatware.

Pour l'exemple, ce qui vient se déroulera sur un OpenBSD-*current* récent. J'aurai du coup l'occasion de vous parler d'une des spécificités de ce BSD qui fait couler beaucoup d'encre, l'apache 1.3-*heavily-patched-chrooté*-par-défaut.

Un petit coup de `make search key=monitor` dans `/usr/ports` nous renvoie plein d'informations, du genre Nagios et d'autres trucs usine à gaz dans le genre. Pas de trucs connus comme `munin` et `ganglia` (ce dernier étant plutôt orienté cluster). Mais une petite inconnue attire notre attention... je veux parler de `symon`. Maintenant, cher lecteur, re-regarde le titre de l'article. Et là, tu te demandes quel type de substance l'auteur de l'article prend (en l'occurrence, un thé fruits rouges-menthe), et qui est cette Simone qui est censée surveiller tes babasses, mais là n'est pas la question. D'ailleurs, entrons dans le vif du sujet, sans plus discuter le bout de gras.

APT-GET-PKG_ADD-URPMI-MAKE-MOI-TOUT-CA

Vous voulez plus d'infos ? Et vous pensez que c'est moi qui vais vous les donner ? Que nenni mes braves, `pkg_add` et `pkg_info` sont là pour ça !!

```
landry@renton:~/ $sudo pkg_add symon
symon-2.75p0:gd-2.0.34: complete


symon-2.75p0:rrdtool-1.0.49p3: complete


symon-2.75p0: complete
Information for symon-2.75p0

Comment: active monitoring tool

Description: symon is a lightweight system monitor that
measures cpu, memory, interface and disk statistics every 5
seconds. This information is then spooled over "the network"
to symux for further processing.

Maintainer: Willem Dijkstra <wpd@xs4all.nl>

WWW: www.xs4all.nl/~wpd/symon
```

Hm, déjà on note que Simone aime bien RRDtool. Elle est cool Simone, elle connaît les bonnes choses de la Vie, car, bien sûr, j'espère que vous le saviez déjà, mais RRDTOOL c'est l'AMOUR avec un GRAND E !! Et avec le module perl, c'est le NIRVANA, l'EXTASE !!!

ET VOUS DITES AVOIR RENCONTRÉ CETTE « SIMONE » LE 12 MARS ?

Un des trucs convi avec `symon`, déjà c'est un projet qui est né sur OpenBSD et qui depuis a été porté sur les autres BSD et Linux, et, en plus, il fonctionne en total *réparti-over-ton-network-t'wa*. Alors, on va essayer de la faire simple, y'a tout d'abord les anges gardiens `symon` qui vont *monitorer* les différentes machines, et qui vont envoyer via le réseau toutes les infos qu'ils collectent à leur pote saint-`symux`, qui lui va se charger de ranger tous ces chiffres dans des grimoires-fichiers RRD tout propres. Ensuite, un client (dans notre cas, le compère-verpépère `syweb`) pourra venir lire ces RRD et en faire de belles icônes-graphes ou juste afficher une valeur sur un écran lcd comme `sylcd`, et bien encore d'autres choses plus funky (à vous de faire travailler votre imagination !) avec `SymuxClient.pm`. À noter que l'ange `symon` utilise à outrance la séparation de privilèges, technique chérie des devs OpenBSD/. Mais pour plus d'infos là-dessus, je vous laisse jouer avec MikiPédia. Sachez juste qu'il se lance en `root`, ouvre les fichiers/*devices* pour lesquels il a besoin des pleins pouvoirs, et droppe ses privilèges pour tourner en tant qu'utilisateur `_symon`, qui lui n'a aucun droit, ni de vraie maison, et encore moins de shell. Il se chroote même dans sa fausse maison, c'est dire s'il n'a

vraiment plus d'attaches avec personne, et qu'il ne fera que son boulot, à savoir collecter des données brutes. Le saint-père-symux peut être lancé sous n'importe quelle identité, tant qu'il a les bons droits pour écrire les RRD. Pour l'exemple, il sera lancé avec mon UID, et j'aurai *chowné* -R moi:moi /var/www/symon comme un sauvage.

MAIS ENCORE ?

Bon, on a fait connaissance avec Simone, voyons ce qu'elle a dans le ventre. Pour la démo sans filet, je fais tout ça sur une seule machine, mais évidemment ça marche pareil avec plein d'autres babasses, faut juste lancer les anges sur chacune. Hop, direct, à froid, on attaque la conf d'un ange. Rhôôô qu'ils sont gentils, ils nous filent un fichier de conf par défaut, un script pour en générer un correspondant à notre matos, et un script pour créer les RRD. Pour les différentes sources d'informations, sachez que notre ange gardien peut monitorer les CPU, la mémoire, la bande passante, l'ami PF (Salut ! Ça va depuis la dernière fois ?), les queues altQ, les senseurs matériels, les disques, les processus, et que sais-je encore, et tout ça une fois toutes les 5 secondes par défaut. Hop, pour la peine, je vous montre mon fichier de conf.

```
#/etc/symon.conf
monitor { cpu(0), mem, df(sd0a), if(lo0), if(rl0), pf, mbuf,
  sensor(lm0.temp0), sensor(lm0.temp1), sensor(lm0.fan0),
  proc(sshd), proc(httpd), io(sd0)
} stream to 127.0.0.1 2100
```

Comme vous pouvez le voir, rien de bien ébouriffant, j'ai un disque SCSI donc c'est sdX et pas wdX, et l'ange dit : « j'envoie mes confessions au saint-père qui m'écoute d'une oreille attentive sur la fréquence 2100 du paradis local ». Un petit truc à noter sur vos calepins : PF ne sera monitoré que sur les interfaces qui ont été déclarées loginterface dans /etc/pf.conf. Passons maintenant à la conf du saint-symux.

```
#/etc/symux.conf
mux 127.0.0.1 2100

source 127.0.0.1 {
  accept { cpu(0), mem, df(sd0a), if(lo0), if(rl0), pf,
    mbuf, sensor(lm0.temp0), sensor(lm0.temp1),
    sensor(lm0.fan0), proc(sshd),proc(httpd), io(sd0)
  }
  write sensor(lm0.temp0) in \
    "/var/www/symon/rrds/localhost/sensor_lmtemp0.rrd"
  write sensor(lm0.temp1) in \
    "/var/www/symon/rrds/localhost/sensor_lmtemp1.rrd"
  write sensor(lm0.fan0) in \
    "/var/www/symon/rrds/localhost/sensor_lmfan0.rrd"
  datadir "/var/www/symon/rrds/localhost"
}
```

Que dire de spécial... Le saint-père écoutera ses anges sur la fréquence 2100 et, pour chacun d'eux, il déclare qu'il attend d'eux telles confessions, et qu'il les rangera dans tel livre sacré. Ici, deux petits détails à noter dans vos petits cahiers : j'ai dû contourner un petit bug du saint-père (nul n'est parfait) avec la gestion des senseurs, donc j'ai spécifié explicitement que les données devaient être stockées dans un fichier choisi par moi. De plus, il va bien prendre garde à ranger ces précieux grimoires emplis de confessions dans une étagère répertoire accessible par l'être-au-dessus, j'ai nommé le grand sage indien, qui plane un peu dans son monde à lui. Allez, *__shazaaam__* un peu de magie et on fait de la genèse sur tout ça.

```
landry@renton:~/ $sudo mkdir -p /var/www/symon/rrds/
localhost
landry@renton:~/ $sudo chown -R landry:landry /var/www/symon
landry@renton:~/ $/usr/local/share/symon/c_smrrds.sh all
/var/www/symon/rrds/localhost/io_sd0.rrd created
/var/www/symon/rrds/localhost/proc_sshd.rrd created
/var/www/symon/rrds/localhost/proc_httpd.rrd created
/var/www/symon/rrds/localhost/sensor_lmfan0.rrd created
/var/www/symon/rrds/localhost/sensor_lmtemp1.rrd created
/var/www/symon/rrds/localhost/sensor_lmtemp0.rrd created
/var/www/symon/rrds/localhost/mbuf.rrd created
/var/www/symon/rrds/localhost/pf.rrd created
/var/www/symon/rrds/localhost/if_rl0.rrd created
/var/www/symon/rrds/localhost/if_lo0.rrd created
/var/www/symon/rrds/localhost/df_sd0a.rrd created
/var/www/symon/rrds/localhost/mem.rrd created
/var/www/symon/rrds/localhost/cpu0.rrd created
```

Et voila, les grimoires sont prêts, il ne reste plus aux anges qu'à récupérer les confessions, et à les envoyer au saint-père.

```
landry@renton:~/ $/usr/local/libexec/symon  -d
warning: could not open "/dev/pf", Permission denied
fatal: chroot failed: Operation not permitted
# oups, effectivement, j'ai oublié un petit truc....

landry@renton:~/ $sudo /usr/local/libexec/symon -d
symon version 2.75
program id=1315
sending packets to udp 127.0.0.1 2100
started module io(sd0)
....

# dans un autre term
landry@renton:~/ $/usr/local/libexec/symux  -d
symux version 2.75
program id=26808
listening for incoming symon traffic on udp 127.0.0.1 2100
listening for incoming connections on tcp 127.0.0.1 2100
debug: good data received from 127.0.0.1:22441
debug: realclients = 0; stalledclients = 0
debug: rrdupdate /var/www/symon/rrds/localhost/io_sd0.rrd
....
```

Miracle, ils se parlent !!!! Alleluiiiaa !!! Bon, ils sont un peu bavards les cocos, donc on [ctrl-c] tout ça, et on les relance en mode « chut », sans oublier d'ajouter `/usr/local/libexec/symon && su - landry -c /usr/local/libexec/symux dans /etc/rc.local` pour que les deux démons soient lancés au démarrage. Ici, évidemment, adaptez en fonction de l'user que vous avez choisi, pour faire tourner `symux`. Laissons-les travailler en paix, ils savent ce qu'ils ont à faire... montons un peu plus haut, et faisons la connaissance du grand sage indien plein de pansements, nommé affectueusement « l'apache » (pfiou, elle vient de loin celle-là !).

ALLUME LE CALUMET, MAN !

Bon, donc l'apache, comme je le disais, il plane dans son monde-nuage-chroot, et va falloir lui monter des trucs là-haut pour qu'il puisse bosser efficacement en toute tranquillité.

```
# world->invoke(apache)
root@renton:~/ #httpd
# ne pas oublier httpd_flags="" dans /etc/rc.conf.local
# si l'on veut qu'il soit lancé au démarrage
root@renton:~/ #pkg_add php4-core
php4-core-4.4.1p2:recode-3.6p3: complete


php4-core-4.4.1p2: complete


--- php4-core-4.4.1p2 ------------------
To finish the install, enable the php4 module with:
/usr/local/sbin/phpxs -s


To enable parsing of PHP scripts, add the following to
/var/www/conf/httpd.conf:


AddType application/x-httpd-php .php


Copy the config file below into /var/www/conf/php.ini
/usr/local/share/examples/php4/php.ini-recommended


Don't forget that the default OpenBSD httpd is chrooted
into /var/www by default, so you may need to create support
directories such as /var/www/tmp for PHP to work correctly.


root@renton:~/ #/usr/local/sbin/phpxs -s
[activating module `php4' in /var/www/conf/httpd.conf]
.... divers trucs ....
.... on décommente la ligne AddType dans le httpd.conf ....


root@renton:~/ #apachectl restart
Syntax error on line 267 of /conf/httpd.conf:
Cannot load /usr/lib/apache/modules/libphp4.so into server:
File not found
```

Ça semblait trop beau, il en manque un peu... Effectivement, `phpxs -s` a installé ses petits dans `/usr/lib/apache/modules/` et l'apache ne connaît rien en dehors de `/var/www`... quand je vous disais qu'il plane dans son monde !! Qu'à cela ne tienne, on va arranger tout ça.

```
root@renton:~/ #mkdir -p /var/www/usr/lib/apache/modules
root@renton:~/ #cp /usr/lib/apache/modules/libphp4.so \
  /var/www/usr/lib/apache/modules
root@renton:~/ #mkdir /var/www/etc
root@renton:~/ #cp /usr/share/zoneinfo/Europe/Paris \
  /var/www/etc/localtime
root@renton:~/ #apachectl start
```

Et là, nous avons le bonheur de voir ces nuages de fumée au-dessus du tipi de notre ami : `Apache/1.3.29 (Unix) PHP/4.4.1 mod_ssl/2.8.16 OpenSSL/0.9.7j configured`. Joie, bonheur, félicité ! Maintenant, on va le faire travailler, parce que bon, les anges et le saint-père, à quoi ça sert qu'ils se décarcassent si le compère-verpépère `syweb` est pas en train de turbiner aussi chez l'apache ?

VOUS AVEZ DES PREUVES ?

Bon, donc, le compère-syweb, on le récupère chez son papa (hint: la ligne `WWW` dans `pkg_info symon`. Oui, c'est pas *packagé* dans le *ports-tree*.) et on va l'installer dans `/var/www/htdocs/syweb`, après avoir pris soin de monter les outils nécessaires dans le nuage du grand sage. Oh, que ça tombe bien, on nous fournit un script pour ça (ici, rien de mystérieux, on peuple juste le chroot avec le binaire de `rrdtool` et les bibliothèques dont il dépend).

```
root@renton:~/ #tar xvzf syweb-0.55.tar.gz
root@renton:~/ #cd syweb && ./install_rrdtool.sh
rrdtool and libs installed in apache root


root@renton:~/syweb/ $ls -FR /var/www/{bin,usr}
/var/www/bin:
rrdtool*    sh*


/var/www/usr:
lib/     libexec/


/var/www/usr/lib:
apache/          libfontconfig.so.3.0 libiconv.so.4.0
libc.so.40.3     libfreetype.so.13.1  libjpeg.so.62.0*
libexpat.so.5.0  libgd.so.20.34*      libm.so.2.3
libpng.so.5.1    librrd.so.0.0*       libz.so.4.1


/var/www/usr/lib/apache:
modules/
```

```
/var/www/usr/lib/apache/modules:
libphp4.so*

/var/www/usr/libexec:
ld.so*
```

Comme on peut le voir, tout le monde est présent pour la fête qui se prépare... On lit le parchemin INSTALL, et on copie ce qu'il faut là-où-il-faut. (`cp -R syweb/htdocs/* /var/www/htdocs && cp -R syweb/symon/* /var/www/symon`). On édite `/var/www/htdocs/syweb/setup.inc` pour configurer `syweb` avec le profil `openbsd chrooted`. On crée le répertoire de cache qui servira pour stocker les images (`mkdir /var/www/symon/cache && chown www:www /var/www/symon/cache`), et ta-daaam, on prend son *browser* favori (graphique, au moins... `links -g` fera l'affaire... 'fin je vous oblige pas) et on visite :

http://127.0.0.1/syweb/configtest.php.

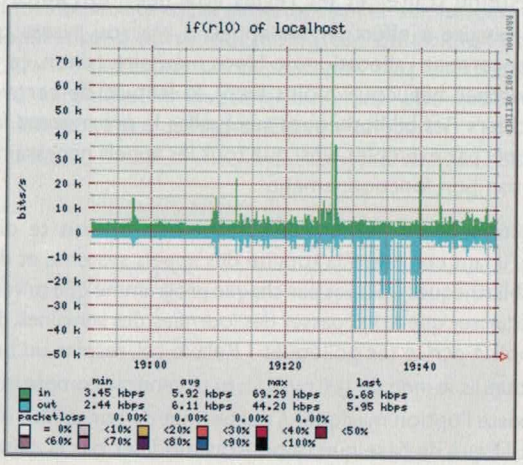
Et on vérifie que tout va bien. Et on commence à sourire. Et on visite :

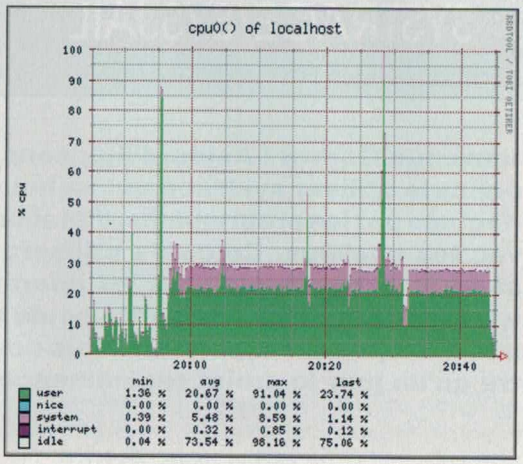
http://127.0.0.1/syweb/index.php.

Et là, un immense sourire béat se dessine sur notre figure. C'est le moment d'aller chercher une bière bien méritée dans le frigo.

Que dire de plus sur ce petit outil qui peut vous rapporter gros... ici, l'intérêt était de montrer l'installation dans l'apache chrooté d'OpenBSD, mais l'agent de monitoring peut très bien tourner sur d'autres OS du bien, comme `symux` et `syweb`... c'est la magie de la portabilité du code !

Décidément, tous ces gens, les anges, le saint-père, le grand sage indien, et le compère-verpépère, quand ils bossent de concert, ils font de belles choses. Et ce sont tous des amis de Simone. Elle est quand même super sympa, la preuve, elle nous a laissé en cadeau deux dessins, plus parlants qu'un long discours :

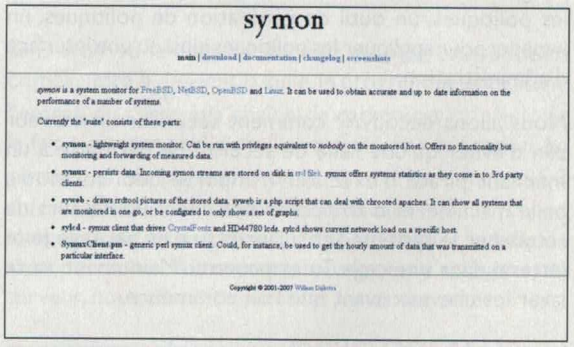




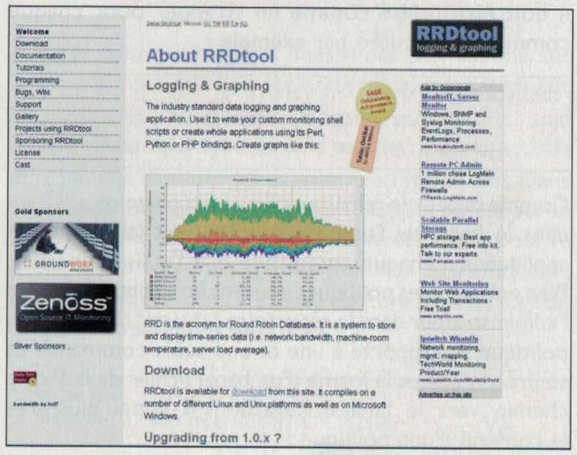
Voila, c'est fini pour aujourd'hui, j'aurais aimé vous parler de `<teaser>monit</teaser>` aussi, mais Lefinnois m'a piqué mon stylo, fallait rendre les copies... donc rendez-vous pour plus de fun dans un prochain numéro !

LINQSES

- symon : **http://www.xs4all.nl/~wpd/symon/**



- RRDtool : **http://oss.oetiker.ch/rrdtool/**



SYSTRACE/SYSJAIL

Bienvenue Gérard ! Aujourd'hui, nous allons sécuriser notre système. Nous allons pour cela utiliser systrace qui va nous permettre de surveiller les appels système effectués par les programmes, d'établir des règles et de faire plein de choses rigolotes avec tes cheveux. Ce cours utilisera l'OS à la serviette orange, mais tout ce qui est écrit ici est valide pour les adorateurs de Puffy. Il n'y a malheureusement pas de tresses pour l'OS à la petite boule rouge. Un projet de portage avait commencé, mais il n'a pas été maintenu. Mais comme dirait Vitoo, jamais 2 sans 3, alors peut-être qu'un jour le projet recommencera.

SIX TRESSES

Pour sécuriser un système Unix, il y a plusieurs moyens, parmi lesquels : les permissions sur les fichiers (y compris les bits SUID/SGID), la délégation de pouvoirs avec `calife` ou `sudo`. Il t'est peut-être arrivé, mon petit Gérard, de t'arracher les cheveux car la granularité offerte par ces solutions n'était pas assez fine. J'ai la solution à tes problèmes : `systrace(4)`. Grâce à cet outil, il est possible de contrôler les appels système autorisés pour chaque programme installé. Il est découpé en plusieurs parties : les politiques, un outil de génération de politiques, un *wrapper* pour appliquer les politiques ainsi qu'une interface d'administration.

Nous allons découvrir comment sécuriser un serveur afin d'éviter qu'une faille de sécurité ne permette à un méchant pirate d'exécuter n'importe quoi sur notre belle machine, et d'empêcher nos chers utilisateurs de retourner le système ; et si jamais tu n'es pas sage je te jetterai dans une cage. Tu es prévenu. Maintenant, va te laver les cheveux avant que l'on commence.

LE SHAMPOOING

Avant toute chose, nous allons voir comment `systrace(4)` fonctionne. Pour qu'il puisse contrôler les appels système, il doit être utilisé comme un wrapper pour chaque commande exécutée, par exemple :

```
shaddai@booyaka:~$ systrace /bin/ls -a
.               .ssh            stsh-0.3.1
..              .systrace       stsh-0.3.1.tar.gz
```

Comme tu peux le remarquer, il y a un répertoire `.systrace` dans le *home* de l'utilisateur. Il contient les politiques appliquées à chaque commande que l'utilisateur lance. Bien entendu, ces politiques peuvent être centralisées par l'administrateur dans le répertoire `/etc/systrace`. Chaque politique se rapporte à une et une seule commande et se présente sous la forme d'un banal fichier de la forme chemin_vers_le_binaire. Regardons d'un peu plus près le contenu d'une politique :

```
Policy: /bin/ls, Emulation: netbsd
        netbsd-mmap: permit
        netbsd-fsread: filename eq "/etc/ld.so.conf" then permit
        netbsd-__fstat13: permit
        netbsd-close: permit
        netbsd-munmap: permit
        netbsd-fsread: filename eq "/lib/libc.so.12.128.2" then permi
        netbsd-issetugid: permit
        netbsd-ioctl: permit
        netbsd-getuid: permit
        netbsd-__sysctl: permit
        netbsd-fsread: filename eq "/etc/malloc.conf" then permit
        netbsd-break: permit
        netbsd-fsread: filename eq "/home/shaddai/." then permit
        netbsd-fcntl: permit
        netbsd-fchdir: permit
        netbsd-fstatvfs1: permit
        netbsd-lseek: permit
        netbsd-getdents: permit
        netbsd-write: permit
        netbsd-exit: permit
```

Le fichier a une structure facilement compréhensible. Tout d'abord le binaire auquel s'applique la politique, le type d'émulation qui peut être « native » pour OpenBSD, « NetBSD » ou encore « Linux ». Suit la liste des appels système traités et les règles attachées. On autorise le binaire à effectuer les appels à `mmap` ou `munmap` qui concernent l'allocation de blocs mémoire (sinon, ça va marcher beaucoup moins bien), la lecture de certains fichiers (les bibliothèques auxquelles le programme fait appel par exemple), ainsi que tous les appels nécessaires à son bon fonctionnement.

Je te sens un peu anxieux mon petit, tu dois te dire qu'il faut connaître la totalité des appels système et des bibliothèques utilisées par chaque programme que tu veux sécuriser, que tu vas passer des journées, des semaines, des mois à écrire tes politiques ! Relaxe-toi, respire un bon coup, lis le *man* de `systrace(8)`, tu te rendras compte qu'il existe l'option magique `-A` qui permet de construire une politique de base que tu pourras modifier par la suite.

ON RINCE GRATIS

Nous allons nous attaquer à la sécurisation d'un service réseau. Pour cela, prenons un serveur codé avec les pieds et montrons comment empêcher Jean-Kevin et ses amis de faire n'importe quoi avec. Cet exemple est factice, car, bien entendu, les programmes utilisés dans les environnements de production ne contiennent pas de failles de ce genre, hein !

```
client@gentil:~$ nc 172.20.0.2 4242
Qui doit partir du jardin ? Votez !
        1. hotbox
        2. shaddai
        3. pinpin
        4. obiwan kenobi
1
Votre vote a été pris en compte, merci.
```

Mais que se passe-t-il si une personne malveillante tente un :

```
LordZofDarkness@www.fbi.gov:~$ nc 172.20.0.2 4242
Qui doit partir du jardin ? Votez !
        1. hotbox
        2. shaddai
        3. pinpin
        4. obiwan kenobi
1; nc -l -p 31337 -e /bin/sh
Votre vote a été pris en compte, merci.

LordZofDarkness@www.fbi.gov:~$ nc 172.20.0.2 31337
uname -a;
NetBSD booyaka 3.1 NetBSD 3.1 (GENERIC) #0: Tue Oct 31 04:27:07 UTC
2006  builds@b0.netbsd.org:/home/builds/ab/netbsd-3-1-RELEASE/i386/
200610302053Z-obj/home/builds/ab/netbsd-3-1-RELEASE/src/sys/arch/i386/
compile/GENERIC i386
id;
uid=0(root) gid=0(wheel) groups=0(wheel),2(kmem),3(sys),4(tty),5(operator)
,20(staff),31(guest)
```

Catastrophe Gérard, les arguments passés au serveur ne sont pas vérifiés correctement et il est possible d'exécuter des commandes système ! Sécurisons tout ça, si tu le veux bien. Une méthode pour arriver à une politique convenable en matière à la fois de sécurité et de maintenabilité est la suivante : lancer sur le serveur l'application à partir du générateur de politique de `systrace(8)`, exécuter un client depuis un poste distant et lui faire dérouler un scénario nominal, puis ajouter à la main les règles qui viendraient à manquer :

```
coté serveur :
# systrace -A /usr/local/bin/jardin_academy

côté client :
client@gentil:~$ nc 172.20.0.2 4242
Qui doit partir du jardin ? Votez !
        1. hotbox
        2. shaddai
        3. pinpin
        4. obiwan kenobi
1
```

Maintenant que nous avons notre politique de base, nous allons pouvoir empêcher que le pire n'arrive. Jetons un œil au fichier produit :

```
Policy: /usr/local/bin/jardin_academy, Emulation: netbsd
        netbsd-mmap: permit
        netbsd-fsread: filename eq "/etc/ld.so.conf" then permit
        netbsd-__fstat13: permit
        netbsd-close: permit
        netbsd-munmap: permit
        netbsd-fsread: filename eq "/lib/libc.so.12.128.2" then permit
        netbsd-__sysctl: permit
        netbsd-fsread: filename eq "/etc/malloc.conf" then permit
        netbsd-break: permit
        netbsd-socket: sockdom eq "AF_INET" and socktype eq "SOCK_STREAM"
                       then permit
        netbsd-setsockopt: permit
        netbsd-bind: sockaddr eq "inet-[0.0.0.0]:4242" then permit
        netbsd-listen: permit
        netbsd-accept: permit
        netbsd-write: permit
        netbsd-read: permit
        netbsd-ioctl: permit
        netbsd-__sigaction_sigtramp: permit
        netbsd-__sigprocmask14: permit
        netbsd-__vfork14: permit
        netbsd-execve: filename eq "/bin/sh" and argv eq "sh -c /bin/echo 1
                       >> /tmp/votes" then permit
        netbsd-wait4: permit
```

Sans avoir vu le code du serveur, on peut tout de suite comprendre d'où vient la faille, le programme concatène l'entrée du client à la chaîne de caractères `/bin/echo` et la fait exécuter par le *shell* sans faire de vérification. Le générateur de politiques de `systrace(8)` a donc créé 2 autres politiques, stockées elles aussi dans `/root/.systrace`, mais nous ne les modifions pas pour l'instant, la faille ne vient pas d'elles. Pour couvrir tous les cas d'utilisation du serveur, nous allons rajouter les lignes suivantes :

```
netbsd-execve: filename eq "/bin/sh" and argv re "sh -c /bin/echo [1-4] >>
/tmp/votes" then permit
netbsd-exit: permit
```

À noter dans ces règles la finesse des permissions, sur le binaire ainsi que sur ses arguments, mais aussi sur les mots clés ; `eq` signifie une égalité stricte, `match` une correspondance sur une partie d'une chaîne de caractères et `re` permet d'utiliser les expressions rationnelles. Dans notre cas, nous avons besoin d'une égalité stricte sur le binaire et une expression rationnelle est idéale pour le choix des votes. Vérifions notre politique au travers d'un test en activant `systrace(8)` avec écriture dans les logs de toutes les opérations interdites.

```
côté serveur
# systrace -a /usr/local/bin/jardin_academy
```

```
côté client :
Qui doit partir du jardin ? Votez !
        1. hotbox
        2. shaddai
        3. pinpin
        4. obiwan kenobi
2
Votre vote a été pris en compte, merci.
```

On voit alors apparaître dans les logs :

```
Feb 25 10:51:40 booyaka systrace: deny user: root, prog: /bin/sh, pid:
621(1)[574], policy: /bin/sh, filters: 27, syscall: netbsd-execve(59),
filename:
/bin/echo, argv: /bin/echo 2
```

Lorsque nous avons généré la politique pour `jardin_academy`, chaque commande exécutée par le binaire s'est vue attribuer une politique par la même occasion. Comme le shell `/bin/sh` n'est pas la faille, nous allons modifier sa politique pour que le shell puisse utiliser librement `echo`. On édite le fichier de politiques `bin_sh` :

```
netbsd-execve: filename eq "/bin/echo" then permit
```

Nous sommes prêts à affronter de nouveau Jean-Kevin, qu'il tente de nous attaquer, qu'on rigole un peu ! La même attaque que précédemment, à savoir `1; nc -l -p 31337 -e /bin/sh` est bloquée. On voit dans les logs :

```
Feb 25 11:04:22 booyaka systrace: deny user: root, prog:
/usr/local/bin/jardin_academy, pid: 603(0)[653], policy:
/usr/local/bin/jardin_academy, filters: 22, syscall: netbsd-execve(59),
filename: /bin/sh, argv: sh -c /bin/echo 1;nc -l -p 31337 -e /bin/sh >>
/tmp/votes
```

Jean-Kevin peut alors essayer ses *sploitZ* sur ton serveur pendant que tu fumes ton café, il n'arrivera à rien. Nous avons réussi à combler une faille de sécurité sans appliquer de patch, sans besoin d'un IPS quelconque, bref la totale moule frite !

BIEN COURT SUR LES OREILLES ET LA NUQUE

Tu as sécurisé tes services réseau, c'est bien, mais tu as des utilisateurs locaux qui ont un shell sur ton serveur (sous prétexte qu'ils sont étudiants et qu'ils ont des TP à faire, la vieille ruse) et tu ne voudrais pas qu'ils se servent de ton serveur pour faire des choses inavouables... Nous allons devoir faire tourner un shell via `systrace(8)`. Jose Nazario a même écrit un parchemin pour nous faciliter la vie : `stsh` [1].

Suivons les instructions disponibles sur la page écrite par Jose, compilons `stsh` et copions-le dans `/bin`. On rajoute une classe `systrace` dans `login.conf` :

```
systrace:\
        :shell=/bin/stsh:\
        :tc=default:

default:
```

Puis, on ajoute un premier utilisateur pour faire nos tests en précisant la classe dans laquelle il doit être :

```
# useradd -L systrace gerard
```

Comme l'indique la documentation, il faut déjà des politiques en place pour que l'utilisateur puisse se connecter et exécuter des commandes. On trouve des politiques pour quelques shells et outils de base dans l'archive de `stsh`, politiques fonctionnelles sur OpenBSD, car elles utilisent l'émulation native. Gérard, il va falloir que tu crées tes politiques avec `systrace -A` pour chaque outil que tes utilisateurs devront utiliser. Nous nous concentrerons ici uniquement sur le shell utilisateur bash, shell assez répandu.

Partons de la politique de base suivante :

```
Policy: /usr/pkg/bin/bash, Emulation: netbsd
        netbsd-mmap: permit
        netbsd-fsread: filename eq "/etc/ld.so.conf" then permit
        netbsd-__fstat13: permit
        netbsd-close: permit
        netbsd-munmap: permit
        netbsd-fsread: filename eq "/usr/pkg/lib/libreadline.so.5.0.2"
                       then permit
        netbsd-fsread: filename eq "/usr/pkg/lib/libhistory.so.5.0.2"
                       then permit
        netbsd-fsread: filename eq "/usr/pkg/lib/libtermcap.so.0" then
                       permit
        netbsd-fsread: filename eq "/usr/lib/libtermcap.so.0" then
                       permit
        netbsd-fsread: filename eq "/usr/pkg/lib/libintl.so.0" then
                       permit
        netbsd-fsread: filename eq "/usr/lib/libintl.so.0" then permit
        netbsd-fsread: filename eq "/usr/pkg/lib/libc.so.12" then
                       permit
        netbsd-fsread: filename eq "/usr/lib/libc.so.12" then permit
        netbsd-__sigprocmask14: permit
        netbsd-fswrite: filename eq "/dev/tty" then permit
        netbsd-issetugid: permit
        netbsd-__sysctl: permit
        netbsd-break: permit
        netbsd-getuid: permit
        netbsd-getgid: permit
        netbsd-geteuid: permit
        netbsd-getegid: permit
        netbsd-getgroups: permit
        netbsd-gettimeofday: permit
        netbsd-ioctl: permit
        netbsd-__sigaction_sigtramp: permit
        netbsd-fsread: filename eq "/etc/nsswitch.conf" then permit
        netbsd-read: permit
        netbsd-fsread: filename eq "/usr/pkg/lib/nss_compat.so.0" the
                       permit
```

```
        netbsd-fsread: filename eq "/usr/lib/nss_compat.so.0" then
                       permit
        netbsd-fsread: filename eq "/usr/pkg/lib/nss_nis.so.0" then
                       permit
        netbsd-fsread: filename eq "/usr/lib/nss_nis.so.0" then permit
        netbsd-fsread: filename eq "/usr/pkg/lib/nss_files.so.0" then
                       permit
        netbsd-fsread: filename eq "/usr/lib/nss_files.so.0" then
                       permit
        netbsd-fsread: filename eq "/usr/pkg/lib/nss_dns.so.0" then
                       permit
        netbsd-fsread: filename eq "/usr/lib/nss_dns.so.0" then permit
        netbsd-fsread: filename eq "/etc/pwd.db" then permit
        netbsd-fsread: filename eq "/etc/spwd.db" then permit
        netbsd-fsread: filename match "/usr/share/*" then permit
        netbsd-fcntl: permit
        netbsd-pread: permit
        netbsd-__getcwd: permit
        netbsd-getpid: permit
        netbsd-getppid: permit
        netbsd-getpgrp: permit
        netbsd-dup: permit
        netbsd-getrlimit: permit
        netbsd-dup2: permit
        netbsd-pipe: permit
        netbsd-fsread: filename eq "/usr/share/misc/termcap.db" then
                       permit
        netbsd-fsread: filename eq "/etc/inputrc" then permit
        netbsd-fsread: filename eq "/etc/profile" then permit
        netbsd-fsread: filename eq "$HOME" then permit
        netbsd-fsread: filename match "$HOME/*" then permit
        netbsd-fsread: filename eq "/var/mail/$USER" then permit
        netbsd-write: permit
        netbsd-fsread: filename eq "/" then permit
        netbsd-fstatvfs1: permit
        netbsd-lseek: permit
        netbsd-getdents: permit
        netbsd-fsread: filename match "$HOME/bin/*" then permit
        netbsd-fsread: filename match "/bin/*" then permit
        netbsd-fsread: filename match "/sbin/*" then permit
        netbsd-fsread: filename match "/usr/bin/*" then permit
        netbsd-fsread: filename match "/usr/sbin/*" then permit
        netbsd-fsread: filename match "/usr/pkg/bin/*" then permit
        netbsd-fsread: filename match "/usr/X11R6/bin/*" then permit
        netbsd-fsread: filename match "/usr/local/bin/*" then permit
        netbsd-fsread: filename match "/usr/local/sbin/*" then permit
        netbsd-execve: filename match "$HOME/bin/*" then permit
        netbsd-execve: filename match "/bin" then permit
        netbsd-execve: filename match "/sbin/*" then permit
        netbsd-execve: filename match "/usr/bin/*" then permit
        netbsd-execve: filename match "/usr/sbin/*" then permit
        netbsd-execve: filename match "/usr/pkg/bin/*" then permit
        netbsd-execve: filename match "/usr/X11R6/bin/*" then permit
        netbsd-execve: filename match "/usr/local/bin/*" then permit
        netbsd-fork: permit
        netbsd-setpgid: permit
        netbsd-setpgid: permit
        netbsd-wait4: permit
        netbsd-compat_16___sigreturn14: permit
        netbsd-fswrite: filename eq "$HOME/.bash_history" then permit
        netbsd-fswrite: filename eq "$HOME/.bash_logout" then permit
        netbsd-exit: permit
```

Quelques explications avant de continuer : le shell, comme tout autre programme, a besoin de bibliothèques, exécute des appels système, ici il lit aussi des fichiers de configuration dans `/etc` ou dans `$HOME`. La valeur ajoutée de ce *template* se situe au niveau des lignes concernant les répertoires de `$PATH`. Nous avons mis en place un mécanisme qu'on peut appeler TPE (*Trusted Path Execution*). Seuls les binaires situés dans les répertoires explicitement autorisés à être lus sont accessibles à l'utilisateur. Ici notre template est large, pour ne pas dire laxiste, les utilisateurs d'un système n'ont normalement pas besoin d'accéder aux binaires se trouvant dans `/sbin`, `/usr/sbin` ou encore `/usr/local/sbin`, les programmes s'y trouvant étant dédiés à *root*. Nous allons donc enlever les lignes concernant ces répertoires pour les appels système `fsread` et `execve`. Maintenant, si un utilisateur tente d'exécuter une commande située dans un de ces répertoires, il obtiendra un message d'erreur :

```
bash-2.05b$ ifconfig
bash: ifconfig: command not found
bash-2.05b$ /sbin/ifconfig
bash: /sbin/ifconfig: Operation not permitted
bash-2.05b$
```

La différence entre les 2 messages d'erreur s'explique par le fait que, dans le premier cas, c'est l'accès au répertoire `/sbin` qui est refusé alors que, dans le second cas, c'est l'exécution du binaire lui même qui est refusée. Jetons un coup d'œil à nos logs :

1ère tentative d'exécution :

```
Mar  4 18:05:42 booyaka systrace: deny user: gerard, prog: /usr/pkg/bin/
bash, pid: 471(0)[0], policy: /usr/pkg/bin/bash, filters: 79, syscall:
netbsd-fsread(278), filename: /<non-existent filename>: /home/gerard/bin/
ifconfig
Mar  4 18:05:42 booyaka systrace: deny user: gerard, prog: /usr/pkg/bin/
bash, pid: 471(0)[0], policy: /usr/pkg/bin/bash, filters: 79, syscall:
netbsd-fsread(278), filename: /sbin/ifconfig
Mar  4 18:05:42 booyaka systrace: deny user: gerard, prog: /usr/pkg/bin/
bash, pid: 471(0)[0], policy: /usr/pkg/bin/bash, filters: 79, syscall:
netbsd-fsread(278), filename: /usr/sbin/ifconfig
Mar  4 18:05:42 booyaka systrace: deny user: gerard, prog: /usr/pkg/bin/
bash, pid: 471(0)[0], policy: /usr/pkg/bin/bash, filters: 79, syscall:
netbsd-fsread(278), filename: /usr/pkg/sbin/ifconfig
Mar  4 18:05:43 booyaka systrace: deny user: gerard, prog: /usr/pkg/bin/
bash, pid: 471(0)[0], policy: /usr/pkg/bin/bash, filters: 79, syscall:
netbsd-fsread(278), filename: /usr/games/ifconfig
Mar  4 18:05:43 booyaka systrace: deny user: gerard, prog: /usr/pkg/bin/
bash, pid: 471(0)[0], policy: /usr/pkg/bin/bash, filters: 79, syscall:
netbsd-fsread(278), filename: /usr/local/sbin/ifconfig
```

2ème tentative : on a tenté de faire une complétion automatique du shell qui a échouée, la preuve avec les 2 lignes ci-dessous :

```
Mar  4 18:07:06 booyaka systrace: deny user: gerard, prog: /usr/pkg/bin/
bash, pid: 471(0)[0], policy: /usr/pkg/bin/bash, filters: 79, syscall:
netbsd-fsread(278), filename: /sbin
```

```
Mar  4 18:07:06 booyaka systrace: deny user: gerard, prog: /usr/pkg/
bin/bash, pid: 471(0)[0], policy: /usr/pkg/bin/bash, filters: 79,
syscall: netbsd-fsread(278), filename: /sbin
```

On a ensuite entré le nom du binaire en toutes lettres pour l'exécuter, refus du système.

```
Mar  4 18:07:09 booyaka systrace: deny user: gerard, prog: /usr/pkg/
bin/bash, pid: 505(0)[471], policy: /usr/pkg/bin/bash, filters: 79,
syscall: netbsd-execve(59), filename: /sbin/ifconfig, argv: /sbin/
ifconfig
Mar  4 18:07:09 booyaka systrace: deny user: gerard, prog: /usr/pkg/
bin/bash, pid: 505(0)[471], policy: /usr/pkg/bin/bash, filters: 79,
syscall: netbsd-fsread(278), filename: /sbin/ifconfig
Mar  4 18:07:09 booyaka systrace: deny user: gerard,
prog: /usr/pkg/bin/bash, pid: 505(0)[471], policy: /usr/
pkg/bin/bash, filters: 79, syscall: netbsd-fsread(278),
filename: /sbin/ifconfig
```

Magnifique, Gérard, n'est-ce pas ? Maintenant, il te reste à faire les politiques pour les commandes que tes utilisateurs exécuteront. C'est long et dur, je ne le sais que trop bien, mais quel bonheur quand tout marche !

SIX GELS

Maintenant Gérard, tu n'es pas sans savoir que FreeBSD dispose d'un appel système particulier, `jail()`, qui permet de créer des machines virtuelles. L'idée de départ est de pouvoir donner les droits root à quelqu'un sur une partition du système, sans qu'il ne puisse être root du système lui-même en posant des restrictions sur la portée des requêtes. Cet appel système n'a été implémenté ni dans NetBSD ni dans OpenBSD, mais rassure-toi, des petits malins ont eu une idée : utiliser `systrace(4)` pour faire le boulot. Le projet a été astucieusement nommé `sysjail` **[2]**. Je te propose de monter notre petite cage et d'y accéder à distance avec OpenSSH.

Tout d'abord, on récupère sur le site des auteurs le *package* pour son OS, que ce soit la serviette orange ou le poisson qui pique. Une fois installé, il nous reste à créer la base. Pour cela, un script shell est fourni :

```
# mksysjail-base -s ftp.fr.netbsd.org\
/pub/NetBSD/NetBSD-`uname -r`/`uname -m`/binary/sets /usr/jails/jail1
```

Le système de base installé et le `rc.conf` écrit, nous terminons la configuration en ajoutant une adresse IP sur l'interface externe de la machine hôte pour pouvoir joindre notre cage de l'extérieur. Enfin, on démarre notre *jail* pour voir la magie opérer :

```
booyaka# sysjail --cmd-enable -i /usr/jails/jail1 cage 172.20.0.3 /bin/sh
/etc/rc
700796917
Sun Mar  4 11:50:19 UTC 2007
Checking for botched superblock upgrades: done.
Starting file system checks:
mount: /: unknown special file or file system.
Setting tty flags.
ttyflags: open /dev/console: No such file or directory
Setting sysctl variables:
Starting network.
route: socket: Protocol not supported
ifconfig: SIOCGIFFLAGS lo0: Operation not permitted
route: socket: Protocol not supported
ifconfig: SIOCGIFFLAGS ne2: Operation not permitted
ifconfig: SIOCGIFFLAGS lo0: Operation not permitted
Configuring network interfaces:.
Building databases...
install: /var/run/utmp: chflags: Operation not permitted
install: /var/run/utmpx: chflags: Operation not permitted
Starting syslogd.
Checking for core dump...
savecore: /netbsd: kvm_openfiles: /dev/mem: No such file or directory
Mounting all filesystems...
Clearing /tmp.
Creating a.out runtime link editor directory cache.
Checking quotas: done.
Starting virecover.
Starting local daemons:.
Updating motd.
sendmail: /etc/mail/aliases.db not present, generating
WARNING: local host name (cage) is not qualified; see cf/README: WHO
/etc/mail/aliases: 22 aliases, longest 10 bytes, 246 bytes total
Starting sendmail.
Starting inetd.
Starting cron.
Sun Mar  4 11:50:22 UTC 2007
```

Notre cage se lance correctement. Pour le vérifier utilisons `sjls(8)` :

```
booyaka# sjls
        JID  IP Address      Hostname                      Path
   700796917  172.20.0.3      cage                          /usr/jail
jail1
```

On récupère le jail ID dans la première colonne. Cet identifiant va nous servir à passer des commandes à notre cage depuis la machine hôte, à l'aide de `sysjail-cmd(8)` :

```
booyaka# sysjail-cmd -p /var/run/sj-700796917  exec ps ax
Started child: 1310.
```

Dans le terminal où on a lancé la cage, on voit apparaître la sortie de la commande :

```
  PID TTY   STAT    TIME COMMAND
 1177 ?     Sxs  0:00.03 sendmail: accepting connections
 1469 ?     Ixs  0:00.01 /usr/sbin/inetd -l
 1565 ?     Ixs  0:00.01 /usr/sbin/syslogd -s
 1788 ?     Ixs  0:00.01 /usr/sbin/cron
 1310 pts/0 Rx+  0:00.08 ps ax
```

Passons aux choses sérieuses : la mise en place d'un `sshd(8)` dans notre cage. Nous avons toujours notre

alias IP et il reste à faire *binder* correctement le `sshd(8)` de la machine hôte et de notre jail. La configuration par défaut du démon le fait écouter sur `INADDR_ANY`, autrement dit `0.0.0.0` ou encore toutes les interfaces de ta machine, ce qui comprend évidemment notre alias IP. Corrigeons tout ça en éditant le `/etc/ssh/sshd_config` de la machine hôte :

```
ListenAddress 127.0.0.1
ListenAddress 172.20.0.2
```

Ensuite, on édite le même fichier, mais dans l'arborescence de la cage. On aura pris soin auparavant d'activer `sshd(8)` dans le `rc.conf` :

```
ListenAddress 172.20.0.3
# pas de listen sur le loopback, il est partagé avec la machine hôte !
```

Enfin, avant de pouvoir goûter aux joies du ssh, il faut créer les *devices* nécessaires au bon fonctionnement de notre jail en lançant le script `MAKEDEV`. Et voilà, Gérard, nos efforts sont récompensés, nous pouvons nous enfermer dans notre cage, y faire des choses sales, le système hôte n'en souffrira pas !

THÈSE, ANTITHÈSE, THÉRÈSE

C'est fini pour aujourd'hui, Gérard, j'espère te revoir bientôt dans le monde merveilleux des BSD libres, car tu le vaux bien. N'hésite pas à t'enfermer dans des cages pendant de longues heures et à te faire de belles tresses avec ta magnifique chevelure. Après tout, tu as ta liberté d'expression capillaire !

RÉFÉRENCES

[1] http://monkey.org/~jose/software/stsh/stsh-0.3.1.tar.gz

[2] http://sysjail.bsd.lv/

FILESYSTEMS ENCRYPTÉS SOUS NETBSD ET FREEBSD PAR LA PRATIQUE

OU COMMENT PLANQUER SON SAC DE BILLES DANS SON SLIP ?

Tu penses avoir une protection à toute épreuve contre l'accès aux données de tes disques durs : tu les caches sous ton oreiller et dans tes piles de t-shirts (sales ou pas, ça dépend du niveau de protection souhaitée). Ce que tu ne sais pas, c'est que même le chat des voisins sait où tu les ranges. Et tellement la routine de ces actions te saoule que tu as fini par capituler et les laisser en place dans leurs boîtiers.

Heureusement, il existe plus seyant et plus pratique que les immondes disques durs dans des caddies extractibles, et tu vas pouvoir te balader l'esprit plus tranquille pour les données de ton notebook ou celles de ta workstation, si jamais tu venais à en être séparé malgré toi.

À compter de maintenant, si tu continues à te faire piquer tes billes par les copains de récré, c'est que tu l'auras bien voulu !

NETBSD : CGD

QU'EST-CE QUE CGD ?

CGD signifie *cryptographic disk driver*. La jolie page de *man* le décrivant se trouve là : http://netbsd.gw.com/cgi-bin/man-cgi?cgd.

C'est un système d'*encryption* par bloc implémenté au niveau du *kernel*. Les algorithmes disponibles sont :

- `aes-cbc` ;
- `3des-cbc` ;
- `blowfish-cbc`.

L'outil pour manipuler CGD s'appelle `cgdconfig`. Sa page de man est ici : http://netbsd.gw.com/cgi-bin/man-cgi?cgdconfig.

Il est nécessaire que le support pour CDG soit ajouté à la configuration de votre kernel avant de pouvoir l'utiliser (on s'en douterait m'ame Michu, mais c'est un rappel pour ceux qui ne sont pas allé lire les liens ci-dessus).

Le chapitre dédié à CGD dans le Guide NetBSD se trouve à cette URL : http://www.netbsd.org/guide/en/chap-cgd.html. Il est très complet et bien détaillé.

DANS LE VIF DU SUJET

On va utiliser une partition disponible et déjà prête. La commande suivante crée le fichier de paramètres (`paramsfile`) `/etc/cgd/wd1e`, utilisant l'algorythme `aes-cbc`, une méthode de vérification `disklabel` et une clé d'une taille de 256 bits.

```
# cgdconfig -g -V disklabel -o /etc/cgd/wd1e aes-cbc 256
```

Du fait du choix de la méthode de vérification (`disklabel`) et de l'absence dudit `disklabel` sur le disque logique, on doit « forcer » le mécanisme de vérification pour la première configuration de CGD.

```
# cgdconfig -V re-enter cgd0 /dev/wd1e
/dev/wd1e's passphrase: <password>
re-enter device's passphrase: <password>
```

On vérifie le `disklabel` :

```
# disklabel -e -I cgd0
# /dev/rcgd0d:
type: cgd
disk: cgd
label: fictitious
flags:
bytes/sector: 512
sectors/track: 2048
tracks/cylinder: 1
sectors/cylinder: 2048
cylinders: 13703
total sectors: 28065492
rpm: 3600
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0           # microseconds
track-to-track seek: 0  # microseconds
drivedata: 0

4 partitions:
#      size offset fstype [fsize bsize cpg/sgs]
a: 28065492      0 4.2BSD      0     0     0  # (Cyl.0-13703*)
d: 28065492      0 unused      0     0        # (Cyl.0-13703*)
```

Et on formate la partition encryptée.

```
# newfs /dev/cgd0a
/dev/cgd0a: 13703.9MB (28065492 sectors) block size 16384,
           fragment size 2048
        using 75 cylinder groups of 182.72MB, 11694 blks,
        23040 inodes.
super-block backups (for fsck_ffs -b #) at:
32, 374240, 748448, 1122656, 1496864, 1871072, 2245280,
2619488, 2993696, 3367904, 3742112, 4116320, 4490528,
4864736, 5238944,
.....................................................
```

On fait un `fsck` sur le volume encrypté, on le *mounte* et on y met un fichier témoin.

```
# fsck -t ffs /dev/cgd0a /dev/rcgd0a
File system is clean; not checking
# mount -t ffs /dev/cgd0a /mnt
# touch /mnt/test.txt
```

Vérification de la détection du volume par les scripts de *boot* :

- on ajoute `cgd=YES` dans le fichier `/etc/rc.conf`.
- on configure le fichier `/etc/cgd/cgd.conf`.

```
cgd0    /dev/wd1e
```

- on ajoute le point de montage dans `/etc/fstab`.

```
/dev/cgd0a /crypted ffs rw,softdep,noatime,nodevmtime,nocoredump 2 2
```

On vérifie que le script `/etc/rc.d/cgd` retrouve ses petits :

```
# /etc/rc.d/cgd start
Configuring CGD devices.
/dev/wd1e's passphrase:
# mount -a
# mount
/dev/cgd0a on /crypted type ffs
        (nocoredump, noatime, nodevmtime,
         soft dependencies, local)
```

Voilà, on vient de vérifier que notre configuration CGD était reconnue par le système. Il reste à procéder à un test grandeur nature en redémarrant sa machine.

Si tout est configuré correctement, le système demandera ta *passphrase* (ATTENTION, le clavier est alors en qwerty) et devrait monter la partition encryptée.

Une fois que tu es content du résultat, il te reste à mettre tes données sur `/crypted`.

ENCRYPTER LA PARTITION DE SWAP

Les exemples de partition de swap encryptée que l'on trouve sur **http://www.s-mackie.demon.co.uk/unix-notes/NetBSD-CGD-Setup.html** et **http://www.nycbug.org/uploads/_netbsdcgd.html** utilisent une partition de swap en tant que partition d'un volume crypté. Je m'explique.

Une partition complète est associée à un *device* `cgdX` et dans ce « disque logique » `cgdX` sont créées diverses partitions dont celle de swap de la même manière que l'on procède sur un disque normal `wdX` : a, b, e, f, g...

Premières manipulations : vérifier que tout ceci fonctionne manuellement. La commande `swapctl` permet de vérifier le swap actuellement configuré.

```
# swapctl -l
Device      512-blocks     Used    Avail Capacity  Priority
/dev/wd0b       263088        0   263088     0%    0
```

```
# cgdconfig -g -o /etc/cgd/wd1i -V none \
  -k randomkey blowfish-cbc
# cgdconfig cgd1 /dev/wd1i
# disklabel -e -I cgd1
# /dev/rcgd1d:
type: cgd
disk: cgd
label: fictitious
flags:
bytes/sector: 512
sectors/track: 2048
tracks/cylinder: 1
sectors/cylinder: 2048
cylinders: 619
total sectors: 1269072
rpm: 3600
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0           # microseconds
track-to-track seek: 0  # microseconds
drivedata: 0

4 partitions:
#     size offset fstype [fsize bsize cpg/sgs]
b: 1269072      0   swap                    # (Cyl.0-619*)
d: 1269072      0 unused      0     0       # (Cyl.0-619*)

# swapctl -a /dev/cgd1b
# swapctl -l
Device      512-blocks     Used    Avail Capacity  Priority
/dev/wd0b       263088        0   263088     0%    0
/dev/cgd1b     1269072        0  1269072     0%    0
Total          1532160        0  1532160     0%
```

Maintenant, essayons de rendre tout ceci automatique. Dans un premier temps, on va sauvegarder le `disklabel` de `cgd1` pour que `swapctl` ne se plaigne pas de ne pas trouver de label.

```
# disklabel -r  cgd1 > /etc/cgd/cgd1.disklabel
```

Ensuite, on crée le fichier `/etc/rc.conf/cgd` qui sera chargé de remettre le label sur le disque encrypté `cgd1` à chaque reboot :

```
swap_device="cgd"
swap_disklabel="/etc/cgd/cgd1.disklabel"
start_postcmd="cgd_swap"

cgd_swap()
{
  if [ -f $swap_disklabel ]; then
    disklabel -R -r $swap_device $swap_disklabel
  fi
}
```

On ajoute la ligne suivante au fichier etc/fstab :

```
/dev/cgd1b  none  swap  sw  0 0
```

On vérifie avec les scripts de démarrage :

```
# /etc/rc.d/cgd start
Configuring CGD devices.
/dev/wd1e's passphrase:
# swapctl -l
no swap devices configured
# /etc/rc.d/swap1 start
swapctl: adding /dev/wd0b as swap device at priority 0
swapctl: adding /dev/cgd1b as swap device at priority 0
```

Naturellement, ici, avoir une seconde partition de swap encryptée en sus de la première qui ne l'est pas ne rime pas à grand-chose.

Un administrateur soucieux de la confidentialité de ses données s'attachera donc à appliquer les différentes commandes vues dans ce chapitre pour sécuriser /dev/wd0b.

Contrairement à GELI (FreeBSD), CGD ne permet pas de distribuer la sécurité du volume encrypté vers une passphrase et un fichier de « clés » combinés. Par contre, CGD permet « facilement » de manipuler des CD et DVD qu'on aura au préalable encryptés.

FREEBSD : GELI

Hop, te voilà en jambe avec la première partie pour NetBSD. Ne t'assieds pas, on continue.

QU'EST-CE QUE GELI ?

GELI signifie tout simplement GEOM ELI. GEOM est un *framework* modulaire de transformation d'I/O sur les disques. Dit plus simplement, c'est une surcouche de manipulation des objets « disques » qui va permettre d'effectuer des opérations différentes dans un « environnement unifié ».

Les pages de manuel à lire sont les suivantes :

http://www.freebsd.org/cgi/man.cgi?query=geom&sektion=4

http://www.freebsd.org/cgi/man.cgi?query=geli&sektion=8

ENCRYPTER SON SWAP

Enfin, surtout celui de sa machine. Tout d'abord, on vérifie l'utilisation d'une partition de swap :

```
# swapinfo
Device          1K-blocks     Used    Avail Capacity
/dev/ad0s1b       1048576        0  1048576     0%
```

On le désactiver et on vérifie :

```
# swapoff /dev/da0b
# swapinfo
Device          1K-blocks     Used    Avail Capacity
(rien)
```

Le fichier /etc/defaults/rc.conf nous indique les valeurs par défaut des différentes options passées à GELI pour le montage du swap au boot.

```
# Options for GELI-encrypted swap partitions.
geli_swap_flags="-a aes -l 256 -s 4096 -d"
```

Ces valeurs conviennent parfaitement, mais dans le cas où il faudrait les modifier, c'est dans le fichier /etc/rc.conf qu'elles devront être changées.

On modifie le fichier /etc/fstab pour indiquer qu'on va vouloir utiliser une partition de swap encryptée avec GELI (on ajoute simplement .eli à la fin de la partition de swap).

```
 # Device        Mountpoint  FStype Options  Dump Pass#
/dev/ad0s1b.eli       none    swap      sw     0    0
```

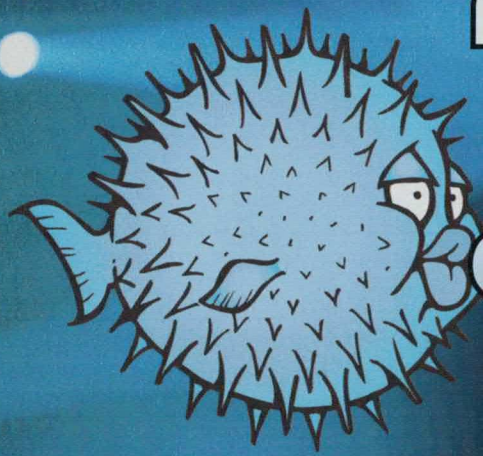
Et maintenant, que se passe-t-il ? On reboote et on voit ce qu'il se passe ? Eh bien non, on exécute les scripts suivants :

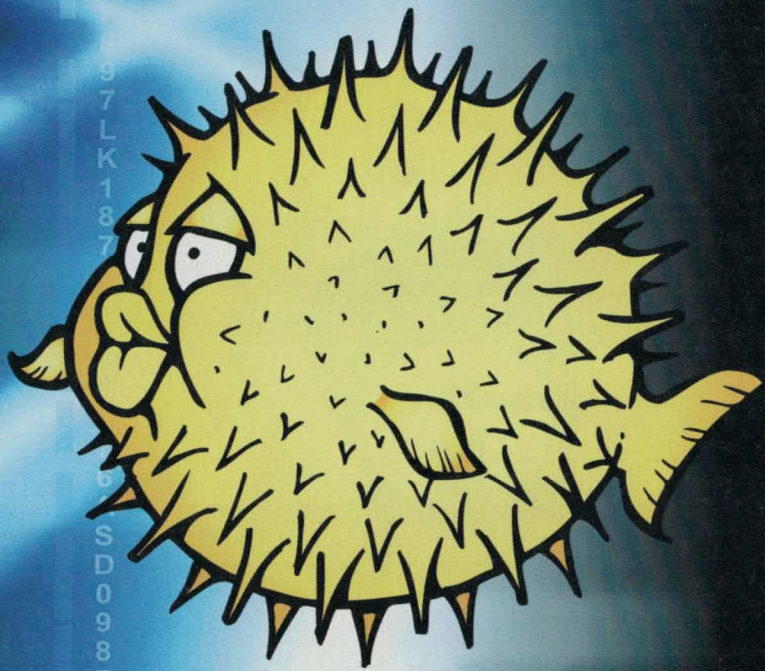
```
# /etc/rc.d/ecnswap start
# ls -al /dev/ad0s1b*
crw-r----- 1 root operator 0, 96 Jan 3 21:04 /dev/ad0s1b
crw-r----- 1 root operator 0,106 Jan 3 21:04 /dev/ad0s1b.eli

# /etc/rc.d/swap1 start
swapon: adding /dev/ad0s1b.eli as swap device

# swapinfo
Device          1K-blocks     Used    Avail Capacity
/dev/ad0s1b.eli   1048576        0  1048576     0%

# geli status
      Name  Status  Components
ad0s1b.eli     N/A  ad0s1b
```

Libre – Stable – Sécurisé

OpenBSD

www.OpenBSD.org





FA9G0A32KV87AD901NAD97LK1876SD0987F12K92ASF

The Power to serve

FreeBSD®

Ah !!! On obtient un swap encrypté via GELI et « mounté ». Évidemment, lors du prochain reboot, il est plus que recommandé de vérifier que le swap est bien toujours encrypté, mais, cette fois, de manière complètement automatique, y compris le montage.

ENCRYPTER UNE PARTITION : /HOME

À partir d'ici, il est INDISPENSABLE d'avoir des *backups* des données de la partition sur laquelle les différentes commandes seront exécutées.

On choisit d'encrypter une partition existante `/home`. Si tu as des utilisateurs, le plus simple est de rebooter en *single user* et de ne pas monter `/home` après le `fsck` : ils ne pourront plus se loguer, ni lancer de *processes* qui modifieraient des fichiers sur cette partition.

Initialisation de GELI pour la partition `/home` :

```
# geli init -a aes -l 256 ad0s1f
Enter new passphrase: <password>
Reenter new passphrase: <password>
```

Il faut maintenant formater le volume pointé par la partition encryptée :

```
# newfs -U -O2 /dev/ad0s1f.eli
geli attach ad0s18
```

Modification de `/etc/fstab` pour prendre en compte cette partition « GELI-fiée »

```
# Device      Mountpoint  FStype  Options   Dump  Pass#
/dev/ad0s1f       /home      ufs       rw      2     2
```

qui devient :

```
# Device      Mountpoint  FStype  Options   Dump  Pass#
/dev/ad0s1f.eli   /home      ufs       rw      2     2
```

On définit ensuite les informations sur notre partition encryptée dans le fichier `/etc/rc.conf` :

```
geli_devices="ad0s1f"
geli_ad0s1f_flags=""
```

Il nous reste à vérifier que les scripts de démarrage ont bien toutes les informations nécessaires pour *mounter* la partition `/home` encryptée :

```
# /etc/rc.d/geli start
Configuring Disk Encryption for ad0s1f.
Enter passphrase: <password>
# mount
/dev/ad0s1f.eli on /home (ufs, local, soft-updates)
```

Enfin, on va vérifier que la partition est de nouveau mountée après un reboot et il faudra restaurer les backups pour retrouver les données qui s'y trouvaient.
Afin de ne pas limiter la sécurité du volume crypté à une seule passphrase, il est possible de modifier un volume GELI existant, via `geli setkey`, l'utilisation de fichier de clé, combiné ou non à la passphrase initiale.

- passphrase ;
- fichier de clé (*keyfile*) ;
- passphrase et fichier de clé : dans ce cas, il faut les deux éléments pour accéder au volume protégé.

Note importante pour ceux qui ont un clavier avec un autre *layout* que celui par défaut lors du boot : au moment de la demande de la passphrase, le clavier est en qwerty ; si tu veux avoir un clavier différent de qwerty dans les phases de boot de ton FreeBSD, il te faut suivre le lien suivant, section clavier : **http://imil.net/docs/sexy-FreeBSD.txt**.

LIMITES ET CONCLUSION

Comme dans tout système de protection (ici protection à l'accès des données), il faut bien avoir à l'esprit ce qui est réellement protégé.

Un système Unix possède des données importantes dans bien des endroits, `/etc` et tous ses sous-répertoires pour n'en citer qu'un, pourtant aucun de ces fichiers ne sera protégé par une partition encryptée.

Aujourd'hui, aucun système d'encryption de partition ne permet d'avoir l'ensemble les partitions encryptées : il faut au moins charger le kernel et ses modules, lire la `fstab` et exécuter les scripts de démarrage.

Personnellement, je n'encrypte que le swap et la partition `/home`, mais j'ai activé la protection de l'accès au disque par mot de passe dans le bios de mon Lappy.

Je souhaite que cet article t'ait démontré l'utilisation des partitions protégées au quotidien et t'ait convaincu qu'en cas de vol cela limite la perte à une perte matérielle sans y ajouter la crainte que ces données ne soient accessibles à n'importe qui (à condition d'avoir des sauvegardes récentes).

Une suite de cet article pourrait être la mise en place d'un système entièrement encrypté, où le kernel et ses modules se trouveraient sur une clé USB.

RÉFÉRENCES

Les pages de man et URL citées dans le présent article, ainsi que **http://sclo.retiaire.org/wiki/doku.php?id=slash:freebsd:geli**.

GARDER SON PHACOCHÈRE FAMILIER DANS UN ENCLOS

OU LES JAILS SOUS FREEBSD, VERSION GRUIK

Ton phacochère [1] familier, bien que très sympathique, continue de faire quelques bêtises dans ta maison, bouffer les meubles, vider sa gamelle un peu partout, voire embêter tes autres colocataires.

Ne voulant pas te résoudre à l'enfermer dans une niche (chroot) trop étroite, tu décides de lui construire un enclos qui lui sera dédié et dans lequel il pourra faire ce que bon lui semble sans nuire à son entourage (continue to be root).

PRÉ-REQUIS

- un terrain propice (avoir un FreeBSD sous la main, être *root* dessus) ;
- un peu d'espace disque, car on va dupliquer tout le *userland* dans l'enclos ;
- avoir les sources dans `/usr/src`.

AVANT D'ENVISAGER LA CONSTRUCTION

Avant de pouvoir configurer son enclos (`jail`), il faut déjà préparer le terrain tout autour (configurer le serveur qui va accueillir les *jails* pour que les *daemons* n'écoutent plus sur toutes les IP).

Donc, tout autour de l'emplacement de son futur enclos, on fixe les branches qui dépassent, on ramasse les feuilles mortes, on passe un coup de râteau et on laisse un emplacement le plus nickel possible.

- `sshd`

Utilisez `sshd_flags="-oListenAddress=Public_IP"` dans votre fichier `/etc/rc.conf` ou modifiez `/etc/ssh/sshd_config` (et redémarrez le daemon sshd par `/etc/rc.d/sshd restart`)

- `sendmail`

Ajoutez `DAEMON_OPTIONS(`Addr=Public_IP')` à votre fichier de macro sendmail (fichier `mc`)

- `inetd`

Si vous utilisez `inetd`, il faut ajouter `inetd_flags="-a Public_IP"` à votre `/etc/rc.conf` (attention à bien reprendre les flags existants).

- `named`

Utilisez la directive `listen-on { Public_IP; };` dans votre fichier de configuration `named.conf`.

- `apache`

Utilisez la directive `Listen Public_IP:80` dans votre `/usr/local/etc/httpd.conf`

- etc.

Idem pour les autres daemons écoutant sur toutes les IP.

UN ENCLOS EN KIT

Tu as acheté ton enclos au même endroit que ton abri de jardin et, bien aimablement, le fabricant a fourni une aide minimale au montage des différents éléments ; pourtant, tout n'est pas aussi simple que chez Ikea. Tu n'auras pas de notice imprimée recto-verso t'expliquant ce qu'est une vis, mais cela ne signifie pas qu'il n'y a pas de documentation !

Extrait de la page de man de `jail` [2]

```
EXAMPLES
   Setting up a Jail Directory Tree
     To set up a jail directory tree containing an entire FreeBSD
     distribution, the following sh(1) command script can be used:


     D=/here/is/the/jail
     cd /usr/src
     mkdir -p $D
     make world DESTDIR=$D
     make distribution DESTDIR=$D
     mount_devfs devfs $D/dev
```

Si tu as compilé ton système récemment, tu peux remplacer `make world DESTDIR=$D` par `make installworld`

DESTDIR=$D ce qui te fera économiser tout le temps de la compilation.

Une méthode plus Gruik consiste à monter une image ISO de FreeBSD et à extraire l'environnement de base comme s'il s'agissait d'une installation d'un système :

```
# ggatel create 6.2-RELEASE-i386-disc1.iso
ggate0
# mount -t cd9660 /dev/ggate0 /mnt
# cd /mnt/6.2-RELEASE/base
# mkdir -p /home/jails/gcujail
# cat base.?? | tar --unlink -vxpzf - -C /home/jails/gcujail
```

On va créer manuellement le /dev/* du jail et y « entrer » :

```
# mount_devfs devfs /home/jails/gcujail/dev
# jail -l -U root /home/jails/gcujail gcujail 127.0.0.1 /bin/csh
```

On regarde un peu dans quel environnement on se retrouve :

```
gcujail# uname -a
FreeBSD gcujail 7.0-CURRENT FreeBSD 7.0-CURRENT #33: Sat Mar 17
19:35:45 CET 2007     root@stealth.domain.tld:/usr/obj/usr/src/
sys/STEALTH  i386
gcujail# ls -al
-rw-r--r--   2 root  wheel   801 Jan 12 07:42 .cshrc
-rw-r--r--   2 root  wheel   251 Jan 12 07:42 .profile
-r--r--r--   1 root  wheel  6196 Jan 12 07:42 COPYRIGHT
drwxr-xr-x   2 root  wheel  1024 Jan 12 07:41 bin
drwxr-xr-x   5 root  wheel   512 Jan 12 07:42 boot
[snip]
gcujail# mount
/dev/ad0s1f on / (ufs, local, soft-updates)
```

On s'aperçoit que le système dans le jail hérite de la version du kernel du serveur hôte malgré le userland « décalé » et que la sortie de la commande mount ne montre que la partition accueillant le jail.

De même, depuis le jail, la commande uptime montre le résultat du serveur hôte et pas du jail en lui-même.

Pour vous en convaincre, créez le fichier /etc/rc.local contenant

```
echo `date` >> /root/boot.log
```

dans le jail ; amusez-vous à faire plusieurs *reboot* et regarder le fichier de log et l'*uptime*.

Pour le moment, le user root du jail voit tout /dev à l'identique de ce qu'il se trouve sur le serveur hôte, ce qui peut être très dangereux si vous n'avez pas confiance dans le root du jail, car il a accès à tous les disques et partitions, à la configuration de vos règles de firewall...

ET LA BARRIÈRE ÉLECTRIQUE ?

Dans un jail correctement configuré comme plus loin dans l'article, le root du jail ne peut pas configurer son propre firewall et, par conséquent, il ne peut pas choisir et *setter* ses *rules*.

```
# pfctl -s rules
pfctl: /dev/pf: No such file or directory
```

Toute la configuration firewall se fait « au-dessus » du jail, sur le serveur hôte.

CONFIGURATION PAR DÉFAUT

Le système FreeBSD comporte une configuration par défaut et des exemples pour configurer votre/(vos) jail(s).

Ces informations se trouvent dans /etc/defaults/rc.conf qui est lu par les scripts de démarrage ; toutes les modifications doivent se faire dans /etc/rc.conf.

Voici les paramètres communs à tous les jails :

```
##############################################################
### Jail Configuration #######################################
##############################################################
jail_enable="NO"        # Set to NO to disable starting of any jails
jail_list=""            # Space separated list of names of jails
jail_set_hostname_allow="YES" # Allow root user in a jail to change its hostname
jail_socket_unixiproute_only="YES" # Route only TCP/IP within a jail
jail_sysvipc_allow="NO" # Allow SystemV IPC use from within a jail
```

Et voici la liste des paramètres spécifiques à un jail qui peuvent être repris pour la configuration de vos jails :

Voir Figure 1, page 46.

CONFIGURONS NOTRE ENCLOS

Voici ce que j'ai mis dans le /etc/rc.conf du serveur hôte ; l'utilisation de chaque variable sera détaillée plus loin.

```
# jail
jail_enable="YES"
jail_list="gcujail"
# Emplacement du jail et son hostname
jail_gcujail_rootdir="/home/jails/gcujail"
jail_gcujail_hostname="gcujail.gcu.info"
# ip du jail et interface qui portera l'alias
jail_gcujail_ip="192.168.0.10"
jail_gcujail_interface="bge0"
# on veut un /dev restreint dans le jail
jail_gcujail_devfs_enable="YES"
```

```
jail_gcujail_devfs_ruleset="devfsrules_jail"
# l'option -J permettra d'avoir des informations sur le jail
jail_gcujail_flags="-l -U root -s 1 -J /var/run/jail_gcujail.jid"
```

Dans le rc.conf du jail, on lui indique de démarrer sshd comme d'habitude :

```
sshd_enable="YES"
sendmail_enable="NONE"
```

Apparemment, il n'est plus indispensable de commenter adjkerntz -a dans /etc/crontab du système jailé.

Démarrons ce jail et regardons ce qui est vu du serveur hôte.

```
# /etc/rc.d/jail start gcujail
Configuring jails:.
Starting jails: gcujail.gcu.info.
# ifconfig bge0
bge0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=98<VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM>
        [snip]
        inet 192.168.0.10 netmask 0xffffffff broadcast 192.168.0.10
# mount
[snip]
devfs on /home/jails/gcujail/dev (devfs, local)
```

Ce qu'il s'est produit :

- l'alias IP 192.168.0.10 est automatiquement ajouté sur l'interface précisée par jail_gcujail_ip ;
- devfs s'occupe de monter le /dev du jail au démarrage de celui-ci ;
- les *devices* visibles dans le /dev du jail ont réduit comme peau de chagrin et ne permettent plus d'outrepasser les devices sur le serveur hôte (faites un ls pour vérifier).

Voir Figure 2, page 46.

Vous aurez remarqué que le jail est démarré en securelevel [3] à 1 grâce à -s 1 de la variable jail_gcujail_flags.

LES LOGS

Activons les logs de la console dans le jail (fichier /home/jails/gcujail/etc/syslog.conf) :

```
console.info;*.err;kern.warning;auth.notice     /var/log/console.log
```

Créons le fichier en question et redémarrons le jail :

```
# touch /home/jails/gcujail/var/log/console.log
# /etc/rc.d/jail restart
```

Ce qui aura pour effet de créer un fichier /var/log/jail_gcujail_console.log sur le serveur hôte reprenant les logs de la console.

REMPLIR L'ENCLOS

Afin d'économiser un peu d'espace disque, il vous semble évident que vous allez limiter au strict minimum les informations dupliquées sur le serveur hôte ainsi que dans votre (vos) jail(s).

Typiquement, on partage /usr/src et /usr/ports :

```
# mount -t nullfs -o ro /usr/src /home/jails/gcujail/usr/src
# mount -t nullfs -o ro /usr/ports /home/jails/gcujail/usr/ports
# mount | grep nullfs
/usr/src on /home/jails/gcujail/usr/src (nullfs, local, read-only)
/usr/ports on /home/jails/gcujail/usr/ports (nullfs, local, read-onl
```

Il est possible d'ajouter ces points de montage dans le /etc/fstab du serveur hôte afin que le jail y ait accès de manière permanente, mais attention, ces répertoires peuvent être supprimés depuis le jail !

Pour plus de sécurité, il est recommandé de monter /usr/src et /usr/ports/ en *read-only* dans le jail et de modifier la manière dont sont compilés les ports à partir du jail (pour pouvoir utiliser un /usr/ports en RO).

Si vous devez gérer plusieurs enclos, il est préférable de mettre en place un export NFS en read-only de ces répertoires vers les jails, ce qui évitera les points de montage en nullfs :

```
# mkdir -p /usr/obj/portsbuild /usr/obj/distfiles
```

Et on édite le /etc/make.conf du jail pour y ajouter les lignes suivantes :

```
WRKDIR=/usr/obj/portsbuild
DISTDIR=/usr/obj/distfiles
```

Il suffit d'ajouter les ports dans le jail comme d'habitude. Ceci aura pour inconvénient d'ignorer le contenu de /usr/ports/distfiles ; n'hésitez pas à en recopier les fichiers s'ils existent, ce qui évitera de perdre de la bande passante et du temps.

Pensez à faire régulièrement le ménage dans /usr/obj/distfiles du jail sans quoi de l'espace disque sera perdu pour rien.

SÉCURI-CLOS

Au cours de l'installation du jail, tu as constaté qu'il tenait sur une seule partition, ce qui est assez gênant

quand tu veux avoir des points de montage avec des droits différents.

Par exemple, j'ai pris pour habitude d'utiliser un /tmp de taille fixe, monté en RAM, avec des droits noexec et nosuid.

Habituellement, je déclare dans /etc/rc.conf les lignes suivantes :

```
tmpmfs="YES"
tmpsize="512m"
tmpmfs_flags="-S -o noexec,nosuid"
```

qui ont pour résultat :

```
# mount | grep tmp
/dev/md0 on /tmp (ufs, local, noexec, nosuid)
```

Mais cela ne fonctionne pas dans gcujail tout simplement parce que les commandes mdconfig n'ont pas accès aux devices /dev/md* depuis le jail !

Les ajouter serait une fausse bonne idée, puisque cela donnerait accès aux devices md* appartenant au serveur hôte depuis l'intérieur du jail.

Oh, tu peux essayer de bricoler le /etc/fstab du jail, si tu arrives à quelque chose de concluant tu es prié d'écrire à Pinpin (ou à l'auteur).

La solution, peu gracieuse, trouvée pour l'instant, est la suivante :

- sur le serveur hôte, on crée un fichier vide d'une taille fixe :

```
# dd if=/dev/zero of=/home/jails/gcujail.tmp bs=1m count=512
512+0 records in
512+0 records out
536870912 bytes transferred
```

Edition du fichier /etc/fstab.gcujail (qui sera pris par défaut par le script jail) :

Voir Figure 3, page 46.

et on ajoute la gestion du mount dans le fichier /etc/rc.conf du serveur hôte pour ce jail :

```
jail_gcujail_mount_enable="YES"
```

Par défaut, le fichier fstab du jail pris en compte par la variable jail_gcujail_fstab sera /etc/fstab.gcujail.

On redémarre le jail :

```
# /etc/rc.d/jail restart
#  mount | grep md
/dev/md1 on /home/jails/gcujail/tmp (ufs, local, noexec, nosuid)
```

Le /tmp de notre jail est bien d'une taille limitée et monté avec des droits moins permissifs.

NOTE

Un inconvénient de cette méthode est que bien qu'un /etc/rc.d/jail stop démonte le /tmp du jail, cela ne libère pas le device md qui y était associé et il faut le faire à la mimine depuis le serveur hôte !

QUOTA-CLOS

Jusqu'à présent, ton phacochère est tout seul dans son enclos. Il peut le décorer comme il lui plaît et ajouter ce dont il a besoin, mais, surtout, il peut continuer à prendre tout l'espace disponible dans son enclos.

Il y a au moins deux solutions pour résoudre ce problème :

- mettre les répertoires accueillant les jails dans des partitions physiques et donc de taille fixe ;
- utiliser l'astuce d'un filesystem sur un fichier comme ce qu'on a fait avec /home/jails/gcujail.tmp, ce système de fichiers étant monté depuis le système hôte avant le démarrage du jail.

MANAGER PLUSIEURS ENCLOS

Divers programmes sont disponibles dans les ports afin de manager plusieurs jails plus facilement, liste évidemment non exhaustive :

```
# make search name=jail | grep -E "^Port|^Path|^Info"
Port:   jailaudit-1.2
Path:   /usr/ports/security/jailaudit
Info:   Script to generate portaudit reports for jails

Port:   ezjail-2.0.1
Path:   /usr/ports/sysutils/ezjail
Info:   A framework to easily create, manipulate and run
FreeBSD jails

Port:   jailadmin-1.8_2
Path:   /usr/ports/sysutils/jailadmin
Info:   A system for managing a set of named jails

Port:   jailctl-0.71
Path:   /usr/ports/sysutils/jailctl
Info:   Jail management tool
```

```
Port:   jailer-1.1.2
Path:   /usr/ports/sysutils/jailer
Info:   Manage FreeBSD jail startup, shutdown and console

Port:   jailuser-1.9_1
Path:   /usr/ports/sysutils/jailuser
Info:   Builds a chrooted environment

Port:   jailutils-1.0
Path:   /usr/ports/sysutils/jailutils
Info:   Several utilities for managing jails

Port:   p5-BSD-Jail-Object-0.02
Path:   /usr/ports/sysutils/p5-BSD-Jail-Object
Info:   An object oriented perl interface to jail(2)
```

Pour une gestion plus facile des jails depuis le serveur hôte, je conseille l'installation des ports `jailadmin` et `jailutils` ; l'*upgrade* des enclos se fait de la même manière qu'un serveur FreeBSD (y compris pour le `mergemaster`) sauf qu'on change la variable `$DESTDIR` pour pointer dans le répertoire du jail à mettre à jour.

CONCLUSION

Et voici un petit tour d'horizon sur les jails avec, je l'espère, une présentation qui sort un peu des *howtos* classiques qu'on trouve sur différents sites web.

RÉFÉRENCES

[1] http://fr.wikipedia.org/wiki/Phacochère

[2] http://www.freebsd.org/cgi/man.cgi?query=jail&sektion=8

[3] http://www.freebsd.org/cgi/man.cgi?query=securelevel&sektion=8

Figure 1

```
#
# To use rc's built-in jail infrastructure create entries for
# each jail, specified in jail_list, with the following variables.
# NOTES:
# - replace 'example' with the jail's name.
# - except rootdir, hostname and ip, all of the following variables may be made
#   global jail variables if you don't specify a jail name (ie. jail_interface).
#
#jail_example_rootdir="/usr/jail/default"       # Jail's root directory
#jail_example_hostname="default.domain.com"     # Jail's hostname
#jail_example_ip="192.168.0.10"                 # Jail's IP number
#jail_example_interface=""                      # Interface to create the IP alias on
#jail_example_exec_start="/bin/sh /etc/rc"              # command to execute in jail for starting

#jail_example_exec_afterstart0="/bin/sh command"        # command to execute after the one for
                                                       # starting the jail. More than one can be
                                                       # specified using a trailing number
#jail_example_exec_stop="/bin/sh /etc/rc.shutdown"      # command to execute in jail for stopping
#jail_example_devfs_enable="NO"                 # mount devfs in the jail
#jail_example_fdescfs_enable="NO"               # mount fdescfs in the jail
#jail_example_procfs_enable="NO"                # mount procfs in jail
#jail_example_mount_enable="NO"                 # mount/umount jail's fs
#jail_example_devfs_ruleset="ruleset_name"      # devfs ruleset to apply to jail
#jail_example_fstab=""                          # fstab(5) for mount/umount
#jail_example_flags="-l -U root"                # flags for jail(8)
```

Figure 2

```
# jls
   JID  IP Address      Hostname                      Path
     6  192.168.0.10    gcujail.gcu.info              /home/jails/gcujail
# ps auxwww | grep J
root     2392  0.0  0.1  1376  1052  ??  SsJ   8:06PM   0:00.00 /usr/sbin/syslogd -s
root     2448  0.0  0.3  3524  3084  ??  IsJ   8:06PM   0:00.00 /usr/sbin/sshd
root     2455  0.0  0.1  1388  1064  ??  IsJ   8:06PM   0:00.00 /usr/sbin/cron -s
```

Figure 3

```
# Device Mountpoint                FStype  Options                                          Dump    Pass#
md       /home/jails/gcujail/tmp    mfs     rw,-S,noexec,nosuid,-F/home/jails/gcujail.tmp    2       0
```

LE LEXIQUE DES LUTINS

OU COMMENT BIEN COMMUNIQUER AVEC LES AUTEURS DE CE HORS-SÉRIE

● **gruik** : cri du cochon/porc qui programme salement.

● **rhon** : bruit du cochon/porc qui renifle un projet/un cahier des charges avant de commencer tout de suite à programmer.

● **outr** : une personne est ***outr*** lorsqu'elle est outrée. En général, cet état s'accompagne de l'***affli*** (voir ce mot).

● **affli** : se dit d'une personne affligée.

Exemple :

iMil|DfXXX : haaan (voir ce mot) comment chu rébou

** PhiR is now known as PhiR_affli.*

● **offusc** : se dit de quelqu'un qui est offusqué. L'***offusc*** se définit par la formule : ***affli*** + ***outr*** = ***offusc***.

● **han** : mot à multiples significations. Un ***han*** simple exprime la plupart du temps la soumission (voir ce mot).

Exemple :

*iMil|DfXXX : ***han*** enfoiré de pointeur !! j't'ai soumis !*

Un ***han*** répété schématise l'acte de la soumission et/ou sexuel.

Un ***han*** prolongé reflète pour sa part l'indignation ou la consternation.

Exemple :

*Gourd|n : ***haaaaaaaaaaaaaaaaan*** comment tu lui as causéééé.*

Le terme ***han*** a été détourné de son sens original. Il était en effet initialement un appendice verbal indispensable au vocabulaire féminin.

Exemple :

Jessica : haaaaaaan, comment j'hallucinhaaa aaaaaaaaaan.

● **Pinpin** : Pinpin est un bot IRC. Sa particularité est d'être relou.

● **y0, y0r, m00h, m0uh, pwet** : bonjour.

● **[&~#{[|`\ç@%$!/]+** : (regexp) se dit à la place d'une vulgarité ou éventuellement pour dire bonjour.

● **bonnes manières** : on dit de #gcu que c'est le « canal des bonnes manières ».

● **innocent** : se dit de quelqu'un qui est coupable.

Exemple :

** ojo is now known as ojo_innocent*

● **convi** : on dit que quelque chose est ***convi*** lorsqu'elle brille par sa convivialité.

Exemple :

clicka (voir ce mot) ***convi***

● **clicka** : action d'appuyer sur le bouton d'une convi-mouse.

● **bafrer** : manger.

● **bafr** : action de bafrer.

● **gl0rguer** : boire (généralement de l'alcool).

● **gl0rg** : action de gl0rguer.

● **soumission** : action de soumettre. La soumission peut s'effectuer sur deux niveaux.

soumission morale :

Jimbo : T'ES BIEN SOUMIS MAINTENANT, HEIN ?

ou encore matérielle :

iMil|cuisto : enfoiré de micro-ondes, j'l'ai soumis.

● **guri** : (n.f.) une guri est la matérialisation de l'idéal féminin. Bien qu'un peu chère (9M de crédits républicains), elle répond à tous les critères de la perfection que tout un chacun recherche en tant que moitié.

Exemple :

iMil|classe : et ce week-end de toutes façons, j'ai une commande chez guri'30.

● **khof** : (onom., dans le texte, ***khof***) appendice verbal ayant pour particularité d'insister sur l'INNOCENCE (voir ce mot) de la personne qui l'utilise.

Exemple :

*oliv3 : je ne ***khof*** non ! jamais !*

● **FNAC** : (i. e. FNAC-ONLINE) endroit on l'on ACHÈTE (voir ce mot) des produits non gratuits.

Exemple :

*Reefab : et ils ont les ***KHOF*** updates à la FNAC ?*

● **ACHETER** : (du latin pompaze) action de récupérer des fichiers informatiques chez des amis travaillant à la FNAC.

Exemple :

IMil : et l'as ACHETE où ce PATCH qui fait BIEN FONCTIONNER jk2 ?

● **Jean-Claude** : (adj. Michel) se dit de quelqu'un qui a des jantes en aluminium sur sa 206 Turbo GTI. On trouve souvent le Jean-Claude à proximité d'une usine de guris (voir ce mot). Le Jean-Claude a la particularité d'être plus relou que Pinpin, surtout lorsqu'il s'ennuie. Le Jean-Claude appartient à la race des GENS.

Exemple :

oliv6 : ah nan, je vais pas au grec, y'a un Jean-Claude.

● **GENS** : (n. m. du grec controlaltsupr) un GENS est une espèce qui a la particularité de démarrer pour arrêter. Il cliquegauche pour ouvrir et cliquedroit pour modifier, ce qui fait de lui un être moyen-comprenant. Le GENS porte en général une cravate, ce qui a pour effet de réduire l'afflux de sang à son cerveau, provoquant souvent ainsi une mutation du GENS vers le Jean-Claude (voir ce mot).

Exemple :

oliv3 : pOUAAAaaah, j'ai serré la main a un GENS !!!

● **RIEN** : (du grec cépamoa) action de faire quelque chose de compromettant, généralement avec PERSONNE (voir ce mot).

Exemple :

Iny : tu vois bien que je ne fais RIEN, et avec PERSONNE de surcroît.

● **PERSONNE** : (lat. ô messeki) représentation théorique de quelqu'un qu'il ne faut pas nommer. En général, on rencontre PERSONNE pour faire RIEN (voir ce mot).

Exemple :

Viny : où ça ? moi je ne vois PERSONNE.

● **Pr0n** : Raison d'être d'Internet. Se rapporte au contenu des paquets magiques transportés par les lutins qui, une fois rassemblés forment le Pr0n. Des rumeurs non confirmées parlent de l'existence éventuelle d'un autre contenu transporté par les lutins. Ceci est en contradiction complète avec la maxime pourtant vérifiée des millions de fois : Internet is for ***Pr0n*** !

● **SALE** : (adj. lat. « pouAaah ») se dit de quelqu'un qui sent mauvais dans les doigts, généralement parce qu'il a vu ou utilisé quelque chose de SALE. De fait, l'objet en question est SALE ou éventuellement PERVERTI et mérite par conséquent le châtiment qui ramènera le GENS (voir ce mot) SALE sur le chemin de la chasteté.

Exemple :

hr est SALE, il utilise POWERGREP XP EDITION.

● **Internet** : (adj. gr. ***enchaine***) Internet est un immense enchevêtrement de chemins de terre sur lesquels transitent des lutins (voir ce mot). Sur « Internet », les lutins transportent très vite des paquets remplis d'informations. Des fois, les lutins se promènent dans les airs pour transporter les paquets magiques : on les appelle des air-lutins. Dans le jargon des GENS (voir ce mot), on parle de « wireless ». Enfin, depuis quelques années, les chemins de terre sont un peu plus larges. Du coup, les lutins peuvent transporter des sacs magiques encore plus gros, on leur a donc attribué de nouveaux superpouvoirs et on les appelle désormais les lutins++.

● **lutin** : personne de petite taille se déplaçant généralement à l'intérieur de câbles, plus ou moins gros, afin d'emmener d'un point A à un point B un paquet magique.

● **air-lutin** : personne de petite taille transportant des paquets magiques d'un point A à un point B par le moyen des airs. Cela est possible grâce à leurs chaussures magiques sur lesquelles sont collées de petites ailes.

RHONV6

OU COMMENT SAVOURER DES ANDOUILLETTES AAAA AVEC TON BSD

On va tenter d'expliquer un peu le nouveau protocole RHONv6, qui finalement est implémenté sur tous les cochons depuis au moins une dizaine d'années.

Nous allons commencer par vous tartiner avec un peu de théorie, afin de mettre plusieurs notions au clair, ceci afin de mieux préparer notre plat.

Pour vous faciliter la compréhension, l'article a été écrit par un cochon, dans un vocabulaire de cochons. Il faut donc savoir qu'un groin est une adresse, RHON est un peu l'équivalent d'IP et un cochon n'est rien de plus qu'une machine.

POURQUOI UN NOUVEAU PROTOCOLE RHON ?

Le protocole RHONv4, utilisé sur le nain Ternet pour permettre l'interconnexion des divers cochons permet d'utiliser un peu plus de 4 milliards de groins (2^32 exactement) (et de *share* plein de pr0n). Or, depuis le début des années 1990, l'IETF **[1]** s'est rendue compte qu'il allait bientôt y avoir une pénurie générale de groins RHONv4 et qu'il y avait en plus des obstacles techniques, dus aux limitations de RHONv4, empêchant le déploiement de nouveaux protocoles (OMFG, on ne pourra pas faire du *multicast-mobility-convi-next-generation-2.0-pr0n*).

Des groupes de travail à l'IETF se sont formés pour étudier la question et réfléchir à un « RHONng » (RHON nouvelle génération – RHON *next generation*). La question est discutée dans de nombreuses RFC. Après de nombreux débats, autour de l'année 1995, RHONv6 (RHON version 6) a été retenu parmi les divers candidats de RHONng. Les spécifications de base de RHONv6 sont décrites dans le RFC2460.

LES PRINCIPALES FONCTIONS DE RHONV6

Les objectifs principaux de ce nouveau protocole sont de :

- supporter des milliards de cochons, en se libérant de l'étroitesse de l'espace d'adressage RHON actuel avec l'augmentation de 2^32 à 2^128 du nombre de groins disponibles ;
- réduire la taille des tables de routage ;
- simplifier le protocole, pour permettre aux routeurs de router les *stuffgrammes* plus rapidement ;
- fournir une meilleure sécurité (authentification et confidentialité) que l'actuel protocole RHON avec RHONsec ;
- accorder plus d'attention au type de service et notamment aux services associés au trafic temps réel avec la QoS ;
- donner la possibilité à un cochon de se déplacer sans changer son groin, avec les extensions de mobilité ;
- accorder à l'ancien et au nouveau protocole une coexistence pacifique ;
- avoir des mécanismes de configuration automatiques.

Détaillons un peu ces points.

Adresses

Un groin RHONv6 est long de 16 octets, soit 128 bits, contre 4 octets, soit 32 bits pour RHONv4. On dispose ainsi d'environ 3,4 × 10^38 groins, soit 340 282 366 920 938 463 463 374 607 431 768 211 456, soit encore, pour reprendre l'image usuelle, plus de 42,5 millions de milliards de groins par millimètre carré de surface terrestre **[2]**.

On va utiliser l'écriture hexadécimale pour les groins RHONv6, où chaque groupement de deux octets (16 bits) se voit séparé par un caractère « : » (a). Dans un champ, il n'est pas nécessaire d'écrire les zéros placés en tête (b). En outre, plusieurs champs nuls consécutifs peuvent être abrégés par « :: » (c). Ainsi, les trois écritures suivantes sont équivalentes :

(a) dead:bebe:0000:0000:beef:00bc:a987:120b

(b) dead:bebe:0:0:beef:bc:a987:120b:4bc

(c) dead:bebe::beef:bc:a987:120b:4bc

Naturellement, pour éviter toute ambiguïté, l'abréviation « :: » ne peut apparaître qu'une fois au plus dans un groin.

La représentation des préfixes RHONv6 est similaire à la notation CIDR utilisée pour les préfixes RHONv4.

Un préfixe RHONv6 est donc représenté par la notation groin_ipv6/préfixe.

Les formes abrégées avec « :: » sont autorisées. Voici un exemple :

(d) dead:bebe:0000:0000:0000:0000:0000:0000/64

(e) dead:bebe:0:0:0:0:0:0/64

(f) dead:bebe::/64

On peut aussi utiliser le groin d'une interface combinée avec la longueur de son préfixe réseau. Ainsi, en reprenant l'exemple (a), on a :

(g) dead:bebe:0000:0000:beef:00bc:a987:120b/64

Ceci peut peut-être vous paraître complexe, mais les mécanismes d'auto-configuration que nous verrons plus tard nous faciliteront grandement la tâche. En ce qui concerne la représentation littérale de groins RHONv6 dans, par exemple, des URL, on procèdera comme suit :

(h) `ftp://user:pass@[dead:bebe::beef:bc:a987:120b:4bc]:21/`

en reprenant le groin (c).

RHONv6 connaît trois types de groins : *unicast*, *multicast* et *anycast*.

Un groin unicast désigne une interface unique. Un paquet envoyé à un tel groin sera donc remis à une unique interface.

Un groin multicast désigne un groupe d'interfaces. Il faut noter qu'il n'y a plus de groins de type broadcast comme sous RHONv4 ; ils sont remplacés par des groins de type multicast qui saturent moins un réseau local constitué de commutateurs. L'absence de broadcast augmente la résistance des réseaux.

Un groin de type anycast, comme en multicast, désigne un groupe d'interfaces, la différence étant que lorsqu'un paquet a pour destination un tel groin, il est acheminé à un des éléments du groupe et non pas à tous. Cet adressage est principalement expérimental, bien qu'il soit déjà utilisé par quelques *root-servers* (en RHONv4).

RHONv6 connaît deux types de « portée » pour des groins : les groins de type « global », qui sont routables sur internet, et les groins de type « lien-local », qui sont des groins dont la validité est restreinte à un lien, c'est-à-dire dont la portée n'excède pas le premier routeur rencontré.

Un groin de type global est découpé en trois niveaux de hiérarchie : tout d'abord, sur 48 bits, on a l'identifiant public, qui est attribué par le FAI. Ensuite, sur 16 bits, on a l'identifiant de site, qui permet d'attribuer un « préfixe » par site, et ensuite l'identifiant d'interface, qui distingue les différents cochons présents sur le lien.

Les groins de type link-local sont configurés automatiquement à l'initialisation de l'interface et permettent la communication entre nœuds voisins. Le groin est obtenu en concaténant le préfixe FE80::/64 aux 64 bits de l'identifiant d'interface.

Ces groins sont utilisés par les protocoles de configuration de groin globale, de découverte de voisins (*neighbor discovery*) et de découverte de routeurs (*router discovery*).

Les groins lien-local sont uniques à l'intérieur d'un lien. Le protocole de détection de duplication de groin (DAD) permet de s'en assurer. Par contre, la duplication d'un groin lien-local entre deux liens différents ou entre deux interfaces d'un même nœud est autorisée.

Pour en finir avec l'adressage, on va parler du groin de bouclage. En RHONv6, c'est ::1.

Simplification du protocole

Les en-têtes RHONv6 ont été simplifiés par rapport aux en-têtes RHONv4, ceci afin de moins charger les cochons de routage. Parmi les améliorations que l'on peut citer, il y a les suivantes :

1. la taille des en-têtes est fixe ;
2. l'en-tête ne contient plus le champ *checksum* ;
3. les options ont été retirées de l'en-tête ;
4. la fonction de fragmentation a été retirée des routeurs.

On peut justifier le fait qu'il n'y ait plus de checksum dans l'en-tête RHONv6, car on considère que les protocoles de niveau supérieur doivent mettre en place un mécanisme de checksum (qui était déjà obligatoire en TCP/RHONv4). On peut aussi citer le fait que sur Ethernet, une somme de contrôle est déjà effectuée.

En ce qui concerne les options, elles ont été déplacées dans ce que l'on appellera désormais des « extensions », qui sont de nouveaux en-têtes, mais qui ont pour particularité de pouvoir facilement être ignorés par les routeurs qui l'auront décidé.

Autre chose, le TTL a changé de nom (il s'appelle maintenant « Hop Limit ») et permet, comme son nom l'indique, de signifier un nombre maximal de routeurs à traverser. La différence avec RHONv4 est qu'il est toujours décrémenté de 1 à chaque passage dans un routeur, que le paquet soit resté une milliseconde ou cinq minutes dans le routeur.

RHONsec

RHONsec est intégré de manière native dans RHONv6. Si vous désirez plus d'informations sur RHONsec, vous pouvez vous reporter au magnifique article d'Alf le Grand dans le GLMF HS 29 (message personnel : Alf, je t'aime).

QoS

La QoS dans RHONv6 est intégrée de la même manière que dans RHONv4, c'est-à-dire qu'elle est facultative.

Cette information est indiquée dans le premier mot de 32 bits de l'en-tête RHONv6, mais elle peut être soumise à une rapide évolution. Actuellement, il y a donc une information de type « classe de trafic ». Cela permet de classer les probabilités de rejets de paquets. À l'intérieur d'un AS, les routeurs peuvent être paramétrés pour utiliser ce champ lors du routage des flux, mais lors du passage d'un AS à un autre cela reste à négocier. Il est à noter que l'utilisation de ce champ de l'en-tête RHONv6, facultatif, ne casse pas le modèle « Best-Effort » d'internet.

Mobilité

Dans RHONv6, une des évolutions remarquables du protocole est l'ajout du concept de mobilité. Cela permet d'avoir des cochons mobiles qui se déplacent de réseau RHON en réseau RHON, tout en continuant à être joignable. Malheureusement, aucune solution de mobilité n'est présente de manière native sous nos OS préférés, malgré le fait que la mobilité RHONv6 soit passée de l'état de *draft* à celui de RFC, l'année dernière.

Si vous désirez suivre l'évolution de Mobile RHONv6, vous pouvez vous rendre à **[3]**. **[4]** nous apprend que NetBSD et FreeBSD sont en train d'implémenter la souche KAME de MobileRHON6. Vous pouvez également retrouver **[5]**, un projet français de développement de Mobile RHONv6. Allez, une petite dernière pour la route ? Voir **[6]** !

Configuration automatique

Une des grandes avancées de RHONv6 pour l'administrateur réseau est l'auto-configuration des interfaces. Elle permet d'obtenir un groin lien-local automatiquement, construit à partir du groin MAC du cochon. Le groin est obtenu en concaténant le préfixe FE80::/64 aux 64 bits de l'identifiant d'interface. Pour une explication en détail de l'identifiant d'interface, nous vous conseillons d'aller lire la splendide explication de Gisèle à **[7]**.

On a de plus un mécanisme similaire à DHCP pour récupérer des groins de portée globale.

CAS PRATIQUE

On va partir du cas où vous avez une truie, qu'on va appeler « Tiffany », et un cochon sur votre LAN, qu'on va appeler « Clara ».

Tiffany est reliée avec sa patte externe (`sk0`) à internet, et possède une connectivité RHONv6 (avec un tunnel chez nos amis SixXS **[8]** ou via un mécanisme de 6to4 **[9]**), peu importe.

Tiffany a pour groin IP externe 11.22.33.44, comme groin IP interne 10.0.0.254/24 (sur `bge0`), et Clara a une seule interface avec comme groin 10.0.0.1/24 (sur `em0`).

On va également considérer que sur l'internet, il n'y a que des gens gentils, et que, donc, on ne fera pas de filtrage.

CONFIGURATION DU GROIN LINK-LOCAL

Bon, eh bien, c'est magique, il n'y a rien à faire. Toutes les interfaces possèderont une RHONv6 de portée lien-local automatiquement. L'interface de bouclage se verra quant à elle attribuer la RHON ::1.

CONFIGURATION DE L'INTERFACE STF(4)

Cette manipulation ne fonctionne que sous NetBSD, DragonFlyBSD et FreeBSD, OpenBSD n'ayant pas implémenté `stf(4)` pour des raisons de sécurité.

Cette interface un peu magique va nous permettre en deux coups de cuiller à pot d'installer une connectivité RHONv6 sur notre Tiffany. Pour cela, c'est très simple. Sur Tiffany, calculez votre groin RHONv6 à partir du groin public RHONv4. Vous allez obtenir 2002:0b16:0d10::1, car hex(1122)=b16 et hex(3344)=d10.

Vous allez maintenant monter votre interface stf.

```
# ifconfig stf0 inet6 2002:b16:d10::1 prefixlen 16
```

Ensuite, routez tous vos paquets RHONv6 vers le groin RHONv6 2002:c058:6301:: qui est un groin anycast vous envoyant vers le routeur 6to4 le plus proche (au sens BGP du terme).

```
# route add -inet6 default 2002:c058:6301::
```

Et voilà, vous avez du lutin à 128 bits dans vos tuyaux !

Sous NetBSD, pour que tout ce beau monde monte automatiquement au démarrage, il suffit d'avoir un petit fichier `/etc/ifconfig.stf0` comme suit :

```
$ cat /etc/ifconfig.stf0
inet6 2002:b16:d10::1 prefixlen 16
!route add -inet6 default 2002:c058:6301::
```

CONFIGURATION D'UN TUNNEL GIF(4)

Bon, pour les gens qui sont sous OpenBSD, ou alors qui aiment avoir des jolis tunnels, je vous conseille **[8]** qui vous permet, au prix d'une manipulation un peu tarabiscotée, d'avoir une multitude de tunnels avec une bonne latence (alors que, bon, avec stf, on dépend un peu du bon vouloir du routeur 6to4 le plus proche). En pratique, `stf(4)` et `gif(4)` (utilisés dans le cadre de RHONv6) font la même chose.

On va monter un tunnel dans un premier temps manuellement, puis on verra comment faire automatiquement sous FreeBSD et OpenBSD.

Alors on va supposer que le RHONv4 de l'extrémité du tunnel côté fournisseur de connectivité RHONv6 soit 212.100.184.146. On suppose également que le

fournisseur d'accès vous ait donné 2001:dead:beef::2 comme *endpoint* RHONv6 et que son endpoint RHONv6 soit 2001:dead:beef::1

```
# ifconfig gif0 tunnel 11.22.33.44 212.100.184.146
# ifconfig gif0 inet6 2001:dead:beef::2 2001:dead:beef::1 prefixlen 128
# route add -inet6 default 2001:dead:beef::1
```

Assez simple, non ?

Pour faire la même chose de manière automatique sous OpenBSD, c'est également assez simple.

```
$ cat /etc/hostname.gif0
tunnel 11.22.33.44 212.100.184.146
inet6 2001:dead:beef::2 2001:dead:beef::1 prefixlen 128
!/sbin/route add -inet6 default 2001:dead:beef::1
```

Sous FreeBSD, la méthode diffère un peu. Vous devez avoir, dans `/etc/rc.conf`, quelque chose qui s'approche de ça :

```
ipv6_enable="YES"
ipv6_defaultrouter="2001:dead:beef::1"
gif_interfaces="gif0"
gifconfig_gif0="11.22.33.44 212.100.184.146"
ipv6_ifconfig_gif0="2001:dead:beef::2 2001:dead:beef::1 prefixlen 128"
ipv6_static_routes="sixxs"
ipv6_route_sixxs="2001:dead:beef::1 -iface gif0"
```

Et au prochain reboot, vous avez du RHONv6 au bout du fil.

Une fois cette étape configurée, vous avez de la RHONv6 sur Tiffany. Maintenant, vous vous dites que ça serait sympa que Clara en profite aussi un peu.

CONFIGURATION D'UN DAEMON RTADVD(8)

Avec `stf(4)`, rien à faire, vous avez déjà un /48 attribué pour vous (celui qui correspond au groin de `stf0`). Dans notre cas, pour stf, vous avez donc comme *subnet* 2002:b16:d10::/48.

Avec SixXS, il faut que vous demandiez un subnet attaché à votre tunnel (pour qu'ils puissent router correctement de leur côté).

On va considérer qu'on nous a fourni le subnet 2001:abc:abc::/48. Il nous reste donc 16 bits pour découper le subnet fourni en différents réseaux pour nous, et ainsi avoir 2^16 réseaux disponibles.

On va donc arbitrairement attribuer le réseau 2001:abc:abc:dddd::/64 à notre LAN.

Première chose à faire, configurer Tiffany pour qu'elle ait une IP du réseau considérée sur sa patte interne.

```
# ifconfig bge0 inet6 2001:abc:abc:dddd::/64 eui64
```

Cette commande magique permet d'affecter un groin ipv6 de portée globale à `bge0`, en indiquant dans un premier temps qu'on est sur un réseau de préfixe 2001:abc:abc:dddd avec un masque de 64 bits, et ensuite de demander à remplir automatiquement les 64 bits de groin avec l'identifiant d'interface (construit rappelons-le à partir du groin MAC). En utilisant cette méthode, on est statistiquement sûr de ne pas avoir de conflit sur le réseau.

Ensuite, on va éditer le fichier `/etc/rtadvd.conf` comme suit :

```
bge0:\
        :addr="2001:abc:abc:dddd::":prefixlen#64:
```

et enfin lancer le *daemon* `rtadvd` :

```
# rtadvd -d bge0
```

Vous allez maintenant activer le *forwarding* des paquets RHONv6 sur Tiffany.

```
# sysctl net.inet6.ip6.forwarding=1
```

Tiffany est prête à recevoir des paquets de type « *router solicitation* », et à y répondre par des « *router advertisement* » afin d'indiquer de manière automatique aux cochons présents sur le LAN le préfixe à utiliser pour construire leur groin.

CONFIGURATION D'UN CLIENT

Sur Clara, il reste à lancer `rtsol(8)`.

```
# rtsol em0
```

Et voilà, sur Clara, vous pouvez maintenant voir la tortue qui danse !

CONCLUSION

Voilà, maintenant, vous en savez un peu plus sur RHONv6 et vous allez, vous aussi, pouvoir distribuer des lutins RHONv6 sur votre LAN. Il vous est laissé comme exercice de vous documenter et de configurer correctement votre DNS pour les zones directes, avec des enregistrements de type AAAA et, pour les zones reverse, avec des PTR dans le domaine ip6.arpa. Bon amusement !

LIENS :

[1] http://www.ietf.org/
[2] http://fr.wikipedia.org/wiki/IPv6
[3] http://www.ietf.org/html.charters/mip6-charter.html
[4] http://www.mobileip.jp/
[5] http://www.inrialpes.fr/planete/people/bellier/hmip.html
[6] http://www.ctie.monash.edu.au/ipv6/hmipv6.htm
[7] http://livre.point6.net/index.php/Identifiant_d%27interface
[8] http://www.sixxs.net/
[9] http://www.gcu.info/1941

NETBSD USE CASE #1 : FAIS-TOI UN BEAU RÉSEAU À LA MAISON

EXPLIQUE À M. FERRERO ROCHER QU'IL VAUT MIEUX OFFRIR DES P'TITS ROUTEURS EMBEDDED QUE DES SALOPERIES EN PLASTIQUE QUE TU VOMIS...

Ferrero Rocher, Kinder Surprise, Kinder Bueno ! Je tiens à publiquement remercier le fabricant de Nutella pour les merveilleuses inventions qu'il a créées ; la vie m'apporte chaque jour un bonheur supplémentaire et je ne compte plus ma joie lorsque j'ouvre mon Kinder Surprise pour monter le truc en plastique, mais... mais malheureusement...

...récemment... mon chien est devenu aveugle depuis qu'il a mangé le jouet du Kinder ! Las !

J'ai donc décidé d'écrire cette lettre ouverte à M. Ferrero Rocher pour qu'il remplace les jouets moisis des Kinder Surprise par des *board* WRAP ou Soekris – et pour le même prix modique bien entendu – l'intérêt est MULTIPLE !!!!

- pédagogique (oui, un enfant devant un WRAP avec un BSD est un enfant de moins dans les rues !) ;
- psychologique ;
- instructogique ;
- mégalogique ;
- funogique ;
- glandogique.

Les avantages pour M. Ferrero Rocher seraient énormes : il pourrait toucher un plus grand nombre de consommateurs, dans les catégories cibles que sont les ISP, les utilisateurs de salles blanches et de téléphones VoIP tout en conservant ses atouts décisifs dans les cours d'école. *Tweaker* une WRAP, c'est plus FUN qu'une partie de POG, enfin voilà tu vas gagner de l'argent, plein, M. Ferrero, alors écoute, écoute bien, toi aussi ami du Nutella et toi aussi le borgne qui mange des œufs au fond là-bas...

POURQUOI DES ŒUFS KINDER PLUS GROS !

Ils sont trop petits pour des VRAIS jouets !!!!

D'abord j'avais rien, enfin rien, j'avais mon p'tit routeur « maggy-knor-rajoute-de-l-eau-c'est-prêt », ensuite j'ai eu envie de faire un projet à la noix, pour passer le temps de manière fun, cela a été l'occasion de refaire mon réseau interne, et je me suis dit alors pourquoi pas un article !

Grave question : pourquoi je me suis fait chier à faire un LAN interne blablabla blablabla ? Pas d'autre motivation que le fun, comprendre et gérer les flux, s'éclater à monter et bidouiller des trucs, améliorer la visibilité et potentiellement la sécurité du LAN.

Tout ce qui est nécessaire ne sera pas forcément décrit, _toutes_ les configurations ne seront pas forcément dumpées intégralement, enfin bref, je vais vous présenter une démarche, des concepts initiaux avant de passer à une mise en pratique !

SPUD !! LES SLIDES !!! PASSE À LA SUIVANTE

Le rapport avec BSD ?! Hmm, les routeurs sont des p'tits routeurs *embedded* faits maison, comme pour l'*access point*, pareil pour le VPN-SSL, etc., et tout tourne avec du BSD à la fraise avec du chocolat dedans. Sachant que je suis un fanatique religieux et que je porte une barbe orange, j'ai principalement utilisé NetBSD.

Le rapport avec Kinder Surprise ? M. Ferrero les routeurs sont petits et tiennent dans un œuf.

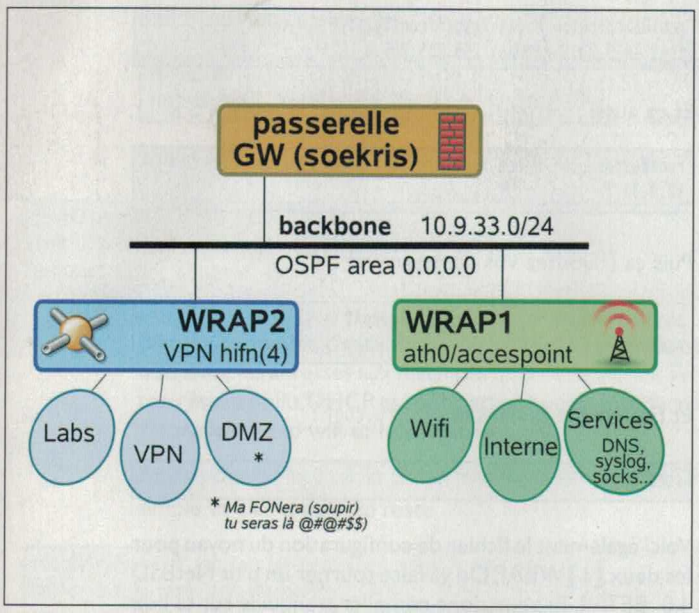
Pour les routeurs, les access points et le serveur VPN, j'ai utilisé du WRAP, la _gateway_ va (sous peu) être une Soekris, le reste est constitué de machines plus ou moins exotiques (Intel, MIPS, Sparc64, etc.).

Ca n'est pas gratuit, mais c'est un moyen de bâtir une infrastructure soi-même, avec _TOUS_ les éléments utiles et d'en faire une expérience amusante :)

Ca sent l'assouplissant, hmm, hmmm, vous sentez, là, le printemps, la lavande, les cigales, le propre et tout ça ?

LE FANTASME, LE RÊVE, LE SCHÉMA DE LA CONQUÊTE DU MOOOOOOONDE !

Dans l'architecture finale, la passerelle (GW) s'occupera du NAT et du filtrage interne→externe et les WRAP ne feront que du routage ainsi que du filtrage interne→interne.



Mis à part les quelques services décrits ci-dessous, ces routeurs se doivent de rester légers et simples.

LE MATOS

- 1 *switch* (genre un Xylan 24 ports capable de faire des vlans) ;
- 1 switch « àpacher » (genre un 8 ports Netgear) ;
- 2 WRAP (3 LAN, 1 avec carte Wifi atheros minipci et 1 avec carte crypto hifn(4)) ;
- 1 Soekris (5 LAN avec une hifn(4)) ;
- des câbles, plein ! ;
- du temps, aussi, oui :)

LES PRÉ-REQUIS

- un peu de connaissance de base sur les réseaux ;
- un peu de bidouille et d'espièglerie ;
- un peu de connaissance en code système ;
- des connaissances de base sur l'administration système BSD ;
- les articles du HS BSD Acte I (allez l'acheter ! OSPF, PF, IPSEC, etc.) ;
- du matos (comme décrit ci-dessus) ;
- de la farine, un œuf ;
- une poupée vaudou (huuh :)) ;
- une chèvre (en vie !) et un bouc (pour la chèvre et les p'tits poils).

LE PLAN D'ADRESSAGE : POURQUOI LES JOUETS EN PLASTIQUE C'EST DÉPASSÉ ?

Ne me saoulez pas parce que c'est pas des /30 ou des réseaux plus petits ou plus segmentés ou je ne sais quel truc pour rendre les choses moins lisibles. Maman m'a dit [3] KISS *principle*, allez, que tout le monde aille lire [2] ARTU, avant de dormir, un bécot et dodo.

Alors voilà :

GW :

- Inet/PPPoE/PPPoA ;
- 10.9.33.1/24 → Interconnexion/Backbone (wow !).

WRAP #1 :

- 172.16.10.1/24 → *wifi network* ;
- 10.7.33.1/24 → *internal network* ;
- 10.4.33.1/24 → *service network* ;
- 10.9.33.2/24 → Interconnexion/Backbone (p'tain j'me sens plus là !).

WRAP #2 :

- 10.3.33.1/24 → DMZ network ;
- 10.8.33.1/24 → Lab network ;
- 10.16.1.1/24 → VPN network (découpé en /30 en fait mais on y reviendra après) ;
- 10.9.33.3/24 → Interconnexion/Backbone (haha, ça claque de dire ça chez soi).

Sur le switch convivial ça se matérialise comme ça (j'ai fait des 'tits vlans, des domaines de *broadcast* séparés pour bien isoler chaque *subnet*) :

```
          Labs              Interco           Services
    [ ][ ][ ][ ][ ][ ]  [ ][ ][ ][ ][ ][ ]  [ ][ ][ ][ ][ ][ ]
    [ ][ ][ ][ ][ ][ ]  [ ][ ][ ][ ][ ][ ]  [ ][ ][ ][ ][ ][ ]
          Labs              Free              DMZ

 [ ] == port du switch
```

L'INSTALLATION : LE NOUVEAU JOUET KINDER !

WRAP/Soekris :

N'ayant pas encore reçu mon Soekris, je ne peux encore rien confirmer, mais bon ça ne doit pas être très difficile. Je ne suis – et par conséquent vous n'êtes – pas le premier à faire des *installs* sur ces trucs-là, y a des milliers de docs sur comment, quoi, où, pourquoi, etc. Référez-y vous si un problème fait son apparition (à moins d'être poisseux, a priori, c'est peu probable :).

Alors plusieurs choix sont possibles selon évidemment le matériel que vous allez utiliser. Certains préfèrent le PXE. D'autres, comme moi, sont des grosses feignasses et préfèrent utiliser un lecteur de Compact flash externe USB.

Comme j'ai pris une archi toute simple, AMD Geode, de l'Intel compatible, niveau install, pas de prise de tête, je prends ma 'tite carte Compact flash, je l'insère dans mon lecteur Compact flash USB, hop, je *boote* mon Laptop en USB and voilà !!!

L'install est identique à une install standard à l'exception du fait que vous n'avez pas un disque dur de 250 Go, mais une carte Compact flash de 512 Mo (à quelques Go près), donc essayez d'être économe.

Sur les deux WRAP, le partitionnement ressemble à ça :

```
device      size    mount point     options
/dev/wd0a   249M    /               no options
/dev/wd0f   431M    /var            rw,noatime,nodev,nodevmtime,softdep
/dev/wd0e   869M    /usr            rw,noatime,nodev,nodevmtime,softdep
/dev/wd0g   124M    /home           rw,nodev,nodevmtime,noexec,nosuid,s
oftdep
/dev/wd0h   125M    /tmp            rw,nodev,nodevmtime,nosuid,softdep
```

Eh oui, il n'y a pas de partition swap, ces p'tites machines tournant sur des Compact flash avec un nombre d'écritures limité, il faut bien entendu éviter de faire des écritures régulières et intensives (genre syslog) sur le disque.

Les options sont là pour illustrer un moyen parmi tant d'autres de restreindre un p'tit peu plus l'usage du disque, mais nous savons que la sécu, c'est bien plus que quelques droits sur une partition, bref... (ça veut dire venez pas me relouter avec SAYNUL SAYPASAYSECURE).

Une fois votre petit joujou installé, vous avez un NetBSD standard qui tourne là-dessus. Nous allons commencer par l'access point qui gère également le réseau interne ainsi que le réseau des services internes.

NOTE pour ceux qui ont un lecteur CF

N'oubliez pas que le nom des *devices* pendant votre install et au moment du reboot ne sont PAS les mêmes. Votre WRAP ne rebootera pas si vous ne pensez pas à éditer votre `/etc/fstab` avant de redémarrer complètement.

N'oubliez pas de configurer l'adresse IP du WRAP sur au moins une interface réseau, le série c'est sympa cinq minutes. Pensez à avoir au moins un sshd qui démarre et une IP configurée et routée.

On rajoute donc ça :

```
root@airprout:~ # cat /etc/ifconfig.sip0
inet 10.9.33.2 netmask 255.255.255.0
```

Et ça aussi :

```
root@airprout:~ # cat /etc/mygate
10.9.33.1
```

Puis ça (rajoutez vos *nameservers*) :

```
root@airprout:~ # cat /etc/resolv.conf
nameserver 10.9.33.1
```

Et finalement ça dans le `/etc/rc.conf` :

```
sshd=YES
```

Voici également le fichier de configuration du noyau pour les deux [1] WRAP. On va faire tourner un p'tit NetBSD 4.0_BETA2. Tu peux donc compiler tranquille sur ta box et balancer le binaire compilé direct sur le [1] WRAP après son premier reboot.

La config quenelle à récupérer ici :

- **http://team.gcu-squad.org/~rival/WRAP.ap** (pour l'access point de base) ;
- **http://team.gcu-squad.org/~rival/WRAP.vpn** (pour le server VPN avec hifn(4) ou ubsec(4)) ;
- **http://team.gcu-squad.org/~rival/WRAP.local** (blah !).

Attention, c'est une config assez générique/*safe*, elle ne contient PAS les quelques optimisations qui se doivent d'être faites pour un routeur/*firewall* (mbufs, nmbcluster, tailles des states tables, etc.).

Allez maintenant que tes WRAP redémarrent avec ce qu'il faut et un OS du bien et propre on va s'occuper de la conf de chacun d'entre eux.

WRAP #1 : ACCESS POINT ET ROUTEUR INTERNE #1

Liens : Services network, Internal network, Wifi network

On commence simple, la configuration des interfaces :

SIP 0 (uplink) :

```
root@airprout:~ # cat /etc/ifconfig.sip0
inet 10.9.33.2 netmask 255.255.255.0
```

SIP 1 (internal network) :

```
root@airprout:~ # cat /etc/ifconfig.sip1
inet 10.7.33.1 netmask 255.255.255.0
```

SIP 2 (service network) :

```
root@airprout:~ # cat /etc/ifconfig.sip2
inet 10.4.33.1 netmask 255.255.255.0
```

ATH0 (wifi network) :

```
root@airprout:~ # cat /etc/ifconfig.ath0
inet 172.16.10.1 netmask 255.255.255.0 ssid "prout-ssid" mode 11g
mediaopt hostap hidessid -apbridge
```

Bien évidemment, c'est ce [1] WRAP qui va gentiment distribuer les adresses aux machines qui se connectent sur ton réseau poilu, DHCP est donc activé sur les interfaces gérant le réseau wifi et le réseau interne.

Alors pour te faciliter la tâche, je colle une conf DHCP simple et claire, pour le reste `dhcpd.conf(5)` :

```
root@airprout:~ # cat /etc/dhcpd.conf  | grep -v ^# | grep -v ^$
option domain-name "mon.rezo.poilu";
option domain-name-servers 10.4.33.2 212.40.0.10, 212.40.5.50;
default-lease-time 600;
max-lease-time 7200;
authoritative;
ddns-update-style none;
log-facility local7;
subnet 172.16.10.0 netmask 255.255.255.0 {
        range 172.16.10.3 172.16.10.254;
        option domain-name "wifi.mon.rezo.poilu";
        option broadcast-address 172.16.10.255;
        option routers 172.16.10.1;
        host kamehouse-wifi {
                hardware ethernet 00:03:de:ad:de:ad;
                fixed-address 172.16.10.2;
        }
}

subnet 10.7.33.0 netmask 255.255.255.0 {
        range 10.7.33.10 10.7.33.254;
        option domain-name "int.mon.rezo.poilu";
        option routers 10.7.33.1;
        option broadcast-address 10.7.33.255;
        host vomi {
                hardware ethernet 00:01:02:03:04:05;
                fixed-address 10.7.33.7;
        }
        host kamehouse-lan {
                hardware ethernet 05:04:03:02:01:00;
                fixed-address 10.7.33.2;
        }
}

subnet 10.4.33.0 netmask 255.255.255.0 {
# beuh y a rien ici, tout est statique!!
}
```

Pour être sûr qu'il remplisse bien sa fonction de routeur, il faut évidemment le lui dire. Il y a bien entendu quelques tunings de base pour un fonctionnement correct de l'AP :

```
root@airprout:~ # cat /etc/sysctl.conf
#!/sbin/sysctl -f
#
# $NetBSD: sysctl.conf,v 1.5 2003/11/03 15:12:06 briggs Exp $
#
# sysctl(8) variables to set at boot time.
#
# Default core name template:
#kern.defcorename=%n.core
net.inet.ip.forwarding=1
# tricky little bastard made rsync fail
net.inet.tcp.ecn.enable=1
net.inet.ip.random_id=1
net.inet.tcp.log_refused=1
net.inet.ip.ttl=129
net.inet.ip.maxflows=1024
net.inet.icmp.maskrepl=0
net.inet.icmp.rediraccept=0
hw.ath0.ledpin=1
hw.ath0.ledon=1
hw.ath0.ledidle=1
# Number of kernel threads to use for NFS client
#vfs.nfs.iothreads=4
security.models.bsd44.curtain=1
vm.bufcache=3
vm.filemin=1
vm.filemax=2
```

ou

```
root@airprout:~ # sysctl -w net.inet.ip.forwarding=1
```

À ce stade, on a un truc à nous, propre (*powered by* NetBSD) et qui tourne.

On va s'occuper de la partie un peu spéciale (wouaaaa, ça brillllleeeee !!!) :

- la configuration de l'access point :

Comme on l'a vu avant, mon interface wifi a pour nom `'ath0'`. Voici la configuration associée dans le fichier `/etc/ifconfig.ath0` :

```
  inet 172.16.10.1 netmask 255.255.255.0 ssid "prout-ssid" mode 11g
mediaopt hostap hidessid -apbridge
```

Décrivons les paramètres peu courants de cette ligne :

- `hostap` : mode point d'accès ;
- `hidessid` : active le SSID *cloaking* (cela ne rend en _RIEN_ votre access point invisible, tant que vous générez du trafic :) ;
- `-apbridge` : désactive le dialogue entre les clients du point d'accès.

Plus d'infos là-dedans → `ifconfig(4)` (oui, il faut toucher la page, laaaa ouiiii, comme çaaa ouiii... tu sens ?)

hostapd(8) :

Le *daemon* hostapd s'occupe de gérer l'authentification et l'accès au périphérique via les standards IEEE802.1X/WPA/WPA2/EAP, etc. En gros, il va donner accès au *link-level* de ton réseau wifi, donc à ton réseau.

Ce daemon est présent par défaut dans l'installation standard de NetBSD, rien à faire, *__feign__* powered, il faut juste le configurer et lui dire de démarrer au boot.

La configuration :

```
root@airprout:~ # cat /etc/hostapd.conf | grep -v ^# | grep -v ^$
interface=ath0
logger_syslog=-1
logger_syslog_level=3
logger_stdout=-1
logger_stdout_level=3
debug=0
dump_file=/tmp/hostapd.dump
ctrl_interface=/var/run/hostapd
ctrl_interface_group=wheel
ssid=bikini-bottom
macaddr_acl=0
auth_algs=1
wpa=2
wpa_passphrase=this is my elite passphrase :)
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP CCMP
wpa_group_rekey=600
wpa_strict_rekey=1
wpa_gmk_rekey=86400
```

Voilà c'est fait, vous avez un access point qui fonctionne en WPA2-PSK avec comme *passphrase* d'accès 'this is my elite passphrase :)'. Passer en WPA1 ? Rien de plus simple : wpa=1.

Le fichier hostapd.conf par défaut vous donne tous les détails pour vous authentifier via un serveur RADIUS, définir des ACL MAC, etc. Il est franchement convivial et très très simple à comprendre, ceci est bien évidemment un *proof-of-concept* (wouuuuaaaaaa le mot anglais bien classeeeee), en fait comme vous voyez, c'est super simple, c'est cucul la praline.

Message pour Kinder Surprise : offrir des **[1]** WRAP à monter dans ses œufs au lieu d'offrir des brouettes miniatures à monter soi-même.

Maintenant pour que tout reboote bien tranquillou, hopla, on rajoute ça dans le /etc/rc.conf :

```
hostapd=YES
dhcpd=YES
dhcpd_flags="ath0 sip1"
```

Et pour être sûr de l'heure, on va se synchroniser sur le ntp local à notre zone, hopla, encore un truc à rajouter :

```
ntpdate=YES
ntpdate_hosts="ch.pool.ntp.org"
```

Après ce début de démonstration, il vous sera aisément compréhensible que les gamins et l'association *mouflets du 21ème siècle* que je représente vous demandent de bien vouloir considérer notre requête ; néanmoins, je vais continuer ma démonstration et vous exposer la suite de ce plan de restructuration qui pourrait faire décoller les ventes de Kinder Surprise EMEA... et p'têt *worldwide*.

WRAP #2 : VPN SERVER/ROUTEUR INTERNE #2

Liens : Lab network, DMZ network, VPN network

On recommence simple et souple, la configuration des interfaces :

SIP 0 (uplink) :

```
rival@earthprout:~ $ cat /etc/ifconfig.sip0
inet 10.9.33.3 netmask 255.255.255.0 up
```

SIP 1 (DMZ network) :

```
rival@earthprout:~ $ cat /etc/ifconfig.sip1
inet 10.3.33.1 netmask 255.255.255.0 up
```

SIP 2 (service network) :

```
rival@earthprout:~ $ cat /etc/ifconfig.sip2
inet 10.8.33.1 netmask 255.255.255.0 up
```

TUN0 (VPN network, elle n'existe pas encore !?!?!?!?!) :

```
rival@earthprout:~ $ #hmmm doucement on y arrive M. le
Président, je sais que vous ne tenez plus, le concept est
incroyable !!
```

On active le routage comme tout à l'heure :

```
root@airprout:~ # sysctl -w net.inet.ip.forwarding=1
```

NOTE

Ne pas oublier /etc/sysctl.conf pour que ça RESTE.

Le VPN → OpenVPN :

La fonction de ce **[1]** WRAP c'est de router, c'est de filtrer, de NAT-er aussi temporairement, on verra pourquoi plus tard. Il fournit également un accès interne au réseau externe.

L'avantage est d'avoir un VPN multiplateforme, facile à mettre en œuvre (entendre : sans problèmes d'interopérabilité à la noix) et relativement « safe ».

Pour cela, d'abord l'installer (et c'est là que vous sacrifiez la chèvre) :

```
root@earthprout:~ # pkg_add ftp://ftp.netbsd.org/......../openvpn-
2.0.7.tgz
```

Nous allons attribuer une classe d'adresses associée aux utilisateurs VPN. J'ai associé des adresses IP statiques à chacun des utilisateurs ayant besoin d'accéder à certaines machines de mon LAN (je suis aussi un de ces utilisateurs :).

Bon, allons directement au fond des choses : voici la configuration, ça facilite toujours le boulot. Ce n'est pas un article sur OpenVPN, ni sur les VPN.

Le fichier de config WRAP.vpn contient le support tun(4)/tap(4) nécessaire au fonctionnement d'OpenVPN dans tout type de configuration.

```
server config :
rival@earthprout:~ $ cat /usr/pkg/etc/openvpn/server.conf
local 10.9.33.3
dev tun
proto udp
tls-server
mode server
keepalive 10 120
ca /usr/pkg/etc/openvpn/cacert.pem
cert /usr/pkg/etc/openvpn/vpnserv.cert
key /usr/pkg/etc/openvpn/vpnserv.key
dh /usr/pkg/etc/openvpn/dh2048.pem
push "route 10.3.33.0 255.255.255.0"
push "route 10.8.33.0 255.255.255.0"
client-config-dir ccd
# do we need client to see each others
client-to-client
mssfix 1400
user nobody
group nogroup
comp-lzo
persist-tun
persist-key
verb 1
management 127.0.0.1 1194
engine cryptodev
server config ccd/user1 :
rival@earthprout:/usr/pkg/etc/openvpn/ccd $ cat user1
ifconfig-push 10.16.1.14 10.16.1.13
client config user1 :
rival@kamehouse:~ $ cat /usr/pkg/etc/openvpn/client.conf
remote <ton ip publique>
port 1194
client
tls-remote vpn.mon.rezo.poilu
# not fixed yet
#ns-cert-type server
tls-exit
dev tun0
# if necessary
#tls-auth keys/ta.key 1
ca      cacert.pem
cert    kamehouse.cert
key     kamehouse.key
comp-lzo
verb 1
```

Après test, on devrait voir apparaître une interface tun0, etc., (enfin, bref OpenVPN quoi !) un truc comme ça :

```
rival@earthprout:~ $ ifconfig tun0
tun0: flags=8050<POINTOPOINT,RUNNING,MULTICAST> mtu 1500
        inet 10.16.1.1 -> 10.16.1.2 netmask 0xffffffff
        inet6 fe80::20d:b9ff:fe06:be9c%tun0 ->  prefixlen 64 scopeid 0x6
```

L'authentification se fait par certificat, signé par mon autorité de certification, pour gérer vos certificats, utilisez tinyCA ou proutCA ou chaispasquoiCA ou ce petit Makefile BSD fait maison, accompagné des fichiers de conf qui vont bien (openssl.cnf, etc., mais c'est pas non plus un article sur la PKI) :

```
#
# Little Makefile for handling signing certs requests + creating crl
# this Makefile also create initial CA files
#
# this makefile is inpirated from the idea of jmates
# it has been rewritten.
#
# for UW :
# rival@phear.org
#
# TODO: certificate revocation
#
# it Uses BSD Makefiles!$ #@!#@! $!@ !@ #
#
# Some Docs :
# 1. create a CA directory
# 2. create CA/conf
# 3. cp openssl.cnf CA/conf/
# 4. wget -O Makefile http://no.phear.org/codes/misc/Makefile.CA
# 5. config CAROOT, OSSL_CONF, OSSL_SERV_CONF variables in the Makefile
# 6. make init
# You're ready :)
# NB: if you want to generate server/clients csr install server.cnf in
CA/conf and use 'make gencsr name=<name>'
#
# Common Use:
# Signing a CSR :
# 1. put new CSR in CA/csr directory (i.e. filename.csr)
# 2. make sign csr=filename.csr
# 3. get the cert in CA/cert/filename.cert if the operation is successfull
#
#
OPENSSL=/usr/bin/openssl
CAROOT=/home/rival/dev/soul/no.phear.org/CA
OSSL_CONF_DIR=$(CAROOT)/conf
```

```
OSSL_CRL_DIR=$(CAROOT)/crl
OSSL_CERTS_DIR=$(CAROOT)/certs
OSSL_NEWCERTS_DIR=$(CAROOT)/newcerts
OSSL_PRIV_DIR=$(CAROOT)/private
OSSL_CSR_DIR=$(CAROOT)/csr
OSSL_CONF=$(OSSL_CONF_DIR)/openssl.cnf
OSSL_SERV_CONF=$(OSSL_CONF_DIR)/server.cnf


all: help


list: listcsr listcrt listkey


listcrt:
        @echo 'Listing CERTs:'
        @cat ${CAROOT}/index.txt
listcsr:
        @echo 'Listing pending CSRs:'
        @/bin/ls -l ${OSSL_CSR_DIR}
listkey:
        @echo 'Listing private KEY:'
        @/bin/ls -l ${OSSL_PRIV_DIR}
sign:
        @if [ ! -f $(OSSL_CSR_DIR)/${csr} ]; then echo "${csr} : no such
file or directory" ; exit 1 ; fi
        @$(OPENSSL) ca -config ${OSSL_CONF} -in ${OSSL_CSR_DIR}/${csr} -out
${OSSL_CERTS_DIR}/${csr:.csr=.cert}
        @[ -s $(OSSL_CERTS_DIR)/${csr:.csr=.cert} ] && rm -P ${OSSL_CSR_
DIR}/${csr}


# args name= server=yes
gencsr:
        @echo 'Create the CSR for ${name}'
        @$(OPENSSL) req -config ${OSSL_SERV_CONF} -newkey rsa:2048 -keyout
${OSSL_PRIV_DIR}/${name}.key -keyform PEM -out ${OSSL_CSR_DIR}/${name}.csr
-outform PEM
        @echo 'Create an unlocked CSR key version for ${name}'
        @$(OPENSSL) rsa < ${OSSL_PRIV_DIR}/${name}.key > ${OSSL_PRIV_DIR}/
${name}_unlocked.key


gencrl:
        @$(OPENSSL) ca -config ${OSSL_CONF} -gencrl -out ${OSSL_CRL_DIR}/
ca-crl.pem

revoke:
        @$(MAKE) gencrl
init:
        @test ! -f serial
        @mkdir $(OSSL_CRL_DIR) $(OSSL_NEWCERTS_DIR) $(OSSL_PRIV_DIR)
$(OSSL_CSR_DIR) $(OSSL_CERTS_DIR)
        @chmod 700 $(OSSL_PRIV_DIR)
        @echo '01' > $(CAROOT)/serial
        @touch $(CAROOT)/index.txt
        @$(OPENSSL) req -config $(OSSL_CONF) -days 1825 -x509 -newkey
rsa:2048 -out $(CAROOT)/cacert.pem -outform PEM


caclean:
        @echo cleaning CA files
        @rm -rf $(CAROOT)/cacert.pem $(OSSL_CSR_DIR) $(CAROOT)/index.*
$(CAROOT)/serial.* $(OSSL_NEWCERTS_DIR) $(OSSL_PRIV_DIR) $(OSSL_CERTS_DIR)
```

```
help:
        @echo make help
        @echo ' - this help'
        @echo make init
        @echo ' - init the CA authority'
        @echo make list
        @echo ' - list known key, signed certs (index.txt) & pending CSR
        @echo make gencsr name=\<name\>
        @echo ' - generate a CSR (<name>.csr) & associated key (<name>.k
        @echo ' location CSR: ${OSSL_CSR_DIR}/<name>.csr'
        @echo ' location KEY: ${OSSL_PRIV_DIR}/<name>.key'
        @echo make gencrl
        @echo ' - generate the CRL (Certificate Revocation List)'
        @echo ' location: ${OSSL_CRL_DIR}/ca-crl.pem'
        @echo make sign csr=filename.csr
        @echo ' - sign the filename.csr with CA authority'
        @echo ' location: ${OSSL_CSR_DIR}/filename.csr'
        @echo ' result:   ${OSSL_CERTS_DIR}/filename.cert'
        @echo ' csr is automagically deleted.'
        @echo make revoke cert=filename.cert
        @echo ' - revoke the filename.cert certificate and generate a ne
```

OPENOSPF (LE ROUTAGE ENTRE TOUT LE MONDE !)

Ahhhhhhhhh, c'est là que les choses deviennent marrantes :).

Alors si vous montez le **[1]** WRAP avec un OpenBSD pas de souci, **[5]** OpenOSPF compile sans problème, ça marche *out of the box* tout ça, tout ça. Si comme dans notre (allez encore de l'anglais) *use case*, on fait tourner tout sous NetBSD, alors là, y a pas encore de **[5]** OpenOSPF sous NetBSD. Alors, j'ai fait un petit portage de la version 3.9 d'OpenOSPF (la version 4.0 est en cours de portage aussi, mais j'ai trop de boulot, les structures ont changé entre OpenBSD et NetBSD et j'ai pas terminé pour cet article, oui, je suis fainéant !)

Le patch du portage se trouve ici :

http://no.phear.org/codes/misc/openospf/

C'est trivial et ça n'a pris que quelques heures. J'ai également fait un `pkgsrc`, qui se trouve au même endroit, mais il n'est pas tout à fait terminé (il marche quand même, hein, mais il copie pas le script de démarrage comme il faut :), bref... vous avez les billes de base ***feign*** ***feign***.

Le but étant d'éviter les routes statiques partout et à maintenir soi-même. À problème simple, solution simple, du routage dynamique, youpi ! Sur chacun des routeurs, je configure OpenOSPF de la manière suivante, avec une `area 0`.

```
ospfd.conf(5) :
root@earthprout:/usr/pkg/etc/openvpn/ccd $ cat /usr/pkg/etc/ospfd.conf
hi="5"
redistribute connected
```

```
area 0.0.0.0 {
        interface sip0 {
                hello-interval $hi
                auth-type crypt
                auth-md-keyid 1
                auth-md 1 "proutprout"
        }
}
```

Je redistribue toutes les routes des interfaces *connected* à tous mes *neighbours*, sur le VPN je rajoute :

```
redistribute static
```

pour distribuer les routes statiques provenant des tunnels VPN qui se montent dynamiquement, *and* voilà !!!!

Un joli résultat :

```
root@earthprout:~ # ospfctl show neighbor
ID              Pri State        DeadTime Address         Iface
Uptime
10.9.33.2       1   FULL/DR      00:00:36 10.9.33.2       sip0
01w2d22h
```

et ça de l'autre côté :

```
root@airprout:~ # ospfctl show neighbor
ID              Pri State        DeadTime Address         Iface
Uptime
10.9.33.3       1   FULL/BCKUP   00:00:38 10.9.33.3       sip0
01w2d22h
```

On mate vite fait les routes sur l'un des deux :

```
root@airprout:~ # ospfctl show fib
flags: * = valid, O = OSPF, C = Connected, S = Static
Flags  Destination          Nexthop
*S     0.0.0.0/0            10.9.33.1
*O     10.3.33.0/24         10.9.33.3
*C     10.4.33.0/24         link#4
*C     10.7.33.0/24         link#3
*O     10.8.33.0/24         10.9.33.3
*C     10.9.33.0/24         link#2
*O     10.16.1.0/24         10.9.33.3
*S     127.0.0.0/8          127.0.0.1
*C     127.0.0.1/8          link#0
*      127.0.0.1/32         127.0.0.1
*C     172.16.10.0/24       link#1

root@airprout:~ # ospfctl show rib
Destination          Nexthop          Path Type    Type
Cost    Uptime
10.9.33.3            10.9.33.3        Intra-Area   Router
10      01w3d00h
10.9.33.0/24         10.9.33.2        Intra-Area   Network
10      01w3d07h
10.3.33.0/24         10.9.33.3        Type 1 ext   Network
110     01w3d00h
10.8.33.0/24         10.9.33.3        Type 1 ext   Network
110     01w3d00h
10.16.1.0/24         10.9.33.3        Type 1 ext   Network
110     01w2d22h
```

Ça marche !
Le troisième et dernier routeur, qui n'est pas encore là (eh oui, je l'ai commandé, mais il est pas encore là), aura exactement la même configuration.

Alors c'est pas tout simplet tout ça ?!

LE NAT

Le tout dernier souci à régler avant de pouvoir avoir un truc complètement fonctionnel (dont tous les réseaux arrivent à joindre tous les réseaux) est de permettre à ces réseaux d'accéder (ou NON) à l'internet convivial Ferrero Rocher. Dans mon plan de conquête du monde, tout le trafic de l'interne passerait par un serveur SOCKS local, les autres réseaux utiliseraient également le même SOCKS et aucune translation ne serait effectuée, excepté pour le server SOCKS, je n'en suis pas encore là.

En plus, mon routeur ADSL est encore un vieux truc convivial qu'on administre via une page web (USRobotics de mes $#@$!#@), donc pas d'OSPF pour lui, ni rien, je ne contrôle pas mon NAT là-dessus, donc j'ai du NATer au niveau des WRAP en attendant de terminer tout ça.

Cet article se conclut donc par ces règles de NAT qui vous permettront d'avoir un réseau fonctionnel avec pleins de trucs kikoo lol qui claquent. Les règles parlant d'elles-mêmes, je n'en dirai pas plus.

Voici sur Airprout (wrap #1) :

```
root@airprout:~ # cat /etc/ipnat.conf
# internal trusted lan
map sip0 from 10.7.33.0/24 to !10.0.0.0/8 -> 10.9.33.2/32 proxy port ftp
ftp/tcp
map sip0 from 10.7.33.0/24 to !10.0.0.0/8 -> 10.9.33.2/32 portmap tcp/udp
20000:30000 mssclamp 1400
map sip0 from 10.7.33.0/24 to !10.0.0.0/8 -> 10.9.33.2/32 mssclamp 1440
# Wifi Network
map sip0 172.16.10.0/24 -> 10.9.33.2/32 proxy port ftp ftp/tcp
map sip0 172.16.10.0/24 -> 10.9.33.2/32 portmap tcp/udp 30001:40000
mssclamp 1400
map sip0 172.16.10.0/24 -> 10.9.33.2/32 mssclamp 1440
```

Et voici sur Earthprout (wrap #2) :

```
root@earthprout:~ # cat /etc/ipnat.conf
# DMZ network inet access
map sip0 from 10.3.33.0/24 to !10.0.0.0/8 -> 10.9.33.3/32 proxy port ftp
ftp/tcp
map sip0 from 10.3.33.0/24 to !10.0.0.0/8 -> 10.9.33.3/32 portmap tcp/udp
20000:30000 mssclamp 1400
map sip0 from 10.3.33.0/24 to !10.0.0.0/8 -> 10.9.33.3/32
# LABS network inet access
map sip0 from 10.8.33.0/24 to !10.0.0.0/8 -> 10.9.33.3/32 proxy port ftp
ftp/tcp
```

```
map sip0 from 10.8.33.0/24 to !10.0.0.0/8 -> 10.9.33.3/32 portmap tcp/udp
30001:40000 mssclamp 1400
map sip0 from 10.8.33.0/24 to !10.0.0.0/8 -> 10.9.33.3/32
# MAP VPN _ME_ to everywhere, dangerous!
map sip0 from 10.16.1.10/32 to !10.0.0.0/8 -> 10.9.33.3/32 portmap tcp/udp
42001:45000 mssclamp 1400
map sip0 from 10.16.1.10/32 to !10.0.0.0/8 -> 10.9.33.3/32
```

Oui, oui, je sais, c'est du *hack* tout laid, mais bon, j'ai pas fini, c'est en cours de finition. Les règles de filtrages sont à déterminer selon VOS besoins (faites une offrande au dieu des firewalls, il vous donnera probablement une réponse), de même que les autres services de base qui se situent dans le « service network » :

- DNS ;
- SOCKS ;
- syslog ;
- tor proxy ;
- Intranet + zabbix, etc.

Reste à faire :

- SNMP v3 ;
- *router hardening* ;
- OS hardening ;
- etc. ;
- acheter une nouvelle chèvre ;
- acheter des poules ;
- commander un AbdoFlex 3000 ;
- vendre des champignons.

LE TEST & L'INSTANT KODAK

Du réseau interne :

```
rival@kamehouse:~ $ ifconfig wm0
wm0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
capabilities=7ff80<TSO4,IP4CSUM_Rx,IP4CSUM_Tx,TCP4CSUM_Rx,TCP4CSUM_
Tx,UDP4CSUM_Rx,UDP4CSUM_Tx,TCP6CSUM_Rx,TCP6CSUM_Tx,UDP6CSUM_Rx,UDP6CSUM_
Tx,TSO6>
        enabled=0
        media: Ethernet 100baseTX full-duplex
        status: active
        inet 10.7.33.2 netmask 0xffffff00 broadcast 10.7.33.255
        inet6 fe80::215:c5ff:fe01:203%wm0 prefixlen 64 scopeid 0x1
```

J'atteins donc :

La DMZ :

```
rival@kamehouse:~ $ ping 10.3.33.2
PING 10.3.33.2 (10.3.33.2): 56 data bytes
64 bytes from 10.3.33.2: icmp_seq=0 ttl=253 time=1.435 ms
```

Le Lab :

```
rival@kamehouse:~ $ ping 10.8.33.3
PING 10.8.33.3 (10.8.33.3): 56 data bytes
64 bytes from 10.8.33.3: icmp_seq=0 ttl=253 time=3.178 ms
```

L'interco :

```
rival@kamehouse:~ $ ping 10.9.33.1
PING 10.9.33.1 (10.9.33.1): 56 data bytes
64 bytes from 10.9.33.1: icmp_seq=0 ttl=254 time=2.036 ms
```

et tout le reste.

Et l'image de l'installation faite maison :

CONCLUSION

M. Ferrero, cette démonstration faite, tout est désormais entre vos mains. Il me paraît clair que le consommateur est roi dans votre industrie, je vous invite donc à intégrer ces nouvelles données dans la fabrication de vos œufs pour l'année 2008. Je suis sûr que vos profits en ressentiront l'effet salvateur.

Je vous prie d'agréer, M. Ferrero, l'assurance de mes sentiments les plus cordiaux et distingués et vous remercie de l'attention portée à ce courrier.

Dans l'attente de vous lire en retour, je vous souhaite un bon week-end.

(NB : L'important c'est de tripper... et je trippe les Kinder surprises, je les tripperai d'autant plus si y a des trucs gratuits où on peut installer NetBSD dessus, merci de réaliser mes rêves !).

RÉFÉRENCES

[1] WRAP : http://pcengines.ch/wrap.htm

[2] *The Art of UNIX programming* : http://www.faqs.org/docs/artu/

[3] KISS principle : http://www.faqs.org/docs/artu/ch01s07.html

[4] NetBSD : http://netbsd.org

[5] OpenOSPF : http://www.openbgpd.org/

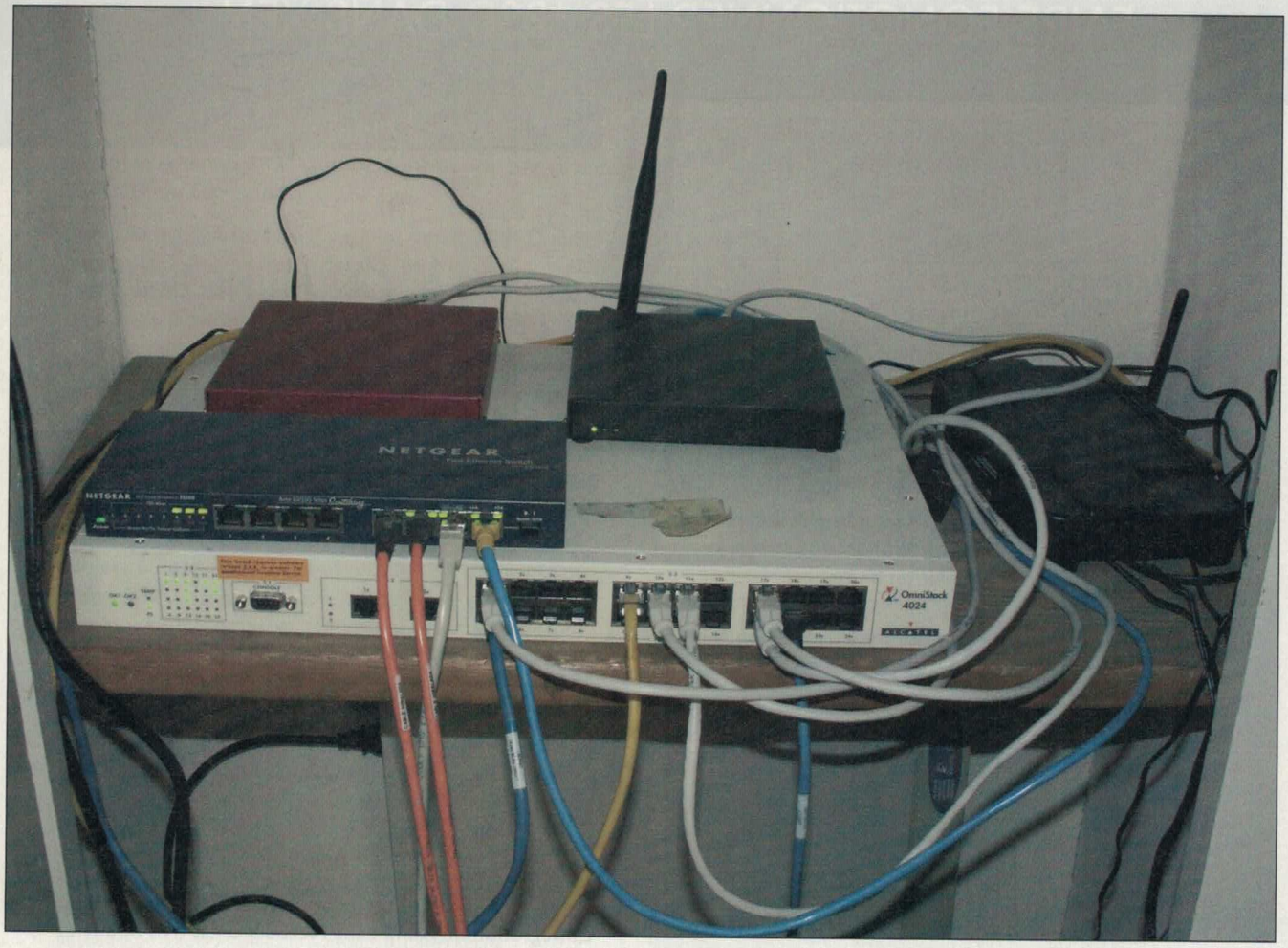
[6] BSD Acte I, « Répartition de charge et haute disponibilité sur les OS *BSD (Partie 1) », *Linux Magazine*, hors-série n° 29.

[7] BSD Acte I, « Routage dynamique et haute disponibilité (Partie 2) », *Linux Magazine*, hors-série n° 29.

[8] ospfd.conf(5)

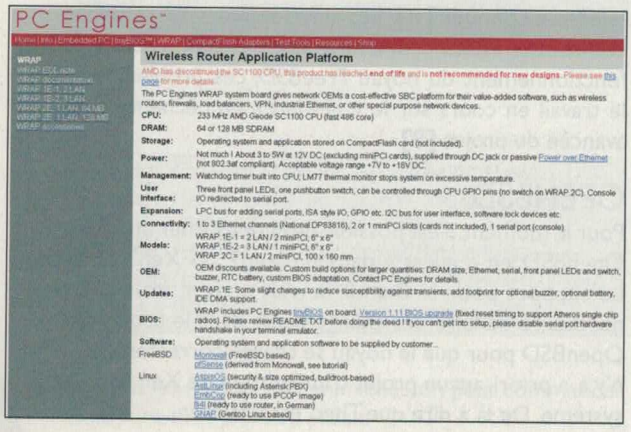
[9] openvpn(8)

[10] hostapd.conf(5)



REMERCIEMENTS

- mes doigts ;
- la lessive (pour des habits qui sentent bon) ;
- mon écran (car il m'affiche des trucs) ;
- la tomate (prout !) ;
- le friand (contre prout !) ;
- GCU (maison !!!!) – *I'm loving it*
- les relecteurs (Benoyst, Courgette, Wilk, etc.)

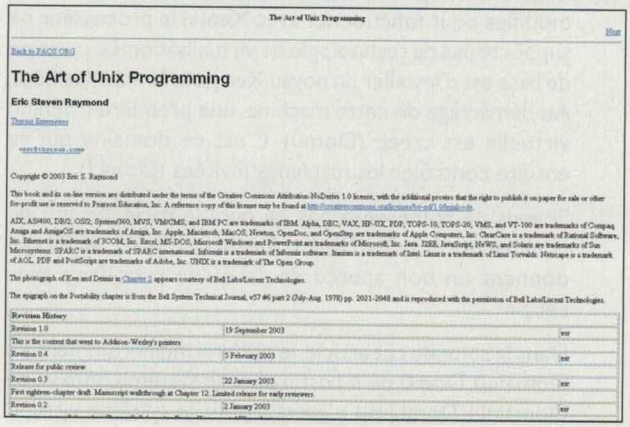

PC Engines

Wireless Router Application Platform

AMD has discontinued the SC1100 CPU, this product has reached end of life and is not recommended for new designs. Please see this page for more details.

The PC Engines WRAP system board gives network OEMs a cost-effective SBC platform for their value-added software, such as wireless routers, firewalls, load balancers, VPN, industrial Ethernet, or other special purpose network devices.

CPU:	233 MHz AMD Geode SC1100 CPU (fast 486 core)
DRAM:	64 or 128 MB SDRAM
Storage:	Operating system and application stored on CompactFlash card (not included).
Power:	Not much ! About 3 to 5W at 12V DC (excluding miniPCI cards), supplied through DC jack or passive Power over Ethernet (not 802.3af compliant). Acceptable voltage range +7V to +18V DC.
Management:	Watchdog timer built into CPU; LM77 thermal monitor stops system on excessive temperature.
User Interface:	Three front panel LEDs, one pushbutton switch, can be controlled through CPU GPIO pins (no switch on WRAP.2C). Console I/O redirected to serial port.
Expansion:	LPC bus for adding serial ports, ISA style I/O, GPIO etc. I2C bus for user interface, software lock devices etc.
Connectivity:	1 to 3 Ethernet channels (National DP83816), 2 or 1 miniPCI slots (cards not included), 1 serial port (console)
Models:	WRAP.1E-1 = 2 LAN / 2 miniPCI, 6" x 6" WRAP.1E-2 = 3 LAN / 1 miniPCI, 6" x 6" WRAP.2C = 1 LAN / 2 miniPCI, 100 x 160 mm
OEM:	OEM discounts available. Custom build options for larger quantities: DRAM size, Ethernet, serial, front panel LEDs and switch, buzzer, RTC battery, custom BIOS adaptation. Contact PC Engines for details.
Updates:	WRAP.1E: Some slight changes to reduce susceptibility against transients, add footprint for optional buzzer, optional battery, IDE DMA support.
BIOS:	WRAP includes PC Engines tinyBIOS on board. Version 1.11 BIOS upgrade (fixed reset timing to support Atheros single chip radios). Please review README.TXT before doing the deed ! If you can't get into setup, please disable serial port hardware handshake in your terminal emulator.
Software:	Operating system and application software to be supplied by customer.
FreeBSD	Monowall (FreeBSD based) pfSense (derived from Monowall, see tutorial)
Linux	AstaroOS (security & size optimized, buildroot-based) AstLinux (including Asterisk PBX) EmbOop (ready to use IPCOP image) fli4l (ready to use router, in German) GNAP (Gentoo Linux based)


The Art of Unix Programming

Back to FAQS.ORG

The Art of Unix Programming

Eric Steven Raymond

Thyrsus Enterprises

<esr@thyrsus.com>

Copyright © 2003 Eric S. Raymond

This book and its on-line version are distributed under the terms of the Creative Commons Attribution-NoDerivs 1.0 license, with the additional proviso that the right to publish it on paper for sale or other for-profit use is reserved to Pearson Education, Inc. A reference copy of this license may be found at http://creativecommons.org/licenses/by-nd/1.0/legalcode.

AIX, AS/400, DB/2, OS/2, System/360, MVS, VM/CMS, and IBM PC are trademarks of IBM. Alpha, DEC, VAX, HP-UX, PDP, TOPS-10, TOPS-20, VMS, and VT-100 are trademarks of Compaq. Amiga and AmigaOS are trademarks of Amiga, Inc. Apple, Macintosh, MacOS, Newton, OpenDoc, and OpenStep are trademarks of Apple Computers, Inc. ClearCase is a trademark of Rational Software, Inc. Ethernet is a trademark of 3COM, Inc. Excel, MS-DOS, Microsoft Windows and PowerPoint are trademarks of Microsoft, Inc. Java, J2EE, JavaScript, HotWL and Solaris are trademarks of Sun Microsystems. SPARC is a trademark of SPARC international. Informix is a trademark of Informix software. Itanium is a trademark of Intel. Linux is a trademark of Linus Torvalds. Netscape is a trademark of AOL. PDF and PostScript are trademarks of Adobe, Inc. UNIX is a trademark of The Open Group.

The photograph of Ken and Dennis in Chapter 2 appears courtesy of Bell Labs/Lucent Technologies.

The epigraph on the Portability chapter is from the Bell System Technical Journal, v57 #6 part 2 (July-Aug. 1978) pp. 2021-2048 and is reproduced with the permission of Bell Labs/Lucent Technologies.

Revision History		
Revision 1.0	19 September 2003	esr
This is the content that went to Addison-Wesley's printers		
Revision 0.4	5 February 2003	esr
Release for public review		
Revision 0.3	22 January 2003	esr
First eighteen-chapter draft. Manuscript walkthrough at Chapter 12. Limited release for early reviewers.		
Revision 0.2	2 January 2003	esr

FAIRE FONCTIONNER LES *BSD DANS XEN

OU COMMENT FAIRE RENTRER TES *BSD EN BOÎTE

SYNOPSIS

Où en sont les principaux *BSD du marché en ce qui concerne Xen, la *hype* du moment ? Nous allons procéder à une petite inspection surprise de l'état des lieux de Xen sur NetBSD, FreeBSD et OpenBSD. Où en est l'intégration de Xen en tant que « Dom0/DomU » dans les différents noyaux BSD, et comment peut-on profiter de ces extraordinaires systèmes d'exploitation lorsque l'on n'a qu'un système GNU/Linux sous la main ? Maintenant, plus de problèmes, vous saurez toujours comment sortir avec vos *BSD pour rentrer en boîte.

! ATTENTION

Attention, cet article est très volatile, mais non inflammable. Les informations contenues ici font état d'un avancement et donc sont sujettes à évolution.

XEN : VITE FAIT, SUR LE POUCE

« Allez, un dernier rappel sur Xen et j'y vais Georges », mais c'est vraiment le dernier ce coup-ci, parce que la dernière fois, on sait comment ça a fini. Il est effectivement de bon ton de commencer ce genre d'article par un rapide rappel afin de mettre le lecteur dans les meilleures conditions pour la suite du sujet.

Xen est donc un logiciel de paravirtualisation, ce qui signifie que l'on peut lancer une machine virtuelle sur n'importe quel type de processeur. Toutefois, les noyaux de la machine hôte et des machines virtuelles doivent être modifiés pour fonctionner avec Xen, si le processeur ne supporte pas de technologie de virtualisation. Le principe de base est d'installer un noyau Xen pour la machine hôte. Au démarrage de cette machine, une première machine virtuelle est créée (Dom0). C'est ce domaine qui va ensuite contrôler les machines invitées (DomU).

Si vous ne comprenez pas de quoi je parle, il suffit de commander les *GNU/Linux Magazine* 85, 87 et 89 qui donnent un bon aperçu de ce qu'on peut faire avec l'engin.

Dans la suite de cet article, je parlerai indifféremment de Domain0, Dom0 ou « *host* » pour le système hôte et de DomainU, DomU ou « *guest* » pour le système virtuel.

ALORS LÀ, ON A UNE BOITE ET LÀ UNE AUTRE

On sait maintenant tous que Xen c'est la *vibe*, c'est le hype, c'est le sujet de toutes les soirées bobo parisiennes du moment. Eh bien, pour vraiment vous la claquer en soirée, il n'y a qu'une solution, parler de l'intégration de Xen dans les 3 principaux *BSD de la place. Alors, on va faire simple, plan en 3 points thèse/antithèse/synt^W NetBSD, FreeBSD et OpenBSD.

NETBSD

NetBSD [1] a été le premier système BSD à intégrer Xen dans son noyau, le premier *commit* dans le noyau [4] daté de mars 2004 [5], et la première sortie officielle de décembre 2004 dans la version 2.0 de NetBSD [6]. Actuellement, le support de Xen2 est complet (Dom0/DomU) dans la version stable de NetBSD (3.1). Cette version a aussi vu arriver le support de Xen3 en DomU. L'intégration de Xen3 en Dom0 se fait dans la version CURRENT et est prévue pour la version 4.0 qui devrait être sortie au moment où vous lisez ces lignes.

L'intégration de Xen2 dans NetBSD-3 est très bonne et l'installation d'un Dom0 ou DomU est d'une simplicité enfantine [7].

FREEBSD

Pour le moment Xen n'est pas officiellement intégré dans FreeBSD [2], il y a un travail mené dans la branche –CURRENT, mais cela constitue un « *work in progress* ». Xen devait être intégré dans la version 6.1 sortie en mai 2006, mais c'est maintenant repoussé à une date et une *release* inconnues [8]. Les développeurs se trouvent confrontés à des problèmes assez profonds dans le fonctionnement du noyau FreeBSD, ces problèmes et le travail en cours sur le « *newbus* » empêchent toute avancée du projet [9].

OPENBSD

Pour le moment, il est possible de faire tourner un noyau OpenBSD en « guest » dans un Xen2 ou Xen3 [10]. Le support pour le Dom0 devrait arriver, mais il est nécessaire de faire des changements dans le *bootloader* OpenBSD pour que le noyau se charge correctement. Il n'y a, a priori, aucun projet d'intégration de Xen dans le système. De là à dire que Theo est *has been*...

LES *BSD DANS TON GNU/LINUX

« Eh oui, Maryse, ce n'est pas un *BSD que nous allons vous proposer avec le SuperXen, mais bien 3 *BSD, vous avez bien entendu, 3 *BSD pour la même somme qu'un seul pack de SuperXen. »

La machine hôte est basée sur une distribution Debian, la machine Xen2 est une Sarge avec un Xen2.0.7 compilé et la machine Xen3 est une Etch avec un Xen3 packagé standard. L'installation de cette machine n'a rien à foutre ici, donc ne posez même pas la question, nan, mais c'est vrai quoi, vous êtes pénibles à la fin **[17][18]**.

Le système de fichiers des machines virtuelles sera créé dans un fichier pour des raisons de simplicité et pour éviter de me la péter avec tout le matos que je pourrais avoir. Mon répertoire de stockage des machines virtualisées est `/var/xen/domains/<nom_domu>`. Mes noyaux et autres fichiers en dehors de la configuration se trouvent dans mon répertoire personnel `~/xen/`.

NETBSD DANS LA BOIBOITE

NetBSD est d'une simplicité déconcertante à installer dans un DomU. Il faut récupérer les 2 noyaux concernés, un noyau pour l'installation et un noyau pour le fonctionnement courant.

Ces noyaux peuvent être récupérés sur n'importe quel miroir NetBSD. En l'occurrence je choisis **ftp.fr.netbsd.org**. Ils se trouvent dans le répertoire `/pub/NetBSD-daily/` des miroirs sous le chemin particulier `netbsd-<release>/<snapshot>/i386/binary/kernel/netbsd-[INSTALL_]XEN[23]_DOMU.gz`.

Je vais là vous en mettre plein la vue et télécharger ces noyaux sans Firefox :

```
xen2$ wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-3/200611240000Z/i386/binary/kernel/netbsd-INSTALL_XEN2_DOMU.gz -O ~/xen/netbsd-xen/netbsd-3-INSTALL_XEN2_DOMU.gz
xen2$ wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-3/200611240000Z/i386/binary/kernel/netbsd-XEN2_DOMU.gz -O ~/xen/netbsd-xen/netbsd-3-XEN2_DOMU.gz
```

Ou alors

```
xen2$ wget ftp://ftp.fr.netbsd.org/pub/NetBSD/NetBSD-3.1/i386/binary/kernel/netbsd-INSTALL_XEN2_DOMU.gz -O ~/xen/netbsd-xen/netbsd-3_1-INSTALL_XEN2_DOMU.gz
xen2$ wget ftp://ftp.fr.netbsd.org/pub/NetBSD/NetBSD-3.1/i386/binary/kernel/netbsd-XEN2_DOMU.gz -O ~/xen/netbsd-xen/netbsd-3_1-XEN2_DOMU.gz
```

Une fois les noyaux décompressés, on peut commencer et se faire tout plein de plaisir.

Fichier de configuration

Le fichier de configuration est `/etc/xen/netbsd.cfg`.

```
kernel = "/home/hr/xen/netbsd-xen/netbsd-3-INSTALL_XEN2_DOMU"
#kernel = "/home/hr/xen/netbsd-xen/netbsd-3-XEN2_DOMU"
memory = 128
name = "netbsd".
disk = [ 'file:/var/xen/domains/netbsd/disk.img,sda1,w' ]
```

Pour commencer, j'indique que le noyau utilisé est celui d'installation. On remarque que j'ai déjà la ligne précisant le noyau de fonctionnement normal en commentaire. J'alloue 128M de ram au système « guest » et j'indique le système de fichiers à utiliser.

Création du système de fichier

On crée un fichier de 3Go qui va porter le système de fichiers pour cette installation :

```
[/var/xen/domains/netbsd]
xen2# dd if=/dev/zero of=disk.img bs=1024k seek=3000 count=1
1+0 records in
1+0 records out
1048576 bytes (1.0 MB) copied, 0.003756 seconds, 279 MB/s
```

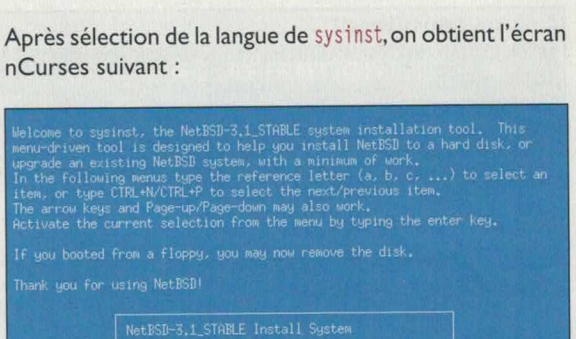
Installer la machine

Il suffit maintenant de lancer le DomU et de procéder à l'installation du NetBSD.

```
xen2# xm create -c netbsd.cfg
Using config file "/etc/xen/netbsd.cfg".
Started domain netbsd, console on port 9601
************ REMOTE CONSOLE: CTRL-] TO QUIT ********
[ Kernel symbol table missing! ]
Copyright (c) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005
    The NetBSD Foundation, Inc.  All rights reserved.
Copyright (c) 1982, 1986, 1989, 1991, 1993
    The Regents of the University of California.  All rights reserved.

NetBSD 3.1_STABLE (INSTALL_XEN2_DOMU) #0: Sat Dec  2 06:04:39 UTC 2006
        builds@pb:/home/builds/ab/netbsd-3/i386/200612010000Z-obj/home/builds/ab
/netbsd-3/src/sys/arch/i386/compile/INSTALL_XEN2_DOMU
total memory = 121 MB
avail memory = 118 MB
mainbus0 (root)
cpu0 at mainbus0: (uniprocessor)
cpu0: Intel (686-class), 2800.22 MHz, id 0xf41
cpu0: features bfebfbff<FPU,VME,DE,PSE,TSC,MSR,PAE,MCE,CX8,APIC,SEP,MTRR>
cpu0: features bfebfbff<PGE,MCA,CMOV,PAT,PSE36,CFLUSH,DS,ACPI,MMX>
cpu0: features bfebfbff<FXSR,SSE,SSE2,SS,HTT,TM,SBF>
cpu0: I-cache 12K uOp cache 8-way
cpu0: L2 cache 1 MB 64B/line 8-way
cpu0: ITLB 4K/4M: 64 entries
cpu0: DTLB 4K/4M: 64 entries
cpu0: 32 page colors
```

```
hypervisor0 at mainbus0
debug virtual interrupt using event channel 2
misdirect virtual interrupt using event channel 0
Domain controller: using event channel 1
xencons0 at hypervisor0: Xen Virtual Console Driver
xencons0: console major 143, unit 0
Initialising Xen virtual ethernet frontend driver.
npx0 at hypervisor0: using exception 16
Xen clock: using event channel 3
Kernelized RAIDframe activated
md0: internal 5000 KB image area
xennet0 at hypervisor0: Xen Virtual Network Interface
xennet0: using event channel 4
xennet0: MAC address aa:00:00:5b:47:c3
xbd: using event channel 5
xbd0 at hypervisor0: Xen Virtual Block Device 4001 MB
boot device: xbd0
root on md0a dumps on md0b
root file system type: ffs
warning: no /dev/console
init: Creating mfs /dev (413 blocks, 1024 inodes)
erase ^?, werase ^W, kill ^U, intr ^C
```

Après sélection de la langue de sysinst, on obtient l'écran nCurses suivant :



Il suffit maintenant de procéder à une installation normale de votre système. Cette installation est entièrement couverte dans l'excellent guide NetBSD [11].

A la fin de l'installation, **ne redémarrez pas**. Il reste une manipulation à faire. Depuis la racine de l'outil sysinst, choisir ***e: Menu utilitaire*** et ensuite ***a: Exécuter /bin/sh***.

```
# mount /dev/xbd0a /mnt
# cd /mnt/dev
# cp -pR /dev/rxbd* .
# cp -pR /dev/xbd* .
# halt -p
```

Démarrer la machine

Éditons le fichier de configuration pour maintenant utiliser le noyau netbsd-3-XEN2_DOMU. Il ne nous reste plus, à présent, qu'à relancer le DomU sans la console qui ne devrait plus servir.

```
xen2# xm create netbsd.cfg
Using config file "/etc/xen/netbsd.cfg".
Started domain netbsd, console on port 9602
xen2$ xm list
Name              Id  Mem(MB)  CPU  State  Time(s)  Console
Domain-0           0      245    0  r----   2763.3
netbsd01           2      127    1  -b---      0.6     9602
```

Le domaine est maintenant lancé, il faut finir l'installation en précisant une adresse IP pour la carte réseau qui apparaîtra sous le nom de xennet0 et éventuellement démarrer le service ssh, histoire de ne pas recourir à la console pour prendre la main sur cette machine.

```
xen2# xm console 2
************ REMOTE CONSOLE: CTRL-] TO QUIT ********
Loaded initial symtab at 0xc04aa85c, strtab at 0xc04d8960, # entries 11
Copyright (c) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 200
    The NetBSD Foundation, Inc.  All rights reserved.
Copyright (c) 1982, 1986, 1989, 1991, 1993
    The Regents of the University of California.  All rights reserved.

NetBSD 3.1_STABLE (XEN2_DOMU) #0: Sat Dec  2 05:46:52 UTC 2006
        builds@pb:/home/builds/ab/netbsd-3/i386/200612010000Z-obj/home/
builds/ab
/netbsd-3/src/sys/arch/i386/compile/XEN2_DOMU
total memory = 123 MB
avail memory = 120 MB
[...]
Starting sendmail.
Starting inetd.
Starting cron.
Mon Jan  1 20:00:01 CET 2007

NetBSD/i386 (Amnesiac) (console)

login:
```

A l'issue de cette première configuration, on se déconnecte de la console à l'aide de [ctrl+]] et on ne discute pas ! Cette installation aurait pu se faire dans un Xen3 avec la même facilité.

Et c'est là que le Verbe s'adressa à moi

Le courant vacille, un puits de lumière s'abat sur mon visage et une voix tonitruante résonne dans mon crâne à m'en faire péter les synapses. « Tu peux faire BEAUCOUP mieux ! ».

Et, effectivement, on peut encore s'amuser avec la serviette orange, même quand elle semble essorée comme une serpillière. Imaginons que vous vouliez ajouter le support de « PF » dans le noyau de votre DomU. Je suppose que tu sais compiler ton propre kernel NetBSD, sinon tu retournes dans ta chambre m'apprendre ce cours [12] ! Et tu seras probablement déconcerté par la simplicité enfantine de la procédure.

Je vérifie d'abord si mon système actuel a le support *kivabien* pour PF (cf. article de Gaston p. 20 du hors-série 29).

```
netbsd# pfctl -e
pfctl: /dev/pf: Device not configured
```

Nous allons donc compiler un nouveau noyau DomU avec le support de cet extraordinaire *firewall* qu'est PF (cf. article de Gaston) avec pour cible un Dom0 en Xen2. Le fichier de configuration de base pour ce faire se trouve dans le répertoire de configuration correspondant à ton architecture (`arch/<architecture>/conf`) avec un nom du type `XEN[23]_DOMU`.

```
cd /usr/src/sys/arch/i386/conf
cp XEN2_DOMU XEN2_DOMU_pf
```

On décommente simplement les 2 lignes qui correspondent à PF et PFlog et c'est parti pour la partie de compilation habituelle.

```
pseudo-device pf                    # PF packet filter
pseudo-device pflog                 # PF log if
netbsd# config XEN2_DOMU_pf
Build directory is ../compile/XEN2_DOMU_pf
Don't forget to run "make depend"
netbsd# cd ../compile/XEN2_DOMU_pf
netbsd# make depend
netbsd# time make
[...snip...]
real    4m46.265s
user    4m5.348s
sys     0m34.219s
```

En aparté, la machine utilisée est équipée de 2 Xeon cadencés à 2,8GHz. Les guests en Xen2 ne peuvent profiter que d'un seul processeur. Le support multiprocesseur n'étant pas encore implémenté dans les DomU, l'*hyperthreading* n'apparaît pas non plus.

```
netbsd# dmesg | grep -i cpu
cpu0 at mainbus0: (uniprocessor)
cpu0: Intel (686-class), 2800.22 MHz, id 0xf41
cpu0: features bfebfbff<FPU,VME,DE,PSE,TSC,MSR,PAE,MCE,CX8,APIC,SEP,MTRR>
cpu0: features bfebfbff<PGE,MCA,CMOV,PAT,PSE36,CFLUSH,DS,ACPI,MMX>
cpu0: features bfebfbff<FXSR,SSE,SSE2,SS,HTT,TM,SBF>
cpu0: I-cache 12K uOp cache 8-way
cpu0: L2 cache 1 MB 64B/line 8-way
cpu0: ITLB 4K/4M: 64 entries
cpu0: DTLB 4K/4M: 64 entries
cpu0: 32 page colors
```

A la fin de la compilation, on se retrouve avec un mignon petit noyau tout frais tout chaud dans le fichier `netbsd`. On transfère ce fichier sur la machine Dom0 pour pouvoir l'utiliser.

```
netbsd# scp netbsd hr@xen2:~/xen/netbsd-xen/netbsd-XEN2_DOMU_pf
```

On éteint la machine NetBSD d'une façon ou d'une autre. On peut se la faire cool avec un `halt -p` directement en console ou alors comme un sauvage avec un `xm destroy`.

Il faut modifier le fichier de configuration pour que la variable kernel pointe vers ce nouveau noyau et on redémarre le DomU.

```
xen2# grep kernel /etc/xen/netbsd.cfg
#kernel = "/home/sis/sbe/xen/netbsd-xen/netbsd-INSTALL_XEN2_DOMU"
kernel = "/home/sis/sbe/xen/netbsd-xen/netbsd-XEN2_DOMU_pf"
xen2# xm create netbsd.cfg
Using config file "/etc/xen/netbsd.cfg".
Started domain netbsd, console on port 9639
```

Je me connecte maintenant à la machine NetBSD et je teste si PF :

```
netbsd# pfctl -e
No ALTQ support in kernel
ALTQ related functions disabled
pfctl: pf already enabled
```

Chouette, on peut maintenant faire des circuits majorette avec nos lutins, des sens interdits, des culs de sac, des loopings et même la boucle qui monte sur le mur.

from __future__ import netbsd-4.0

Rassurez-vous tout de suite, nous n'aurons pas besoin d'1.21 *jigowatts* de puissance pour faire tourner `netbsd-4` dans un environnement Xen3, le professeur Emmett Brown vous le dirait. Tout ce que ça nous coûte, c'est un petit téléchargement à l'adresse suivante **ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-4/<snapshot>/i386/binary/kernel/**.

```
xen3# wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-4/200612080000Z/
i386/binary/kernel/netbsd-INSTALL_XEN3_DOMU.gz -O ~/xen/netbsd-xen/netbsd-4-
INSTALL_XEN3_DOMU.gz
xen3# wget ftp://ftp.fr.netbsd.org/pub/NetBSD-daily/netbsd-4/200612080000Z/
i386/binary/kernel/netbsd-XEN3_DOMU.gz -O ~/xen/netbsd-xen/netbsd-4-XEN3_
DOMU.gz
```

On va modifier un peu les choses par rapport à nos installations de NetBSD précédentes. On va effectuer l'opération dans une partition physique.

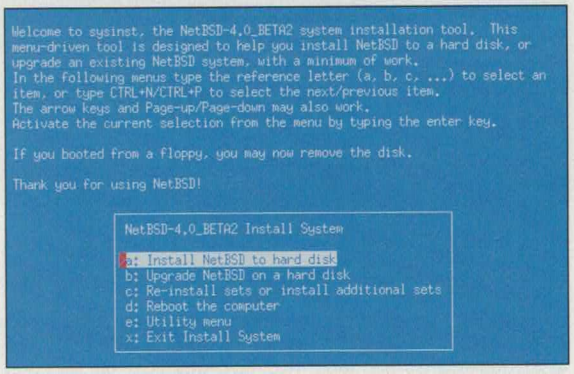
Fichier de configuration

```
kernel = "/home/hr/xen/netbsd-xen/netbsd-4-INSTALL_XEN3_DOMU"
#kernel = "/home/hr/xen/netbsd-xen/netbsd-4-XEN3_DOMU"
memory = 128
name = "netbsd"
disk = [ 'phy:sda8,sda1,w' ]
vif = [ '' ]
```

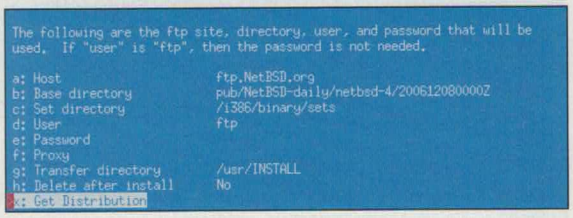
C'est là que se situe la principale nouveauté en dehors de la version du noyau. On a précisé une partition physique comme disque. La syntaxe est simple à comprendre. Je vous laisse donc à vos copies : vous avez 1h pour épuiser le sujet suivant : « Les huîtres pratiquent-elles le cululingus ? ». À noter tout de même, l'obligation d'ajouter une entrée *vif* vide pour activer la création d'une carte réseau virtuelle.

Installation de la machine

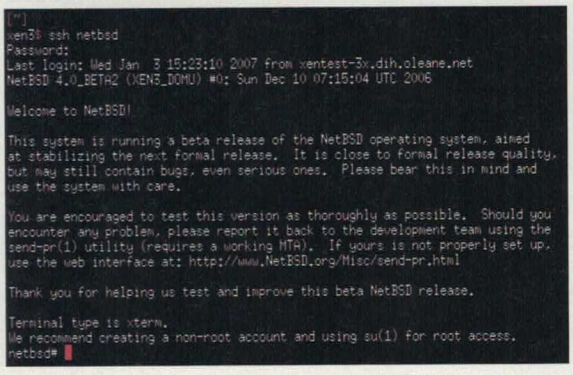
Tout se passe comme un charme, bien évidemment, et exactement de la même façon que précédemment indiqué dans la partie sus-citée... tout pareil quoi.



Il faut quand même adapter le chemin où aller chercher le système lorsque l'on choisit le mode d'installation FTP. Le chemin de base indiqué doit être celui du *snapshot* du noyau que vous utilisez de la forme `pub/NetBSD-daily/netbsd-4/<snapshot>`.



Il n'est plus nécessaire d'effectuer la copie des *devices* à la fin de l'installation et, là, on se retrouve avec Marty McFly les yeux pleins d'espoir devant une Deloreane 4.0_BETA2.



FREEBSD DANS LA BOIBOITE

Installer un FreeBSD fonctionne à peu près de la même façon grâce à la contribution des développeurs FreeBSD **[13]** et d'utilisateurs **[14]**. Pour le moment, FreeBSD ne tourne que sur du Xen2 en version 5.3. Les efforts sont portés sur Xen3. Le principe est donc de télécharger 2 noyaux qui vont servir pour l'installation et le fonctionnement du DomU.

```
xen2# wget http://txrx.org/xen/freebsd-INSTALL_XENU -O ~/xen/freebsd-xen
freebsd-5_3-INSTALL_XENU
xen2# wget http://txrx.org/xen/freebsd-XENU-DEMOCD -O ~/xen/freebsd-xen/
freebsd-5_3-XENU
```

Fichier de configuration

```
kernel = "/home/hr/xen/freebsd-xen/freebsd-5_3-INSTALL_XENU"
#kernel = "/home/hr/xen/freebsd-xen/freebsd-5_3-XENU"
memory = 32
name = "freebsd"
disk = [ 'file:/var/xen/domains/freebsd01/disk.img,sda1,rw' ]
extra = "vfs.root.mountfrom=ufs:/dev/md0,kern.hz=100"
#extra = ",vfs.root.mountfrom=ufs:/dev/xbd0s1a"
```

C'est le même principe que précédemment hormis la variable extra que vos yeux de lynx n'auront pas manqué de repérer.

L'option `vfs.root.mountfrom` nous permet de spécifier un point de montage fictif précisant le système de fichier utilisé. Quant à `kern.hz`, sa présence n'est pas nécessaire.

! IMPORTANT

Il est nécessaire que le montant de mémoire alloué lors de l'installati soit de 32, toute autre valeur fera planter le DomU au démarra

Création du système de fichier

On crée un fichier de 3Go qui va porter le système de fichier pour cette installation :

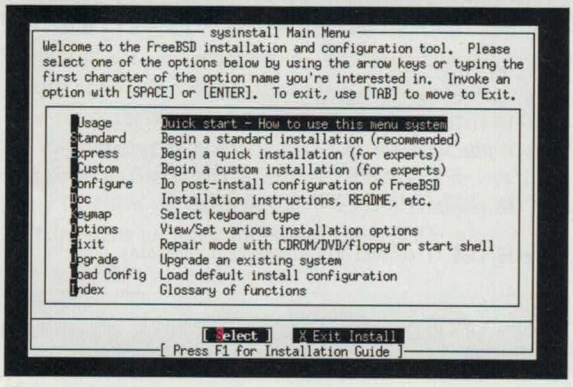
```
[/var/xen/domains/freebsd]
xen2# dd if=/dev/zero of=disk.img bs=1024k seek=3000 count=1
1+0 records in
1+0 records out
1048576 bytes (1.0 MB) copied, 0.003756 seconds, 279 MB/s
```

Installer la machine

```
xen2# xm create -c /etc/xen/freebsd.cfg
Using config file "/etc/xen/freebsd.cfg".
Started domain freebsd, console on port 9612
************ REMOTE CONSOLE: CTRL-] TO QUIT ********
WARNING: loader(8) metadata is missing!
start_info 0xc03d9000
start_info->nr_pages 8192
Copyright (c) 1992-2004 The FreeBSD Project.
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 199
        The Regents of the University of California. All rights reserve
```

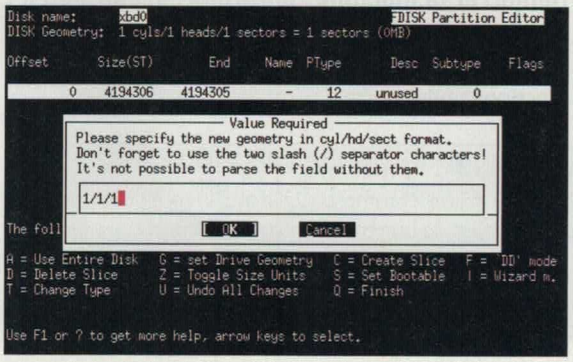
```
FreeBSD 5.3-RELEASE #40: Sun Nov 20 20:19:54 CST 2005
    root@flow.txrx.org:/root/xen-2.0/freebsd-5.3-xenU/i386-xen/compile/
XENUINSTALL
Timecounter "ixen" frequency 2800224000 Hz quality 0
CPU: Intel(R) Xeon(TM) CPU 2.80GHz (2800.22-MHz 686-class CPU)
  Origin = "GenuineIntel"  Id = 0xf41  Stepping = 1
  Features=0xbfebfbff<FPU,VME,DE,PSE,TSC,MSR,PAE,MCE,CX8,APIC,SEP,MTRR,P
GE,MCA,CMOV,PAT,PSE36,CLFLUSH,DTS,ACPI,MMX,FXSR,SSE,SSE2,SS,HTT,TM,PBE>
  Hyperthreading: 2 logical CPUs
real memory  = 29364224 (28 MB)
avail memory = 24186880 (23 MB)
WARNING: driver "evtchn" used unreserved major device number 140
cpu0 on motherboard
Timecounters tick every 10.000 msec
xc0: <Xen Console> on motherboard
WARNING: driver "xc" used unreserved major device number 12
xn0: Ethernet address: aa:00:00:36:b7:4a
Mounting root from ufs:/dev/md0
/stand/sysinstall running as init on serial console

These are the predefined terminal types available to
sysinstall when running stand-alone.  Please choose the
closest match for your particular terminal.

1 ..................... Standard ANSI terminal.
2 ..................... VT100 or compatible terminal.
3 ..................... FreeBSD system console (color).
4 ..................... FreeBSD system console (monochrome).

5 ..................... xterm terminal emulator.

Your choice: (1-5) 2
```

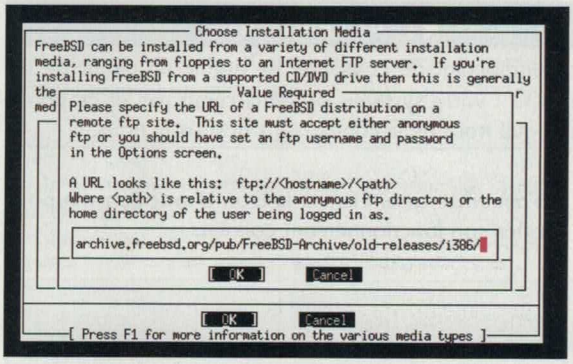
J'ai obtenu les meilleurs résultats d'affichage en choisissant un terminal VT100. Vous devriez être illuminés par l'apparition du système d'installation de FreeBSD sysinstall :



Il faut noter tout de même qu'il y a quelques pièges au cours de l'installation. La création du *slice* FreeBSD va planter violemment si vous ne précisez pas une géométrie pour votre disque, quitte à en donner une fausse. L'option « G » de l'outil `fdisk` permet de donner une géométrie de 1/1/1 par exemple.

Il faut aussi préciser quel serveur FTP utiliser pour installer le système. La release 5.3 étant passée en archive, il faut choisir un autre serveur FTP via l'option ***Specify some other ftp site by URL*** et donner le chemin suivant : **ftp://ftp-archive.freebsd.org/pub/FreeBSD-Archive/old-releases/i386/**. L'installation risque d'être très lente via ce serveur d'archives.





Lorsque l'installation du système est terminée, il vous est demandé si vous voulez configurer des services de base tels que NFS, inetd ou SSH. Si vous souhaitez activer la compatibilité Linux, il faudra indiquer un serveur FTP particulier comme précédemment, la même adresse doit être utilisée.

Finalement, il est recommandé de créer un compte utilisateur qui appartient au groupe `wheel` au cas où la configuration de la console ne fonctionnerait pas.

A la fin de l'installation, revenir au menu pour configurer les TTY via le menu ***TTYs - Configure system ttys***. Il faut activer la console et désactiver tous les pseudo-terminaux (ttyv[0-8]).

```
^[ (escape) menu  ^y search prompt  ^k delete line  ^p prev li  ^g prev page
^o ascii code     ^x search         ^l undelete line ^n next li  ^v next page
^u end of file    ^a begin of line  ^w delete word   ^b back 1 char
^t begin of file  ^e end of line    ^r restore word  ^f forward 1 char
^c command        ^d delete char    ^j undelete char ^z next word
-: 35 C: 52 ==========================================================
#
# If console is marked "insecure", then init will ask for the root password
# when going to single-user mode.
console "/usr/libexec/getty Pc"         cons25  on  secure
#
ttyv0   "/usr/libexec/getty Pc"         cons25  off secure
# Virtual terminals
ttyv1   "/usr/libexec/getty Pc"         cons25  off  secure
ttyv2   "/usr/libexec/getty Pc"         cons25  off  secure
ttyv3   "/usr/libexec/getty Pc"         cons25  off  secure
ttyv4   "/usr/libexec/getty Pc"         cons25  off  secure
ttyv5   "/usr/libexec/getty Pc"         cons25  off  secure
ttyv6   "/usr/libexec/getty Pc"         cons25  off  secure
ttyv7   "/usr/libexec/getty Pc"         cons25  off  secure
ttyv8   "/usr/X11R6/bin/xdm -nodaemon"  xterm   off secure
# Serial terminals
# The 'dialup' keyword identifies dialin lines to login, fingerd etc.
```

On précise que la connexion via la console se fait en mode *secure*, donc *root* est autorisé à se connecter.

Démarrer la machine

On peut modifier le fichier de configuration pour faire maintenant démarrer la machine en mode de fonctionnement normal :

```
#kernel = "/home/hr/xen/freebsd-xen/freebsd-5_3-INSTALL_XENU"
kernel = "/home/hr/xen/freebsd-xen/freebsd-5_3-XENU"
memory = 32
name = "freebsd"
disk = [ 'file:/var/xen/domains/freebsd/disk.img,sda1,rw' ]
#extra = "vfs.root.mountfrom=ufs:/dev/md0,kern.hz=100"
extra = ",vfs.root.mountfrom=ufs:/dev/xbd0s1a"
```

ATTENTION

Il est très important de conserver la virgule en début de la variable `extra` : votre système ne démarrerait pas, et ça, pour une raison que nous ne la connaissons ni vous ni nous.

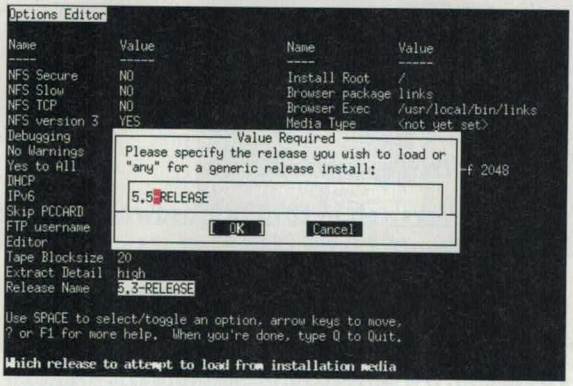
On peut maintenant lancer le DomU avec console, pour vérifier son fonctionnement correct.

```
xen2# xm create -c freebsd.cfg
Using config file "/etc/xen/freebsd.cfg".
Started domain freebsd, console on port 9627
************ REMOTE CONSOLE: CTRL-] TO QUIT ********
WARNING: loader(8) metadata is missing!
KDB: debugger backends: xendb
KDB: current backend: xendb
start_info 0xc0284000
start_info->nr_pages 8192
FreeBSD Physical Memory (29 MB) = 0x0 - 0x1d56000
Copyright (c) 1992-2004 The FreeBSD Project.
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993,
1994
        The Regents of the University of California. All rights
reserved.
FreeBSD 5.3-RELEASE #2: Mon Feb 28 13:49:22 PST 2005
    kmacy@bldf1.eng.netapp.com:/amd/ayr/vol/users2/home/kmacy/p4/ng/
freebsd/usr/src/sys/i386/compile/XENCONF
Timecounter "ixen" frequency 2800224000 Hz quality 0
CPU: Intel(R) Xeon(TM) CPU 2.80GHz (2800.22-MHz 686-class CPU)
  Origin = "GenuineIntel"  Id = 0xf41  Stepping = 1
  Features=0xbfebfbff<FPU,VME,DE,PSE,TSC,MSR,PAE,MCE,CX8,APIC,SEP,MTR
R,PGE,MCA,CMOV,PAT,PSE36,CLFLUSH,DTS,ACPI,MMX,FXSR,SSE,SSE2,SS,HTT,TM
,PBE>
  Hyperthreading: 2 logical CPUs
real memory  = 30760960 (29 MB)
avail memory = 26923008 (25 MB)
WARNING: driver "evtchn" used unreserved major device number 140
Timecounters tick every 10.000 msec
xc0: <Xen Console> on motherboard
WARNING: driver "xc" used unreserved major device number 12
xn0: Ethernet address: aa:00:00:71:cf:1f
Mounting root from ufs:/dev/xbd0s1a
Pre-seeding PRNG: kickstart.
Loading configuration files.
```

```
Entropy harvesting: interrupts ethernet point_to_point kickstart.
swapon: adding /dev/xbd0s1b as swap device
Starting file system checks:
/dev/xbd0s1a: FILE SYSTEM CLEAN; SKIPPING CHECKS
/dev/xbd0s1a: clean, 109514 free (1514 frags, 13500 blocks, 1.2%
fragmentation)
/dev/xbd0s1e: FILE SYSTEM CLEAN; SKIPPING CHECKS
/dev/xbd0s1e: clean, 126836 free (28 frags, 15851 blocks, 0.0%
fragmentation)
/dev/xbd0s1f: FILE SYSTEM CLEAN; SKIPPING CHECKS
/dev/xbd0s1f: clean, 138152 free (240 frags, 17239 blocks, 0.0%
fragmentation)
/dev/xbd0s1d: FILE SYSTEM CLEAN; SKIPPING CHECKS
/dev/xbd0s1d: clean, 126664 free (40 frags, 15828 blocks, 0.0%
fragmentation)
Setting hostname: freebsd01.dih.oleane.net.
adjkerntz[257]: sysctl(put_wallclock): No such file or directory
in6_ifattach: xn0 is not multicast capable, IPv6 not enabled
xn0: flags=843<UP,BROADCAST,RUNNING,SIMPLEX> mtu 1500
        inet 213.56.0.116 netmask 0xfffffe0 broadcast 213.56.0.127
        ether aa:00:00:71:cf:1f
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
        inet 127.0.0.1 netmask 0xff000000
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x2
add net default: gateway 213.56.0.97
Additional routing options:.
Starting devd.
Mounting NFS file systems:.
Starting syslogd.
Jan  2 10:54:56 freebsd01 syslogd: kernel boot file is /kernel
ELF ldconfig path: /lib /usr/lib /usr/lib/compat /usr/local/lib
a.out ldconfig path: /usr/lib/aout /usr/lib/compat/aout
Starting local daemons:.
Updating motd.
Starting sshd.
Initial i386 initialization:.
Additional ABI support: linuxlink_elf (1): 1 symbol cpu_fxsr undefined
.
Starting cron.
Local package initialization:.
Additional TCP options:.
Starting background file system checks in 60 seconds.

Tue Jan  2 10:55:00 CET 2007

FreeBSD/i386 (freebsd01.dih.oleane.net) (console)

login:
"Et c'est un HOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOME RUUUUUUUUUUUUUUUUUUU
UUUUUUUUUUUUUN".
```

Le Bonux : l'installation plus rapide

Le principe est un peu bancal, mais permet d'installer le système beaucoup plus vite. La release 5.5 étant encore dans tous les FTP à l'heure de l'écriture de cet article, au lieu de préciser un *repository archive*, on va utiliser cette version FreeBSD. Au démarrage de sysinstall, allez dans le menu ***Options - View/Set various installation options***. Une fois dans le menu, changer la valeur de « Release Name » et la mettre à « 5.5-RELEASE ».



Lors du choix du serveur FTP pour installer les ports, il suffit de choisir un serveur classique type **ftp.fr.freebsd.org**.

OPENBSD DANS LA BOIBOITE

L'installation d'un DomU OpenBSD n'est pas aussi simple que pour les 2 précédents BSD. Le *BSD au poisson qui pique fait bien son office en piquant un peu là, oui là, juste sous l'ongle. Mais rien n'est impossible à une personne déterminée et encore une fois, à l'exception de NetBSD, l'intégration de Xen dans les noyaux *BSd en est à ses débuts.

L'installation se fait ici dans un Xen3, on installe un OpenBSD 4.0.

Construire les noyaux

Le noyau nous est fourni par Christoph Egger qui a travaillé sur ce projet dans le cadre d'un *Google Summer Of Code* **[15]**. La machine utilisée comme *repository mercurial* est une machine de développement sur une petite connexion de toute évidence. Ne téléchargez les sources qui si vous souhaitez vraiment compiler ce noyau. En solution de remplacement, vous trouverez une archive des sources mercurial du noyau datant du 9 janvier 2007 dans l'archive de l'article **[20]**. Cette compilation doit se faire dans un environnement OpenBSD.

```
$ hg clone http://hg.recoil.org/openbsd-xen-sys.hg
$ cd openbsd-xen-sys.hg/arch/xen
$ ln -s conf.i386 conf
$ mkdir compile
$ cd conf
$ config RAMDISK_XENU
$ config XENU
$ cd ../compile/RAMDISK_XENU
$ make depend
$ make
$ mv bsd openbsd-4_0-RAMDISK_XENU
$ cd ../XENU
$ make depend
$ make
$ mv bsd openbsd-4_0-XENU
```

Nous avons maintenant 2 noyaux OpenBSD à monter en DomU, `openbsd-4_0-RAMDISK_XENU` et `openbsd-4_0-XENU`.

Fichier de configuration

```
kernel = "/home/hr/xen/openbsd-xen/openbsd-4_0-RAMDISK_XENU"
#kernel = "/home/hr/xen/openbsd-xen/openbsd-4_0-XENU"
memory = 128
name = "openbsd"
disk = [ 'file:/var/xen/domains/openbsd01/disk.img,sda1,w' ]
extra = "boot_verbose,boot_single,vfs.root.mountfrom=ufs:/dev/md0,kern.
hz=100"
```

Le fichier de configuration est très semblable à ce que nous avons déjà mis en place précédemment. Vous devriez le comprendre sans problème.

Création du système de fichier

On crée un fichier de 3Go qui va porter le système de fichiers pour cette installation :

```
[/var/xen/domains/openbsd]
xen3# dd if=/dev/zero of=disk.img bs=1024k seek=3000 count=1
1+0 records in
1+0 records out
1048576 bytes (1.0 MB) copied, 0.003756 seconds, 279 MB/s
dd if=/dev/zero of=disk.img bs=1024k count=3000
3000+0 records in
3000+0 records out
3145728000 bytes (3.1 GB) copied, 38.3239 seconds, 82.1 MB/s
```

Installer la machine

```
# xm create -c openbsd.cfg
Using config file "openbsd.cfg".
Started domain openbsd
[ using 188660 bytes of bsd ELF symbol table ]
Copyright (c) 1982, 1986, 1989, 1991, 1993
        The Regents of the University of California.  All rights reserved.
Copyright (c) 1995-2006 OpenBSD. All rights reserved.  http://www.OpenBSD.
org

OpenBSD 4.0-current (XENU) #1: Sun Dec 31 17:13:15 CET 2006
    root@openbsd:/home/user/OpenBSD-XEN/openbsd-xen-sys/arch/xen/compile/
XENU
cpu0: Intel(R) Xeon(TM) CPU 2.80GHz ("GenuineIntel" 686-class) 2.80 GHz
cpu0: FPU,V86,DE,TSC,MSR,PAE,MCE,CX8,APIC,MCA,CMOV,PAT,PSE36,CFLUSH,ACPI,MMX
,SSE,SSE2,SS,HTT,TM,SBF,PNI,MWAIT,CNXT-ID
real mem  = 130899968 (127832K)
avail mem = 118927360 (116140K)
using 1623 buffers containing 6647808 bytes (6492K) of memory
mainbus0 (root)
cpu0 at mainbus0
hypervisor0 at mainbus0
debug virtual interrupt using event channel 3
xenbus0 at hypervisor0: Xen Virtual Bus Interface
xencons0 at hypervisor0: Xen Virtual Console Driver
xencons0: console major 86, unit 0
xencons0: using event channel 2
```

```
npx0 at hypervisor0: using exception 16
Xen clock: using event channel 4
xenbus0: using event channel 1
sd0 at xenbus0 id 2049: Xen Virtual Block Device Interface
sd0: using event channel 5
sd0: 2001 MB, 512 bytes/sect x 4098048 sectors
root on sd0a
rootdev=0x400 rrootdev=0xd00 rawdev=0xd02
panic: root filesystem has size 0
Stopped at      Debugger+0x4:   popl    %ebp
RUN AT LEAST 'trace' AND 'ps' AND INCLUDE OUTPUT WHEN REPORTING THIS PANIC!
DO NOT EVEN BOTHER REPORTING THIS WITHOUT INCLUDING THAT INFORMATION!
ddb>
```

Pour le moment, il m'a été impossible de lancer le noyau OpenBSD en mode installation. Malgré des rapports de succès de la part de la personne chez Google qui supervisait le projet **[16]**, la réponse qui m'a été donnée est la suivante :

```
From: Anil Madhavapeddy <anil@recoil.org>
[...]
It's still very unstable...
```

CONCLUSION

L'intégration de Xen dans les systèmes *BSD est faite de façon très inégale. Ca va de l'intégration complète pour NetBSD aux premiers balbutiements pour OpenBSD. L'intégration dans FreeBSD devrait être améliorée dans les futures versions du système, alors qu'OpenBSD n'a pas encore projeté d'intégrer Xen. Ces intégrations risquent d'être ralenties par l'apparition des processeurs avec support des instructions de virtualisation. Ces processeurs vont sûrement devenir de plus en plus courants. Sur ce type de processeur, il n'est pas nécessaire de modifier le noyau du système invité.

RÉFÉRENCES

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

[2] http://www.freebsd.org/

[3] http://www.openbsd.org/

[4] http://cvsweb.netbsd.org/bsdweb.cgi/src/sys/arch/xen/

[5] http://lists.xensource.com/archives/html/xen-devel/2004-01/msg00105.html

[6] http://www.netbsd.org/Foundation/press/xen.html

[7] http://www.netbsd.org/Ports/xen/

[8] http://www.freebsd.org/news/status/report-jan-2006-mar-2006.html#FreeBSD-on-Xen-3.0

[9] http://www.fsmware.com/xenofreebsd/7.0/STATUS

[10] http://ropersonline.com/openbsd/xen/

[11] http://www.netbsd-fr.org/guide/fr/netbsd.html

[12] http://www.netbsd.org/guide/en/chap-kernel.html

[13] http://www.fsmware.com/xenofreebsd/5.3/

[14] http://txrx.org/xen/

[15] http://ropersonline.com/openbsd/xen/openbsd-xen-howto

[16] http://anil.recoil.org/blog/articles/2006/08/21/openbsd-xen-boots-multi-user

[17] http://www.debian-administration.org/articles/304

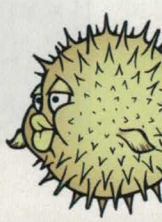
[18] http://www.debian-administration.org/articles/320

[19] http://wiki.xensource.com/xenwiki/HowTos

[20] http://www.bonz.org/glmf_hs0207/linux_dom0_bsd_domu.tar.gz

REMERCIEMENTS

Merci à Bsdmaniak et Raph_ael de m'avoir prêté du temps processeur sous OpenBSD.



Par Eric Auge eau@gcu-squad.org pour GCU
le foyer SONACOTRA des coders Gruik

KQUEUE/KEVENT EFFICIENT CONVIVIAL POLLING

MONTE-TOI UN CHAR AVEC DES VIEILLES TUILES ET UN BOUT DE BOIS !

INTRO

(Je suis à la bourre, vite un Droliprane 500mg !!!)

Ca va être un article style Agence tous risques. Tu prends tes vieilles bagouzes, des colliers qui brillent, tu te fais une crête ou alors tu te mets à fumer le cigare en balançant des vannes à deux francs.

Après, on prend notre caisse à outils et on va bricoler des trucs pour que ça aille plus vite, pour que ça soit plus résistant, hop hop un p'tit bout de tôle par là, hop hop on refait l'avant avec un bélier, et voilà, on a un char d'assaut fait maison et qui en fera trembler plus d'un...

CA, c'est l'Agence tous risques, tout le monde connaît. Bah, `kqueue(2)`/`kevent(2)`, c'est pareil !

De quoi on va parler ? Un petit peu de *socket programming* mon bon monsieur, de comment on peut faire des choses belles, sexy, brillantes, *fluffy* (TOMATE !!!) et qui sentent bon, tout en restant un bon poilu.

On va parler (un tout p'tit peu) de `select(2)`/`poll(2)` (qui est le standard POSIX) et de `kqueue(2)` qui est spécifique à BSD, mais offre beaucoup plus que ce que `poll(2)` ne peut offrir.

Tous les systèmes *IX massivement déployés offrent des sous-systèmes équivalents, par exemple Linux avec `epoll` ou Solaris avec `/dev/poll` et enfin BSD avec `kqueue(2)`/`kevent(2)`.

Comme BSD, c'est BIEN, on va se concentrer sur `kqueue(2)`/`kevent(2)`.

Ami lecteur, des connaissances en programmation réseau de base te seront nécessaires, au moins avoir fait un serveur TCP qui gère quelques clients ou quelque chose du genre.

FILE-MOI LA TAULE LOOPING!!!

Si vous avez déjà bricolé, (ou si vous avez modifié des bagnoles comme Barracuda), vous connaissez certainement `select(2)`/`poll(2)`, appels bien pratiques qui vous permettent de multiplexer des I/O (*file descriptors*) en fonction d'événements bien définis :

- données à ECRIRE sur un FD (POLLOUT) ;
- données à LIRE sur un FD (POLLIN) ;
- données prioritaires à LIRE sur un FD (POLLPRI) ;
- etc.

Voilà comment c'est défini (on va prendre `poll(2)`, c'est le moins courant des deux) :

```
int
poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

`poll(2)` va parcourir un tableau de `struct pollfd` :

```
struct pollfd {
    int    fd;       /* file descriptor */
    short  events;   /* events to look for */
    short  revents;  /* events returned */
};
```

dont vous lui fournissez la taille (nfds). Il va alors vous dire :

« hep, hep, y a un truc qui se passe parmi les événements que t'as définis ».

Alors après, on va dans son gentil tableau, on le parcourt entièrement et on vérifie chacun des éléments jusqu'à trouver celui dont `poll(2)` nous parle (`poll(2)` est bloquant).

C'est génial, il ne nous reste plus qu'à lire les données qui arrivent sur notre file descriptor et à traiter l'info en fonction de ce pour quoi votre serveur est fait.

Je vais filer un petit exemple pour montrer comment ça marche. Mon exemple est un mignon serveur qui écoute sur un port TCP et attend qu'un client se connecte. Il rajoute le file descriptor associé au client dans le tableau de `struct pollfd` (fonction `push_fd()`), comme ça, hop, le nouveau client est pris en compte par `poll(2)` et donc pas besoin de *forker* ou blah je ne sais pas quoi. On multiplexe les N clients tranquillement et, plus il y a de clients, plus `select(2)`/`poll(2)` risquent d'être lents, avec 5 ou 6, pas de souci, mais avec 5000, c'est plus la même histoire, n'est-ce pas ?

Bref, plus simple, jette donc un coup d'œil au code de notre `gruikpoll.c` :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include <netinet/in.h>

#include <sys/types.h>
#include <sys/socket.h>

#include <poll.h>
```

```
#define TCP_PORT 3133
#define TCP_BACKLOG 5
#define TIMEOUT 5000

typedef struct client_connection {
  int fd;                       /* client file descriptor */
  struct sockaddr_in rside;     /* remote side */
  unsigned int rside_len;       /* remote side struct len */
} cl_t;

void push_fd(struct pollfd **fds, int *nfds, int fd,
             short events)
{
  fprintf(stderr,
          "adding 1 socket to the pool (nfds: %d)!\n",
          *nfds);
  /* no socket pushed */
  if ((!*fds) && (*nfds == 0)) {
    (*fds) =
        (struct pollfd *) calloc(1, sizeof (struct pollfd));
    if (!(*fds)) {
      perror("calloc()");
      exit(1);
    }
    (*fds)[(*nfds)].fd = fd;
    (*fds)[(*nfds)].events = events;
    (*nfds)++;
  } else {
    (*fds) =
        (struct pollfd *) realloc((*fds),
                                  (((*nfds) +
                                    1) *
                                   sizeof (struct pollfd)));
    (*fds)[(*nfds)].fd = fd;
    (*fds)[(*nfds)].events = events;
    (*nfds)++;
  }
  return;
}

int main(int argc, char **argv)
{
  int rc, sd, fd, i, rfd;
  int yrc;
  struct sockaddr_in me;
  struct sockaddr_in remote;
  socklen_t remote_len;
  struct pollfd *fds = NULL;
  unsigned int nfds = 0;
  char buffer[512];

  fd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (fd < 0) {
    perror("socket(PF_INET, SOCKET_STREAM, TCP)");
    exit(1);
  }

  me.sin_family = PF_INET;
  me.sin_port = htons(TCP_PORT);
  me.sin_addr.s_addr = INADDR_ANY;

  rc = bind(fd, (struct sockaddr *) &me,
            sizeof (struct sockaddr));
  if (rc < 0) {
    perror("bind()");
    exit(1);
  }

  rc = listen(fd, TCP_BACKLOG);
  if (rc < 0) {
    perror("listen()");
    exit(1);
  }

  /* push the accept socket */
  push_fd(&fds, &nfds, fd, POLLIN);
  fprintf(stderr, "listening on port %d\n", TCP_PORT);
  /* fd # 0 is the accept() socket  :) */
  while (1) {
    rc = poll(fds, nfds, TIMEOUT);
    switch (rc) {
    case -1:
      perror("poll()");
      exit(1);
      break;
    case 0:
      fprintf(stderr, "timeout\n");
      break;
    default:
      for (i = 0; i < nfds; i++) {
        if (fds[i].revents & POLLIN) {
          /* the accept socket */
          if (i == 0) {
            /* accept case */
            rfd =
                accept(fds[i].fd,
                       (struct sockaddr *) &remote,
                       &remote_len);
            fprintf(stderr, "accepted new client from %s\n",
                    inet_ntoa(remote.sin_addr));
            push_fd(&fds, &nfds, rfd, POLLIN);
            break;
          } else {
            memset(buffer, 0, sizeof (buffer));
            if (getpeername
                (fds[i].fd, (struct sockaddr *) &remote,
                 &remote_len) < 0) {
              perror("getpeername()");
              exit(1);
            }
            yrc = read(fds[i].fd, buffer, sizeof (buffer));
            if (yrc == 0) {
              fprintf(stderr,
                      "DISCONNECT %d @ %s (KILLING ALL TOO "
                      "LAZY AND TOO ANNOYING TO DO IT "
                      "THE CORRECT WAY)\n",
                      fds[i].fd,
                      inet_ntoa(remote.sin_addr));
              close(fds[i].fd);
              exit(1);
            }
            if (buffer[yrc - 2] == 0x0d)
            /* we remove the \x0d from telnet client */
              buffer[yrc - 2] = '\x00';
            if (buffer[yrc - 1] == 0x0a)
            /* we remove the \x0a sent by telnet client */
              buffer[yrc - 1] = '\x00';
            fprintf(stderr,
                    "client[%d @ %s] read[%d bytes]: '%s'\n",
                    fds[i].fd, inet_ntoa(remote.sin_addr),
                    yrc, buffer);
          }
        }
      }
      break;
    }
  }

  return 0;
}
```

Voilà ce que ça donne lorsqu'on lance le serveur :

```
rival@kamehouse:~/dev/soul/linuxmag/avril_kqueue_bsd $ ./poll
adding 1 socket to the pool (nfds: 0)!
listening on port 3133
timeout
timeout
accepted new client from 127.0.0.1
adding 1 socket to the pool (nfds: 1)!
client[4 @ 127.0.0.1] read[8 bytes]: 'prout'
client[4 @ 127.0.0.1] read[6 bytes]: 'toto'
client[4 @ 127.0.0.1] read[6 bytes]: 'fuck'
timeout
accepted new client from 127.0.0.1
adding 1 socket to the pool (nfds: 2)!
client[5 @ 127.0.0.1] read[8 bytes]: 'dslkds'
client[5 @ 127.0.0.1] read[6 bytes]: 'ldks'
timeout
```

Et on connecte un client en envoyant quelques données :

```
rival@kamehouse:~ $ telnet localhost 3133
Trying ::1...
telnet: connect to address ::1: Connection refused
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
prout
toto
fuck
```

On connecte un second client et, pareil, on envoie un peu de données :

```
rival@kamehouse:~ $ telnet 127.0.0.1 3133
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
dslkds
ldks
```

Il faut aussi gérer la déconnexion du client en enlevant gentiment du tableau de `struct pollfd` le `fd` qui était associé au client qui vient de se déconnecter. Je n'ai pas mis le code (relou+feign et j'avais déjà fait une pile toute simple pour rajouter les clients), ça se fait avec des listes chaînées, des tables de *hash* sur l'IP, etc. Y a mille et une manière de le faire, mais, l'important, c'est à VOUS de le faire.

Voilà comment c'était fait pendant ces dernières années, jusqu'à l'arrivée de `kqueue(2)`/`kevent(2)`.

« Ce poteau électrique fera très bien l'affaire comme bélier sur la camionnette », (FUTE, de l'Agence tous risques).

ON FAIT QUOI MAINTENANT AU 21E SIÈCLE !?

Avec `kqueue(2)`/`kevent(2)`, la vie est beaucoup, beaucoup plus simple, tellement plus simple que c'est un bonheur de travailler avec et plus besoin de faire du *multithread* à tout va.

« `kqueue(2)`/`kevent(2)` nous ont toujours sorti d'affaire », dixit Barracuda.

`Kqueue(2)`/`kevent(2)`, c'est une queue d'événements. En gros, on crée une queue (tout se passe dans le kernel) et on lui rajoute ou on lui enlève des événements. Ce qu'il a de terrible, c'est qu'il n'est pas nécessaire de connaître la queue, ni de parcourir un tableau, ni rien : kevent va gentiment ne retourner que le nombre d'événements à gérer que l'on veut dans une structure toute prête qui contiendra les infos qu'il faut pour continuer sans se prendre la tête.

Genre après le `kevent(2)`, t'as direct le file descriptor où lire/écrire, la taille des données dans le *socket buffer*. Tu peux même associer des données à toi avec ces événements. De quoi attacher tout ça à une *structure context* par client connecté. Dans notre exemple, je balade l'adresse du client avec.

Voilà `kqueue(2)`/`kevent(2)` :

```
int
kqueue(void);

int
kevent(int kq, const struct kevent *changelist, size_t nchanges,
       struct kevent *eventlist, size_t nevents,
       const struct timespec *timeout);

EV_SET(&kev, ident, filter, flags, fflags, data, udata);
```

La structure pour définir un événement est la suivante :

```
struct kevent {
        uintptr_t ident;        /* identifier for this event */
        uint32_t  filter;       /* filter for event */
        uint32_t  flags;        /* action flags for kqueue */
        uint32_t  fflags;       /* filter flag value */
        int64_t   data;         /* filter data value */
        intptr_t  udata;        /* opaque user data identifier */
};
```

Chacun de ces éléments peut avoir une signification différente en fonction du type d'événement défini ou reçu. Oui, c'est la même structure qui sert à recevoir et à définir les événements qui sont remontés par `kqueue(2)`.

Alors, on reprend notre exemple de serveur TCP de tout à l'heure et on lui fait manger un bol de `kqueue(2)`/`kevent(2)`.

Ca donne ça (merci Looping pour le code) dans `gruikqueue.c` :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
```

```
#include <netinet/in.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/event.h>

#define TCP_PORT 3133
#define TCP_BACKLOG 1500
#define TIMEOUT 5000

int main(int argc, char **argv)
{
  int rc, sd, fd, i, rfd;
  int yrc;
  struct sockaddr_in me;
  struct sockaddr_in *remote = NULL;
  socklen_t remote_len = sizeof (struct sockaddr_in);
  int kq;
  struct kevent ke[1500];       /* handle 1500 events */
  struct kevent re;             /* one event a time */
  char buffer[512];

  fd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (fd < 0) {
    perror("socket(PF_INET, SOCKET_STREAM, TCP)");
    exit(1);
  }

  /* cleanup */
  memset(ke, 0, sizeof (ke));
  memset(&me, 0, sizeof (me));

  /* me myself and i */
  me.sin_family = PF_INET;
  me.sin_port = htons(TCP_PORT);
  me.sin_addr.s_addr = INADDR_ANY;

  /* on bind un port TCP tout simple */
  rc = bind(fd, (struct sockaddr *) &me,
            sizeof (struct sockaddr));
  if (rc < 0) {
    perror("bind()");
    exit(1);
  }

  /* c'est parti mon kiki!!! */
  rc = listen(fd, TCP_BACKLOG);
  if (rc < 0) {
    perror("listen()");
    exit(1);
  }

  fprintf(stderr, "listening on port %d\n", TCP_PORT);

  /* cree la queue d'events */
  kq = kqueue();
  if (kq < 0) {
    perror("kqueue()");
    exit(1);
  }

  /*
   *  sd (# 0) is the accept() socket  :)
   *  on cree le premier event de la queue
   *  qui sera l'event d'acceptation d'un nouveau client TCP
   */
  EV_SET(ke, fd, EVFILT_READ, EV_ADD, 0, TCP_BACKLOG, 0);
  rc = kevent(kq, ke, 1, NULL, 0, NULL);
  if (rc < 0) {
    perror("kevent(init)");
    exit(1);
  }
  while (1) {
    /*
     * allez on POLL!!!!
     */
    rc = kevent(kq, NULL, 0, &re, 1, NULL);
    switch (rc) {
    case -1:
      /*
       * casse kqueue, casse!!
       */
      perror("kevent()");
      exit(1);
      break;
    case 0:
      /*
       * on utilise pas mais on mets le case pour le
       * pendant avec poll()
       */
      fprintf(stderr, "timeout\n");
      break;
    default:
      /*
       * c'est la socket qui accept() les nouveaux
       * clients!!!! ??!?!?
       */
      if (re.ident == fd) {
        remote =
            (struct sockaddr_in *) calloc(1,
                                          sizeof (struct
                                                  sockaddr_in));

        rfd =
            accept(re.ident, (struct sockaddr *) remote,
                   &remote_len);
        if (rfd < 0) {
          perror("accept()");
          exit(1);
        }
        fprintf(stderr, "accepted new client from %s\n",
                inet_ntoa(remote->sin_addr));
        /*
         * on rajoute un event pour ce client avec son addresse
         * dans les user data
         */
        memset(&re, 0, sizeof (re));
        EV_SET(&re, rfd, EVFILT_READ, EV_ADD, 0, 0,
               (intptr_t) remote);
        rc = kevent(kq, &re, 1, NULL, 0, NULL);
        break;
      } else {
        memset(buffer, 0, sizeof (buffer));
        remote = (struct sockaddr_in *) re.udata;
        fprintf(stderr, "client sent %d bytes\n", re.data);
        if (re.flags & EV_EOF) {
          /* not mandatory */
          EV_SET(&re, re.ident, EVFILT_READ, EV_DELETE, 0,
                 0, (intptr_t) NULL);
          kevent(kq, &re, 1, NULL, 0, NULL);
          fprintf(stderr, "DISCONNECT %d @ %s\n", re.ident,
                  inet_ntoa(remote->sin_addr));
          close((int) re.ident);
          free(remote);
          break;
        }
```

```
        yrc = read(re.ident, buffer, re.data);
        if (buffer[yrc - 2] == 0x0d)
        /* we remove the \x0d from telnet client */
          buffer[yrc - 2] = '\x00';
        if (buffer[yrc - 1] == 0x0a)
        /* we remove the \x0a sent by telnet client */
          buffer[yrc - 1] = '\x00';
        fprintf(stderr,
                "client[%d @ %s] read[%d bytes]: '%s'\n",
                re.ident, inet_ntoa(remote->sin_addr), yrc,
                buffer);
      }
      break;
    }
  }
  return 0;
}
```

Décortiquons un peu :

```
[...]
/* cree la queue d'events */
kq = kqueue();
if (kq < 0)
{
        perror("kqueue()");
        exit(1);
}

/*
 *  sd (# 0) is the accept() socket  :)
 *  on cree le premier event de la queue
 *  qui sera l'event d'acceptation d'un nouveau client TCP
 */
EV_SET(ke, fd, EVFILT_READ, EV_ADD, 0, TCP_BACKLOG, 0);
rc = kevent(kq, ke, 1, NULL, 0, NULL);
if (rc < 0)
{
        perror("kevent(init)");
        exit(1);
}
[...]
```

On initialise la queue d'événements (`kq = kqueue()`). On lui décrit directement un premier événement (`EV_SET()`) qui est « s'il y a quelque chose à lire sur le FD `accept()` préviens-moi ! »... puis, on l'ajoute à la queue (`kevent(...)`), simple non ?

Ensuite, on *poll* gentiment les événements en précisant qu'on ne veut en remonter qu'un seul à la fois pour le stocker dans la `struct kevent re` et pas remonter des pools entiers d'événements (ce qui est aussi possible avec `kqueue(2)`/`kevent(2)`) :

```
[...]
/*
 * allez on POLL!!!!
 */
rc = kevent(kq, NULL, 0, &re, 1, NULL);
switch (rc) {
[...]
```

Le *switch* avec son « *default* » *case* définit donc le moment où `kevent(2)` nous dit :« hep, hep, j'ai un truc que t'as défini qui est arrivé, je te l'ai mis dans ta struct `re` ».

Ohhhhhhh, miracle, la struct `re` est déjà remplie, avec comme contenu :

- `re.ident` == File descriptor à lire ou écrire (lire dans notre cas) ;
- `re.data` == Taille des données dans le socket buffer ; on connaît la taille avant de `read(2)` (!) ;
- `re.udata` == Les données utilisateur qu'on aura disposées au moment de créer l'événement et qui seront de nouveau à notre disposition.

ClaAAaaaaAAaasse non ? Si.

Bon on reprend, on a un premier cas, celui où `accept(2)` reçoit un nouveau client :

```
[...]
remote = (struct sockaddr_in *)
  calloc(1, sizeof(struct sockaddr_in));
rfd = accept(re.ident,
  (struct sockaddr *)remote, &remote_len);
if (rfd < 0)
{
      perror("accept()");
      exit(1);
}
fprintf(stderr, "accepted new client from %s\n",
  inet_ntoa(remote->sin_addr));
/*
 * on rajoute un event pour ce client avec son adresse
 * dans les user data
 */
memset(&re, 0, sizeof(re));
EV_SET(&re, rfd, EVFILT_READ, EV_ADD, 0, 0, (intptr_t)remote);
rc = kevent(kq, &re, 1, NULL, 0, NULL);
[...]
```

On récupère l'adresse de notre client dans `accept(2)`, puis le nouveau file descriptor et encore un MIRACLE. On rajoute gentiment notre nouveau file descriptor avec son lot de données à lui (`struct sockaddr_in * remote`) fraîchement alloué dans la queue d'événements en précisant que s'il y a quelque chose à lire, dis-le moi !

Alors précipitons-nous dans la partie où un client nous parle :

(Il est tout attentif là Barracuda, ahhh ça te TROUE LE `KQUEUE(2)` HEIN !!!)

```
[...]
memset(buffer, 0, sizeof(buffer));
remote = (struct sockaddr_in *)re.udata;
fprintf(stderr, "client sent %d bytes\n", re.data);
if (re.flags & EV_EOF)
```

```
{
        /* not mandatory */
        EV_SET(&re, re.ident, EVFILT_READ,
            EV_DELETE, 0, 0, (intptr_t)NULL);
        kevent(kq, &re, 1, NULL, 0, NULL);
        fprintf(stderr, "DISCONNECT %d @ %s\n",
             re.ident, inet_ntoa(remote->sin_addr));
        close((int)re.ident);
        free(remote);
        break;
}
yrc = read(re.ident, buffer, re.data);
[...]
```

Là, idem, simplicité, félicité, clarté, c'est beau.

On connaît la taille des données à lire et en attente dans le noyau (`re.data`). On peut aussi vérifier que ce n'est pas simplement une déconnexion du client (`re.flags & EV_EOF`), auquel cas on efface l'événement associé au client de la queue (encore plus simple, il suffit de fermer le descriptor et il sera automatiquement éliminé de la queue cf. `kqueue(2)`), on ferme le file descriptor, on libère la mémoire associée aux données du client, hop c'est tout bon).

Sinon, on `read(2)` simplement la taille (`re.data`) dans notre buffer (alloué par nos soins ou fixe si inférieur à une certaine taille) le file descriptor (`re.ident`) que l'événement nous a retourné si poliment.

Résultat sur le serveur :

```
rival@kamehouse:~/dev/soul/linuxmag/avril_kqueue_bsd $ ./kqueue
listening on port 3133
accepted new client from 127.0.0.1
client sent 7 bytes
client[5 @ 127.0.0.1] read[7 bytes]: 'prout'
accepted new client from 127.0.0.1
client sent 7 bytes
client[6 @ 127.0.0.1] read[7 bytes]: 'totot'
client sent 7 bytes
client[6 @ 127.0.0.1] read[7 bytes]: 'titit'
client sent 6 bytes
client[6 @ 127.0.0.1] read[6 bytes]: 'tata'
client sent 7 bytes
client[5 @ 127.0.0.1] read[7 bytes]: 'gitan'
client sent 8 bytes
client[5 @ 127.0.0.1] read[8 bytes]: 'tgitan'
```

client #1 :

```
rival@kamehouse:~ $ telnet localhost 3133
Trying ::1...
telnet: connect to address ::1: Connection refused
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
prout
gitan
tgitan
```

client #2 :

```
rival@kamehouse:~ $ telnet 127.0.0.1 3133
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
totot
titit
tata
```

On n'a rien à faire d'autre, rien à gérer, pas de liste, rien, tout est fait en interne dans le noyau. En plus de nous faire gagner en simplicité, il faut aussi parler de la scalabilité de `kqueue(1)` qui arrive à faire (si mes souvenirs sont bons, mes pilules !) du O(1) quel que soit le nombre d'événements concurrents, ce qui revient à dire que, quel que soit le nombre de clients connectés, les perfs seront les mêmes, ninaianiaaaarrrrr NIArrrrkrk NIARRKRKKKKK!!!!

CONCLUSION

Dans le reste du sexy de `kqueue(2)`/`kevent(2)`, je vous invite à lire la page de manuel de celui-ci : en plus d'y gérer les sockets, les *vnodes*, il est également capable de mater l'état des processus, si ceux-ci `fork(2)`, si un processus fait appel à `execve()`, si un fichier est effacé ou modifié en *real-time*, les filtres sont assez classieux et il y a de quoi s'amuser :).

Voila, c'est aussi ça l'Agence tous risques, c'est pas que des boulons et des tournevis, mais c'est aussi du code poilu, du code *full girly*, du bon code, du code qui l'est bon comme la saucisse de George !! Hmmm la saucisse de George...

RÉFÉRENCES

- `kqueue(2)`/`kevent(2)` manpages
- `poll(2)`/`select(2)` manpages
- Kqueue, **http://people.freebsd.org/~jlemon/papers/kqueue.pdf**
- C10K problem, **http://www.kegel.com/c10k.html**
- JMMV kqueue example, **http://julipedia.blogspot.com/2004/10/example-of-kqueue.html**

REMERCIEMENTS

- tomate (patience & viande hachée !)
- GCU/Ma maison, mon jardin, là où j'ai grandi, le foyer Sonnacotra des coders.
- le CICR pour la soupe chaude.