Secure SHell

Faites communiquer vos ordinateurs !

Romain Vimont (®om)

Résumé

Ce document est un complément détaillé à l'atelier *SSH* lors de l'ubuntu party des 7 et 8 juin 2008. Cet atelier avait pour but de présenter brièvement ce que permet de faire *SSH*, et comment mettre en place des choses intéressantes lorsque l'on possède plusieurs ordinateurs.

Prérequis :
- savoir taper des commandes dans un shell ;
- connaître un mimimum l'adressage IP ;
- savoir utiliser le gestionnaire de paquets.

Si vous avez des questions ou des remarques, vous pouvez me contacter par *Jabber* à `rom1v@jabber.fr`.

Table des matières

Chapitre 1

Présentation

SSH, pour *Secure SHell*, désigne à la fois un protocole de communication sécurisé et le logiciel qui permet d'utiliser ce protocole. Son utilisation principale, la plus basique, est l'accès à un terminal distant, dans lequel on peut taper des commandes.

1.1 Installation

Pour faire communiquer deux machines, il faut deux programmes :
– le serveur, qui écoute un port (par défaut, le 22) en attente de connexion de clients ;
– le client.
Sous **Ubuntu 8.04**, le client `ssh` est préinstallé.
Sur la ou les machines serveurs, il faut installer le paquet `openssh-server` :

```
sudo apt-get install openssh-server
```

Pour simplifier, nous utiliserons les paramètres par défaut (en particulier nous utiliserons le port 22[1]).
Il faut ensuite démarrer le serveur :

```
sudo /etc/init.d/ssh start
```

1.2 Utilisation

Supposons que la machine serveur ait pour IP locale `192.168.0.1`, et que l'on veuille s'y connecter à partir d'une machine cliente en tant que `monlogin` (`monlogin` doit être un compte *unix* existant sur la machine serveur) :

```
ssh monlogin@192.168.0.1
```

Si sur la machine cliente on est déjà identifié en tant que `monlogin`, on peut omettre le login :

```
ssh 192.168.0.1
```

Plusieurs options sont disponibles, pour choisir le port d'écoute du serveur. Une option qui peut parfois être utile, `-C`, concerne la compression *gzip* de la connexion :

```
ssh -C 192.168.0.1
```

Lors de la première connexion, le client demande s'il peut faire confiance à ce serveur (en indiquant le fingerprint de sa clé host). Il faut simplement écrire `yes` si c'est bien celle du serveur.
Pour connaître le fingerprint de la clé host du serveur :

[1] Pour modifier le port, il faut éditer le fichier `/etc/ssh/sshd_config` et remplacer ou ajouter une ligne `Port 1234` par exemple. Ceci impliquera de spécifier un paramètre supplémentaire dans les programmes clients.

```
ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key.pub
```

Le mot de passe du compte `monlogin` doit ensuite être renseigné, et s'il est valide, on a un terminal sur la machine serveur à partir de la machine cliente, et on peut y exécuter des commandes.

Si la machine serveur n'est pas sur le réseau local de la machine client, il faut indiquer l'IP (ou l'alias DNS) par laquelle on veut y accéder (par exemple celle de la **MachinBox**, qui fait routeur), et configurer la **MachinBox** pour qu'elle redirige le port 22 (ou autre) sur la machine qui a un serveur SSH.

Il est alors possible d'exécuter des commandes (`ls` par exemple).

Par contre, pour des programmes qui ont une interface graphique, cela ne fonctionne pas directement. Si l'on tente de lancer `gedit` en SSH par exemple, voici ce que l'on obtient :

```
$ gedit
cannot open display:
```

Pour qu'il se lance sur le serveur, il faut changer la valeur de la variable d'environnement `DISPLAY` :

```
$ export DISPLAY=:0
$ gedit
```

En tant que client, on ne voit pas la fenêtre, elle est simplement lancée sur le serveur. Cela peut être pratique dans certains cas.

1.3 Astuce

Si vos machines ont des IP fixes sur le réseau local, plutôt que de les appeler par leur IP (192.168.0.1, 192.168.0.2...), il est possible de leur donner des petits noms.

Pour cela, sur chaque machine cliente, éditer le fichier `/etc/hosts` :

```
gksudo gedit /etc/hosts
```

et rajoutez des alias. Par exemple :

```
192.168.0.1 tux
192.168.0.2 gnu
```

Pour accéder à `tux`, il suffira alors de taper :

```
ssh monlogin@tux
```

ou encore, si on utilise le même login sur les 2 machines :

```
ssh tux
```

Chapitre 2

Authentification par clés

Pour l'instant, nous nous sommes connectés à une machine distante, qui nous authentifiait par le mot de passe du compte avec lequel on voulait se connecter.

Cette authentification a pour avantage de n'avoir rien à configurer. Par contre, elle a plusieurs inconvénients :

1. À chaque connexion SSH, il faut retaper le mot de passe ;
2. Certaines fonctions peuvent nécessiter une authentification par clés (car pas de prompt) ;
3. Si quelqu'un connaît le mot de passe (qui, pour 90% est le même que leur mot de passe VNC ou MSN, euh, pardon, Jabber), il a accès à votre machine.

2.1 Principe

L'authentification par clés est similaire à un système de badges.

Tout d'abord, un client doit générer une paire de clés publique/privée. La clé publique doit être copiée sur chacun des serveurs qui doit faire confiance à ce client. La clé privée doit rester secrète (c'est ce qui prouve que c'est bien le client qui se connecte à une machine, et pas quelqu'un qui se fait passer pour lui). Tout ce qui sera signé par une clé sera vérifiable par l'autre, et une autre clé ne pourra pas produire la même signature.

Lorsque le client se connecte à un serveur sur lequel il est autorisé, le serveur lui dit "ok, j'ai bien ta clé publique dans ma liste d'autorisation, maintenant, prouve-moi que c'est bien toi". Le client lui envoie alors un message signé avec sa clé privée, si le serveur parvient à vérifier le message, c'est ok, le client est bien celui qu'il prétend être.

L'authentification est réciproque, le client, de la même manière, vérifie qu'il se connecte au serveur auquel il veut se connecter (même pour une authentification par mot de passe).

Toute la sécurité réside dans le fait qu'il est très difficile de deviner la clé privée à partir de la clé publique.

Pour éviter que le client se fasse "voler" sa clé privée par quelqu'un qui a accès à sa machine, celle-ci est chiffrée et protégée par une passphrase (un password, plus long).

Il est possible de désactiver l'authentification par mots de passe. Pour cela, il faut éditer le fichier `/etc/ssh/sshd_config` :

```
PasswordAuthentication no
```

Il peut cependant être préférable d'autoriser les deux types d'authentification (si vous voulez vous connecter en SSH à partir de chez un ami, il ne possèdera pas votre clé privée).

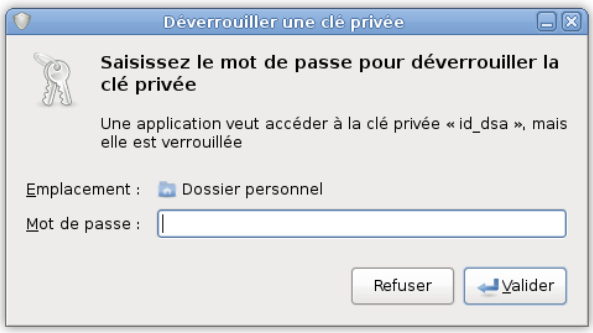

FIG. 2.1 – Demande de passphrase sous Gnome

2.2 Mise en œuvre

2.2.1 Génération des clés

Le serveur a, dès l'installation, généré une paire de clés pour la machine (même deux, une RSA et une DSA, mais peu importe).

Ce qu'il faut, c'est générer une paire de clés pour chaque client. Pour cela, il suffit d'exécuter :

```
ssh-keygen
```

puis d'accepter le fichier proposé par défaut pour stocker la clé.

La passphrase est ensuite demandée pour chiffrer la clé privée. Choisissez ce que vous voulez, mais quelque chose de pas trop court (mais pas 150 caractères non plus, il faudra la taper de temps en temps).

2.2.2 Autorisation de la clé sur un serveur

La liste des clés publiques autorisées est stockée dans le fichier `~/.ssh/authorized_keys` (ce fichier peut ne pas exister si aucune clé n'est autorisée, il faut alors le créer).

La clé publique du client est stockée dans `~/.ssh/id_rsa.pub` ou `~/.ssh/id_dsa.pub`. Il faut alors rajouter le contenu de ce fichier dans le fichier d'autorisation du serveur.

Pour cela, plusieurs solutions : clé usb, recopie à la main (déconseillé quand même !)... La plus simple consiste à copier la clé sur le serveur en utilisant SSH (avec authentification par mot de passe, évidemment). Pour cela, il suffit de taper cette commande :

```
ssh login@serveur "echo ‘cat ~/.ssh/id_rsa.pub‘ >> .ssh/authorized_keys"
```

Exécutez ces commandes à partir du client pour tous les serveurs sur lesquels vous voulez vous authentifier par clés.

2.3 Utilisation

Ensuite, vous pouvez vous connecter au serveur de la même manière :

```
ssh monlogin@monserveur
```

Il vous demandera la passphrase pour déchiffrer la clé privée la première fois (voir Fig. 2.1), et ne le redemandera plus pour toute la durée de la session et pour tous les serveurs ! Si l'on considère que votre clé privée est votre badge, vous avez prouvé que vous êtes bien celui à qui appartient le badge (en tapant votre passphrase), et votre badge vous donne accès à tous les serveurs sur lequel il est autorisé.

Chapitre 3

Transfert de fichiers sécurisé

Il est possible d'accéder aux fichiers d'une machine qui possède un serveur SSH, avec les mêmes droits d'accès que l'utilisateur avec lequel on est connecté.

3.1 Clients

3.1.1 SCP

Le client "officiel" pour partager des fichiers est `scp`, en ligne de commande.
Pour copier du client vers le serveur :

```
scp /un/dossier/monfichier monlogin@monserveur:/le/dossier/distant/voulu
```

Pour copier du serveur vers le client :

```
scp monlogin@monserveur:/le/dossier/distant/voulu/monfichier /un/dossier
```

Il est également possible, comme avec `cp` en local, de copier récursivemement un répertoire, ou plusieurs fichiers définis par un motif.
Contrairement à certains clients graphiques, il a l'avantage de permettre l'authentification par clés, comme `ssh`.

3.1.2 SFTP

Le client `sftp` en ligne de commande permet de naviguer dans l'arborescence à la manière d'un ftp (avec la commande `ftp`).

3.1.3 Nautilus

Le navigateur de fichiers de *Gnome* peut parcourir l'arborescence de fichiers lors d'une connexion SSH. Pour cela, il suffit de taper dans la barre d'adresse l'url suivante :

```
sftp://monlogin@monserveur
```

Les données d'authentification vous seront alors demandées.
Nautilus permet plus que simplement parcourir les fichiers et dossiers partagés en SSH, cela sera expliqué dans le chapitre suivant.

3.1.4 Konqueror

La navigateur de fichiers de *KDE* permet également de parcourir l'arborescence, il suffit de taper dans la barre d'adresse l'url suivante :

```
fish://monlogin@monserveur
```

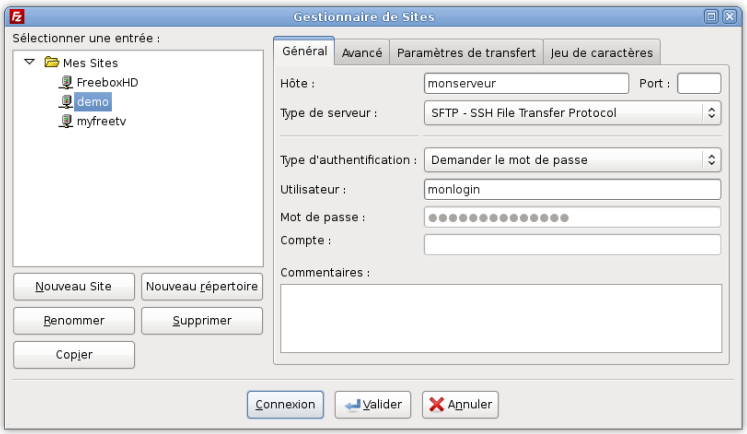

FIG. 3.1 – Gestionnaire de sites de FileZilla

3.1.5 FileZilla

Le célèbre client FTP permet également de se connecter en SSH. Pour cela, il faut créer un nouveau site dans le gestionnaire de site, et utiliser comme type de serveur `SFTP - SSH File Transfer Protocol`, comme indiqué à la Fig. 3.1.

Ce client est également disponible sous Windows.

3.2 MySecureShell

L'accès aux fichiers de son compte SSH à distance, c'est très pratique. Mais on ne peut pas donner un accès à quelqu'un sans qu'il ait également un accès shell, et un accès à tous les fichiers du système (du moins ceux qui n'ont pas des droits de la forme `xy0`). Il peut donc lire vos fichiers, et exécuter des commandes, lancer des programmes, etc...

C'est là qu'intervient **MySecureShell**. Il va permettre de créer des utilisateurs qui n'ont pas de shell, et qui ne peuvent pas sortir de leur home.

3.2.1 Installation

Malheureusement, il n'est pas présent dans les dépôts par défaut d'**Ubuntu 8.04**. Actuellement en version 1.0, il est possible de télécharger le **.deb** sur sourceforge, ou encore de rajouter une "source de logiciels" :

```
deb http://mysecureshell.free.fr/repository/debian testing main
```

Il suffit alors de mettre à jour les dépôts puis d'installer le paquet `mysecureshell` :

```
sudo apt-get install mysecureshell
```

3.2.2 Création d'un utilisateur

Créons maintenant un utilisateur `demo` :

```
sudo adduser --home /var/sftp --shell /bin/MySecureShell sftpuser
```

Entrez ensuite son mot de passe (qui sera nécessaire à la connexion).

Cet utilisateur sera enfermé dans son home (ici `/var/sftp`), et n'aura pas la possibilité d'exécuter des commandes shell.

Il est possible de se connecter pour transférer des fichiers, avec les différents clients présentés précédemment.

Vous pouvez copier des fichiers dans son répertoire. Les liens symboliques ne suffisent pas pour donner un accès à un autre répertoire, cependant vous pouvez “monter” des répertoires dans son home :

```
sudo mount --bind /un/répertoire /var/sftp/lelien
```

```
sudo umount /var/sftp/lelien
```

Par défaut, ce répertoire a les droits de `sftpuser`, et non les vôtres. Faites donc bien attention à donner les bons droits aux fichiers partagés.

3.2.3 Configuration avancée

Des paramétrages peuvent être faits dans le fichier `/etc/ssh/sftp_config`, par exemple pour le choix du dossier racine, et la limitation du débit et du nombre d'utilisateurs.

Chapitre 4

Système de fichiers virtuel

Avoir accès aux fichiers pour les télécharger, c'est bien... Y avoir accès comme si c'étaient des fichiers locaux, c'est mieux !

4.1 Principe

Grâce à `mount`, on peut monter un répertoire local dans l'arborescence locale :

```
mount --bind olddir newdir
```

Le principe ici est de *monter* dans un répertoire **distant** dans l'arborescence locale.

4.2 Intérêt

Lors du parcours des fichiers et dossiers comme un simple FTP, vous ne pouvez que télécharger (ou envoyer) des fichiers. Pour regarder une vidéo accessible à distance par exemple, avec un "FTP-like", il faut télécharger la vidéo, puis la regarder (éventuellement il est possible regarder le début de la vidéo avant qu'elle ne soit terminée de télécharger). Pareil pour la musique.

Avec un système de fichiers virtuel, vous pouvez les manipuler comme des fichiers locaux. Ainsi, vous pouvez ouvrir votre vidéo directement avec votre lecteur habituel, vous déplacer dans la vidéo (et ça réagit "en direct", sans avoir à posséder l'intégralité de la vidéo). On peut donc faire de la pseudo-vidéo-à-la-demande[1], de la pseudo-musique-à-la-demande, ou encore accéder à ses documents, son album photo...

On peut donc imaginer, dans le cadre d'une utilisation domestique, avoir un serveur pour la maison qui contient les vidéos, la musique, les albums photos... et tous les membres de la famille qui accèdent à ces fichiers directement à partir de leur ordinateur.

Étant donné que ce système est bâti sur SSH, il est possible de faire tout cela de manière sécurisée sur internet. Cependant, cet avantage est à relativiser, car si cela fonctionne très bien avec les documents et la musique, les débits montant de votre connexion domestique ne permettent que rarement de voir une vidéo "en direct" avec ce mécanisme. Cela est surtout très utile en local.

4.3 Utilisation

4.3.1 sshfs

Il faut tout d'abord installer le paquet `sshfs` sur le poste client. :

[1] "pseudo" seulement, car il ne s'agit que d'amener le contenu du fichier là où il doit être lu, un "vrai" serveur de vidéo-à-la-demande gèrerait le transcodage, le changement de résolution vidéo en fonction du débit réseau ; ici, le SSH ne "sait" même pas qu'il manipule de la vidéo.

```
sudo apt-get install sshfs
```

Ensuite, pour monter un répertoire distant dans l'arborescence locale (le répertoire local doit exister) :

```
sshfs monlogin@monserveur:/mon/repertoire/distant /mon/repertoire/local
```

Vous pouvez alors naviguer dans `/mon/repertoire/local` dans tous les logiciels, qui le considèrent vraiment comme un répertoire local.

Ainsi, il suffit de monter son répertoire de photo distant (celui de **digiKam**[2] par exemple) dans un répertoire local, et de configurer son **digiKam** local pour utiliser ce répertoire comme album photo. Il est alors possible de visionner les photos, et même de mettre sa carte mémoire d'appareil photo en local pour en transférer le contenu, comme si les données étaient locales.

Pour démonter un répertoire (possible que si non utilisé) :

```
fusermount -u /mon/repertoire/local
```

L'utilisation d'**autofs**[3] pour **sshfs** posant quelques problèmes (connexion en root, clé non chiffrée), il peut être pratique de se créer des raccourcis pour monter et démonter les répertoires.

sshfs fonctionne quelque soit l'environnement de bureau (et même sans environnement de bureau).

Il a l'avantage de pouvoir monter n'importe quel répertoire distant dans n'importe quel répertoire local.

4.3.2 gvfs

Depuis *Gnome 2.22* (et donc **Ubuntu 8.04**), le parcours d'un serveur SSH provoque le montage de la racine (`/`) du serveur dans `~/.gvfs/sftp sur HOST`, où `HOST` est le nom du serveur (ip, dns ou alias hosts). Pour se connecter à un serveur, cliquez sur le menu *Raccourcis*, puis *Se connecter à un serveur...* (voir Fig. 4.1). Cet outil graphique s'occupe simplement d'écrire l'url qu'il faut dans **nautilus**.

Un raccourci est alors créé (sur le bureau et dans le menu *Raccourcis* du système et de **nautilus**). Pour démonter, il suffit, comme pour n'importe quel périphérique (clé usb, disque externe...), de cliquer droit, puis *Démonter le volume*.

gvfs apporte quelque chose de remarquable : ne pas taper une seule ligne de commande. C'est indéniablement un gros avantage pour **gvfs**. Et si vous avez le paquet `nautilus-open-terminal`, et que vous demandez à ouvrir un dossier distant dans une console, il ouvre un terminal où la connexion SSH est déjà ouverte, et dans le répertoire voulu.

Par contre, ceci ne fonctionne que sous *Gnome*, et on ne choisit pas le répertoire de montage. Pour configurer un gestionnaire de photos pour le faire pointer sur un répertoire distant, il peut donc être préférable d'utiliser **sshfs**.

Autre point en faveur de **sshfs**, **gvfs** est très récent, et a encore quelques petits problèmes. Par exemple, il arrive que **nautilus** se ferme purement et simplement, ou encore qu'il ne monte pas le serveur dans `~/.gvfs/sftp sur HOST` (et donc le seul lien accessible est `sftp://...`), ce qui empêche totalement la lecture de vidéos distantes avec **VLC** par exemple.

[2] **digiKam** est un gestionnaire de photos sous KDE.

[3] **autofs** permet de monter un répertoire que lorsque l'on demande à voir son contenu, et de le démonter automatiquement après un timeout ; il est très pratique et utilisé pour les partages **NFS**.

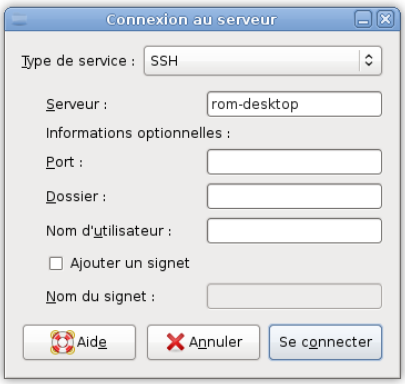

FIG. 4.1 – Connexion à un serveur sous Gnome

Chapitre 5

Affichage à distance

5.1 Principe

Nous avons vu qu'il était possible avec SSH d'exécuter des commandes. Or, certaines commandes lancent des applications graphiques. Comme nous l'avons vu à la page 4, il est possible de lancer l'application graphique totalement sur le serveur. Ici nous nous intéresserons à lancer l'application sur le serveur, mais avec l'affichage exporté sur le client.

5.2 Intérêt

À première vue, avoir un programme qui s'exécute à distance, avec l'affichage uniquement en local, ne présente que peu d'intérêt. Et pourtant...

Supposons que l'on possède un ordinateur fixe sur lequel est branché un ampli avec des enceintes, et que cet ordinateur possède toute notre collection de musique ; il a un peu le rôle de serveur. Par ailleurs, on possède un ordinateur portable, qui est notre ordinateur principal. On peut trouver utile de contrôler la musique de la maison directement à partir de son PC portable connecté en wifi... C'est ce que l'on peut faire en lançant le lecteur audio sur le PC fixe, mais avec l'affichage sur le PC portable.

Contrairement à l'utilisation de **VNC**, qui permet de visionner l'intégralité d'un écran d'ordinateur distant dans une fenêtre, avec SSH, le lecteur audio s'intègre parfaitement au PC client, il a même une icône dans le *systray*.

5.3 Utilisation

Pour se connecter en activant la redirection graphique, il suffit de rajouter le paramètre `-X` :

```
ssh monlogin@monserveur -XC
```

(on active également la compression, cela permet de gagner en réactivité)

On peut même lancer directement l'application distante, et en faire un raccourci, par exemple **Amarok**[1] :

```
ssh monlogin@monserveur -XC amarok
```

Une capture d'écran est disponible Fig. 5.1.

[1] **Amarok** est le lecteur audio de KDE.

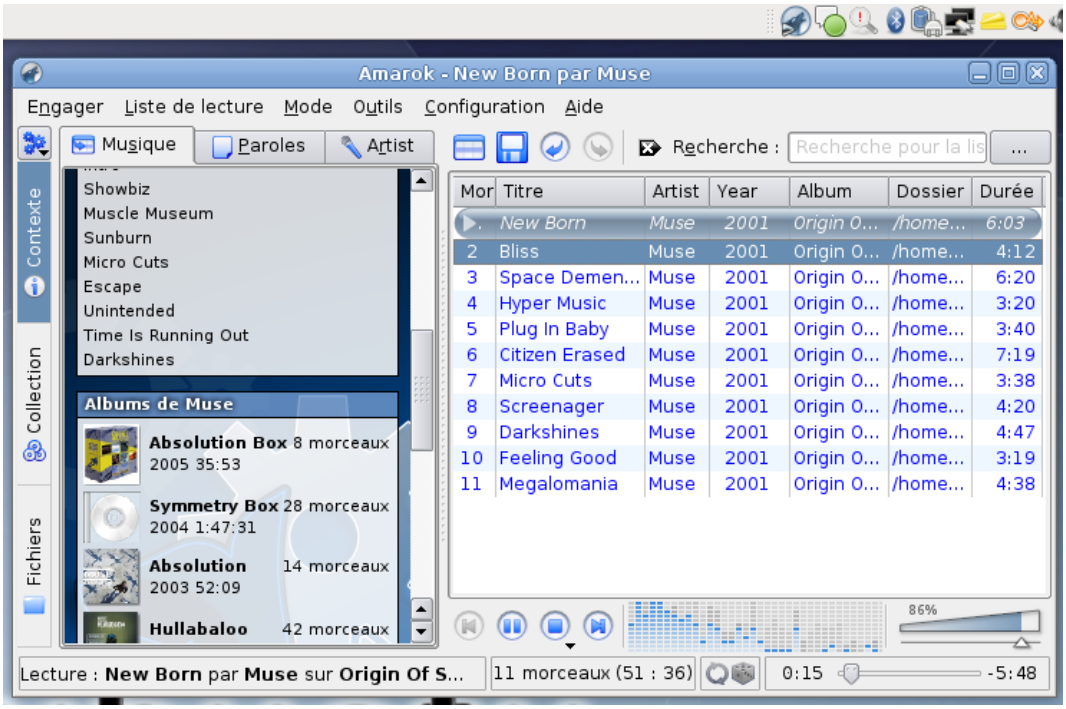

FIG. 5.1 – Amarok à distance

5.4 Plus généralement...

En fait, lorsque l'on veut écouter de la musique, il y a 3 entités :

DATA l'entité où se trouve la musique (en *Ogg Vorbis* par exemple) ;

SOUND l'entité où doit être lue la musique ;

CONTROL l'entité à partir de laquelle on veut contrôler la musique.

Le but du jeu est donc de "déplacer" la musique de **DATA** vers **SOUND**, où on lance le lecteur audio, et de "déplacer" l'affichage du lecteur audio sur **CONTROL**.

Pour "déplacer" la musique, il suffit d'utiliser un système de fichier virtuel (voir chapitre 4) : **SOUND** monte à distance le dossier audio se trouvant sur **DATA** dans son arborescence locale. Pour "déplacer" l'affichage du lecteur, il suffit de lancer le lecteur sur **SOUND** en SSH à partir de la machine **CONTROL**.

Chapitre 6

Redirection de ports

SSH permet également de faire de la redirection de ports.

6.1 Principe

Le principe consiste à rediriger :
– un port local vers un port de la machine distante ;
– un port de la machine distante vers un port local.
La redirection se fait à l'intérieur de la connexion SSH.

6.2 Intérêt

Il est ainsi possible, dans le premier cas, se connecter à VNC en passant dans la connexion SSH. Ainsi, la connexion VNC, qui n'est pas chiffrée, est sécurisée par la connexion SSH, et de plus seul le port SSH a besoin d'être ouvert sur le routeur (puisque le client n'utilise QUE la connexion SSH). Il est alors possible de fermer (de ne pas ouvrir) le port VNC sur le routeur domestique, et ainsi de désactiver le mot de passe VNC (l'authentification de SSH étant évidemment bien plus forte).

Dans le second cas, supposons que vous êtes derrière un routeur, que vous ne pouvez pas configurer (au travail par exemple), et vous avez justement besoin de lancer un serveur (*apache* par exemple) sur votre machine (PC portable par exemple) sur l'un de ces ports. Il est possible d'ouvrir une connexion SSH vers votre ordinateur fixe, qui lui, a un port ouvert utilisable, pour rediriger ce port vers votre serveur local. En utilisant l'adresse de votre PC fixe avec le port adéquat, l'utilisateur sera redirigé vers votre ordinateur portable, même si celui-ci est "caché" derrière un routeur.

6.3 Utilisation

Port local

Pour rediriger un port local :

```
ssh monlogin@monserveur -NL1234:localhost:5900
```

`1234` est le port local redirigé ; `localhost:5900` est l'adresse, à partir de `monserveur`, vers laquelle ce port est redirigé.

Ainsi, pour se connecter à VNC (par défaut le port 5900) sur `monserveur` en passant dans la connexion SSH, il suffit de se connecter à `localhost:1234`.

Port distant

Pour rediriger un port distant, il faut d'abord activer la redirection de ports sur le serveur. Pour cela, éditez `/etc/ssh/sshd_config`, et rajoutez :

```
GatewayPorts yes
```

puis redémarrez le serveur :

```
sudo /etc/init.d/ssh restart
```

Ensuite, pour activer la redirection :

```
ssh monlogin@monserveur -NR1234:localhost:5900
```

Ainsi, pour qu'un utilisateur se connecte à votre serveur VNC local, il suffira de se connecter à `monserveur:1234`.

Chapitre 7

Traverser un proxy avec une machine rebond

7.1 Objectifs

Il vous est peut-être déjà arrivé d'être derrière un proxy en entreprise, et évidemment, seuls quelques ports (sortants) sont ouverts, et certains sites sont bloqués.

Vous ne pouvez pas lire vos mails dans votre logiciel de courrier, vous n'avez pas accès à des sites qui sont bloqués (si le blocage est injustifié, il faut envoyer un mail à l'admin, etc...), vous n'avez pas accès à IRC (c'est pratique pour avoir de l'aide sur du développement)... Bref, vous êtes bloqués.

Ce chapitre va présenter comment passer à travers le proxy, et comment, avec une machine "rebond" (votre pc fixe chez vous par exemple), vous pouvez accéder à tous les sites, à irc, à votre messagerie instantanée...

7.2 Passer à travers le proxy

Le proxy filtre tout le trafic, et décide de ce qu'il doit accepter ou refuser. Mais bien sûr, il existe des connexions sécurisées (pour accéder à des sites sécurisés sur `https` par exemple, comme les sites bancaires), et par définition, il ne peut pas analyser ce traffic, car c'est chiffré !

Il suffit donc de se connecter à un serveur qui écoute sur le port 443 (port du `https`) pour le proxy vous laisse passer. C'est le cas par exemple avec *Jabber*, dans votre client jabber, vous pouvez spécifier d'utiliser SSL sur le port 443 (si votre serveur jabber le supporte), dans ce cas vous pouvez vous y connecter sans problème.

7.3 Mettre en place un tunnel SSH

Mais bon, c'est bien gentil, mais tous les serveurs n'ont pas un démon qui tourne sur le port 443 pour vous accueillir. Donc vous allez configurer un serveur (votre pc fixe chez vous par exemple) pour que lui, écoute sur le port 443 et vous obéisse.

7.3.1 Côté serveur

Pour cela, il faut modifier la configuration du serveur SSH pour qu'il écoute (aussi) sur le port 443. Éditez `/etc/ssh/sshd_config` et rajoutez (ou remplacez) :

```
Port 443
```

Alternativement, vous pouvez laisser 22, et configurer votre routeur domestique pour qu'il redirige le port 443 sur le port 22, ça permet d'utiliser le SSH en local sur le port 22, qui est le port par défaut.

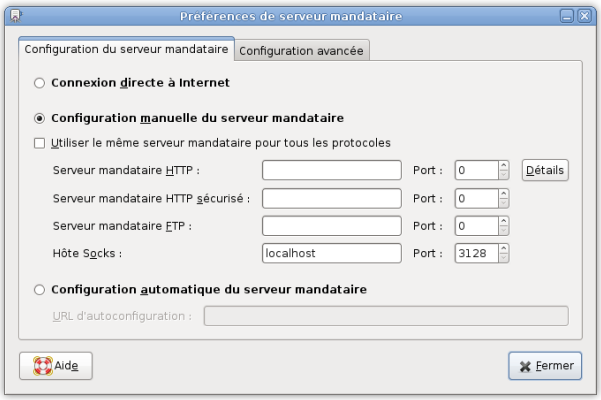

FIG. 7.1 – Réglages du serveur mandataire sous Gnome

Redémarrez le serveur SSH :

```
sudo /etc/init.d/ssh restart
```

Voilà c'est tout pour le serveur.

7.3.2 Côté client

Maintenant le client.
Installez `corkscrew` :

```
sudo apt-get install corkscrew
```

Cela vous permettra de passer une connexion SSH au dessus de `https`.
Supposons que le proxy que vous essayez de passer est : `proxy.entreprise.fr` sur le port 3128, et que votre PC fixe est accessible depuis l'extérieur par `moi.mondomaine.fr` (si vous n'en n'avez pas, utilisez votre adresse ip).
Sur le pc client, tapez la commande :

```
ssh moi@moi.mondomaine.fr -Cp443 -ND3128 \
-o"ProxyCommand corkscrew proxy.entreprise.fr 3128 %h %p"
```

Si le proxy nécessite une authentification :

```
ssh moi@moi.mondomaine.fr -Cp443 -ND3128 \
-o"ProxyCommand corkscrew proxy.entreprise.fr 3128 %h %p monlogin monmotdepasse"
```

(vous pouvez choisir le port que vous voulez au lieu de 3128)
Voilà, maintenant, `localhost:3128` est un proxy SOCKS5 !
Vous pouvez le configurer dans les paramètres de votre système (voir Fig. 7.1) ou dans chacun des logiciels.
Toutes vos connexions passeront dans le tunnel SSH de manière sécurisée (entre votre pc client et votre pc fixe), et c'est votre pc fixe qui effectuera la connexion vers les sites que vous voulez accéder, mais lui, contrairement au proxy de l'entreprise, ne refusera pas les connexions :)
Pour éviter de passer toute la commande `ssh` en paramètre de `ssh`, vous pouvez la configurer définitivement pour un host particulier. Pour cela, éditez chez le client `/etc/ssh/ssh_config`, à la fin du fichier, rajoutez (en adaptant) :

```
Host moi.mondomaine.fr
        ProxyCommand corkscrew proxy.entreprise.fr 3128 %h %p
```

ou, si le proxy nécessite une authentification :

```
Host moi.mondomaine.fr
    ProxyCommand corkscrew proxy.entreprise.fr 3128 %h %p monlogin monmotdepasse
```

Maintenant vous pouvez simplement taper

```
ssh moi@moi.mondomaine.fr -Cp443 -ND3128
```

pour créer le tunnel.

De la même manière (avec *corkscrew*), vous pouvez monter vos répertoires distants en local à travers une connexion SSH sur `https`, et lire votre musique ou regardez vos vidéos à distance (si la connexion le permet), rediriger des ports...

Enjoy :)