ROYAUME DU MAROC

مكتب التكوين المهني وإنعاش الشغل

Office de la Formation Professionnelle et de la Promotion du Travail

# Ssh sous Gnu/Linux

www.ofppt.info

OFPPT

**D**IRECTION **R**ECHERCHE ET **I**NGENIERIE DE **F**ORMATION

**SECTEUR NTIC**

# Sommaire

## 1.1. Principe

Approche de ssh, des tunnels et des services mandataires

## 1.2. Présentation

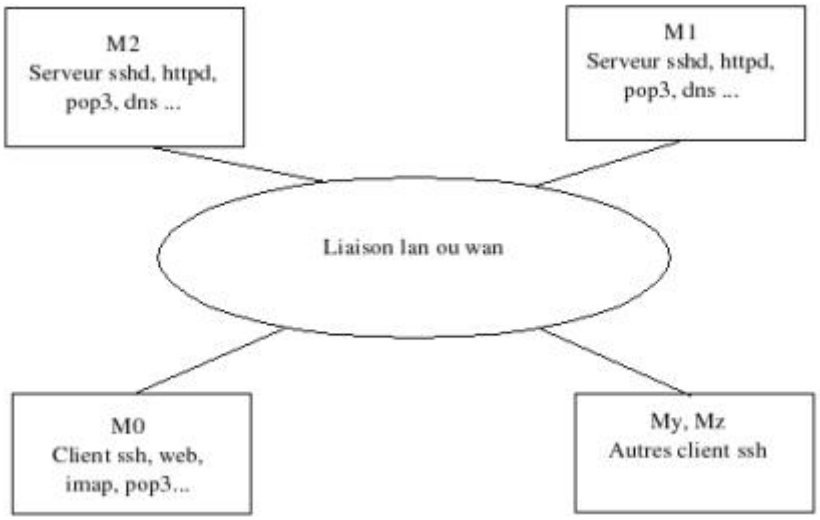
Un des principaux risques sur les réseaux provient de "l'écoute" possible puisque toutes les données transitent, si rien n'est fait, en clair sur les résaeux. C'est à dire qu'elles ne sont pas cryptées.

Il est ainsi possible de récupérer sans difficulté les mots de passe des personnes utilisant le réseau, leur messages, et d'espionner toutes leurs transactions, y compris celles passées sur des serveurs HTTP. Ceci est lié à la méthode d'accès des réseaux. Le principe de la commutation par switch permet de limiter un peu les risques mais n'est pas imparable.

Il existe des solutions permettant de sécuriser un minimum les transactions. Nous allons voir rapidement quelques modes d'utilisation de `ssh` et de logiciels comme scp, sftp, unisson et rsync afin de voir comment évoluer dans un environnement plus sûr.

Il sera fait référence à la maquette suivante :

**Figure 13.1. Schéma maquette**




Qu'est ce que ssh ?

En fait ssh ou Secure SHell, propose un shell sécurisé pour les connexions à distance et se présente dans ce domaine comme le standard "de fait". Mais ce n'est pas que cela. Nous allons essayer de voir quelques modes d'utilisation de ce produit.

Dans une transaction traditionnelle, un client (personne ou programme) émet une requête TCP vers un serveur. Il y a un processus serveur utilisant un port et un processus client utilisant

également un port. Par exemple pop3. Il y a donc un processus serveur et un processus client.

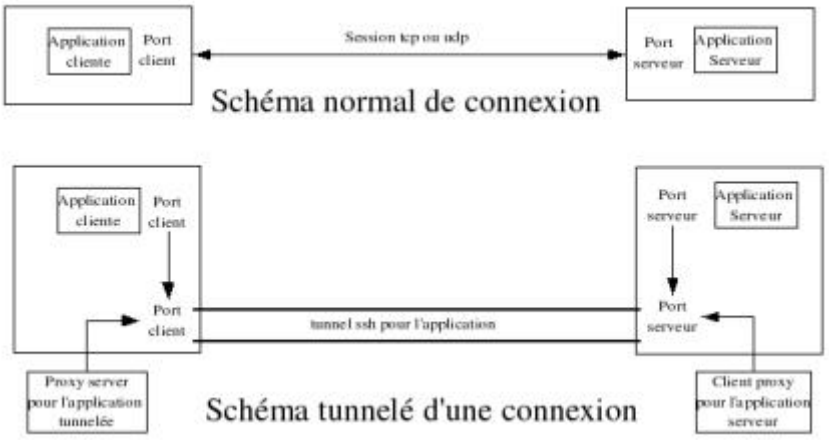
Avec ssh, il sera possible d'établir un tunnel crypté (ou chiffré) entre le client et le serveur.

Il faut bien comprendre ce dont il s'agit et les processus mis en oeuvre.

Sur la machine serveur vous allez avoir 2 processus serveurs. Le serveur pop3 et le serveur SSH. Sur le client, vous allez également avoir 2 processus. Le client pop3 et le client ssh. Le client pop3 se connecte au tunnel (le client ssh local). Le serveur pop3, est relié au serveur ssh distant. Les transactions passent dans le tunnel.

Le client ssh devient un serveur mandataire (proxy) pour le protocole tunnelé. Le serveur ssh devient le client proxy pour le serveur.

**Figure 13.2. Schéma du fonctionnement**




Sur le diagramme, le canal entre le port tcp de l'application cliente et le port client du tunnel n'est pas chiffré. Il en est de même entre le port tcp de l'application serveur et le port serveur du tunnel. Seul, le canal entre les 2 ports du tunnel est chiffré.

L'application cliente n'utilise plus le port par défaut qu'écoute normalement le serveur.

L'utilisation d'un tunnel ssh impose des contraintes. Par exemple, dans l'exemple ci-dessus, si vous voulez utiliser l'application cliente avec un autre serveur, il faudra recréer un autre tunnel et reconfigurer le client. Vous pouvez créer deux tunnels différents, mais vous devrez avoir deux applications clientes utilisant des ports différents. Tout cela convient bien sur des environnements simples, mais reste un peu contraignant si l'environnement est complexe.

Pour tunneler un réseau ou de multiples clients, le mieux est d'installer un serveur ssh capable de recevoir plusieurs connexions et servant ainsi de serveur proxy aux clients du réseau pour un service donné et routant les requêtes dans un tunnel sécurisé vers le serveur d'application concerné.

L'objectif est de pouvoir supprimer d'un serveur toute application utilisant des protocoles "non sûrs" (ftp, telnet, rsh...), pour les remplacer par des applications plus sûres, ou alors, s'il n'est pas

possible de les désactiver, de les faire passer dans un tunnel crypté.

Des programmes de la même famille comme "scp" ou "sftp" remplacent les commandes ftp ou rcp.

Avec ssh la totalité de la transaction entre un client et le serveur est cryptée. Le client a la possibilité d'utiliser des applications X distantes (X11 forwarding) à partir d'une invite shell dans un environnement sécurisé. On se sert également de SSH pour sécuriser des transactions non sûres comme pop3 ou imap par exemple. De plus en plus, ces applications supportent maintenant SSL aussi bien pour les clients que pour les serveurs.

SSH sur GNU/Linux est généralement composé de 3 packages :

```
OpenSSH général, (openssh),
le serveur OpenSSH (openssh-server)
le client (openssh-clients).
```

Les packages OpenSSH requièrent le paquetage OpenSSL (openssl).

Chaque fois que cela est possible vous devez utiliser une solution qui chiffre vos transactions.

## *1.3. Mode de fonctionnement de SSH*

L'établissement du dialogue entre le client et le serveur suit un protocole particulier :

1. établissement d'une couche transport sécurisée
2. chiffrement des données à l'aide de clefs symétriques pendant la transaction

Le client peut s'authentifier en toute sécurité, et accéder aux applications conformes aux spécifications du protocole.

### 1.3.1. Mode de fonctionnement de la couche transport SSH

La couche transport assure le chiffrement et le déchiffrement des données. Elle assure également la compression pour améliorer le transfert. Le client et le serveur négocient plusieurs éléments afin que la session puisse s'établir.

1. l'échange des clés
2. l'algorithme de clé publique à utiliser
3. l'algorithme de chiffrement symétrique à utiliser
4. l'algorithme d'authentification de message à utiliser
5. l'algorithme repère (hash) à utiliser

Lors du premier échange, le client ne connaît pas le serveur. Le serveur propose alors une clé hôte qui servira par la suite au client de moyen d'identification du serveur.

```
M0:$ ssh -l mlx M1.foo.org
Warning: Permanently added 'M1,x.y.z.t' (DSA) to the list of known hosts.
mlx@M1.foo.org's password:
Last login: Sat Nov  2 11:37:32 2002 from 212.47.248.114
```

```
Linux 2.2.19.
mlx@M1.foo.org:$
```

Voilà une idée de ce que cela donne :

```
freeduc-sup.alt.eu.org,144.85.15.72 ssh-rsa AAAAB3NzaC1y (...)
pegase,195.115.88.38 ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIE (...)
freeduc-sup.eu.org,137.194.161.2 ssh-dss AAAAB3NzaC1kc3M (...)
(...)
```

Le risque de ce processus, vient donc surtout de la première transaction, où le client n'a pas le moyen d'identifier de façon fiable le serveur avec qui il communique. Un pirate peut dans certains cas, tenter de détourner cette opération. Au cours d'un échange, le client et le serveur modifient régulièrement leurs clés. A chaque opération de renouvellement de clé, un pirate qui aurait réussi à décrypter les clés, devrait refaire toute l'opération.

**Authentification :**

Une fois le tunnel sécurisé mis en place, le serveur envoie au client les différentes méthodes d'authentification qu'il supporte. Dans le cadre d'une authentification par mot de passe, celui-ci peut être envoyé en toute sécurité puisqu'il est chiffré.

**Connexion** :

Une fois l'authentification réalisée, le tunnel SSH peut multiplexer plusieurs canaux en délégant la tâche à des agents.

```
[mlx@M1 X11]$ pstree -l 24245
sshd---bash-+-drakfw
            |-pstree
            |-2*[xclock]
            `-xlogo
```

Ici on voit dans la même session ssh, 2 canaux pour xclock, 1 pour xlogo et 1 pour la commande pstree. Chaque canal est numéroté. Le client peut fermer un canal sans pour autant fermer toute la session.

### 1.3.2. Fichiers de configuration d'OpenSSH

OpenSSH est constitué de deux ensembles de fichiers de configuration, comme c'est en général le cas sur les services sous linux (ldap, samba..). Il y a un fichier de configuration pour les programmes clients (ssh, scp et sftp) et l'autre pour le service serveur(sshd). Les informations de configuration SSH qui s'appliquent à l'ensemble du système sont stockées dans le répertoire */etc/ssh*. Voici les principaux fichiers de configuration :

```
ssh_config              fichier de configuration client SSH pour l'ensemble
                        du système. Il est écrasé si un même fichier est
                        présent dans le répertoire personnel de
                        l'utilisateur
                        (~/.ssh/config).
sshd_config             fichier de configuration pour sshd.
ssh_host_dsa_key        clé DSA privée utilisée par sshd.
ssh_host_dsa_key.pub    clé DSA publique utilisée par sshd.
```

```
ssh_host_key             clé RSA privée utilisée par sshd pour la
                         version 1 du protocole SSH.
ssh_host_key.pub         clé RSA publique utilisée par sshd pour la
                         version 1 du protocole SSH.
ssh_host_rsa_key         clé RSA privée utilisée par sshd pour la
                         version 2 du protocole SSH.
ssh_host_rsa_key.pub     clé RSA publique utilisée par sshd pour la
                         version 2 du protocole SSH.
```

Les informations spécifiques à un utilisateur sont stockées dans son répertoire personnel à l'intérieur du répertoire ~/.ssh/:

```
authorized_keys ou parfois   authorized_keys2
                           ce fichier contient une liste de clés
                           publiques "autorisées". Si un utilisateur
                           se connecte et prouve qu'il connaît la clé
                           privée correspondant à l'une de ces clés,
                           il obtient l'authentification. Notez qu'il
                           ne s'agit que d'une méthode d'authentification
                           facultative.

id_dsa                     contient l'identité d'authentification
                           DSA de l'utilisateur.
id_dsa.pub                 la clé DSA publique de l'utilisateur.
id_rsa                     la clé RSA publique utilisée par sshd pour
                           la version 2 du protocole SSH.
identity                   la clé RSA privée utilisée par sshd pour la
                           version 1 du protocole SSH.
known_hosts                ce fichier contient les clés hôte DSA des
                           serveurs SSH auxquels l'utilisateur s'est

connecté.
```

Exemple sur une machine :

```
mlx@M1:~$ ls -al .ssh/
total 16
drwx------   2 mlx mlx     4096 Jan 16  2004 ./
drwxr-xr-x   5 mlx mlx     4096 Oct 18 16:12 ../
-rw-------   1 mlx mlx     1192 Mar 11  2004 authorized_keys2
-rw-------   1 mlx mlx      240 Jan 16  2004 known_hosts
```

## *1.4. Configurer et utiliser SSH*

Nous allons faire nos premières expérences avec ssh. Il vous faut un client et un serveur. C'est mieux.

### 1.4.1. Premiers pas

L'idée est de mettre en place une procédure entre un client et un serveur qui garantit des transactions sécurisées. A la fin, vous pourrez utiliser les ressources du serveur distant dans un tunnel, sans avoir à vous authentifier à chaque fois.

Pour ceux qui ont déjà utilisé les commandes rlogin, rsh, rcp... et les fichiers .rhosts, le principe

est identique. La différence fondamentale est que, avec ssh, tout est crypté.

Pour cela vous devez mettre en place les clés qui serviront à ssh pour vous authentifier. Concrètement cela consiste à définir une paire de clés, une publique que vous mettrez sur le serveur distant, une privée que vous conserverez sur votre machine.

La première chose à faire est de vous créer une clé. Voyons comment réaliser cela.

Tout d'abord allez dans votre répertoire personnel.

```
$ cd
$ ssh-keygen -t dsa
$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/home/mlx/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/mlx/.ssh/id_dsa.
Your public key has been saved in /home/mlx/.ssh/id_dsa.pub.
The key fingerprint is:
5c:d9:d8:c5:3d:8d:0b:10:33:42:9c:83:45:a1:d0:61 mlx@neptune.foo.org
```

Vérification de .ssh. Attention il y a un "." devant ssh

```
[mlx@neptune mlx]$ ls -alR .ssh
.ssh:
total 20
drwx------    2 mlx      mlx          4096 nov 11 18:19 ./
drwx--x--x   37 mlx      mlx          4096 nov 11 18:09 ../
-rw-------    1 mlx      mlx           736 nov 11 18:19 id_dsa
-rw-r--r--    1 mlx      mlx           616 nov 11 18:19 id_dsa.pub
-rw-r--r--    1 mlx      mlx          1956 nov 10 19:38 known_hosts
```

Cette commande a généré une clé DSA par défaut de 1024 bits. La clé privée sera stockée dans ~/.ssh/id_dsa et la clé publique dans ~/.ssh/id_dsa.pub.

Si vous voulez générer une clé RSA2, utilisez l'option "-t rsa" et pour du RSA1 "-t rsa1". Vous devrez entrer une "passphrase". Entre 10 et 30 caractères. Mélangez majuscules, minuscules et chiffres. La clé privée doit ensuite être mise en lecture seule pour le propriétaire et aucun accès pour les autres.

Pour modifier votre "passphrase" sur une clé privée DSA, utilisez la commande :

```
M0:$ ssh-keygen -p -f ~/.ssh/id_dsa
Enter old passphrase:
Key has comment '/home/mlx/.ssh/id_dsa'
Enter new passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved with the new passphrase.
```

Attention, il ne faut pas oublier la "passphrase", elle vous sera redemandée. Il va falloir maintenant copier la clé publique vers le ou les serveurs distants. Il est préférable que vous ayez un compte, sinon vous devrez demander à l'administrateur distant de réaliser la procédure.

La **clé publique**, doit être **copiée sur le serveur distant** dans ~/.ssh/authorized_keys. La clé privée reste sur votre poste client. Vous pouvez mettre plusieurs clés publiques sur le serveur, si vous le souhaitez ou si vous accédez au serveur avec plusieurs comptes d'accès différents.

Copiez la clé avec scp sur le compte que vous avez sur le serveur :

```
MO:$ cat .ssh/id_dsa.pub | ssh mlx@M1.foo.org \
                   "cat - >>.ssh/authorized_keys[2]"
# Attention, sur certaines machines, le fichier se nomme
# authorized_keys, sur d'autres authorized_keys2
# Dasn l'exemple qui est donné, le répertoire .ssh doit exister.

Warning: Permanently added 'M1.foo.org' (RSA) to the list of known hosts.
mlx@M1.foo.org's password:
```

Le système demande votre mot de passe.

Elle est transférée. On doit pouvoir maintenant réaliser des opérations (commandes) comme scp sur le serveur distant, sans avoir à saisir de mot de passe. Ici on va faire un "ls", sans ouvrir de session sur la machine distante.

```
M0:$ ssh mlx@M1.foo.org ls
Enter passphrase for key '/home/mlx/.ssh/id_dsa':
d1
d2
d3
d4
```

Le système distant ne demande plus le mot de passe, par contre il demande la "passphrase". Il va falloir aussi essayer de se passer de ça, car s'il est fastidieux de saisir son mot de passe, il est encore plus embêtant de saisir une "passphrase". Nous verrons comment se passer de ça avec un agent.

Remarque : Envoyer une clé par mail n'est pas un système sûr, et même chiffré et signé cela ne garantit pas au destinataire que vous en êtes l'émetteur s'il ne vous a jamais vu. L'administrateur distant peut demander à ce que l'envoyeur justifie qu'il est bien celui qui a envoyé la clé. Il suffit pour cela de téléphoner à l'administrateur et de communiquer "la signature ou empreinte" (finger print) de la clé (ou par sms). On utilise le même procédé pour identifier et vérifier la validité des clés gpg.

Pour obtenir le "finger print" d'une clé utiliser la commande :

```
M0:$ ssh-keygen -l
Enter file in which the key is (/home/mlx/.ssh/id_rsa): .ssh/id_dsa
1024 5c:d9:d8:c5:3d:8d:0b:10:33:42:9c:83:45:a1:d0:61 .ssh/id_dsa.pub
M0:$ ssh-keygen -l
Enter file in which the key is (/home/mlx/.ssh/id_rsa): .ssh/id_dsa.pub
1024 5c:d9:d8:c5:3d:8d:0b:10:33:42:9c:83:45:a1:d0:61 .ssh/id_dsa.pub
```

### 1.4.2. Utiliser un agent ssh

L'utilisation d'un agent, évite d'avoir à retaper la "passphrase" à chaque fois que l'on sollicite l'utilisation de la clé privée. Un agent stocke en mémoire les clés privées. Voici comment activer

un agent :

```
M0:$ ssh-agent
```

La commande met sur la sortie standard des variables environnement à déclarer et à exporter. Faites le.

```
M0:$ SSH_AUTH_SOCK=/tmp/ssh-XXXB76f4/agent.2888; export SSH_AUTH_SOCK;
M0:$ SSH_AGENT_PID=2889; export SSH_AGENT_PID;
M0:$ echo Agent pid 2889;
```

On va maintenant exporter les clés. Cela consiste à les mettre dans le cache de l'agent avec la commande ssh-add. La commande demandera la "passphrase"..

```
M0:$ ssh-add
Enter passphrase for /home/mlx/.ssh/id_dsa:
Identity added: /home/mlx/.ssh/id_dsa (/home/mlx/.ssh/id_dsa)
```

On vérifie maintenant la connexion.

```
M0:$ ssh mlx@M1.foo.org ls
d1
d2
d3
d4
```

Nous n'avons plus besoin de taper le mot de passe, ni la "passphrase". Pour supprimer une clé (ici RSA) de l'agent, utilisez l'option "-d"

```
M0:$ ssh-add -d ~/.ssh/id_rsa

Utilisez l'option -D pour supprimer toutes les clés de l'agent.
Utilisez l'option -l pour savoir quelles clés sont chargées par l'agent.
```

### 1.4.3. Automatisation dans X

Cela peut être automatisé dans un script.

"ssh-agent" est un daemon dont le but est d'intercepter la clé publique lorsque le programme "ssh-add" la lit après avoir demandé la passphrase. "ssh-agent" doit ensuite transmettre la clé aux processus dont il est le "père" du point de vue du système. "ssh-agent" fait bénéficier les processus fils (fork) de ses propriétés.

Pour étendre cela aux sessions X, si vous lancez l'interface graphique manuellement, cela deviendra par exemple :

```
M0:$ ssh-agent xinit
ou
M0:$ ssh-agent startx
```

Dans tous les cas, il est nécessaire que les variables d'environnement soient définies avant le lancement du serveur X, qui les exportera par défaut à l'ensemble de vos applications graphiques

(lancées sous X).

Cela peut aussi être fait dans le fichier startx ou bien ~/.xinitrc ou bien ~/.xsession, en résumé avant le lancement de votre gestionnaire de fenêtres. Par exemple, si c'est le fichier ~/.xinitrc qui lance le gestionnaire de fenêtres, il ressemblera à ceci:

```
#!/bin/sh

ssh-agent -s > /tmp/ssh.keys     # pour y mettre les variables environnement
.  /tmp/ssh.keys                 # Exporter les variables
rm  /tmp/ssh.keys                # Faire le ménage après
startx
```

Ainsi, au prochain lancement d'un serveur X, vous n'aurez plus qu'à ouvrir une fenêtre xterm et taper la commande: `ssh-add`

Une autre solution consiste à mettre dans son .xinit ou .xsession :

```
/usr/local/bin/ssh-add $HOME/.ssh/id_dsa < /dev/null
```

La commande ssh-add demande la passphrase

Pour les sessions xdm par exemple, modifier les fichiers de démarrage :

```
#Au début de /etc/X11/Xsession:
AGENT=$(type -p ssh-agent)
if [ -x "$AGENT" -a -z "$SSH_AUTH_SOCK" ]; then
    if [ -r $HOME/.ssh/identity -o -r $HOME/.ssh2/identification \
            -o -r $HOME/.ssh/id_dsa -o -r $HOME/.ssh/id_rsa ]; then
        SSH_AGENT="$AGENT --"
    fi
fi
```

## *1.5. Comprendre la redirection de port (Port Forwarding)*

Avant d'aller plus loin il est important de bien comprendre ce qu'est le "port forwarding".

Nous avons vu qu'il y avait en jeu :

1. - l'application cliente
2. - l'application serveur
3. - le client ssh
4. - le serveur ssh

Nous allons voir la redirection locale '-L' et distante '-R'. La différence vient du sens de la connexion. Dans le relayage local, le client tcp et le client ssh sont sur la même machine. Dans le relayage distant ou "remote", le client ssh est sur la même machine que le serveur tcp.

Dans la démarche qui suit, on utilise un client M0 et deux serveurs M1 et M2. On considère que sur chaque serveur fonctionne un serveur ssh. On considère aussi que sur chaque machine on

dispose d'un compte d'accès sur chaque machine identique avec les clés publiques installées. Cela évite une authentification à chaque commande ou l'utilisation de l'option '-l' pour préciser le compte à utiliser sur le serveur.

### 1.5.1. Redirection locale de port (-L Local)

Si on considère ces 3 machines : M0, la machine cliente, M1 et M2, des serveurs. La syntaxe de la commande est la suivante :

```
ssh -L port-local:HOSTNAME:port-distant machine-distante
```

la commande est passée sur M0.

```
M0:$ ssh -L 1234:HOSTNAME:80 M1
```

La commande ouvre une connexion entre le M0:1234 vers HOSTNAME:80 en utilisant le serveur sshd M1:22 (actif sur M1 et sur le port 22). Tout dépend de la valeur que va prendre HOSTNAME. HOSTNAME indique la machine distante sur lequel s'opére la connexion.

Si HOSTNAME est localhost :

```
M0:$ ssh -L 1234:localhost:80 M1
```

Ici localhost indique l'adresse de loopback de la machine distante, c'est à dire ici M1 et sur laquelle tourne le serveur sshd. C'est donc M0:1234 qui est tunnélé vers le port M1:80 en utilisant le service M1:22.

Si HOSTNAME est M1 :

```
M0:$ ssh -L 1234:M1:80 M1 :-/
```

La connexion est identique mais utilisera l'adresse de la M1 (interface réseau ) plutôt que l'interface lo.

Si HOSTNAME est M0 :

```
M0:$ ssh -L 1234:M0:80 M1
```

La commande connecte M1 sur M0:80.

Si HOSTNAME est M2 :

```
M0:$ ssh -L 1234:M2:80 M1
```

Il y aura une connexion (un tunnel créé) entre M0 et M1 mais la redirection est effectuée entre M1:1234 et M2:80 en utilisant M1:22. Les transactions sont chiffrées entre M0 et M1, mais pas entre M1 et M2, sauf si un second tunnel ssh est créé entre M1 et M2.

Les redirections de ports ne sont accessibles en théorie que pour les processus locaux (localhost) (pour nous M0). Si la redirection était possible pour des clients (MX ou MY) ou des requêtes

distantes qui viendraient se connecter su rM0:1234 dans notre exemple, ces clients pourraient être surpris de se voir re-routés. (Sans parler de risques pour la sécurité car cela signifie que d'autres personnes pourraient utiliser à notre insu le tunnel que nous venons de créer. Prenons un exemple :

```
MO$ ssh -L 1234:M1:80 M1
```

les requêtes locales passées sur M1:1234 sont redirigées vers M1:80, mais des requêtes passées d'autres machines sur M0:1234 ne seront pas, par défaut redirigées.

Il est possible toutefois de passer outre avec l'option '-g' qui autorise des clients distants à se connecter à des ports locaux redirigés.

```
MO$ ssh -g -L 1234:M2:80 M1
```

La commande "lynx http://M0:1234" lancé à partir d'une machine tierce (MX ou MY) redirigera vers M2:80.

Une utilisation de cette fonction sera vue en application un peu plus loin.

### 1.5.2. Redirection distante de ports (-R Remote)

Ici le client ssh et le serveur TCP sont du même côté. La syntaxe de la commande est la suivante :

```
ssh -R port-distant:HOSTNAME:port-local  machine-distante
```

Le port distant est donc sur la machine distante (remote)

On reprend les mêmes machines que précédemment :

```
M0:$ ssh -R 1234:HOSTNAME:80 M1
```

Ici M0 à le serveur TCP, il sert donc de relais entre une connexion M1:1234 et HOSTNAME:80. La connexion est chiffrée entre M1 et M0. Le chiffrement entre M0 et HOSTNAME dépend de la liaison mise en place.

Si HOSTNAME est localhost, alors on a :

```
M0:$ ssh -R 1234:localhost:80 M1
```

Cela ouvre une connexion depuis M1:1234 vers M0:80 car localhost correspond, ici, à M0. Si un utilisateur passe une requête sur M1:1234, elle sera redirigée vers M0:80.

Si HOSTNAME est M2, on a :

```
M0:$ ssh -R 1234:M2:80 M1
```

Cela ouvre une connexion entre M0 et M1:22, mais les requêtes allant de M1:1234 sont redirigés

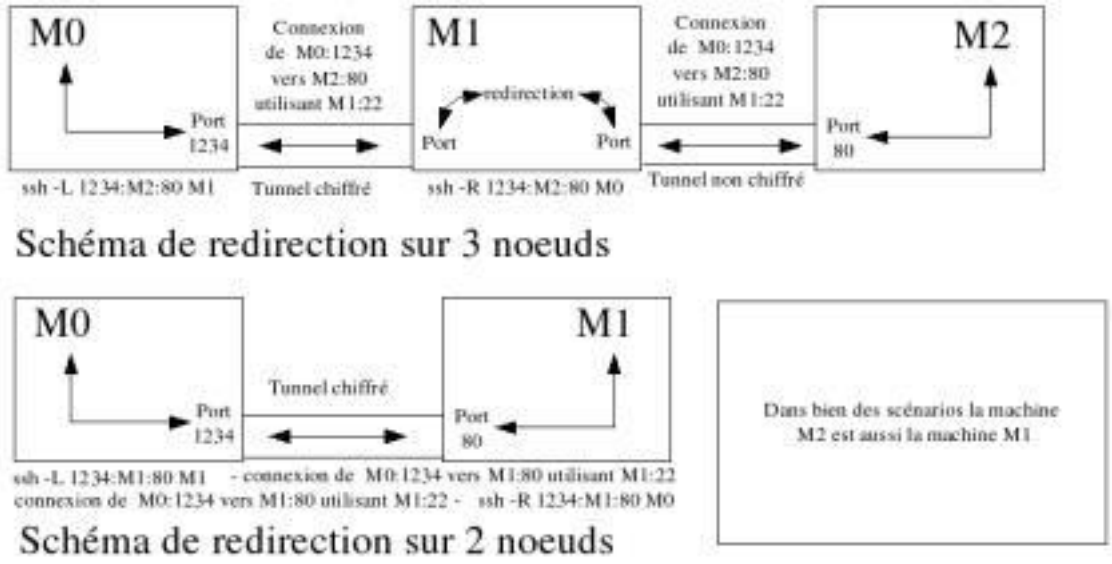
sur M2:80. Le canal est chiffré entre M1 et M0, mais pas entre M0 et M2.

Enfin dernière remarque, il est possible de passer en paramètre le compte à utiliser sur le serveur qui fait tourner le serveur sshd.

Cette option permet par exemple de donner, à des machines distantes, un accès à un service sur une machine inaccessible autrement.

### 1.5.3. Schéma de redirection distante de ports

**Figure 13.3. Schéma du fonctionnement**




### 1.5.4. Exemple de cas d'utilisation

SSH permet de "tunneler" des protocoles applicatifs via la retransmission de port. Lorsque vous utilisez cette technique, le serveur SSH devient un conduit crypté vers le client SSH.

Cela représente un des plus importants intérêt de SSH. Permettre la création de tunnels "sûrs" pour des protocoles de transports "non sûrs". Ce principe est utilisé pour des applications comme pop, imap, mais également pour des applications X Window.

La retransmission de port mappe (redirige) un port local du client vers un port distant du serveur. SSH permet de mapper (rediriger) tous les ports du serveur vers tous les ports du client.

Pour créer un canal de retransmission de port TCP/IP entre 2 machines qui attend les connexions sur l'hôte local, on utilise la commande suivante :

```
ssh -L port-local:HOSTNAME:port-distant nomutilisateur@nomhôte
```

Une fois que le canal de retransmission de port est en place entre les deux ordinateurs, vous pouvez diriger votre client (POP par exemple) pour qu'il utilise le port local redirigé et non plus vers le port distant avec une transmission en clair.

Nous avons bien vu les options '-L' et '-R'. De nombreuses autres options sont utilisables, par exemple :

```
-L     # Forwarder un port local vers un port distant sur une machine
distante
-R     # Forwarder un port distant vers un port local sur la machine locale
-N     # Ne pas exécuter de commande distante
-f     # Excute le programme en tâche de fond
-l     # Passer en paramètre le login de connexion
-g     # Autoriser des machines distantes à se connecter
       # sur des ports locaux exportés
```

Voyons comment créer un tunnel pour le service d'émulation VT par exemple. On va utiliser un port local compris entre 1024 et 65535 qui sont réservés pour des applications utilisateurs. On va prendre par exemple le port local 1025. Pour créer un tunnel on va utiliser la commande :

```
M0:$  ssh -L 1025:localhost:23 mlx@M1.foo.org
```

Ici on utilise (précise) que le compte utilisé sur la machine distante sera M1.foo.org. Vous devez faire cela si le nom du compte que vous utilisez sur le client diffère de celui que vous utilisez sur le serveur.

On crée ici un tunnel entre le port local 1025 et le port distant 23. Le tunnel étant créé il ne reste plus qu'à utiliser le port local 1025 avec un telnet adresse_de_la_machine 1025

Le fait de quitter ssh ferme la session tunnelée.

Il est également possible d'ouvrir une session temporaire pour un temps donné. Cela évite d'avoir à fermer les connexions. Dans la commande :

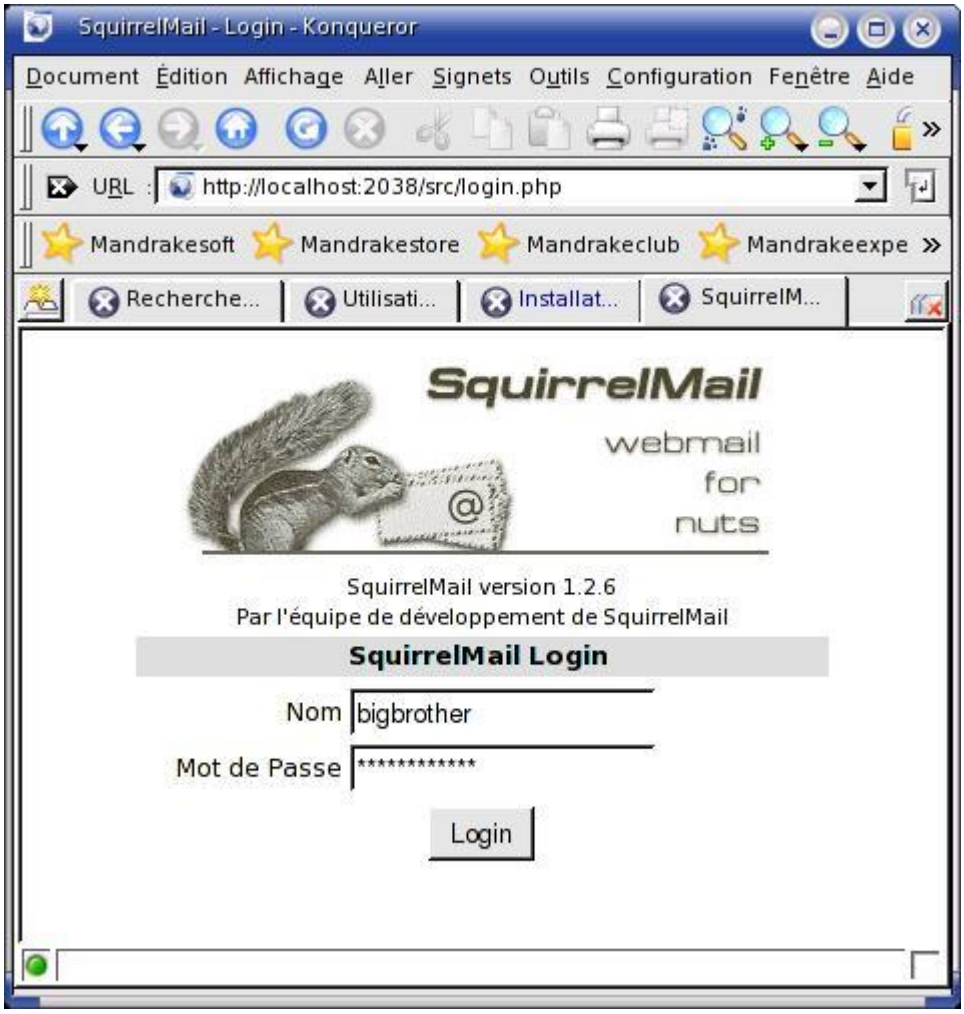
```
M0:$  ssh -f  -L 1025:localhost:23 mlx@M1.foo.org sleep 10
```

L'option -f, met ssh en tâche de fond. La session sera fermée automatiquement au bout du temps déterminé, seulement si aucun processus n'utilise le canal à ce moment.

Le principe peut être appliqué à d'autres services. Voici comment utiliser un canal sécurisé vers une application (ici un webmail) qui ne l'est pas. (En fait dans la réalité, je vous rassure, l'application présentée sur l'image offre toutes les options de connexion en mode sécurisé. L'exemple donné est pris pour illustrer le propos.)

```
# On crée un tunnel entre le port local et le webmail en utilisant
# le compte mlx@foo.org
M0:$ ssh  -N -f -L 2038:M1:80 mlx@foo.org
```

**Figure 13.4. Tunnel HTTP**




On peut également sans risque aller sur sa zone public_html en toute sécurité.

```
lynx http://localhost:2038/~mlx
```

La connexion se fait sur la machine locale et sur le port forwardé. Ni le compte utilisateur utilisé, ni le mot de passe ne transitent en clair.

Avec la machine sur lequel le test est réalisé, les commandes :

```
M0:$ ssh  -N -f -L 2038:M1:80 mlx@foo.org
et
M0:$ ssh  -N -f -L 2038:localhost:80 mlx@foo.org
```

ne produisent pas la même chose.

En effet :

```
M0:$ ssh  -N -f -L 2038:M1:80 mlx@foo.org
```

fait référence à une adresse IP qui est redirigée par un Vhost Apache.

```
M0:$ ssh  -N -f -L 2038:localhost:80 mlx@foo.org
```

fait référence à l'adresse de loopback, ici c'est le serveur par défaut (typiquement sur une debian /var/www/) qui est consulté. Si vous avez plusieurs Vhosts, il vous faudra créer un tunnel par Vhost.

De la même façon, on peut l'utiliser pour une session ftp.

```
M0:$ ssh  -N -f -L 2039:M1:21 mlx@foo.org
M0:$ ftp localhost 2039
Connected to localhost.
220 ProFTPD 1.2.5rc1 Server (Debian) [M1]
Name (localhost:mlx): mlx
331 Password required for mlx.
Password:
230 User mlx logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
```

Attention, dans ce dernier exemple, il n'y a que l'authentification qui est chiffrée car le port 20 (ftp-data) n'est pas forwardé.

### 1.5.5. X and Port Forwarding

Le déport d'application graphique fonctionne sur le même principe. Le client (/etc/ssh/ssh_config), doit avoir l'option "X11Forward" activée pour les systèmes distants. Le serveur (/etc/ssh/sshd_config), doit avoir l'option "X11Forwarding" d'activée. Il est possible de tunneler des applications graphiques dans ssh.

```
# Sur le client /etc/ssh/ssh_config
ForwardX11 yes

# Sur le serveur /etc/ssh/sshd_config
X11Forwarding yes
```

Voyons comment utiliser une application graphique distante :

```
M0:$  ssh -C  mlx@M1.foo.org
Enter passphrase for key '/home/mlx/.ssh/id_dsa':
M0:$ xclock &
[1] 7771
```

L'horloge distante s'affiche. L'option "-C", active la compression.

Vous pouvez tout mettre sur la même ligne de commande, avec l'option "-f", qui "fork" l'application demandée.

```
M0:$  ssh -C -f  mlx@M1.foo.org xclock
```

Notez le prompt, ici l'ouverture de session a été effectuée en tâche de fond (-f). Pour fermer le canal ssh, il suffit de fermer l'application.

### 1.5.6. Automatisation de tâches SSH

Dans la mesure où il est possible de passer des commandes à distances sur un serveur, sans avoir à saisir de mot de passe ou de "passphrase", on peut envisager l'automatisation de tâches entre machines et dans des tunnels, comme par exemple les sauvegardes ou la synchronisation de fichiers. Il suffira pour cela de créer un compte associé à la tâche, lui créer une clé privée et exporter sa clé publique sur les différentes machines.

Attention toutefois, les manipulations sur les serveurs requièrent un accès root. Cela signifie que votre compte opérateur, devra avoir un accès root sur le serveur ce qui n'est jamais très bon.

Vous aurez intérêt à réfléchir à la façon de réaliser le traitement sans que ce compte opérateur ait besoin du compte "super utilisateur".

Si vous avez un serveur ayant un dm (Desktop Manager) comme gdm par exemple, disponible sur le réseau vous pouvez lancer les émulations X Window des clients en appelant le serveur. Par exemple sur le client faire :

```
X -query x.y.z.t :u
ou encore
X -broadcast :u
```

avec u pouvant varier de 0 à 5, suivant que vous voulez avoir la session graphique sur F7...F12. Pour le système vc7 que vous utilisez par défaut avec CTRL ALT F7 correspond au premier "display" :0.

## 1.6. Scénario d'utilisation d'un proxy ssh

Maintenant que nous y voyons plus clair, voici deux scénarios d'utilisation d'un proxy ssh.

### 1.6.1. Proxy HTTP

Vous êtes sur un réseau avec un accès extérieur limité. Pas d'accès http :-(

Vous devez disposer d'une machine à l'extérieur sur laquelle tourne un serveur ssh et un proxy Squid par exemple.

Par défaut Squid utilise le port 3128. On met tout d'abord les autorisations à Squid afin qu'il accepte nos requêtes, sinon elles seront rejetées. (Voir les ACL du fichier /etc/squid.conf). Si vous êtes pressés un "http_access allow all" est brutal, mais ça marche ;-) Nous allons créer un tunnel vers ce service de la machine locale vers la machine distante sur le port 3128.

```
ssh  -N -f -L 3128:M1:3128 mlx@M1
```

Il ne reste plus qu'à configurer votre navigateur pour qu'il utilise le proxy local "localhost:3128" et le tour est joué. Vous pouvez naviguer en toute sécurité, si vos requêtes sont espionnées au niveau du firewall de votre boîte, elle ne pourront plus être lues.

Cette configuration présente une limitation. Vous ne pouvez utiliser le proxy qu'à partir de la machine sur laquelle vous avez créé le tunnel (M0).

Si vous êtes mobile dans votre entreprise et que vous souhaitez accéder à l'extérieur à partir d'autres machines, ou encore que vous voulez laisser cet accès à des amis qui utilisent le même réseau que vous, il faut procéder autrement.

```
ssh  -g -N -f -L 3128:M1:3128 mlx@M1
```

L'option "-g" va transformer votre machine en passerelle pour le tunnel. Les autres n'auront plus qu'à mettre comme proxy, le nom ou l'adresse de votre machine et le port 3128.

Si vous souhaitez par contre ouvrir le service de votre proxy à d'autres amis mais pouvant être dans d'autres boîtes ou sur d'autres réseaux, cela se complique un peu, car chacun d'eux doit pouvoir créer un tunnel vers votre proxy. Il faut donc, sur votre proxy créer plusieurs ports d'écoute pour les tunnels et rediriger tout ça vers votre proxy.

```
ssh -N -f -g -L 3130:localhost:3128 mlx@localhost
ssh -N -f -g -L 3131:localhost:3128 mlx@localhost
etc... etc...
```

Ici on a créé sur le proxy 2 ports, 3130 et 3131 qui redirigent vers le port 3128. Il suffit ensuite à partir des machines distantes (réseaux distants) de créer les tunnels vers ces ports.

```
MY$ ssh  -g -N -f -L 3128:localhost:3130 mlx@M1
MZ$ ssh  -g -N -f -L 3128:localhost:3130 mlx@M1
```

et d'utiliser les redirections comme cela a été expliqué plus haut.

Si vous ne souhaitez rediriger les requêtes que pour une machine (site web) en particulier vous pouvez créer un tunnel de la façon suivante :

```
M0:$ ssh  -N -f -L 3128:SITEWEB:3128 mlx@M1
```

où SITEWEB est l'url du site web que vous souhaitez atteindre (typiquement un webmail). Ici, la configuration du navigateur ne change pas. Le proxy est le même. Vous évitez juste l'utilisation d'un squid si vous n'en avez pas.

Remarque, dans le navigateur vous pouvez taper tout ce que vous voulez (yahoo, wanadoo, google...) vous arriverez toujours sur SITEWEB.

### 1.6.2. Autres scénarios

Si vous avez compris le principe pour le processus décrit ci-dessus, vous pouvez l'appliquer pour tous les autres services (messagerie pop ou imap, news, cvs, vnc ...).

Si un admin faisait trop de rétorsion, ne dites plus rien, vous savez maintenant comment vous y prendre.

## *1.7. Utilisation de rsync*

rsync est un outil largement utilisé pour la synchronisation de répertoires sur la même machine

ou sur des machines distantes. (création de miroirs distants).

rsync est intéressant car il ne mettra à jour sur le "repository" qui reçoit uniquement les fichiers qui ont été créés ou modifiés. Cela procure un gain non négligeable par rapport à une simple "recopie" de toute l'arborescence dans le cas ou peu de fichiers sont modifiés.

rsync et rsyncd, associés à des scripts et à la crontab, est une option remarquable pour la réplication de disques entre 2 ou plusieurs machines.

Si vous souhaitez "mirorrer" un disque local vers un répertoire que vous avez sur une autre machine sur internet, il vous faudra également passer par une procédure d'authentification. Si vous avez exporté votre clé publique sur la machine distante, vous allez pouvoir synchroniser les disques dans un tunnel sécurisé avec ssh et sans avoir à entrer votre mot de passe. Par exemple, la commande :

```
cd & & rsync -e ssh -valptz  * mlx@M1.foo.org
```

synchronisera votre $HOME local, sur le $HOME de la machine distante.

Il existe bien sûr des applications graphiques basées sur le concept rsync.

Une autre option pour la synchronisation de disques distants est "unison". unison est un produit particulièrement performant :

```
http://www.cis.upenn.edu/~bcpierce/unison/index.html
```

unison permet l'utilisation en mode commande (scripts) mais propose aussi une interface graphique.

## *1.8. Utilisation de SCP et de SFTP*

L'utilisation de ces commandes est relativement simple. SCP permet de faire de la copie de fichiers. SFTP est utilisable en mode intéractif ou en mode batch et ressemble plus au FTP.

### 1.8.1. Utilisation de scp

La commande

```
M0:$ ssh mlx@M1.foo.org "ls -al psionic"

-rw-r--r--    1 root     root        64562 Nov 23 23:05 hostsentry-0.02-
4.noarch.rpm
-rw-r--r--    1 root     root        26955 Nov 23 23:05 logsentry-1.1.1-
1.i386.rpm
-rw-r--r--    1 root     root        48804 Nov 23 23:06 portsentry-1.1-
fr7.i386.rpm
-rw-r--r--    1 root     root        48804 Nov 23 23:13 portsentry-1.1-
fr7.i386.rpm.1
```

La commande :

```
ssh mlx@M1.foo.org "ls -al"
```

donne la liste des fichiers distants qui sont dans le répertoire "psionic". Pour copier ces fichiers localement dans un répertoire psionic on va utiliser :

```
cd && mkdir psionic
scp mlx@M1.foo.org:/home/mlx/psionic/* ~/psionic
```

Ou pour envoyer les fichiers du répertoire local psionic, vers le répertoire tmp qui est dans /home/mlx de la machine M1.foo.org :

```
scp ~/psionic/* mlx@foo.org:/home/mlx/tmp
```

### 1.8.2. Utilisation de sftp

sftp peut être utilisé pour du transfert de fichiers en mode sécurisé.

```
sftp  mlx@M1.foo.org
sftp>
```

On obtient un prompt, ici le système ne m'a pas demandé de m'authentifier. Pour avoir une liste des commandes, utiliser "help".

```
sftp> help
Available commands:
cd path                       Change remote directory to 'path'
lcd path                      Change local directory to 'path'
chgrp grp path                Change group of file 'path' to 'grp'
chmod mode path               Change permissions of file 'path' to 'mode'
chown own path                Change owner of file 'path' to 'own'
help                          Display this help text
get remote-path [local-path]  Download file
lls [ls-options [path]]       Display local directory listing
ln oldpath newpath            Symlink remote file
lmkdir path                   Create local directory
lpwd                          Print local working directory
ls [path]                     Display remote directory listing
lumask umask                  Set local umask to 'umask'
mkdir path                    Create remote directory
put local-path [remote-path]  Upload file
pwd                           Display remote working directory
exit                          Quit sftp
quit                          Quit sftp
rename oldpath newpath        Rename remote file
rmdir path                    Remove remote directory
rm path                       Delete remote file
symlink oldpath newpath       Symlink remote file
version                      Show SFTP version
!command                      Execute 'command' in local shell
!                             Escape to local shell
?                             Synonym for help
```

tentatives d'accès.

## 1.9. Extrait de /etc/syslog.conf

```
# Log anything (except mail) of level info or higher.
# Don't log private authentication messages

# The authpriv file has restricted access.
authpriv.*                                      /var/log/auth.log
auth,authpriv.none;                             /var/log/syslog
```

## 1.10. Extrait de /var/log/syslog

```
Feb  3 18:02:52 ns1 ftpd[1051]: FTP session closed
Feb  3 18:03:31 ns1 syslogd 1.3-3: restart.
Feb  3 18:07:34 ns1 in.ftpd[1057]: refused connect from cli1.archinet.edu
Feb  3 18:07:46 ns1 in.ftpd[1058]: connect from ns1.archinet.edu
Feb  3 18:10:57 ns1 login[1063]: LOGIN ON ttyp3 BY mlx FROM puce
```

Remarques:

La commande kill -HUP pid de sysklogd permet de redémarrer ce processus avec prise en compte des paramètres se trouvant dans /etc/syslog.conf. Il est aussi possible d'invoquer le script de lancement du service en lui passant l'argument restart : /etc/init.d/syslogd restart