shh

par BENJAMIN PAVIE

Définition

SSH (**Secure SHell**) désigne un ensemble de programmes et de protocoles qui permettent de se connecter sur une machine distante de manière sécurisée.
Une implémentation libre de SSH est OpenSSH, livrée par une grande majorité de distributions (divers Unix, Linux et MacOSX).

Pourquoi

L'échange de clé de chiffrement en début de connexion permet durant la connexion d'échanger des trames chiffrées, empéchant l'utilisation d'un sniffer permettant de voir ce que fait l'utilisateur.
L'utilisation de **SSH** permet ainsi d'éviter la compromission des mots de passe, qui circulent en «clair» sur le réseau. En outre, celà oblige à disposer d'une authentification renforcée des machines, pas seulement basée sur le nom ou l'adresse IP (qui peuvent être falsifés).
SSH permet donc d'exécuter en toute sécurité des commandes à distance (d'où le mot « shell »), de transférer des fichiers en toute sécurité et de sécuriser les sessions X11.
Pour toutes ces raisons, le protocole **SSH** permet de remplacer efficacement les programmes rlogin, telnet et rsh.

Explications

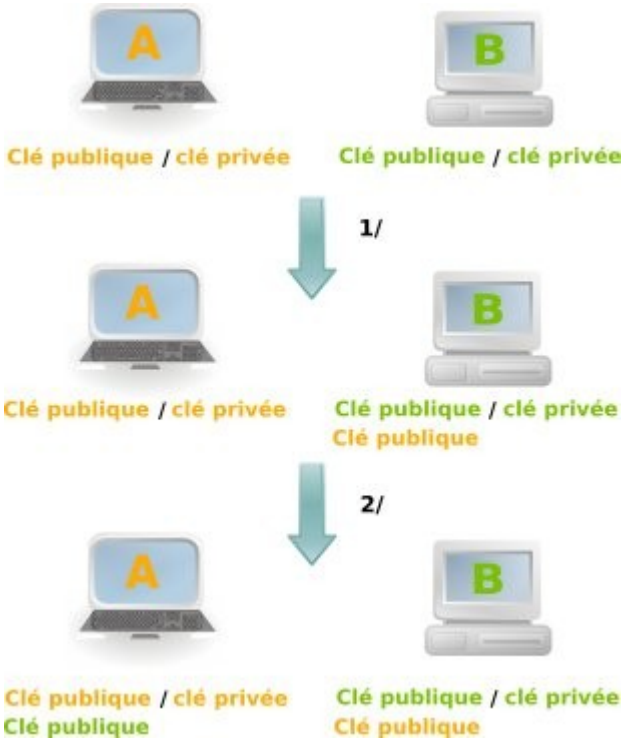
Les méthodes d'authentification :

Par mot de passe

Lors de la connexion, après avoir décliné son identité, l'utilisateur est invité à entrer un mot de passe qui est transmit au serveur, qui le compare à une empreinte de celui associé à l'utilisateur. Le mot de passe en clair est encapsulé dans une communication secrète, et devient «inviolable» sur le réseau. L'apport de **SSH** ici est que tout va fonctionner à l'identique par rapport à telnet, sauf que le mot de passe ne circule pas en clair.
Une variante plus sécurisé est l'utilisation de «mots de passe à usage unique».

Par clés publiques

Principe du chiffrement asymétrique :



1/ **A** envoit sa clé publique à **B**
2/ **B** envoit sa clé publique à **A**

A peut crypter les données destinées à la machine **B** avec la clé publique **B**.
La machine **B** peut décrypter ces données avec sa propre clé privée.
A peut donc maintenant envoyer en toute sécurité des données à **B**.

B peut crypter les données destinées à la machine **A** avec la clé publique **A**.
La machine **A** peut décrypter ces données avec sa propre clé privée.
B peut donc maintenant envoyer en toute sécurité des données à **A**.

A et **B** peuvent maintenant communiquer de manière chiffrée.

Les 3 étapes principales pour établir une connexion sont :

- **Génération d'un «bi-clé asymétrique»** (c'est à dire un couple clé-privée/clé-publique RSA ou DSA) sur une machine, en général la machine cliente (si on utilise plusieurs machines clientes, on effectue cette génération en général sur une seule d'entre elles, puis on recopie les clés sur les autres). Ce bi-clé est stocké dans un sous-répertoire (par exemple *~/.ssh* avec OpenSSH) de l'utilisateur.
- **Copie de la clé publique sur les machines serveur** sur lesquelles on souhaite pouvoir utiliser cette authentification. Cela consiste à ajouter la ligne correspondant à la clé publique présente dans le répertoire ou elle a été générée (par exemple, le contenu du fichier *~/.ssh/id_dsa.pub*) dans un fichier du serveur situé dans le répertoire (par exemple *~/.ssh* avec OpenSSH) de l'utilisateur : le nom de ce fichier dépend de la version de **SSH**, souvent *authorized_keys* ou *authorization*.
- **Entrée de la «passphrase»** au moment de le séquence de connexion. Afin d'automatiser l'entrée de la «passphrase», il est possible d'utiliser un utilitaire permettant de ne la rentrée qu'une seule fois par session locale : ssh-agent.

Configuration du serveur OpenSSH

Editez le fichier /etc/ssh/sshd_config. Vous devriez voir apparaitre un ensemble de directives toutes commentées. En fait celles-ci vous informent sur les valeurs par défaut, il ne faut décommenter l'une d'entre elles que lorsque vous souhaitez en changer la valeur. Il faut être root pour modifier la configuration du serveur :

```
user1@mon_poste$ su root
#
```

signifiant que vous êtes root.

Vous pouvez éditer le fichier /etc/ssh/sshd_config à l'aide d'un éditeur de texte (par exemple vim).

Principales configurations possibles

```
ListenAddress 192.168.0.1
```

Cette option permet de faire écouter le démon du serveur OpenSSH, **sshd**, que sur une interface donnée.
Si par exemple vous possédez 2 cartes réseaux, une permettant une connexion sur le réseau internet (avec une adresse non locale) et une autre permettant une connexion locale (par exemple avec une adresse 192.168.0.1), et que vous souhaitez que les accés SSH ne se fassent qu'en provenance du réseau local entrez votre I.P. locale. Dans ce cas, les adresses extérieurs à votre réseau locale n'auront pas accés à votre serveur OpenSSH.

```
PermitRootLogin no
```

Cette option permet d'autoriser ou non une connection au serveur ssh avec le login root.
Il est fortement conseillé pour des raison de sécurité d'interdire ceci, afin d'éviter les attaques par la force brute via SSH.
Par défaut, cette option est sur « no ».

`Port 666` ou `ListenAddress 192.168.0.1:666`

Change le port d'écoute du serveur ssh (par défaut, le serveur ssh écoute sur le port 22).
C'est appliqué la sécurité par l'obscurité (l'attaquant potentielle cherchera préférentiellement à attaquer sur le port 22).

```
PermitEmptyPasswords no
```

Cette option permet d'autoriser ou non des connexion avec un couple identifiant/mot de passe MAIS en autorisant que le mot de passe soit vide. Ceci est une aberration du point de vue sécurité.
Par défaut, cette option est sur « no ».

```
AllowUsers user1 user2
```

Autorise seulement certains utilisateurs à avoir accès via ssh à cette machine. `user@host` peut également être utilisé pour n'autoriser l'accès qu'à un utilisateur donné depuis un hôte donné.

```
AllowGroups wheel admin
```

Autorise seulement certains membres de groupes à avoir accès via SSH à cette machine.
AllowGroups et AllowUsers ont des directives équivalentes pour interdire l'accès à la machine.

```
PubkeyAuthentication yes
```

Autoriser ou non la connexion par clé publique.
Par défaut, cette option est sur « yes ».

```
AuthorizedKeysFile .ssh/authorized_keys
```

Indique le chemin vers le fichier cntenant les clés autorisée pour l'authentification par clé publique.
Le chemin peut être absolu (de type /mon/chemin/fichier) ou relatif au home de l'utilisateur (c'est le cas si l'on met .ssh/authorized_keys qui est équivalent à /home/user1/.ssh/authorized_keys où user1 est un utilisateur).
Par défaut, cette option est sur « yes ».

```
PasswordAuthentication yes
```

Cette option permet d'autoriser ou non des connexion avec un couple identifiant/mot de passe.
Il est plus sûr d'autoriser l'accès à la machine uniquement aux utilisateurs avec des clés SSH placées dans le fichier `~/.ssh/authorized_keys`.
Si c'est ce que vous voulez, positionnez cette option à « no ».

Désactiver toute forme d'autorisation dont vous n'avez pas réellement besoin si vous n'utilisez pas, par exemple :

```
RhostsRSAAuthentication
HostbasedAuthentication
KerberosAuthentication
RhostsAuthentication
```

Vous devriez les désactiver même s'ils le sont déjà par défaut (voir la page de manuel sshd_config.5).

`Protocole 2`

Désactiver le protocole version 1, car il a des défauts de conception qui facilite le crack de mots de passe. Pour plus d'informations, lisez un article concernant les problèmes du protocole ssh ou le bulletin d'alerte Xforce.

`Banner /chemin/`*`un_fichier`*

Ajouter une bannière (elle sera récupérée du fichier) pour les utilisateurs se connectant au serveur OpenSSH. Dans certains pays, envoyer un avertissement avant l'accès à un système donné avertissant des accès non autorisés ou du suivi des utilisateurs devrait être ajouté pour avoir une protection légale.

Vous pouvez également restreindre l'accès au serveur OpenSSH en utilisant `pam_listfile` ou `pam_wheel` dans le fichier de contrôle PAM. Par exemple, vous pourriez bloquer tous les utilisateurs qui ne sont pas dans `/etc/loginusers` en ajoutant cette ligne à `/etc/pam.d/ssh` :

```
auth required pam_listfile.so sense=allow onerr=fail item=user
file=/etc/loginusers
```

Exemple pour une connexion par mot de passe seulement

```
Port 22
Protocol 2
ListenAddress votre_ip
ServerKeyBits 1024
PermitRootLogin no
PubkeyAuthentication no
IgnoreRhosts yes
PasswordAuthentication yes
UsePAM yes
Compression yes
```

Exemple pour une connexion par authentification via la clé publique

```
Port 22
Protocol 2
ListenAddress votre_ip
ServerKeyBits 1024
PermitRootLogin no
PubkeyAuthentication yes
AuthorizedKeysFile .ssh/authorized_keys
IgnoreRhosts yes
# Mettez ces 2 options à « no » si vous ne voulez
# autoriserl'accès qu'aux utilisateurs ayant
# enregistré leur clés sur votre serveur :
PasswordAuthentication yes
UsePAM yes
Compression yes
```

Renforcer la protection de votre serveur ssh

Afin d'éviter les attaques par force brut (SSH Brute Force Attacks), il peut être utile de paramétrer le pare-feu afin de limiter les tentatives de connexions par minute.

Sous GNU/linux : utilisation d'iptables

Ajouter la ligne suivante permettant de d'ajouter une nouvelle chaîne utilisateur. Par la suite, des règles peuvent êtres ajoutées à ces chaînes, grâce au paramètre "-A". Ceci permettra de créer une chaine d'utilisateur à laquelle ne s'applique pas le filtrage appelé **SSH_WHITELIST**.

```
iptables -N SSH_WHITELIST
```

Ensuite, on peut ajouter des utilisateurs à cette liste (USER1 peut etre une adresse IP xxx.xxx.xxx.xxx):

```
iptables -A SSH_WHITELIST -s USER1 -m recent --remove --name SSH -j
ACCEPT
```

Ensuite, ajoutez les 4 règles suivantes qui limiterons les connexions entrantes sur le port 22 à un maximum 4 tentatives par minutes, les autres étant ensuite directement "jetées" (dropped) :

```
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --set --name SSH
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j SSH_WHITELIST
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --update --seconds
60 --hitcount 4 --rttl --name SSH -j ULOG --ulog-prefix SSH_brute_force
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --update --seconds
60 --hitcount 4 --rttl --name SSH -j DROP
```

Explications :
--state NEW permet de signifier à iptable que seul les nouvelles connexions sont concernées.
--set à la première ligne fera que l'adresse IP de l'hote qui a initié la connexion sera ajoutée à la "liste récente", où elle pourra être testée et utilisée plus tard, par exemple ici par les règles qui suivent.
-j précise la cible de la régle : ici, nous utilisons une cible définie par nous appelé **SSH_WHITELIST**.
-m recent indique à Netfilter qu'il faut utiliser le module **"recent"** afin d'analyser cette règle.
Le module **"state"** permet le suivi de connexion (conntrack).
Le module **"recent"** permet de créer dynamiquement une liste d'adresse IP et d'ensuite de travailler sur cette liste d'adresse IP.
--update test si les les adresses IP sui sont dans la liste des connexions récentes, dans notre cas chaque connexion sur le port 22 seront dans cette liste du fait de l'utilisation de **--set** dans la 1ère régle.
--seconds les adresses IP seront traitées si la dernière connexion était dans le laps de temps précisé (ici, 60 secondes).
--hitcount les adresses IP seront traitées si le nombre de tenative de connexion est supérieur au nombre que nous avons choisi (ici, 4).
--rttl prend en compte les datagram TTL lors du traitement des paquets pour tenter

de pallier les adresses sources "parodiées" (spoofed).
-j ULOG --ulog-prefix SSH_brute_force permet que les tentatives répérées soient précédées dans les logs de la mention "SSH_brute_force".

Si vous ne souhaitez bloquer que les accés provenant du réseau extérieur et laisser libre les accés provenant de votre réseau privé, vous pouvez rajouter un paramètre permettant de spécifier l'interface à surveiller, par exemple : `-i eth0` si vous souhaitez établir cette limite de connexion uniquement sur l'interface eth0.

Sous freeBSD/OpenBSD : utilisation de PacketFilter

Il suffit d'ajouter dans le fichier `/etc/pf.conf` :

```
table persist
block in quick from
pass in quick on $external inet proto tcp from any to any port ssh flags S/SA keep
state ( max-src-conn-rate 4/60 ,overload flush global)
```

Explications :
ligne 1 : créer une table qui va conserver les adresses IP des attaquants.
ligne 2 : bloque tout ce qui provient de ces adresses IP.
ligne 3 : autoriser les connexions ssh si il y a moins de 4 tentatives de connexion en 1 minutes (60 secondes), et sinon on enregistre l'adresse IP dans la table, et on détruit toutes les connexions correspondant à cette adresse IP.

Configuration du client OpenSSH

La configuration des clients SSH dépend bien sur de la configuration du serveur. Si vous avez opté pour une authentification par mot de passe (voir installation du serveur OpenSSH), la configuration du client se limite à la création d'une paire de clés.

Dans le cas où vous souhaitez mettre en place une authentification par clé publique quelques manipulations supplémentaires seront nécessaires.

Le client SSH est fourni avec le paquetage OpenSSH, chaque client devra donc installer le même paquetage que le serveur.

Modification du fichier de configuration

Le fichier de configuration se trouve par défaut dans `$HOME/.ssh/config`.
Pour plus d'information concernat les possibilité de modification de ce fichier, reportez vous à la page de man sshd_config.5.

Création d'une paire de clé : ssh-keygen

Pour créer une parie de clé, nous allons utiliser l'exécutable **ssh-keygen**.
Plusieurs options sont disponibles :

-t : (pour *type*)
Précise le type de clé à créer. Les valeurs possibles sont « rsa1 » pour la version 1 du

protocole(non conseillé) et « rsa » ou « dsa » pour la version 2 du protocole.

-b : (pour *bits*)
Spécifie le nombre de bits de la clef à créer. Le minimum est 512 bits. En général, on considère qu'une longueur de 1024 bits est suffisante, et des longueurs de clés supérieures n'améliorent pas la sécurité mais ralentissent le tout. Par défaut 1024.

-f : (pour *filename*)
Précise le nom du fichier de clé. Par défaut, l'on met *id_dsa* si l'on choisit une clé de type « dsa » et *id_rsa* si l'on choisit une clé de type « rsa ».

```
user1@mon_poste ~/.ssh $ ssh-keygen -t dsa -b 1024 -f id_dsa
Generating public/private dsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_dsa.
Your public key has been saved in id_dsa.pub.
The key fingerprint is:
6a:98:70:33:b0:61:02:0c:eb:2d:d5:a6:c8:4c:24:23 user1@mon_poste
```

Pour les deux algorithmes (dsa, rsa), le système nous demande une « passphrase ». Celle-ci est un « mot de passe amélioré », car non limité à un mot ou une petite suite de caractéres. En effet, elle peut permettre via un autre programme, **ssh-add**, de vous authentifier une seule fois puis ensuite de vous connecter à différent serveurs sans vous re-authentifier à chaque fois (ceci est bien sur valable uniquement pour les connexions par clé publique). Il faut cependant prendre des précautions, car en cas de perte de la « passphrase », vous ne pourriez plus vous authentifier en tant que propriétaire authentique.

Authentification uniquement par mot de passe

La première des méthodes, la plus connue, repose sur le modèle employé par rlogin ou rsh : l'hôte distant demande un mot de passe pour s'assurer de votre identité.

Pour se connecter au serveur, on utilise le programme **ssh** fournit par openSSH.

```
user1@mon_poste ~/.ssh $ ssh user1@Serveur
The authenticity of host 'Serveur (192.168.0.1)' can't be established.
RSA key fingerprint is 33:93:84:34:59:8f:2a:d6:4a:fb:51:27:12:36:53:ac.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'Serveur,192.168.0.112' (RSA) to the list of known hosts.
user1@Serveur's password: <votre mot de passe>
Last login: Thu May 1 17:41:29 2005 from 192.168.0.20
Linux Serveur 2.4.24 #1 Wed Feb 11 01:01:56 CET 2004 i686 GNU/Linux
user1@Serveur:~$
```

Si le port du serveur SSH n'est pas par défaut le port 22 mais par exemple 123, il vous faut préciser ce port en ajoutant l'option : `-p 123`

C'est seulement lors de la première connexion que l'on nous demande de répondre par yes ou par no de façon à ajouter la clé publique du serveur dans notre fichier `~/.ssh/known_hosts`. Si le serveur venait à changer de clé, le programme **ssh** refusera de s'y connecter en vous signalent que la correspondance machine/clé n'est plus valable pour le serveur en question. Il vous suffit alors d'éditer à la main votre fichier `~/.ssh/known_hosts` et de supprimer l'entrée concernant le serveur. Re-itérer alors votre demande de connexion, vous devriez obtenir le même comportement que lors d'une nouvelle connexion.

Authentification par clé publique ou par mot de passe

L'avantage de cette méthode est qu'elle permet une connexion sécurisée avec un serveur distant sans demande de mot de passe, ainsi il vous est possible d'écire des scripts (ou des programmes) profitant de cela pour se connecter automatiquement à un serveur afin d'effectuer diverses opérations. Il s'agit d'un mode de connexion non interactif.

Une fois les clés générées, il vous faut les placer dans votre répertoire personel sur le serveur. Vous pouvez faire ce transfert soit grâce à un media (CDROM/clé USB etc...) ou bien utiliser le protocole FTP, ou encore demander à un administrateur de le faire pour vous.
Une fois votre clé copiée sur le serveur au bon endroit créez le fichier `authorized_keys` comme suit :

```
user1@Serveur:~/.ssh$ cat id_dsa.pub >> authorized_keys
```

Puis donnez les permissions correctes à ce fichier :

```
user1@Serveur:~/.ssh$ chmod 400 authorized_keys
```

Votre répertoire personel sur le serveur doit au plus avoir comme permission 755, sinon l'authentification par clé ne fonctionnera pas !

Vous pouvez maintenant vous connecter, l'hôte distant vous reconnaît.
Vous pouvez vous connecter sans mot de passe et avec une « passphrase » si vous en avez entrée une (ce qui est fortement conseillé).

L'agent d'authentification : ssh-agent

Vous savez maintenant vous connecter à un hôte distant connaîssant votre identité par l'intermédiaire de votre clé publique. Nous avons vu précédement que vous étiez libre de mettre ou non une « passphrase » afin de chiffrer votre identification et ainsi de compliquer l'usurpation qui pourrait en être faite (ce qui est quand même fortement conseillé).
S'il paraît fastidieux d'insérer un mot de passe à chaque connexion, cela l'est d'autant plus lorsque l'on doit rentrer une phrase entière. L'agent ssh est là pour nous simplifier la vie : lancé au début de la session (terminal ou graphique), il retient la « passphrase » le temps où **ssh-agent** sera actif (le temps d'une session).

- Pour une session en mode terminal :

```
user1@mon_poste ~/.ssh $ ssh-agent screen
user1@mon_poste ~/.ssh $ ssh-add
Enter passphrase for /home/user1/.ssh/id_dsa: Entrez votre mot de passe
Identity added: /home/user1/.ssh/id_dsa (/home/user1/.ssh/id_dsa)
user1@mon_poste ~/.ssh $
```

Après avoir lancé l'agent ssh, on ajoute la « passphrase » à l'agent par l'intermédiaire de **ssh-add**.
Tous les hôtes distants disposant de votre clef publique ne vous demanderons alors plus la « passphrase ».

- De même, en mode graphique :

```
user1@mon_poste$ ssh-agent startx
```

Il vous suffit ensuite d'ouvrir un terminal et de taper :

```
user1@mon_poste ~/.ssh $ ssh-add
Enter passphrase for /home/user1/.ssh/id_dsa: Entrez votre mot de passe
Identity added: /home/user1/.ssh/id_dsa (/home/user1/.ssh/id_dsa)
user1@mon_poste ~/.ssh $
```

L'agent sera actif pour toutes les applications utilisées en mode graphique.

- Une demière méthode consiste à lancer l'agent-ssh qui fournit un certains nombre de variables à exporter, puis demander l'action de ssh-add :

```
user1@mon_poste$ ssh-agent
```

On récupère les variables fournies et on les exporte :

```
$SSH_AUTH_SOCK=/tmp/ssh-XXy0hiXW/agent.2019; export SSH_AUTH_SOCK;
$SSH_AGENT_PID=2020; export SSH_AGENT_PID;
$echo Agent pid 2020;
```

puis on tappe :

```
user1@mon_poste ~/.ssh $ ssh-add
Enter passphrase for /home/user1/.ssh/id_dsa: Entrez votre mot de passe
Identity added: /home/user1/.ssh/id_dsa (/home/user1/.ssh/id_dsa)
user1@mon_poste ~/.ssh $
```

Pour tuer l'agent ssh, il suffit de faire :

```
user1@mon_poste$ ssh-agent -k
unset SSH_AUTH_SOCK;
unset SSH_AGENT_PID;
echo Agent pid 2020 killed;
Invalidité des clefs d'un hôte connu
```

- Pour généraliser le mécanisme à l'ensemble de votre session X window, il est nécessaire que ces 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).

Celà peut 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 (par exemple, Gnome), il ressemblera à ceci :

```
#!/bin/sh
ssh-agent -s > /tmp/ssh.hskepzj
. /tmp/ssh.hskepzj
rm /tmp/ssh.hskepzj
#lancement de votre gestionnaire de fenêtres, ici Gnome
exec gnome-session
```

Obtenir un shell sur le serveur : ssh

Pour obtenir un shell sur le serveur, vous devez utiliser le programme **ssh** fournit par OpenSSH (ou un autre client de votre choix, voir la liste ici).

Plusieurs options sont disponibles en voici quelques unes :

-p (pour *port*)
Port à connecter sur la machine distante.
On peut aussi le spécifier pour une machine donnée dans le fichier de configuration (cf la page de man de sshd_config.5).

-X
Active le transfert X11. On peut aussi le spécifier pour une machine donnée dans le fichier de configuration.
Le transfert X11 doit être activé avec précaution. En effet, les utilisateurs capables de contourner les permissions des fichiers sur la machines distante (pour accéder à la base de donnée d'accréditation de X) peuvent accéder au à l'écran X11 local via le transfert de connexion. Un attaquant pourrait alors effectuer des opérations telles qu'enregistrer la frappe.

-x
Désactive le transfert X11.

Exemple :

```
user1@mon_poste ~/.ssh $ ssh user1@Serveur
Last login: Thu May 1 17:41:29 2005 from 192.168.0.20
Linux Serveur 2.4.24 #1 Wed Feb 11 01:01:56 CET 2004 i686 GNU/Linux
user1@Serveur:~$
```

Copier des fichiers

SSH permet de copier de manière sécurisée des fichiers ou des répertoire depuis le serveur vers le client ou bien depuis le client vers le serveur en utilisant la commande `scp` .

Pour celà, nous pouvons utilisez plusieurs options :

-r
Copie récursivement des répertoires entiers.

-v
Mode bavard. scp affichent des messages d'information et de débogage sur ce qu'il fait. Utile pour le débogage des problèmes de connexion, d'authentification et de configuration.

D'autre options sont disponibles, voir la traduction du man de scp par Laurent GAUTROT ici .

Exemple : copie de fichiers

Copie d'un fichier vers le serveur :

```
 user1@A ~/$ scp /chemin/vers/mon/fichier/toto.txt
user@chezmoi.com:/chemin/de/destination/
```

Copie de plusieurs fichiers vers le serveur :

```
 user1@A ~/$ scp fichier1.txt fichier2.txt user@chezmoi.com:/chemin/de/destination/
```

Copie d'un répertoire :

```
 user1@A ~/$ scp -r mon_repertoire user@chezmoi.com:/chemin/de/destination/
```

Copie d'un fichier contenant des espaces :

```
 user1@A ~/$ scp "\"/chemin/Mon fichier avec espace.txt\""
user@chezmoi.com:/chemin/de/destination/
```

Copie d'un fichier contenant deux points (cf http://www.openssh.com/fr/faq.html#2.13) :

```
 user1@A ~/$ scp ./chemin/Mon_fichier_avec_deux:points.txt
user@chezmoi.com:/chemin/de/destination/
```

ou bien

```
 user1@A ~/$ scp ./chemin/Mon_fichier_avec_deux\:points.txt
user@chezmoi.com:/chemin/de/destination/
```

Rediriger les ports avec ssh

Le principe des tunnels SSH Tunnels est de permettre d'atteindre des machines auxquelles vous n'avez pas normalement accès, par exemple, des machines sur un réseau local à votre entreprise ou derrière un firewall. Il vous faut néanmoin posséder un accès à une machine connectée au réseau que vous voulez atteindre (en général un firewall). SSH peut créer 2 types de tunnels, appelé "redirections locales" et "redirections distantes".

Pour celà, nous devons utilisez plusieurs options de **ssh** :

-f
Demande à ssh de basculer en arrière-plan juste avant d'exécuter la commande.

-N
N'exécute aucune commande distante. Utilisé pour les transferts de ports (seulement dans la version 2 du protocole).

-C

Active la compression de toutes les données en gzip

-g
Permet à des machines distantes de se connecter à des ports transférés locaux.

-R: `R port:host:hostport`
Spécifie que le port donné `hostport` de la machine distante `host` sera transféré au port `port` de la machine locale. Ceci fonctionne grâce à l'allocation d'une socket qui écoute sur le port `hostport` de la machine distante `host`, et qui, dès qu'une connexion est établie sur ce port, la transfère à travers le canal sécurisé, et se connecte sur le port `port` de la machine locale.

-L `L port:host:hostport`
Spécifie que le port donné de la machine locale (client) sera transféré vers l'hôte et le port donné depuis la machine distante. Ceci fonctionne grâce à l'allocation d'une socket qui écoute sur le port `port` de la machine locale, et qui, dès qu'une connexion est établie sur ce port, la transfère à travers le canal sécurisé, et se connecte à `host` sur le port `hostport` depuis la machine distante.
Seul root peut transférer des ports privilégiés.

Exemple 1 : Redirection distante de ports (-R Remote)

On veut, par exemple, que le poste **chezmoi.com** puisse se connecter au poste **A** possédant l'adresse locale 192.168.0.2 (qui se trouve dans un réseau privé, et est donc normalement inaccessible).

Dans cet exemple, il faut le poste **A** et le poste **chezmoi.com** aient lancés chacun leurs **serveurs SSH**.

Depuis le poste **A** faire la commande suivante :

```
user1@A ~/$ ssh -fN -C -g -R 2222:127.0.0.1:22 user@chezmoi.com
```

Ceci ouvre le port 2222 sur **user@chezmoi.com** et le redirectionne sur le port 22 du poste **A**.

Maintenant, depuis **chezmoi.com**, vous avez accé au poste **A** en faisant :

```
user@chezmoi.com ~/$ ssh -p 2222 user1@127.0.0.1
```

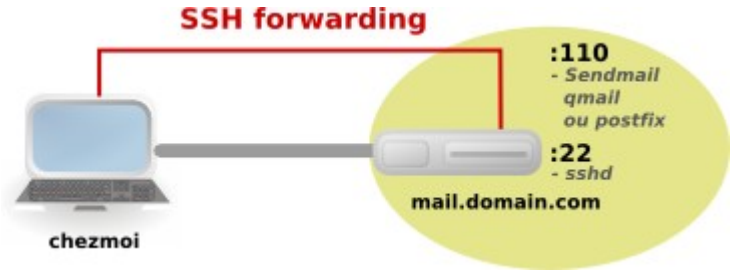
```
user1@A ~/$
```

Donc maintenant, quand je me connecte depuis **chezmoi.com** sur l'adresse locale (127.0.0.1), je suis redirectionné sur le poste **A** : la connexion via le port 22 sur mon adresse locale (127.0.0.1) me redirectionne vers le port 22 du poste **A**.

Exemple 2 : Redirection locale de ports (-L Locale) :

On veut, par exemple, créer un tunnel ssh pour encapsuler le protole **POP** afin de sécuriser le transfert du mot de passe et les mails. Ceci peut être si par exemple vous accéder à votre compte mail depuis une connexion wifi, afin d'éviter une interception de votre mot de passe.

La seule restriction, mais elle est importante, est que le serveur d'accéder aux e-mails possè également un serveur ssh.



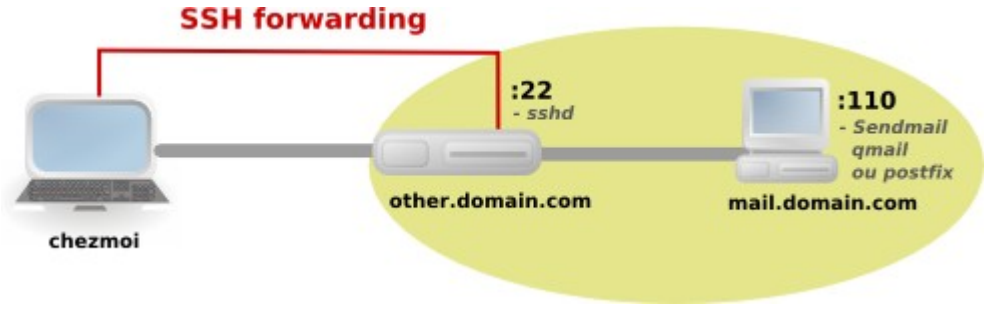
Supposons que votre serveur de mail se nomme **mail.domain.com**, et que votre poste se nomme **chezmoi** :

Depuis le poste **chezmoi** faire la commande suivante :

```
user1@chezmoi ~/# ssh -L 110:mail.domain.com:110 mail.domain.com
```

Donc ici, vous transférez votre requête POP du port 110 de **chezmoi** au moyen de la connexion SSH au port 22 de **mail.domain.com**. Ensuite, **serveur.mail.com** se connecte à son port 110 pour vous permettre de vérifier votre courrier.
Ensuite, au lieu de spécifier "mail.domain.com" comme nom de serveur POP3, vous allez maintenant entrer "localhost".

Si **serveur.mail.com** n'exécute aucun démon de serveur **SSH**, mais que vous pouvez tout de même vous connecter par SSH à un ordinateur tout près, au moyen d'un pare-feu par exemple, vous pouvez quand même utiliser SSH pour rendre sécurisée la partie de la connexion POP qui est faite sur un réseau public.



Dans ce cas, la commande est légèrement différente :

```
user1@chezmoi ~/# ssh -L 110:mail.domain.com:110 other.domain.com
```

Dans cet exemple, vous transférez votre requête POP du port 110 de votre ordinateur (**chezmoi**) au moyen de la connexion SSH au port 22 de **other.domain.com**. Ensuite, **other.domain.com** se connecte au port 110 de **mail.domain.com**. pour vous permettre de vérifier votre courrier. Seule la connexion entre vous et **other.domain.com** est sécurisée, mais dans nombre de cas, cela suffit pour acheminer des informations sur un réseau public de façon sécurisée et pour vous offrir plus de sécurité qu'auparavant.