OpenSSH-Server

1. OpenSSH - Client serveur SSH

SSH signifie "Secure Shell", il s'agit donc d'un "shell" dit "sécurisé". Un shell va permettre de dialoguer avec une machine ou un serveur via l'exécution de différentes commandes qui retourneront des informations.
Un peu d'histoire :
Autrefois, d'autres protocoles étaient utilisés pour accéder à distance à un serveur Linux. Le protocole Telnet a pendant longtemps été utilisé, il permet également d'accéder à distance à une machine Linux, mais Telnet est aujourd'hui délaissé au profit de SSH et cela pour une raison très simple : son manque de sécurité.

En effet, Telnet était un protocole qui faisait tout passer en clair sur le réseau. Cela signifie que lorsque l'on se connectait à un serveur Linux à distance et que l'on fournissait à ce serveur nos identifiants, ceux-ci transitaient en texte claire. Un intrus se trouvant sur la route des paquets qui transitaient entre un serveur et un client communiquant en telnet était alors capable de voir ces identifiants, et les autres messages transportés.
On ne pourra jamais empêcher quelqu'un d'écouter les communications qui transitent par un réseau, il faut alors faire en sorte que celui qui écoute à notre insu ne comprenne pas ce qu'il voit. SSH a principalement été créé pour répondre à cette problématique de confidentialité. En effet avec SSH, tous les échanges entre le client et le serveur sont chiffrés.

SSH a été créé en 1995 par Tatu Ylönen, d'abord dans sa version 1, qui a ensuite connue des problèmes de sécurité, SSH est passé ensuite, et est toujours, en version 2. Cette seconde version a été normalisée en 2006 par l'IETF (Internet Engineering Task Force).

Le fait que le protocole SSH soit normalisé par l'IETF signifie que c'est un protocole stable, qui est considéré comme une "norme", le but de l'IETF étant de développer et de maintenir les normes de l'Internet. La standardisation d'un protocole ou d'une technologie est toujours un signe de fiabilité et de durabilité.

Administration distante et sécurisée
Nous avons vu ce qu'était SSH, un protocole, mais aussi une suite d'application permettant l'accès à distance à des machines Linux par des clients. Les clients utilisent donc un logiciel sur leurs ordinateurs qui va initialiser une connexion vers un serveur Linux, le plus souvent distant, pour aller y saisir des commandes. On obtient alors un shell distant sur un serveur.

Transfert de fichier sécurisé
SSH permet également le transfert de fichier entre des machines de manière sécurisée. À l'instar du protocole Telnet dont nous avons parlé plus haut, le protocole FTP permettant le transfert de fichier entre deux machines, n'est pas sécurisé. Il laisse en effet passer les informations de connexion, mais également les fichiers transférés, en clair sur le réseau. On est alors capable de récupérer, en plus des identifiants, le contenu des fichiers. Le protocole SSH permet de sécuriser les transferts de fichier, notamment via la commande *SCP*, le *SFTP* ou encore le *SSHFS* :

•**SCP** : Secure Copy
•**SFTP** : SSH File Transfert Protocol
•**SSHFS** : SSH File System

X11 Forwarding
Une autre utilisation qui peut être faite du protocole SSH est le X Forwarding. Sous Linux, "X" est souvent employé pour parler du mode graphique ou GUI. Le protocole SSH permet en effet d'effectuer un transfert graphique du serveur vers le client. Autrement dit, on pourra lancer une application graphique sur le serveur, par exemple une visionneuse d'image, et la fenêtre et son contenu apparaîtra sur le poste du client, le tout étant transféré via SSH entre le serveur et le client.

Tunneling et encapsulation de protocole :
Le protocole SSH peut permettre l'encapsulation de n'importe quel protocole. Ainsi, un protocole non chiffré comme HTTP peut être encapsulé (comprendre "emballé", comme un objet dans un colis) dans un le protocole SSH pour passer un pare-feu ou un proxy par exemple. Il sera ensuite désencapsulé à la destination ("déballé").

OpenSSH utilise le protocole SSH pour l'**authentification des utilisateurs**. Ce protocole prend en charge plusieurs méthodes d'authentification, notamment l'authentification par **mot de passe**, l'authentification par **clé publique**, l'**authentification par clé SSH**, l'**authentification basée sur un certificat**, etc. Ces méthodes permettent de vérifier l'identité des utilisateurs et de sécuriser l'accès au serveur.

OpenSSH utilise également le protocole SSH pour le **chiffrement des données** transitant entre le client et le serveur. Le protocole SSH utilise des algorithmes de chiffrement **symétriques** tels que **AES** (Advanced Encryption Standard), des algorithmes de chiffrement asymétriques tels que RSA (Rivest-Shamir-Adleman) et des algorithmes de hachage tels que SHA (Secure Hash Algorithm) pour garantir la confidentialité et l'intégrité des données échangées.

Le protocole Diffie-Hellman est utilisé dans le cadre du **processus d'échange de clés** de la phase d'initialisation de la connexion SSH. Il permet aux deux parties de générer de manière sécurisée un **secret partagé** sans avoir besoin de transmettre directement ce secret sur le réseau, ce qui réduit les risques d'interception par des tiers malveillants.

Installation d'un serveur SSH
Vérifier si Openssh est installer ? : $ apt-cache policy openssh-server

```
openssh-server:
  Installé : 1:8.4p1-5+deb11u3
  Candidat : 1:8.4p1-5+deb11u3
 Table de version :
 *** 1:8.4p1-5+deb11u3 500
        500 http://deb.debian.org/debian-security bullseye-security/main amd64 Packages
        100 /var/lib/dpkg/status
     1:8.4p1-5+deb11u2 500
        500 http://deb.debian.org/debian bullseye/main amd64 Packages
```

Une autre façon de vérifier l'installation d'OpenSSH aurait été de vérifier quel le fichier /etc/ssh/ssh**d**_config est bien présent sur le système. Il s'agit du fichier de configuration du serveur SSH sur les distributions Linux.

Installer OpenSSH
Si ce n'est pas le cas, installons-le :
$ sudo apt-get update
$sudo apt-get install openssh-server
Connaître la version de ssh : srv@srv:~$ ssh -V
OpenSSH_8.4p1 Debian-5+deb11u3, OpenSSL 1.1.1n 15 Mar 2022

Dans tous les cas, nous pourrons voir l'état du service SSH avec la commande "**systemctl**", le serveur est-il en fonctionnement (actif, activé) ou non ? :

```
$ systemctl status sshd
```

```
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: e>
     Active: active (running) since Wed 2024-03-27 17:45:30 CET; 43min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
    Process: 499 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Main PID: 519 (sshd)
      Tasks: 1 (limit: 2321)
     Memory: 5.1M
        CPU: 857ms
     CGroup: /system.slice/ssh.service
             └─519 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
```

Sinon, il suffira de le démarrer avec la commande : **$ sudo systemctl start sshd**

Le service SSH écoute par défaut sur le port 22 en TCP. Nous pouvons alors voir, sur notre serveur Linux, que le service est bien en écoute sur ce port :

```
$ ss -lntp |grep ":22"
LISTEN   0        128              0.0.0.0:22             0.0.0.0:*
LISTEN   0        128                 [::]:22                [::]:*
```

Ici, la commande "**ss**" (anciennement connue sous le nom de "***netstat***") permet de visualiser l'état des ports et des connexions du serveur sous les systèmes Unix :

- "*-l*" permet de ne lister que les ports en écoute,
- "*-n*" permet d'afficher les ports de manière numérique, et non leur translation habituelle (exemple "*telnet*" à la place de "*23*")
- "*-t*" permet de ne lister que les ports TCP,
- "-p" permet enfin de lister les processus derrière chaque ports :

→ Ici, on voit donc que le port *22* est bien écoute, en *IPv4* (0.0.0.0:22) et en *IPv6* ([::]:22.

Connexion client-serveur SSH (192.168.1.22)

```
anonymous@debian:~$ ssh serveur@192.168.1.22
The authenticity of host '192.168.1.22 (192.168.1.22)' can't be established.
ECDSA key fingerprint is SHA256:V8+Lc7tIn59ZhYfrO5O7Sfs/v++TWl2nuzI+8jW3jF8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.22' (ECDSA) to the list of known hosts.
serveur@192.168.1.22's password:
Linux serveur 4.19.0-16-amd64 #1 SMP Debian 4.19.181-1 (2021-03-19) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Apr 27 13:14:27 2021 from 192.168.1.23
```

*On peut préciser le login et numéro du port :***$ ssh *192.168.1.22* -l *user* -p *22***

Remarque : Depuis les dernières versions de Debian, il est impossible de s'authentifier en root via SSH lorsqu'OpenSSH est configuré par défaut. Il s'agit d'une mesure de sécurité et n'essayez donc pas de vous authentifier en tant que root, préféré donc un autre utilisateur présent sur votre système.

→ **Putty** est l'outil le plus couramment utilisé, n'oubliez pas qu'il s'agit d'un logiciel client pour Windows et que d'autre logiciels existe comme **Kitty** ou **TeraTerm**.

OpenSSH : Configuration du serveur SSH

Le répertoire du service *OpenSSH* se trouve dans */etc/ssh*, on pourra y trouver les fichiers suivants dans la configuration par défaut :

- ***moduli*** : Il s'agit d'un fichier contenant des informations utilisées par le système de chiffrement.

Contient une liste de nombres premiers et de générateurs appropriés pour être utilisés avec Diffie-Hellman. Ces nombres premiers et générateurs sont précalculés et inclus dans ce fichier pour des raisons de performance et de sécurité. Ils sont utilisés lors de la négociation de la clé secrète partagée pendant l'établissement de la connexion SSH.

Chaque ligne du fichier `moduli` contient trois valeurs : la longueur en bits du nombre premier, le nombre premier lui-même, et le générateur correspondant :

- **Time** : Il s'agit probablement du moment où les tests ont été effectués ou lorsque les paramètres ont été générés.
- **Type** : Cela pourrait indiquer le type de test utilisé pour vérifier les nombres premiers, par exemple, si des tests de primalité ont été utilisés.
- **Tests** : Nombre de tests de primalité effectués.
- **Tries** : Nombre de tentatives pour trouver un nombre premier satisfaisant les critères.
- **Size** : La longueur en bits du nombre premier.
- **Generator** : Le générateur utilisé dans le protocole Diffie-Hellman.
- **Modulus** : Le nombre premier lui-même utilisé dans le protocole Diffie-Hellman.

- ***ssh_config :*** Il s'agit du fichier de configuration du client (*openssh*-client).
- ***ssh_config.d :*** *même chose que ssh_config mais de manière* modulaire qui facilite la gestion de la configuration SSH en permettant aux administrateurs de séparer les différents aspects de la configuration dans des fichiers distincts, ce qui peut être plus organisé et plus facile à gérer, notamment dans des environnements complexes.
- ***sshd_config*** : Il s'agit du fichier de configuration du serveur OpenSSH.
- ***ssh_host_**** : Il s'agit de fichier contenant certaines clés qui seront utilisées lors des différents processus de chiffrement.
- **ssh_host_ecdsa_key** et **ssh_host_ecdsa_key.pub**

Le fichier ssh_host_ecdsa_key.pub contient la clé publique associée à la clé privée ECDSA utilisée par le serveur SSH pour permettre aux clients de vérifier son identité lors de la connexion.

ECDSA (Elliptic Curve Digital Signature Algorithm) est un algorithme de **signature numérique** basé sur les courbes elliptiques. Dans le contexte de SSH, il est utilisé pour l'authentification du serveur.

- **ssh_host_ed25519_key** et **ssh_host_ed25519_key.pub**

Ed25519 est un algorithme de **signature numérique** basé sur les courbes elliptiques

- **ssh_host_rsa_key** et **ssh_host_rsa_key.pub**

RSA (Rivest-Shamir-Adleman) est un algorithme de cryptographie asymétrique largement utilisé. Dans le contexte de SSH, RSA est utilisé pour l'**authentification du serveur**.

Pour prendre en compte les modifications de configuration : **$ sudo systemctl restart sshd**

Note : Certains processus et services acceptent le "reload", il s'agit alors de recharger la configuration d'une application sans la redémarrer complètement, cela **évite de couper les connexions en cours** par exemple, ce qui est plus pratique. C'est notamment le cas de Apache ou d'OpenSSH : **$ sudo systemctl reload sshd**

Changer le port d'écoute de SSH

Par défaut, le service OpenSSH écoute sur le port TCP 22. Cependant, une bonne pratique de sécurité consiste à changer le port d'écoute du serveur SSH.

Il est important d'éviter de configurer notre port sur la plage des plages réservées. Pour rappel, il s'agit des ports du numéro 0 à 1023. Mettons par exemple notre service SSH en écoute sur le port *7256*, ainsi un attaquant souhaitant s'en prendre à notre service SSH aura déjà plus de difficulté à le trouver, ralentissant ainsi ses attaques.

Dans le fichier de configuration *sshd_config modifier* la ligne suivante :
Port 22 par Port port *7256* *puis redémarrer ou recharger le service SSH*

Une fois que le service redémarré, vérifier que le serveur est bien à l'écoute sur le nouveau port avec la commande "ss" :

```
$ ss -lntp |grep ":7256" |column -t
LISTEN  0  128  0.0.0.0:7256  0.0.0.0:*
LISTEN  0  128  [::]:7256     [::]:*
```

Note : la commande "| column -t" n'impacte pas la sortie de la commande "ss", il s'agit juste d'avoir un meilleur affichage.

Gérer l'IP d'écoute du serveur
Il peut arriver que votre serveur Linux ait plusieurs interfaces réseau. C'est par exemple le cas même lorsque l'on dispose d'une seule interface réseau car le serveur écoute sur l'IPv4 mais aussi sur l'IPv6.

Supposons que le serveur possèdent deux adresses IP : 192.168.10.131 ou 192.168.240.132. On souhaite n'écouter que sur le port 7256 de l'interface eth1 qui possède l'IP 192.168.240.132. Dans le fichier de configuration sshd_config, rajouter la ligne suivante :

ListenAddress 192.168.240.132

Ensuite sauvegarder la modification et redémarrer le service SSH.

```
$ ss -lntp |grep "7256" |column -t
LISTEN  0  128  192.168.240.132:7256  0.0.0.0:*
```

Autorisation et permission, gestion des groupes et utilisateurs
La base d'utilisateur utilisée par SSH est la base utilisateur Linux (/etc/passwd).

L'utilisateur root possède tous les droits sur le serveur. Par sécurité, l'authentification en "root" est interdite depuis les connexions SSH dans la configuration par défaut d'OpenSSH : # permitRootLogin yes (permitRootLogin no)

Permissions et interdictions des groupes et utilisateurs

Il est possible de gérer de façon très précise qui parmi les utilisateurs pourra se connecter en SSH via les directives "AllowUsers" et "AllowGroups". Dans le fichier de configuration *sshd_config modifier* la ligne suivante (autoriser juste u_1 et u_2):

allowUsers u_1 u_2

Une autre possibilité est de gérer cela à l'aide des groupes utilisateurs Linux :
$ sudo addgroup ssh_allowed
$ sudo usermod -a -G ssh_allowed u_1
$ sudo usermod -a -G ssh_allowed u_2

Ici, l'option "-a" permet de garder les groupes auxquels l'utilisateur appartient déjà, "-G" permet de spécifier le nouveau groupe dont fera parti l'utilisateur. Puis on ajoutera dans la configuration OpenSSH la ligne suivante :

allowGroups ssh_allowed
allowUsers u_3

On peut effectuer l'opération inverse en refusant explicitement certains utilisateurs ou certains groupes et en acceptant tous les autres. La politique est alors "tout ce qui n'est pas explicitement interdit est autorisé" (une liste noire) :

denyUsers u_4 u_5
denyGroups ssh_not_allowed

Gestion des logs

Les logs se trouvent dans le fichier /var/log/auth.log, mais dans ce fichier, il n'y a pas que les logs SSH qui s'y trouvent :

```
$ sudo nano /var/log/auth.log
...
Apr 25 18:15:53 debian sshd[737]: Server listening on 0.0.0.0 port 22.
Apr 25 18:15:53 debian sshd[737]: Server listening on :: port 22.
Apr 25 18:32:25 debian sshd[737]: Received signal 15; terminating.
…..
COMMAND=/usr/bin/systemctl restart sshd
Apr 28 01:59:23 debian sshd[11479]: Received signal 15; terminating.
Apr 28 01:59:23 debian sshd[11551]: Server listening on 0.0.0.0 port 7256.
Apr 28 01:59:23 debian sshd[11551]: Server listening on :: port 7256.
…
Apr 28 02:58:00 debian sshd[17378]: Accepted password for anonymous from ::1 port 52990 ssh2
Apr 28 02:58:00 debian sshd[17378]: pam_unix(sshd:session): session opened for user anonymous by (uid=0)
...
Apr 28 02:58:02 debian sshd[17378]: pam_unix(sshd:session): session closed for user anonymous
```

Bannière de connexion SSH

Lorsqu'une connexion en SSH ou en terminal "physique" à un serveur est établie, il y aura un message qui s'affiche juste après que vous vous soyez authentifié.

Ce message est appelé "motd" pour "Message Of the Day". Ce MOTD par défaut peut être personnalisé afin d'afficher des informations utiles ou adresser un message important à l'utilisateur.

Les modifications de ce paramètre s'effectuent non pas dans la configuration SSH mais la configuration du système. La bannière d'origine ou modifiée ne s'affiche pas seulement lors d'une connexion SSH mais également lors de l'ouverture d'une session en terminal directement sur la machine physique. On doit donc se rendre dans le fichier "/etc/motd" pour y trouver le message par défaut :

```
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
```

Pour modifier le message qui sera affiché lors d'une connexion , il suffit de modifier le contenu de ce fichier, par exemple :

```
==================================================
Connexion établie avec succès.
Merci de respecter la politique de sécurité.
Tout accès par une personne non habilité est interdit.
==================================================
```

Exemple :

```
 base) <anonym@debian:>ssh srv@192.168.1.22 -p 1234
srv@192.168.1.22's password:
Linux srv.lsi.lan 5.10.0-21-amd64 #1 SMP Debian 5.10.162-1 (2023-01-21) x86_64
==================================================
Connexion établie avec succès.
```

```
Merci de respecter la politique de sécurité.
Tout accès par une personne non habilité est interdit.
==========================================================
Last login: Thu Apr 13 01:34:12 2023 from 192.168.1.1
srv@srv:~$
```

Nous pouvons également voir le "LastLog", c'est la date et l'heure de la dernière connexion de cet utilisateur en SSH ainsi que l'IP du client ayant établie la connexion. Cette information est affichée par défaut par OpenSSH. Pour supprimer cette notification, on se rend dans le fichier sshd_config pour y trouver la ligne suivante : PrintLastLog yes (On remplacera simplement "yes" par "no".)

2. Transfert de fichier via SSH

Il existe différentes méthodes pour transférer des fichiers d'une machine à une autre via le protocole SSH :

- SCP
- SFTP
- SSHFS

Le plus souvent, SCP est SFTP sont utilisés, SSHFS correspond à des besoins plus précis et spécifiques.

2.1 SCP : Secure Copy

SCP est une commande fournis par le paquet "*openssh-client*". Cette commande permet de façon très simple d'échanger des fichiers et des dossiers entre un client SSH et un serveur SSH.

Copie d'un fichier depuis le répertoire courant vers un répertoire du serveur:

> scp -P 7256 Fichier login@serveur:Chemin

Copie d'un répertoire, avec éventuellement ses sous-répertoires, vers un répertoire du serveur

> scp -r -P 7256 Repertoire login@serveur:Chemin

Copie d'un fichier du serveur vers le répertoire courant

> scp -P 7256 login@serveur:Chemin/Fichier .

Copie d'un répertoire du serveur vers le répertoire courant:

> scp -r -P 7256 login@serveur:Chemin/Repertoire .

Exemples

Envoi d'un fichier via SSH en utilisant SCP :
$scp /home/$u_1$/index.php root@192.168.1.131:/var/www/

Cette commande permet d'envoyer le fichier */home/u_1/index.php* dans le répertoire */var/www* du serveur Linux. Le "*:*" permet de spécifier le chemin dans la machine distante. S'il n'est pas spécifié, les fichiers atterriront dans le dossier par défaut de l'utilisateur : la home. (*/root* pour l'utilisateur *root*, */home/user* pour l'utilisateur "*user*", etc.).

Téléchargement d'un fichier via SSH en utilisant SCP
Pour télécharger un fichier depuis le serveur Linux vers la *Machine de l'utilisateur u_1 la commande est:*
*$*scp root@192.168.1.131:/var/www/index.php /home/u_1

Transfert de dossier : utilisation de la récursivité :
$scp -r /home/$u_1$/site root@192.168.1.131:/var/www/

Note : Il faut spécifier à SCP le port sur lequel aller discuter (si ce n'est pas 22), cela se fait via l'option "*-P*", à spécifier avant la source et la destination :
$scp -r -P 7256 /home/$u_1$/index.php root@192.168.1.131:/var/www/

Utilisation d'un client Windows
Si votre client est une machine *Windows*, vous pouvez utiliser le logiciel *WinSCP* qui se chargera de faire le même type de transfert avec une interface graphique.

2.2 SFTP : Secure FTP

SFTP (*Secure FTP*) est un dérivé du protocole "*FTP*" (*File Transfert Protocol*). *SFTP* est une extension du protocole *SSH*, plus techniquement un "*sub-system*" au niveau /etc/ssh/sshd-config:

subsystem sftp /usr/lib/openssh/sftp-server

On peut décrire le *SFTP* comme l'encapsulation du protocole *FTP* dans une "couche" sécurisée qu'est *SSH*. Autrement dit, en *SFTP*, client et serveur dialoguent en *FTP* mais utilisent *SSH* pour faire transiter les paquets, ce qui permet de sécuriser l'utilisation du *FTP*.

En l'état, il existe peu de différence entre utiliser *SCP* ou *SFTP*. Avec un client comme *WinSCP* sous *Windows*, l'effet sera exactement le même et l'utilisation aussi. Vous n'aurez qu'à choisir "*SFTP*" au lieu de "*SCP*" lors du remplissage des informations de connexion.
Sous Linux, en tant que client, la commande "*sftp*" s'utilise exactement comme la commande "*scp*" :
$ sftp -P 7256 srv@192.168.1.22:/home/srv/tt .

2.3 SSHFS : SSH FileSystem

SSHFS "*SSH File System*" permet de monter, en temps réel, un répertoire d'une machine Linux sur une autre, de la même manière que l'on peut monter un répertoire SMB entre deux machines.

Installation (au niveau client): $sudo apt-get install sshfs
client@client$ mkdir /*home/*client/partage
client@client$ sshfs -r -p 1234 srv@192.168.1.22:/home/srv/tt /*home/*client/partage

client@client:~$ mount |grep partage
srv@192.168.1.22:/home/srv/tt on /home/client/partage type fuse.sshfs
(rw,nosuid,nodev,relatime,user_id=1001,group_id=1001)

Remarques :

- Si vous obtenez l'erreur obscure suivante :«**fuse: bad mount point … »**, il faut vous connecter au moins une fois au serveur distant via ssh. Ceci permettra d'ajouter la clé

- Il est recommandé de démonter le montage effectué une fois terminé. C'est-à-dire de couper la liaison *SSHFS* entre les deux machines.
  $ sudo umount /*home/*client/partage ou $ sudo fusermount -u /*home/*client/partage

- Note : **FUSE**, abréviation de *Filesystem in Userspace*) permet d'implémenter toutes les fonctionnalités d'un système de fichier dans un espace utilisateur.

3. SSH : X11 Forwarding

Configuration X11 Forwarding coté serveur

Du côté du serveur, nous aurons seulement à autoriser le déport d'affichage dans notre configuration SSH. Nous allons donc, dans notre fichier de configuration SSH "*/etc/ssh/sshd_config*" repérer les lignes suivantes :

X11Forwarding yes
X11DisplayOffset 10

On s'assurera que le paramètre "*X11Forwarding*" est à "*yes*", n'oubliez pas de redémarrer votre service SSH si une modification a dû être faite ("*service sshd restart*").

Le paramètre "*X11Forwarding*" permet donc d'autoriser ou non l'utilisation du déport d'affichage.

Test:lancer depuis le client ssh : xclock , xterm et firefox

Méthode 1
au niveau client : $ ssh -X -p 7256 serveur@192.168.1.22
L'option **-X** de SSH permet de se connecter à distance sur une machine et lancer des applications en mode graphique. (tester les commandes : xterm, firefox et xclock, *xcalc*, *xman*, ..) :

Méthode 2 : Utiliser des environnements de bureau à distance les plus couramment utilisés avec SSH :

- VNC (Virtual Network Computing)
- RDP (Remote Desktop Protocol)
- NX (Nomachine X)
- XRDP (X Remote Desktop Protocol)

4. Authentification SSH par clés

Il est également possible d'utiliser un autre mécanisme d'authentification, en l'occurrence, l'authentification par clé. Cela est parfois préféré car la clé permet de ne pas avoir à retenir systématiquement des mots passe différents. Nous allons ici voir le fonctionnement général de cette méthode d'authentification.

La plupart du temps la génération de la clé se fait sous Linux sans trop de problème étant donné qu'*OpenSSH* y est nativement présent. Sous *Windows* toutefois, quelques manipulations sont à effectuer afin de générer et d'envoyer notre clé au serveur.

L'authentification par clés se fait donc via une paire de clés, le client va donc générer une paire de clés, une publique et une privée. Il va bien entendu garder sa clé privée pour lui et envoyer la clé publique au serveur SSH qui la stockera dans un endroit prévu cet effet.

4.1. Génération de la clé

Étant donné qu'un client SSH peut être un client Linux ou un client Windows, nous allons voir comment générer cette paire de clés sous Linux en ligne de commande, et sous Windows via PuttyGen.
Générer une paire de clés sous Linux : $ ssh-keygen
Si aucune option n'est spécifiée, une clé RSA de 2048 bits sera créée, ce qui est acceptable aujourd'hui en termes de sécurité. Si vous souhaitez spécifier une autre taille de clé, vous pouvez utiliser l'option "-b" : $ ssh-keygen -b 4096 (Il va ensuite nous demander de saisir une passphrase)

→ Par défaut, la clé va être stockée dans le répertoire .ssh/ de l'utilisateur courant
Il va ensuite nous être proposé de saisir une passphrase
Néanmoins, si un tiers parvient à nous dérober notre clé privée, il arrivera à se connecter aux serveurs sans mot de passe. Ainsi, une passphrase permet la protection de notre clé privée via un mot de passe, ou plutôt une phrase de passe ("passphrase"). L'avantage par rapport à un mot de passe SSH est que vous n'avez qu'un mot de passe à retenir, celui de votre clé privée et pas un mot de passe par serveur SSH.

Une fois créées, vous pourrez donc voir vos clés dans le répertoire ".ssh" de l'utilisateur :
~$ ls -al .ssh
….
-rw------- 1 anonymous anonymous 1876 mai 19 00:26 **id_rsa**
-rw-r--r-- 1 anonymous anonymous 398 mai 19 00:26 **id_rsa.pub**
-rw-r--r-- 1 anonymous anonymous 666 avril 30 00:41 **known_hosts**

- *known_hosts*": il s'agit d'un fichier permettant d'identifier un serveur.
  Si vous vous connectez en SSH à plusieurs serveurs depuis votre utilisateur (compte utilisé), votre fichier *known_host* va peu à peu se remplir. Cela entraîne entre autre le fait qu'une demande validation de la clé serveur est demandée à la première connexion du serveur mais pas lors des connexions suivantes.
- Le fichier en .pub est la clé publique
- Le fichier en id_rsa est la clé privée

Générer une paire de clés sous Windows

La plupart des connexions SSH sous Windows sont initialisées avec le client Putty qui est simple et efficace. L'outil PuttyGen permet de générer facilement un jeu de clés qui nous permettra de nous authentifier sur un serveur en les utilisant.
Page de télechargement de PuttyGen :
http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

4.2. Mettre la clé sur le serveur

Il faut maintenant envoyer/mettre la clé publique sur le serveur afin de pouvoir s'y authentifier :

$ ssh-copy-id root@192.168.1.22

Notez que par défaut, cette commande va envoyer la clé publique ".ssh/id_rsa.pub". On peut néanmoins, si l'on possède plusieurs paires de clés, spécifier la clé publique à envoyer au serveur, par exemple :

$ ssh-copy-id -i ~/.ssh/id_rsa_1.pub root@192.168.1.22

Maintenant que notre clé publique est présente sur le serveur que nous possédons dans notre répertoire .ssh sa clé privée associée, nous allons pouvoir nous connecter au serveur juste en saisissant la commande suivante : $ ssh utilisateur@serveur

La passphrase de la clé publique nous sera demandée, puis on sera connecté en SSH à notre serveur.

L'avantage est le suivant : si cette même clé publique est envoyée à 30 autres serveurs, la passphrase sera la même pour toutes les connexions alors qu'il faudra retenir 30 mots de passe différents si vous utilisez la méthode d'authentification habituelle par mot de passe.

→ Pour que la passphrase soit toujours demandée, vous devez utiliser l'agent SSH :
au niveau client ssh : $ ssh-add -t 0 -c /home/srv/.ssh/id_rsa (t 0 : vous devez configurer une durée de vie infinie pour la clé privée dans l'agent SSH. Cela garantira que la passphrase est toujours requise à chaque utilisation de la clé privée.

4.4. Authentification SSH par clé avec Putty

Putty a maintenant besoin de savoir où se situe notre clé privée pour établir une communication valable avec le serveur. Après l'avoir lancé et avoir renseigné l'IP de notre serveur, nous nous rendrons dans la partie "Connexion" > "SSH" > "Auth" puis nous cliquerons sur "Browse" pour aller chercher notre clé privée à côté du champ "Private key file for authentication" .