Leçon 10 : Administration à distance et installation d'applications

10.1 Introduction à l'administration à distance

Administrer un serveur Linux à distance peut se faire de plusieurs façons. La plus simple est l'accès en mode terminal (ie. *ligne de commande*) au serveur, en utilisant SSH. Ce service, très utile pour démarrer des tâches à distance est installé par défaut dans le plupart des distributions. Bien sûr, à côté de cette possibilité de connexion *en mode texte*, il existe diverses méthodes permettant également un accès *en mode graphique* à des serveurs Linux, même depuis un poste client Windows.

Ainsi, parmi les solutions très courantes, il y a l'antique XDMCP véritable précurseur dans le domaine, qui permettait à des terminaux très léger de se connecter à un serveur puissant. Une autre solution, répandue également dans le monde Microsoft, est l'utilisation de VNC, cependant la sécurité assez basique de ce protocole le rend inutilisable depuis internet sans la configuration de VPN ou autre mécanisme de sécurité avancé. Enfin, il y a le service NX et son évolution X2Go qui permet de se connecter en mode graphique à un serveur Linux en se basant sur le service SSH.

10.2 Configuration et utilisation de SSH

Le service SSH est un des services les plus déployés sur les machines exécutant Linux. Il est même activé par défaut dans la plupart des distributions, ce qui n'est pas nécessairement une bonne idée[30]. Cependant, grâce à ce service, il est possible de :

1. Se connecter, de manière interactive et sécurisée, à un serveur distant. Sous Windows, l'utilisation de Putty permet de réaliser cette tâche.
2. Echanger des fichiers en utilisant le protocole SCP (*secure copy*) dont nous parlerons plus tard. Sous Windows, des logiciels comme WinSCP ou FileZilla supportent ce protocole.
3. Démarrer une commande à distance, même à partir d'un langage de programmation. Ainsi, il est possible de lancer un script, déployé sur un serveur, pour réaliser des tâches particulières[31]. Citons, par exemple, **TamirSSH** dans l'environnement .NET ou encore **JSch** dans l'environnement Java qui permettent de réaliser cela.

La configuration de SSH se trouve dans le dossier `/etc/ssh` et est scindée en 2 parties distinctes :

1. La configuration **du client SSH** qui se trouve dans `/etc/ssh/ssh_config`. Celle-ci est utilisée lorsqu'on se connecte, depuis la machine linux courante vers un serveur extérieur.
2. La configuration du **service SSH** qui se trouve dans `/etc/ssh/sshd_config`. Celle-ci est utilisée lorsqu'une connexion est établie depuis une autre machine vers ce serveur, en utilisant SSH.

Nous allons dans la suite aborder la configuration *serveur* c'est-à-dire, celle **du service SSH**.

Ainsi, les options suivantes, qui peuvent être placées dans le fichier sshd_config, peuvent être intéressantes :

Option	Explication
`Port 22`	C'est le port d'écoute du service SSH. Le port 22 est, par

[30] En effet, quelle est la réelle utilité d'avoir un service SSH exécuté sur un PC de bureau ?
[31] A HELMo, cette fonctionnalité est utilisée pour la création automatique des boîtes mails utilisateurs lors de l'inscription d'un nouvel étudiant. Un programme .NET appelle un script PERL via SSH en utilisant TamirSSH.

	défaut, dédié à ce service. Il est parfois intéressant (ou nécessaire) de modifier le port par défaut.
ListenAddress 0.0.0.0 **ListenAddress ::**	Ces options permettent de déterminer sur quelle adresse IP le service SSH est actif. Par défaut, il est actif sur l'ensemble des adresses IP (IPv4 et IPv6) de la machine. Si la machine dispose de plusieurs interfaces réseaux, on peut ainsi limiter l'accès au service SSH à une seule adresse.
Protocol 2	Il existe 2 version du protocole SSH. La version 1 est complètement dépassée et non-sûre. Il est donc recommandé de ne jamais activer cette version.
PermitRootLogin yes	Cette option permet à l'administrateur (i.e. l'utilisateur root) de se connecter à distance. Une bonne pratique est de désactiver cette possibilité. Ainsi pour se connecter au serveur, il faut connaître un couple nom d'utilisateur / mot de passe. Il est ensuite possible de *devenir* administrateur grâce à la commande `su`.
PermitEmptyPasswords no	Cette option désactive tous les comptes qui ont un mot de passe vide, ce qui est une très bonne idée. Cette configuration est faite par défaut.
X11Forwarding yes	Permet d'activer le lancement d'application graphique à distance. Cette option est intéressante pour administrer un serveur, et lancer une application dont l'affichage se fait localement mais l'exécution à distance. Pour utiliser cette option, il faut disposer d'un serveur X11. Ce serveur est présent dans les systèmes Unix principalement.
AllowTcpForwarding yes	Cette option autorise la création de tunnel SSH. Pour introduire ce sujet (sur lequel nous reviendrons par la suite), un tunnel SSH permet de se connecter au serveur « *comme si* » une connexion VPN existait. Cela permet de traverser le firewall pour accéder à des ressources internes.

10.2.1 Démarrer / Arrêter le service SSH

Pour démarrer le service SSH, il faut passer par la commande `systemctl` :

```
$ systemctl restart sshd
```

Pour désactiver le service SSH (sur des postes clients) :

```
$ systemctl disable sshd
```

10.2.2 Authentification dans SSH

Le service SSH, comme bien d'autres, utilise les modules PAM (*Pluggable Authentication Modules*) pour authentifier les utilisateurs. PAM est intéressant car il permet de généraliser l'utilisation d'autres mécanismes d'authentification que le traditionnel login / mot de passe. Ainsi, il est possible d'utiliser *Kerberos* (serveur de distribution de clés), *S/Key* (utilisation de mots de passe à usage unique), ou *même l'authentification à 2 facteurs* (Google Authenticator PAM module, Yubico PAM…).

A côté de PAM, le service SSH supporte également l'authentification par clé RSA ou DSA. Cette méthode d'authentification est simple :

1. On génère sur le poste client une paire de clé publique/privée.
2. On ajoute la clé publique dans les clés autorisées pour le compte visé
3. L'authentification n'utilise plus le mot de passe mais la clé privée correspondant à la clé publique déposée.

Etape 1 : générer les clés

La génération de la paire de clés publiques / privées se fait par l'utilisation de la commande `ssh-keygen`. Lors de l'utilisation de la commande, il faut mentionner le type de clé souhaité.

Type de clé	Explication
rsa	Utilisation d'une paire de clés RSA. Ce sont les clés les plus courantes utilisées. Dans un avenir proche, ces clés polynomiales seront remplacées par des clés basées sur des courbes elliptiques
dsa	Utilisation d'une paire de clés DSA. Ces clés sont moins courantes et sont basées sur des logarithmes discrets. Ce chiffrement cédera sa place aux clés basées sur des courbes élliptiques dans un avenir proche.
ecdsa	Standard sur les clés basées par courbes elliptiques. Leur déploiement est encore très restreint. Ainsi, le programme Putty ne supporte pas encore ces clés.
ed25519	Second standard basé sur les clés par courbes elliptiques. Le déploiement est toujours très restreint. Il n'y a pas de grande différence entre ecdsa et ed25519, ni en terme de sécurité, ni en terme de performance.

Actuellement, l'option **rsa** semble est celle qu'il faut privilégier :

```
$ ssh-keygen -t rsa
```

*Cette commande génère une clé **rsa** de 2048 bits. Il est possible de générer une clé plus longue, pour davantage de sécurité en ajoutant à la fin de la commande l'option suivante : `-b 4096`.*

Attention ! Lors de la génération de la clé, le système demande *un code de verrouillage* protégeant la clé. Ainsi, celle-ci ne peut être lue sans entrer le *code de verrouillage* correspondant. L'utilisation d'un code de verrouillage non-vide est recommandée.

Une fois la clé installée, le système ne demande plus le mot de passe pour se connecter mais le code de verrouillage de la clé.

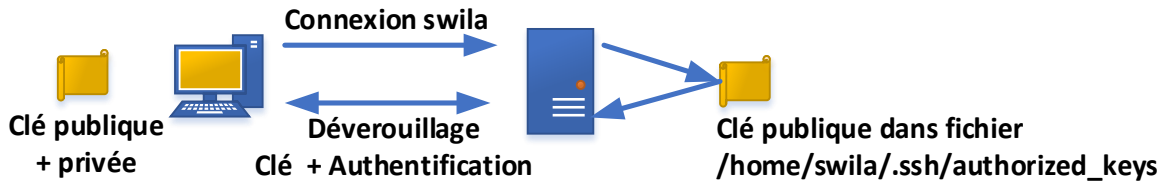

Figure 10.1 : Authentification SSH par clé publique / privée

La figure 10.1 résume le fonctionnement. Le client doit disposer de la clé publique et privée, le serveur distant dispose uniquement de la clé publique. Lors de la connexion, l'authentification utilise

la clé publique / privée au lieu du mot de passe : l'utilisateur est invité à entrer le code de verrouillage de la clé privée pour permettre à son ordinateur client de lire celle-ci.

Etape 2 : Ajout de la clé pour le compte visé

ATTENTION ! Comme nous l'avons annoncé, il faut ajouter la clé publique créée dans la liste des clés autorisées pour un compte donné. Nous avons également vu qu'il était recommandé de mentionner un *code de verrouillage* pour la clé en question. Il est par contre COMPLETEMENT DECONSEILLE DE DEFINIR UNE CLE SANS CODE DE VERROUILLAGE VERS L'UTILISATEUR ROOT.

Et d'une manière générale, l'authentification avec le compte *root* est déconseillée à distance. Il vaut toujours mieux se connecter avec un utilisateur et puis, en utilisant la commande `su`, devenir *root*.

Pour ajouter la clé crée sur la machine distante, il faut pouvoir s'y connecter (avec son login / mot de passe par exemple). La solution la plus simple consiste à utiliser le script fourni par SSH pour réaliser cette opération, à savoir `ssh-copy-id`[32]. Par exemple, à partir de la machine *CentOS7-Client :*

```
[swila@localhost ~]$ ssh-copy-id -i .ssh/id_rsa.pub lsw@172.18.1.2
```

Cette commande ajoute la clé contenue dans le fichier `id_rsa.pub` au compte `lsw` de la machine dont l'IP est `172.18.1.2`. Pour réaliser la copie, il faut entrer le mot de passe correspondant au compte `lsw`. Les authentifications ultérieures utiliseront la clé.

POUR DES RAISONS DE SECURITE, LA CLE PRIVEE DOIT RESTER SECRETE.

```
[swila@localhost ~]$ ssh lsw@172.18.1.2
```

Donc si nous revenons à la figure 10.1, il est possible d'expliquer le mécanisme d'authentification complètement désormais : lorsqu'une connexion est initiée (par exemple vers le compte `lsw` comme ci-dessus), le service SSH va vérifier si une (ou plusieurs) clé publique est présente dans le fichier `/home/lsw/.ssh/authorized_keys`. Si c'est le cas, et si le client dispose de la clé privée correspondante, alors l'authentification sur base de la clé publique / privée est initiée. Le système demande *le code de verrouillage* (si celui-ci existe) et authentifie l'utilisateur.

10.2.3 Code de verrouillage et agent

Comme mentionné plus haut, un code de verrouillage devrait toujours être présent pour protéger les clés privées. **Il y a cependant une exception à cette règle : les tâches planifiées**[33]. En effet, quand une tâche est planifiée et doit réaliser une opération à distance, le service SSH est un bon moyen d'y arriver. Cependant, devoir entrer un mot de passe ou un code de verrouillage est un frein au déploiement du service. Il est dans ce cas possible de ne pas utiliser de code de verrouillage. Cependant, la remarque précédente s'applique : pas de connexion vers le compte *root* sans code de verrouillage.

Ainsi, en cas d'utilisation de `rsync`, `scp` ou toute autre commande se basant sur SSH, utiliser une authentification par clé sans code est possible.

[32] Il est possible de réaliser l'opération à la main : il faut copier-coller le contenu du fichier `id_rsa.pub` dans le fichier `/home/<login>/.ssh/authorized_keys` sur le serveur distant.
[33] Par exemple, une sauvegarde effectuée automatiquement durant la nuit.

Comment faire pour ne pas devoir entrer son code de verrouillage tout le temps ?

Une question importante est de déterminer s'il est possible **de ne pas entrer tout le temps son code de verrouillage**. Il y a une façon simple, utiliser l'agent SSH.

L'agent SSH est un programme hautement sécurisé qui garde en mémoire une copie de la clé privée déverrouillée. Ainsi, lors de l'authentification, aucun code de verrouillage n'est nécessaire puisque la clé privée est déjà lue et disponible. Cette méthode facilite la vie de l'administrateur qui peut, ainsi, charger les clés en mémoire lors du lancement de sa machine et travailler sans avoir à entrer un seul mot de passe par la suite.

Le système *CentOS 7* lance automatiquement l'agent SSH. Ainsi, lors de la 1ère connexion utilisant la clé privée, le code de déverrouillage est demandé et la clé est gardée en mémoire.

Les connexions ultérieures ne nécessite plus de devoir entrer le code. La connexion est immédiate.

Si vous devez lancer l'agent à la main :

```
$ ssh-agent
```

Pour ajouter une clé en mémoire :

```
$ ssh-add ~/.ssh/id_rsa
```

10.2.4 Lancement d'application graphique à distance

Comme nous l'avons mentionné en introduction, il est possible de lancer l'exécution d'un programme à distance **mais avec un affichage local**. Pour ce faire, il faut que le serveur SSH distant ait activé l'option `X11Forwarding`, que le poste client dispose d'un serveur graphique X11 (ce qui est le cas sous la plupart des systèmes UNIX[34]) et que le client démarre sa session avec une option particulière :

```
$ ssh -X p010544@dartagnan.cg.helmo.be
$ xeyes
```

L'application `xeyes` est lancée depuis le serveur *dartagnan* mais est affichée localement.

10.2.5 Etablissement d'un tunnel SSH

En utilisant SSH, il est possible d'établir un tunnel vers un serveur distant et accéder à des applications qui ne sont pas disponibles sur Internet. Pour ce faire, il faut que le serveur soit configuré avec l'option `AllowTcpForwarding` activée et que le client dispose d'un compte SSH sur ce serveur.

Il est ainsi possible d'accéder à un site web depuis l'extérieur :

```
$ ssh -L 8000:dartagnan.cg.helmo.be:80 p010544@dartagnan.cg.helmo.be
```

Cette commande lance un accès SSH sur le serveur dartagnan mais établit, en plus, un tunnel SSH. Cela signifie que lorsqu'on accède à `http://localhost:8000`, la requête est passée dans le tunnel SSH pour être transmise à `dartagnan.cg.helmo.be` sur le port `80` (port web).

[34] Cette remarque n'est pas vraie pour MacOS X. Pour ce dernier, il est nécessaire de démarrer un serveur X particulier comme XQuartz

Cette commande permet donc de voir les sites web publiés par les étudiants et hébergés sur *dartagnan* sans devoir démarrer la connexion VPN. Le firewall de HELMo voit uniquement passer du trafic SSH sans se douter qu'il y a du trafic web à l'intérieur.

Le tunnel reste actif tant que la connexion SSH est lancée. Dès que celle-ci est déconnectée, le tunnel SSH est fermé en même temps.

10.3 Le service X2Go

Le service X2Go utilise SSH pour établir une connexion en mode graphique sur le serveur Linux. Le programme client est disponible sur les principales plateformes.

Toutes les informations d'installation et de configuration sont disponibles sur le site web du projet : http://www.x2go.org. Le serveur existe sous la forme de *package* pour les distributions RedHat /CentOS 7.

10.4 Installation d'applications

Les applications peuvent prendre deux formes distinctes : des ***applications précompilées et prévues pour la distribution utilisée***. Ainsi, les versions CentOS 7 utilise des *packages RPM* (présents aussi dans les distributions Suse, CentOS, RedHat, Scientific Linux ou encore Fedora). Si nous souhaitons installer un package pour CentOS 7, il faut qu'il soit étiqueté el7 (*enterprise linux 7)* ou *CentOS 7* afin d'être compatible. Avec d'autres distributions comme Ubuntu, ce sont des *packages DEB* (initialement prévu pour Debian, mais adopté par toutes les distributions qui sont basées sur celle-ci comme Ubuntu et toutes ses variantes).

La seconde forme pour l'installation d'application est ***de télécharger les sources du programme (en C, C++, …), de compiler celui-ci sur le système et de l'installer***. Cette manière peut sembler difficile d'un premier abord mais elle a l'avantage d'être indépendante de la distribution Linux présente.

10.4.1 Installation d'une application compilée

Avant de commencer, il convient de bien comprendre comment le système fonctionne : des dépôts contenant les applications compilées sont présents sur Internet. Ces dépôts sont utilisés pour installer de nouvelles applications ou encore maintenir le système à jour. Ainsi, les installations utilisent la commande `yum` :

```
$ yum upgrade
```

Cette commande vérifie si des packages installés doivent être mis à jour. Si c'est le cas, les nouvelles versions sont téléchargées et installées.

On peut demander à `yum` d'installer un programme particulier :

```
$ yum install xterm
```

Cette commande installe le package `xterm` (contenant un terminal graphique standard). Il faut noter que ***toutes les dépendances sont automatiquement installées***.

```
$ yum search php
```

Cette commande interroge les dépôts pour obtenir les packages référençant le mot php (soit dans le nom, soit dans la description).

```
$ yum install ./monfichier.rpm
```
Cette commande installe un package RPM dont le fichier a été téléchargé et stocké sur le disque dur. Une vérification des dépendances sera effectuée durant l'installation.

Les dépôts

Les dépôts utilisables peuvent être nombreux. Par exemple, *Adobe* propose un dépôt pour l'installation du plugin *flash* dans les systèmes EL7 (CentOS et Redhat). Les dépôts peuvent être nombreux sur un serveur, en fonction des programmes souhaités. Il est, cependant, toujours préférable de privilégier les dépôts officiels de la distribution pour l'installation d'un programme. Si celui-ci n'est pas disponible sur le dépôt, on peut alors ajouter un dépôt alternatif proposant le logiciel en question.

Les dépôts installés ont un fichier `.repo` installé dans le dossier `/etc/yum.repos.d`. Sur notre machine virtuelle, les dépôts suivants sont installés :

- CentOS
- EPEL
- Webmin

L'ajout d'un autre dépôt ajoutera un fichier dans ce dossier.

Le programme YumEx

La commande `yum` est une commande en mode texte. Il existe une version graphique de ce logiciel nommé `yumex` (*Yum Extender*). Celui-ci peut être installé depuis les dépôts :

```
$ yum install yumex
$ yumex
```

10.4.2 Installation depuis les sources

L'autre méthode pour installer une application est de télécharger son code source et de compiler celle-ci directement sur la machine qui est destinée à l'exécuter. Le processus est assez standardisé. Cependant, il est nécessaire de bien s'assurer que toutes les librairies nécessaires à la compilation du programme sont bien présentes (il n'y a pas de gestion *des dépendances quand on compile depuis les sources*).

La première étape est **de télécharger l'application depuis internet**. Le plus souvent, l'application est fournie dans une archive compressée en `.tar.gz` ou `.tar.bz2`. Il faut décompresser cette archive :
`$ tar xvzf monfichier.tar.gz` (par exemple)

L'étape suivante consiste à **compiler l'application** :
```
$ cd dossier_application
$ ./configure
$ make
```

Une fois l'application compilée, **il faut installer celle-ci sur le système (en root)**:
```
$ make install
```

Une fois l'application installée, elle est le plus souvent présente dans le dossier `/usr/local`.

10.5 Exercices

On vous demande de :

1. Changer le mot de passe du compte *root* par votre mot de passe HELMo

2. Ouvrir le port 22 sur votre firewall pfSense

3. Planifier une sauvegarde automatique en utilisant *rsync* de votre dossier `/home` vers la machine de votre voisin, en utilisant le compte qu'il vous avait créé (voir leçon précédente)
4. Configurer un tunnel SSH pour accéder au serveur web de votre voisin. Pour ce faire, rediriger le port local `8080` vers sa machine, sur le port `80`. Accéder à son site web en utilisant l'adresse `http://127.0.0.1:8080`.

5. En vous basant sur les informations du site http://www.if-not-true-then-false.com/2010/install-adobe-flash-player-10-on-fedora-centos-red-hat-rhel/, ajouter le dépôt *Adobe* et installer *flash player* sur votre machine.

6. En vous basant sur les informations présentes ici : http://wiki.x2go.org/doku.php, installer le serveur X2Go sur votre machine *CenOS7 Serveur* et un client sur l'autre. Tenter ensuite une connexion.
    a. Astuce : lors de la configuration du poste client, remplacer KDE ou GNOME par XFCE.