Module administration réseaux GNU/Linux

SSH avec OpenSSH

REFERENTIEL EMPLOI ACTIVITES COMPETENCES DU TITRE PROFESSIONNEL TSSR	
Activité type	Maintenir, exploiter et sécuriser une infrastructure centralisée
Compétences professionnelles	Maintenir et exploiter un serveur Linux

Compétences professionnelles	Maintenir et exploiter le réseau local et la téléphonie

Table des matières

1. Introduction et mise-en-garde

1.1 Qu'est-ce que SSH ?

SSH signifie ***Secure SHell***. C'est un protocole qui permet de faire des connexions sécurisées (i.e. chiffrées) entre un serveur et un client SSH. Nous allons utiliser le programme OpenSSH, qui est la version libre du client et du serveur SSH.

1.2 Mise en garde sur la sécurité

Nature du problème

Installer un serveur SSH permet aux utilisateurs d'accéder au système à distance, en rentrant leur login et leur mot de passe (ou avec un mécanisme de clefs).
Cela signifie aussi qu'un pirate peut essayer d'obtenir un compte sur le système (pour accéder à des fichiers sur le système ou pour utiliser le système comme une passerelle pour attaquer d'autres systèmes) en essayant plein de mots de passes différents pour un même login (il peut le faire de manière automatique en s'aidant d'un dictionnaire électronique). On appelle ça une attaque *en force brute*.

Il y a donc **trois contraintes majeures pour garder un système sécurisé** après avoir installé un serveur SSH :

- avoir un serveur SSH à jour au niveau de la sécurité, ce qui doit être le cas si vous faites consciencieusement les mises à jour de sécurité en suivant la procédure Debian,
- que les mots de passes de *TOUS* les utilisateurs soient suffisamment complexes pour résister à une attaque en force brute ;
- surveiller les connexions en lisant régulièrement le fichier de log **/var/log/auth.log**.

2. Le système de clefs de SSH

2.1 La théorie de la cryptographie asymétrique

SSH utilise la **cryptographie asymétrique** RSA ou DSA. En cryptographie asymétrique, chaque personne dispose d'un couple de clef : **une clé publique et une clef privée**. La clé publique peut être librement publiée tandis que la clef privée doit rester secrète. La connaissance de la clef publique ne permet pas d'en déduire la clé privée.

Si Alice veut envoyer un message confidentiel à Bob, elle doit le chiffrer avec la clef publique de Bob et lui envoyer sur un canal qui n'est pas forcément sécurisé. Seul Bob pourra déchiffrer ce message en utilisant sa clef privée.

2.2 La théorie de la cryptographie symétrique

SSH utilise également la **cryptographie symétrique**. Son principe est simple : si Alice veut envoyer un message confidentiel à Bob, Alice et Bob doivent d'abord posséder une même clef secrète. Alice chiffre le message avec la clé secrète puis l'envoie à Bob sur un canal qui n'est pas forcément sécurisé. Bob déchiffre alors le message grâce à la clef secrète. Toute autre personne en possession de la clef secrète peut également déchiffrer le message.

Les **algorithmes de chiffrement symétrique sont beaucoup moins gourmands en ressources processeur que ceux de chiffrement asymétrique**… mais le gros problème est

l'échange de la clef secrète entre Alice et Bob. Dans le protocole SSL, qui est utilisé par SSH et par les navigateurs Web, la cryptographie asymétrique est utilisée au début de la communication pour que Alice et Bob puissent s'échanger une clef secrète de manière sécurisée, puis la suite la communication est sécurisée grâce à la cryptographie symétrique en utilisant la clef secrète ainsi échangée.

2.3 L'établissement d'une connexion SSH

Un serveur SSH dispose d'un couple de clefs RSA stocké dans le répertoire /etc/ssh/ et généré lors de l'installation du serveur. Le fichier ssh_host_rsa_key contient la clef privée et a les permissions 600. Le fichier ssh_host_rsa_key.pub contient la clef publique et a les permissions 644.

Nous allons suivre par étapes l'établissement d'une connexion SSH :

1. Le serveur envoie sa **clef publique** au client. Celui-ci vérifie qu'il s'agit bien de la clef du serveur, s'il l'a déjà reçue lors d'une connexion précédente.

2. Le client génère une **clef secrète** et l'envoie au serveur, **en chiffrant l'échange avec la clef publique du serveur** (**chiffrement asymétrique**). Le serveur déchiffre cette clef secrète en utilisant sa clé privée, ce qui prouve qu'il est bien le vrai serveur.

3. Pour le prouver au client, il chiffre un message standard avec la **clef secrète** et l'envoie au client. Si le client retrouve le message standard en utilisant la clef secrète, il a la preuve que le serveur est bien le vrai serveur.

4. Une fois la clef secrète échangée, le client et le serveur peuvent alors établir un canal sécurisé grâce à la clef secrète commune (chiffrement symétrique).

5. Une fois que le canal sécurisé est en place, le client va pouvoir envoyer au serveur le login et le mot de passe de l'utilisateur pour vérification. La canal sécurisé reste en place jusqu'à ce que l'utilisateur se déconnecte.

La seule contrainte est de s'assurer que la clef publique présentée par le serveur est bien sa clef publique… sinon le client risque de se connecter à un faux serveur qui aurait pris l'adresse IP du vrai serveur (ou toute autre magouille).

Une bonne méthode est par exemple de demander à l'administrateur du serveur quelle est le *fingerprint* de la clef publique du serveur avant de s'y connecter pour la première fois. Le *fingerprint* d'une clef publique est une chaîne de 32 caractères hexadécimaux à peu près unique pour chaque clef (un hachage) ; il s'obtient grâce à la commande :

```
# ssh-keygen -lf /etc/ssh/ssh_host_rsa_key.pub
```

3. Installation et configuration de SSH

3.1 Installation du serveur SSH

Le client SSH est disponible dans le paquet *openssh-client*, qui est préinstallé.

Pour pouvoir vous connecter à distance, vous pouvez maintenant installer le serveur SSH :

```
# apt install openssh-server
```

L'installation comporte une étape de génération des clefs de chiffrement. Finalement, le serveur SSH se lance.

3.2 Configuration du serveur SSH

Le fichier de configuration du serveur SSH est **/etc/ssh/sshd_config**. À ne pas confondre avec le fichier **/etc/ssh/ssh_confi**g, qui est le fichier de configuration du client SSH.

Explication des lignes les plus importantes du fichier de configuration **/etc/ssh/sshd_config** :

- `Port 22`

  Signifie que le serveur SSH écoute sur le port 22, qui est le port par défaut de SSH. Vous pouvez le faire écouter sur un autre port en changeant cette ligne. Vous pouvez aussi le faire écouter sur plusieurs ports à la fois en rajoutant des lignes similaires.

- `PermitRootLogin` *`paramètre`*

  Si **yes** est passé en paramètre, cela signifie que vous pouvez vous connecter en root par SSH. La paramètre par défaut est en général ***no***, ce qui signifie que pour vous connecter en root à distance, vous devrez d'abord vous connecter par SSH en tant que simple utilisateur, puis utiliser la commande **su** pour devenir root. Sans cela, un pirate n'aurait qu'à trouver le mot de passe du compte root, alors que là, il doit trouver votre login *et* votre mot de passe.

- `Protocol 2`

  Ici le 2 pour SSH2. SSH peut utiliser 2 protocoles : SSH1 et SSH2, le premier étant moins fiable que le second nous privilégierons SSH2.

- `HostKey /etc/ssh/ssh_host...`

  Spécifie un fichier contenant une clef privée de machine utilisée par SSH. Par défaut /etc/ssh/ssh_host_key pour la version 1 du protocole, et /etc/ssh/ssh_host_rsa_key et /etc/ssh/ssh_host_dsa_key pour la version 2 du protocole. Si vous utilisez un client moderne vous pouvez commenter toutes les lignes HostKey sauf « HostKey /etc/ssh/ssh_host_ed25519_key ».

Si vous avez modifié le fichier de configuration du serveur, il faut lui dire de relire son fichier de configuration :

```
# systemctl reload sshd.service
```

4. Se connecter par SSH

4.1 Installation du client SSH

```
# apt install openssh-client
```

4.2 Authentification par mot de passe

C'est la méthode la plus simple. Depuis la machine cliente, tapez :

```
$ ssh login@ip_du_serveur_SSH (ou nom DNS du serveur SSH)
```

Astuce : Si vous utilisez le même identifiant sur le client et sur le serveur, vous pouvez vous contenter de taper :

```
$ ssh ip_du_serveur_SSH
```

Par défaut le client se connecte sur le port 22. Si le port du serveur a été changé, il faudra le préciser au client ssh. Ex. :

```
$ ssh login@ip_du_serveur_SSH -p PORT
```

Si c'est la première connexion SSH depuis ce client vers ce serveur, il vous demande si le fingerprint de la clef publique présentée par le serveur est bien le bon. Pour être sûr que vous vous connectez au bon serveur, vous devez connaître de façon certaine le fingerprint de sa clef publique et la comparer à celle qu'il vous affiche. Si les deux fingerprints sont identiques, répondez *yes*, et la clef publique du serveur est alors rajoutée au fichier ~/.ssh/known_hosts.

Exemple avec un serveur appelé « **sql** » et un client :

```
Fingerprint clef publique du serveur sql :
[root@sql ~]# ssh-keygen -lf /etc/ssh/ssh_host_ed25519_key
256 SHA256:ohucBZPkNx5w7txPRfQfNuCILKZdCa6ssrol0DFci8Y root@sql
(ED25519)

Sur le client, lancement de la connexion SSH :
$ ssh root@sql
The authenticity of host 'sql (192.168.1.17)' can't be established.
ED25519 key fingerprint is
SHA256:ohucBZPkNx5w7txPRfQfNuCILKZdCa6ssrol0DFci8Y.
ED25519 key fingerprint is
MD5:dc:c0:9c:04:eb:a7:7d:63:5c:2e:4a:17:48:2e:8c:5a.
Are you sure you want to continue connecting (yes/no)?
```

Les deux fingerprint SHA256 sont identiques. Je peux donc continuer la connexion car le serveur est bien celui sur lequel je souhaite ouvrir une session SSH.

Si vous vous êtes déjà connecté depuis ce client vers le serveur, sa clef publique est déjà dans le fichier ~/.ssh/known_hosts et il ne vous demande donc rien.

Ensuite, entrez votre mot de passe... et vous verrez apparaître le prompt, comme si vous vous étiez connecté en local sur la machine.

4.3 Authentification par clef

Au lieu de s'authentifier par mot de passe, les utilisateurs peuvent s'authentifier grâce à la cryptographie asymétrique et son couple de clefs privée/publique, comme le fait le serveur SSH auprès du client SSH.

4.3.1 Générer ses clefs

Pour générer un couple de clefs RSA, tapez (sur la machine cliente) :

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/olivier/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/olivier/.ssh/id_rsa.
Your public key has been saved in /home/olivier/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:sG0CmhFSrCLLF8yZ2+cDipuDYUtzLVtDBeDA9ZrqQ2U olivier@fixe01.ldnr.lan
The key's randomart image is:
+---[RSA 2048]----+
|o++o...          |
| o+..  .         |
| .+.+.o          |
|+  Eoo +         |
|+.=o* o S        |
|o*.= * +         |
|=o* = =          |
|o=.o   o         |
| ++     .        |
+----[SHA256]-----+
```

Les clefs générées ont par défaut une longueur de 2048 bits, ce qui est aujourd'hui considéré comme suffisant pour une bonne protection.

La clef privée est stockée dans le fichier ~/.ssh/id_rsa avec les permissions 600 et la clef publique est stockée dans le fichier ~/.ssh/id_rsa.pub avec les permissions 644.

Lors de la création, OpenSSH vous demande une ***pass phrase*** qui est un mot de passe pour protéger la clef privée. Cette *pass phrase* sert à chiffrer la clef privée. La ***pass phrase*** vous sera alors demandée à chaque utilisation de la clef privée, c'est-à-dire à chaque fois que vous vous connecterez en utilisant cette méthode d'authentification. Un mécanisme appelé *ssh-agent* permet de ne pas rentrer le mot de passe à chaque fois, comme nous le verrons un peu plus loin dans ce chapitre.

Vous pouvez à tout moment changer la *pass phrase* qui protège votre clef privée avec la commande **ssh-keygen -p**

4.3.2 Autoriser votre clef publique

Pour cela, il suffit de copier votre clef publique dans le fichier ~/.ssh/authorized_keys de la machine sur laquelle vous voulez vous connecter à distance. La commande suivante permet de réaliser cette opération via SSH :

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub login@ip_serveur_SSH
Si port différent alors :
$ ssh-copy-id -i ~/.ssh/id_rsa.pub "-p PORT login@ip_serveur_SSH"
```

4.3.3 Se connecter

La commande est la même que pour une authentification par mot de passe.

5. Transfert de fichiers par SSH

5.1 En console

Le transfert de fichiers par SSH est possible d'au moins trois façons :

- avec **scp** (comme Ssh CoPy), qui s'utilise de la même manière que la commande **cp** ;
- avec **sftp**, logiciel très basique qui s'utilise comme **ftp**.

  Ici également, vous pouvez utiliser la méthode d'authentification par mot de passe ou par clefs, l'utilisation restant la même dans les deux cas.

Utiliser SCP

Pour illustrer la syntaxe, je vais vous donner quelques exemples :

- pour transférer le fichier test1.txt situé dans le répertoire courant vers le home du compte *toto* de la machine *ordi1.exemple.org* sur laquelle tourne un serveur SSH :
```
$ scp test1.txt toto@ordi1.exemple.org:
```
- pour récupérer le fichier test2.txt situé dans le répertoire personnel de l'utilisateur *toto* de la machine *ordi2.exemple.org* et l'écrire dans le répertoire courant :
```
$ scp toto@ordi2.exemple.org:test2.txt .
```
- pour récupérer tous les fichiers ayant l'extension .txt situés dans le répertoire /usr/local de la machine *ordi2.exemple.org* et l'écrire dans le sous-répertoire test-scp du répertoire courant :
```
$ scp toto@ordi2.exemple.org:'/usr/local/*.txt' test-scp
```
- pour transférer l'intégralité du sous-répertoire test-scp du répertoire courant vers le sous répertoire incoming du home de l'utilisateur *toto* de la machine *ordi1.exemple.org* :
```
$ scp -r test-scp toto@ordi1.exemple.org:incoming
```

Si le port du serveur est différent alors :
```
$ scp -P PORT test1.txt toto@ordi1.exemple.org
```

6. Se connecter par SSH sans taper de mot de passe

6.1 Le principe

Cette section s'adresse à ceux qui utilisent un couple de clefs publiques / privées, **et qui ont chiffré leur clé privée avec une *pass phrase*** (c'est la configuration la plus sûre). Par conséquent, le client SSH demande la *pass phrase* à chaque utilisation des clefs pour s'authentifier.

Pour éviter d'avoir à taper systématiquement sa *pass phrase*, il faut utiliser ***ssh-agent*** : ce programme tourne en tâche de fond et garde la clef en mémoire. La commande *ssh-add* permet de donner sa clef à *ssh-agent*. Ensuite, quand vous utilisez le client SSH, il contacte *ssh-agent* pour qu'il lui donne la clef.

6.2 La pratique

Dans une console, lancez *ssh-agent* en évaluant les commandes qu'il écrit sur sa sortie :

```
$ eval $(ssh-agent)
Agent pid 4520;
$
```

Puis donnez votre clef à l'agent :

```
$ ssh-add
Enter passphrase for /home/olivier/.ssh/id_rsa:
Identity added: /home/olivier/.ssh/id_rsa (/home/olivier/.ssh/id_rsa)
$
```

Il vous demande alors votre *pass phrase*. Maintenant que votre clef a été transmise à l'agent, vous pouvez vous connecter sans entrer de mot de passe à toutes les machines pour lesquelles vous avez mis votre clef publique dans le fichier ~/.ssh/authorized_keys.

7. Se connecter par SSH depuis un client Windows

Bonne nouvelle ! Depuis fin 2017, Windows 10 propose SSH en mode client et serveur. Pour l'activer :
http://www.kevinsubileau.fr/informatique/astuces-tutoriels/windows-10-client-serveur-ssh-natif.html

https://docs.microsoft.com/fr-fr/powershell/scripting/core-powershell/ssh-remoting-in-powershell-core?view=powershell-6

Si vous êtes sur une version antérieure à Windows 10, il existe le client graphique **Putty** qui fonctionne aussi sous Windows 10 :

http://marc.terrier.free.fr/docputty/chapter02.html#demarrer-une-session

8. Conclusion

Comme vous pouvez le constater, la mise en place d'un serveur SSH, au niveau technique, n'est pas des plus compliquée. De plus, il est important de noter que dans le cadre de l'administration à distance, ce programme est très utilisé.

Ce protocole offre la possibilité de déplacer le port d'écoute du serveur, de bénéficier d'une connexion sécurisée au sein d'un réseau privé. En revanche, si le serveur a accès à Internet, il est préférable de désactiver l'authentification par mot de passe, des scanners balayant sans cesse la toile. L'authentification par clé n'est pas très difficile à mettre en place et apporte une plus grande souplesse et une meilleure sécurité.

9. Annexe : commande screen

screen est un paquet qui permet le multiplexage de terminaux, ce qui permet l'ouverture de plusieurs terminaux dans une même console, de passer de l'un à l'autre et de pouvoir les récupérer plus tard. Ce service s'avère très pratique dans le cas d'une utilisation d'une connexion distante en console. Les fonctionnalités principales permettent :

• la possibilité d'attacher et de détacher une session, pratique par exemple pour reprendre la main sur un serveur distant exécutant une commande lourde lancé à la main via SSH.

• la possibilité de partager un terminal avec un autre utilisateur, idéal pour aider un utilisateur distant.

Installation

```
# apt-get install screen
```

Exemple d'utilisation

Premièrement, il faut nommer une session screen :

```
$ screen -S maSessionScreen
```

Un message annonce la version utilisée et indique que ce programme est publié sous licence GPL s'affiche à l'écran. Il ne reste plus qu'à presser la touche [ESPACE]. La nouvelle session du shell s'affiche et attend qu'on saisisse une commande, par exemple :

```
$ ssh root@ip-serveur
Last login: Sun Jun 24 16:59:18 2018 from fixe01.ldnr.lan
[root@sql ~]
```

Vous pouvez vous déconnecter avec la commande « CTRL + a puis d (seul) » ou en fermant la console. Vous rouvrez un autre terminal et vous vous rattacher à la session maSessionScreen :

```
$ screen -r maSessionScreen
ssh root@ip-serveur
Last login: Sun Jun 24 16:59:18 2018 from fixe01.ldnr.lan
[root@sql ~]#
```

Non seulement vous récupérez votre session screen mais vous ne perdez pas votre connexion SSH ce qui peut être pratique en cas de coupure réseau.

Quelques commandes

• lister les screens :

```
$ screen -ls
There is a screen on:
    15300.maSessionScreen (Detached)
1 Socket in /run/screen/S-olivier
```

• tuer un screen :

```
$ exit
```

• nettoyer la liste de screens pour n'avoir que des instances actives :

```
$ screen -wipe
```

• se rattacher un screen :

```
$ screen -r nom_session
```

• fournir de l'aide à quelqu'un en voyant ce qui est saisi sur le terminal, l'utilisateur tapera :

```
$ screen -x nom_session
```