OpenSSH

1. Utilisations de OpenSSH

Les sessions interactives sur les systèmes Unix ont d'abord été conduites par des terminaux passifs, qui se bornaient à gérer les entrées et sorties, connectées à une unité centrale par un port série. Les frappes au clavier étaient envoyées brutes à l'unité centrale, et l'unité centrale envoyaient en retour des ordres d'affichage à l'écran. Les ordinateurs étant alors hors de prix, le coût relativement modeste des terminaux passifs permettait de mutualiser l'utilisation d'un ordinateur.

Avec la généralisation des réseaux IP et la démocratisation des ordinateurs personnels, l'administration distante des systèmes Unix s'est faite ensuite par le protocole telnet. Le principe est rigoureusement le même qu'avec les terminaux passifs, si ce n'est que les frappes au clavier et ordres d'affichage sont envoyés dans des paquets telnet transportés par IP. Le problème est que la gestion de la sécurité avec le protocole telnet est largement insuffisante : une authentification est réalisée en texte clair, et aucune confidentialité n'est apportée aux échanges entre le client et le serveur.

Le protocole SSH vise à apporter des services d'authentification et de confidentialité à des échanges entre clients et serveur pour le transport sécurisé de données. Il est dans la plupart des cas simples utilisé en tant que « telnet sécurisé » mais il est aussi capable d'assurer le transport sécurisé d'autres protocoles applicatifs. L'implémentation open source du protocole SSH est « OpenSSH », créé et maintenu par les membres du projet OpenBSD.

2. Gestion des authentifications

a. Authentification par mot de passe

L'utilisation la plus simple du client SSH, qui consiste à ouvrir une session shell distante de façon sécurisée sur réseau IP, exploite un mode d'authentification simple, à savoir utiliser un compte local sur le serveur et demander au client de s'authentifier avec le nom et le mot de passe de ce compte présent sur le serveur. Le mot de passe est alors vérifié et l'authentification est validée. Toutefois, cette phase d'authentification par mot de passe sert uniquement à vérifier la validité du client. Lequel client peut à son tour avoir des doutes sur l'identité et la légitimité du serveur : en clair, suis-je bien en train de parler à mon serveur, ou à un faux serveur qui exploiterait les commandes tapées pour récupérer des informations sur mes systèmes ? Pour éviter tout risque d'usurpation du serveur, le client réalise une vérification de l'identité du serveur à la première connexion. En fait, une empreinte numérique du serveur est réalisée, et après validation de cette empreinte par le client, elle est conservée dans un fichier appelé **known_hosts**, présent dans un répertoire caché **.ssh** dans le répertoire personnel de l'utilisateur.

Exemple de fichier known_hosts

Le fichier known_hosts présente une (très longue) ligne par serveur connu.

```
beta:~# cat .ssh/known_hosts
|1|LPx02U8nHnkSb0czyqVrdXPcW04=|jS0/QdS0HydzPZj8QXxHXC4j6EM= ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAv+kXth0/RSAroNfqeV+IkEMetdWRWYBvbNOqUDDSL/fLylBip9le40xfTe1j
FXuYqAWR+mQMo8Pg37/PUWeetlBCvG4F486UbqUn2Ol5B/1GZqzG7nvbOLcp7CDr6vmqgrk2QZvUZcohWc4L9S6z
zvk3EmQ1AMa+BKo4m+FCG9E1mK4bFtvchVqL1amzGg1jd2QuTzMGNibTdrEi9gSr2TrJ5Se9AhNQkIzZPvrqvVAD
itiggcYNetxaNkPKfW8DdClq+qOVVAQuWnZiO63Mp/0+b+JEutFgNsX8mkt9nx34Yws7s3BnIuT7oU+shxnuy/vj
5But4uUry5tFaTxXCw==
beta:~#
```

b. Authentification par clés

Une méthode sans doute plus fiable pour authentifier les connexions SSH consiste à utiliser des clés d'authentification stockées localement sur le disque de l'utilisateur. L'authentification par clés ne dispense pas obligatoirement de la saisie d'un mot de passe, mais garantie à l'utilisateur que la machine distante est bien celle avec laquelle on veut travailler et non pas une usurpatrice.

Création de la paire de clés sur le client

Pour que le serveur puisse être formellement identifié, il doit disposer de la clé publique du client. Cette clé lui permettra de crypter des données déchiffrables par le seul client propriétaire de la clé privée correspondante. Il convient donc dans un premier temps de générer cette clé publique sur le client. Comme il s'agit de cryptographie asymétrique, la génération d'une clé publique est obligatoirement simultanée à celle de la clé privée correspondante.

La commande **ssh-keygen** permet de créer ces clés publiques et privées.

Génération d'un couple de clés

```
ssh-keygen -t algorithme
```

Où *algorithme* représente l'algorithme employé pour la génération des clés du client. Il peut s'agir de RSA (version 1 ou 2 de SSH) ou DSA (version 2 de SSH). RSA et DSA sont deux algorithmes de cryptage asymétriques souvent utilisés pour l'authentification. Si l'algorithme n'est pas précisé, la valeur par défaut RSA est employée.

Génération d'un couple de clés avec les valeurs par défaut

On génère ici un couple de clés avec l'algorithme par défaut (RSA) pour l'utilisateur tata. La représentation graphique (randomart) de la clé n'est pas systématique et dépend de la version de la commande.

```
tata@stotion:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/tata/.ssh/id_rsa):
Created directory '/home/tata/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/tata/.ssh/id_rsa.
Your public key has been saved in /home/tata/.ssh/id_rsa.pub.
The key fingerprint is:
f3:5c:f1:34:6c:1b:a6:4c:5b:c4:6d:30:48:01:76:f4 tata@stotion
The key's randomart image is:
+--[ RSA 2048]----+
|          o+=++o |
|         . ..+..o|
|           o E. |
|          o X + |
|        S   = o  |
|         + .     |
|          o      |
|                 |
|                 |
+-----------------+
tata@stotion:~$
```

La commande **ssh-keygen** provoque la création de deux fichiers, par défaut dans un répertoire **.ssh** situé directement dans le répertoire personnel de l'utilisateur. Ces deux fichiers sont par défaut **id_rsa** pour la clé privée et **id_rsa.pub** pour la clé publique correspondante. Même si ça n'est pas obligatoire, il est vivement recommandé de protéger la clé privée par un mot de passe qui sera demandé lors de sa création.

Contenus de fichiers de clés privées et publiques

On observe le contenu des fichiers de clés privées et publiques. Notez que les droits par défaut sont limités sur le fichier de clé privée, et ouverts sur le fichier de clé publique.

```
tata@stotion:~/.ssh$ ls -l
total 8
-rw------- 1 tata tata 1743 2010-09-03 09:38 id_rsa
-rw-r--r-- 1 tata tata  394 2010-09-03 09:38 id_rsa.pub
tata@stotion:~/.ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAs0jrYKKQKiS4f/cCQMhOcc2WTMmGrbXXv3oyz67KUwkm4JumEU1
YkOaNi+WM4nVbkzC7rkUnlXQMxu/EpZLoraNySMHZjUgYiWiRuM4pI0z/atPfjVlwPtGzfUKlqSsP4NCark/9G0
WlMgEXlgpEdeJDmMBRuj98PJjOI/cRGRTgR6JEoevFWMPTDRpoBix3YizVY+dA+unJQPaNKWhoDnCZg7xWi+ZRg
T2Q1PcbqYKt4xLio+Eei0dvlgu5r5hSvymOdWbXwykywoloIxnzIPiUe7CAxm+KCBA23LQw73pREd1cglS6Gd23
b5Byv/oI6etqs4WOmcJa40Ymvtfbjw== tata@stotion
tata@stotion:~/.ssh$ cat id_rsa
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,B08C4C3C4B021A76

TzO6ofHOv8sVRDoPj+o7dXfPuXDJaOmQSGhDkWUTC9iGHYnGdHgsig5EKWEez0Zj
YucF9doTpLCv9UsRac6WHRjlQb7AUjk9phEjrKYW4gAfoXNcFY5IiC7fca9i8NQk
YCj4mtzmbJAFc0W9Ax8g0UzZ8bwElIacI28pAdSvVqVHQ6omnVBoWhXhgWTUZaKp
```

```
2XbY5gJ7miKW3Y9IPZ3JLukB3j4rTZ0bu8j/UedyXuogpZgYF2vW0GfvtBbfP31F
(...)
RZfBnf+3+KxTvnAtJsMSZc4Glg+9Gch9V+mjU2SfW+T+bUnYLB/6Mpo1aq/akj3r
0G6w12SgjqiOuuXnsCdU8Ox1olCqiHFrk0DyPmwoxcSQygpm2r7FIwL4MPxbELJO
zfk+0wJOmsUANJzeBKd4LXmZykYsAOmf3zZNlS+iU/ZhCBqFmn3/5w==
-----END RSA PRIVATE KEY-----
tata@stotion:~/.ssh$
```

À chaque connexion, le serveur regarde dans le répertoire local de l'utilisateur essayant de se connecter si un répertoire **.ssh/authorized_keys** existe, et s'il contient la clé publique du client. Si c'est le cas, l'authentification du serveur peut être réalisée par le client. Le client devra donc copier son fichier de clé publique dans le répertoire **~/.ssh.authorized_keys** du serveur par le moyen de son choix. (clé usb, copie réseau).

c. L'agent SSH

Pour les administrateurs ayant fréquemment besoin d'accéder à plusieurs machines par SSH, un « agent SSH », lancé par la commande **ssh-agent**, permet de conserver en mémoire les clés privées utilisées pour les authentifications. Les clés privées sont transmises une fois pour toutes à l'agent par la commande **ssh-add**. Si un mot de passe de protection de la clé est nécessaire, il est demandé à cette occasion. Les clés sont ensuite disponibles sans intervention directe de l'utilisateur pour toute authentification.

La commande **ssh-add** consulte le répertoire **.ssh** dans le répertoire personnel de l'utilisateur et recherche d'éventuelles clés privées dans les fichiers **id_rsa, id_dsa**, et **identity**. Les clés stockées par l'agent SSH peuvent être consultées par la commande **ssh-add -l**.

Lancement de l'agent par la commande ssh-agent

L'agent alimente des variables lors de son fonctionnement qui permettent de le gérer plus facilement.

```
tata@stotion:~$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-sRuvox4519/agent.4519; export SSH_AUTH_SOCK;
SSH_AGENT_PID=4520; export SSH_AGENT_PID;
echo Agent pid 4520;
tata@stotion:~$
```

ssh-agent : variables courantes	
SSH_AGENT_PID	Le pid de l'agent en cours d'exécution.
SSH_AUTH_SOCK	Le socket créé par le processus.

Prise en compte de clés par l'agent SSH

La commande ssh-add sans argument permet la prise en compte des clés par l'agent SSH qui doit naturellement avoir été lancé auparavant.

```
tata@stotion:~$ ssh-add
Enter passphrase for /home/tata/.ssh/id_rsa:
Identity added: /home/tata/.ssh/id_rsa (/home/tata/.ssh/id_rsa)
tata@stotion:~$
```

Visualisation des clés privées stockées par le ssh-agent

La commande ssh-add -l permet de vérifier que les clés ont bien été prises en compte par l'agent.

```
tata@stotion:~$ ssh-add -l
2048 f3:5c:f1:34:6c:1b:a6:4c:5b:c4:6d:30:48:01:76:f4 tata@stotion (RSA)
2048 f3:5c:f1:34:6c:1b:a6:4c:5b:c4:6d:30:48:01:76:f4 /home/tata/.ssh/id_rsa (RSA)
tata@stotion:~$
```

L'agent SSH est avant tout une solution de gestion de clés et n'est pas destiné à créer les clés SSH. L'agent SSH ne peut travailler que sur des clés déjà créées par la commande ssh-keygen.

3. Confidentialité des communications

a. Session interactive avec SSH

La session interactive est ouverte depuis un client vers un serveur avec un compte utilisateur présent sur le serveur.

Ouverture de session interactive avec SSH

```
ssh utilisateur@adresse_serveur
```

Session interactive avec SSH : option et paramètres	
utilisateur	Le compte utilisateur présent sur le serveur avec lequel on se connecte.
adresse_serveur	L'adresse IP du serveur auquel on se connecte.

Exemple d'ouverture de session interactive avec SSH

```
alpha:~# hostname ; whoami
alpha
root
alpha:~# ssh toto@192.168.0.11
toto@192.168.0.11's password:

toto@beta:~$ hostname ; whoami
beta
toto
toto@beta:~$
```

b. Copie de fichiers avec SSH

La commande **scp** s'appuie sur le démon SSH et permet de copier des fichiers de façon sécurisée avec les services d'authentification et de confidentialité offerts par SSH. La copie peut se faire du client vers le serveur ou depuis le serveur vers le client.

Copie de fichier du client vers le serveur avec scp

```
scp fichier_local utilisateur@adresse_serveur:fichier_distant
```

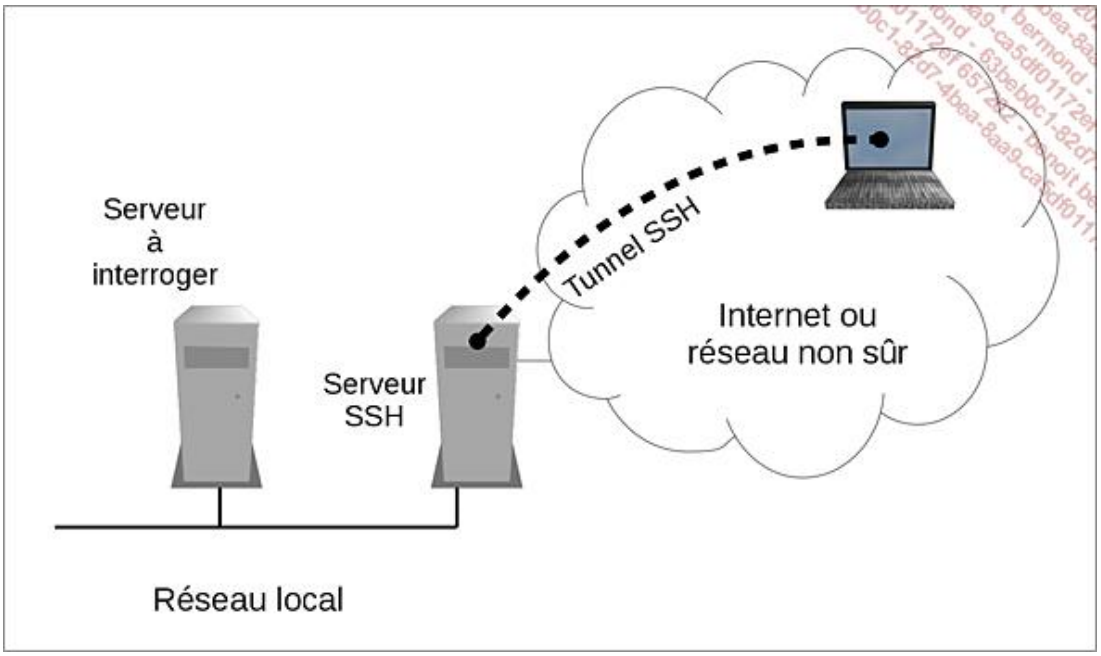
Copie de fichier depuis le serveur vers le client avec scp

```
scp utilisateur@adresse_serveur:fichier_distant fichier_local
```

Copie de fichiers avec scp : options et paramètres	
fichier_local	Chemin relatif ou absolu du fichier local devant être copié.
fichier_distant	Chemin absolu du fichier distant devant être copié.
utilisateur	Compte utilisateur existant sur le serveur utilisé pour la copie.
adresse_serveur	Adresse IP du serveur hébergeant le service SSH.

c. Utilisation d'applications dans des tunnels SSH

La création d'un tunnel SSH permet de sécuriser une communication client-serveur pour un protocole à priori peu sécurisé. On établit depuis le poste client un tunnel SSH vers le serveur, et tout le trafic entre ces deux machines est sécurisé. Le serveur génère alors un autre trafic non sécurisé vers la machine cible du trafic. Les connexions des clients qui souhaitent emprunter le tunnel se font en fait vers le client SSH.



Création d'un tunnel applicatif SSH

```
ssh -L port:cible_trafic:port_cible utilisateur@serveur
```

Tunnel SSH : options et paramètres	
-L	Renvoie un port local vers un serveur SSH (établissement de tunnel).
port	Le port local à renvoyer.
cible_trafic	Adresse IP ou nom de la machine cible du trafic.
port_cible	Port vers lequel renvoyer le trafic sur la machine cible.
utilisateur	Compte utilisateur sur le serveur utilisé pour l'établissement du tunnel.
serveur	Adresse IP ou nom du serveur extrémité du tunnel.

Dans ce fonctionnement, un tunnel est établi entre un client et un serveur. Sur le client, le trafic à destination du port local est renvoyé au travers du tunnel SSH vers la machine cible sur le port cible.

d. Renvoi de sessions X11 via SSH

Le serveur X ne prévoyant nativement pas de sécurité forte pour ses échanges clients-serveurs, un usage courant de SSH consiste à faire circuler dans un tunnel SSH des applications graphiques. Il faut pour cela autoriser le serveur SSH à relayer ce type de trafic, puis d'utiliser un client compatible avec ce mode de fonctionnement.

L'autorisation du renvoi de sessions X via SSH se fait en modifiant le fichier de configuration du serveur SSH **/etc/ssh/sshd_config**.

Autorisation du renvoi des connexions X dans sshd_config.conf

```
X11Forwarding yes
```

Connexion depuis un client SSH

```
ssh -X utilisateur@serveur
```

Où *utilisateur* représente le compte utilisé pour la connexion, et *serveur* l'adresse IP ou le nom du serveur auquel on se connecte. Les applications graphiques peuvent alors être lancées depuis la session SSH cliente.