SSH - Secure Shell

Victor Moraru
IFI, AUF

Plan du cours

- Présentation SSH
- SSH - coté client
- SSH - coté serveur
- Autorisation forte
- Conclusion

Sécuriser des connexions à distance: Secure Shell

SSH permet de sécuriser les communications des réseaux en utilisant la cryptographie

- SSH est composé d'un ensemble d'outils permettant des connexions sécurisées entre des machines. Ces outils ont pour but de remplacer les utilitaires de connexions classiques n'utilisant pas de chiffrage.
    - Remplace : rcp, rlogin, rsh, telnet (ftp par sftp en SSH V2)
- SSH chiffre et compresse un canal de communication qui sécurise les données transmises (permet d'éviter les sniffeurs réseaux)
- Non seulement le mot de passe est chiffré lors de la connexion mais les informations circulant sur le réseau entre les deux machines le sont aussi.

Terminologie de SSH

C’est un protocole

- c'est-à-dire c’est une méthode standard permettant à des machines d'établir une communication sécurisée décliné en 2 versions : Version 1 et version 2 : le protocole v1 possédait une faille permettant à un pirate d'insérer des données dans le flux chiffré

C’est aussi un produit :

- SSH Communications Security Inc (V1 et V2) Initialement développé par Tatu Ylönen (payant) dernière version gratuite v1.2.12
- OpenSSH du projet OpenBSD (V1 et V2) apparaît en 1999, aujourd’hui c’est le plus utilisé
  - Produit en accord avec la législation française sur la cryptographie
  - http://www.ssi.gouv.fr/fr/reglementation/index.html#produits_crypto

Ce sont aussi des commandes en ligne

Terminologie de SSH

Site : http://www.openssh.org

- Version logicielle actuelle : openssh-5.1.tar.gz
  - Existe en format packagé pour toutes distributions
- Subit un audit permanent du code
- OpenSSH utilise principalement :
  - OpenSSL
  - Zlib pour la compression des flux
  - Perl lors de l'installation, pour des applications tiers (ssh-copyid…)
  - Ainsi que des générateurs d'entropie
    - ✔ PRNGD (Pseudo Random Number Generator Daemon)
    - ✔ /dev/random

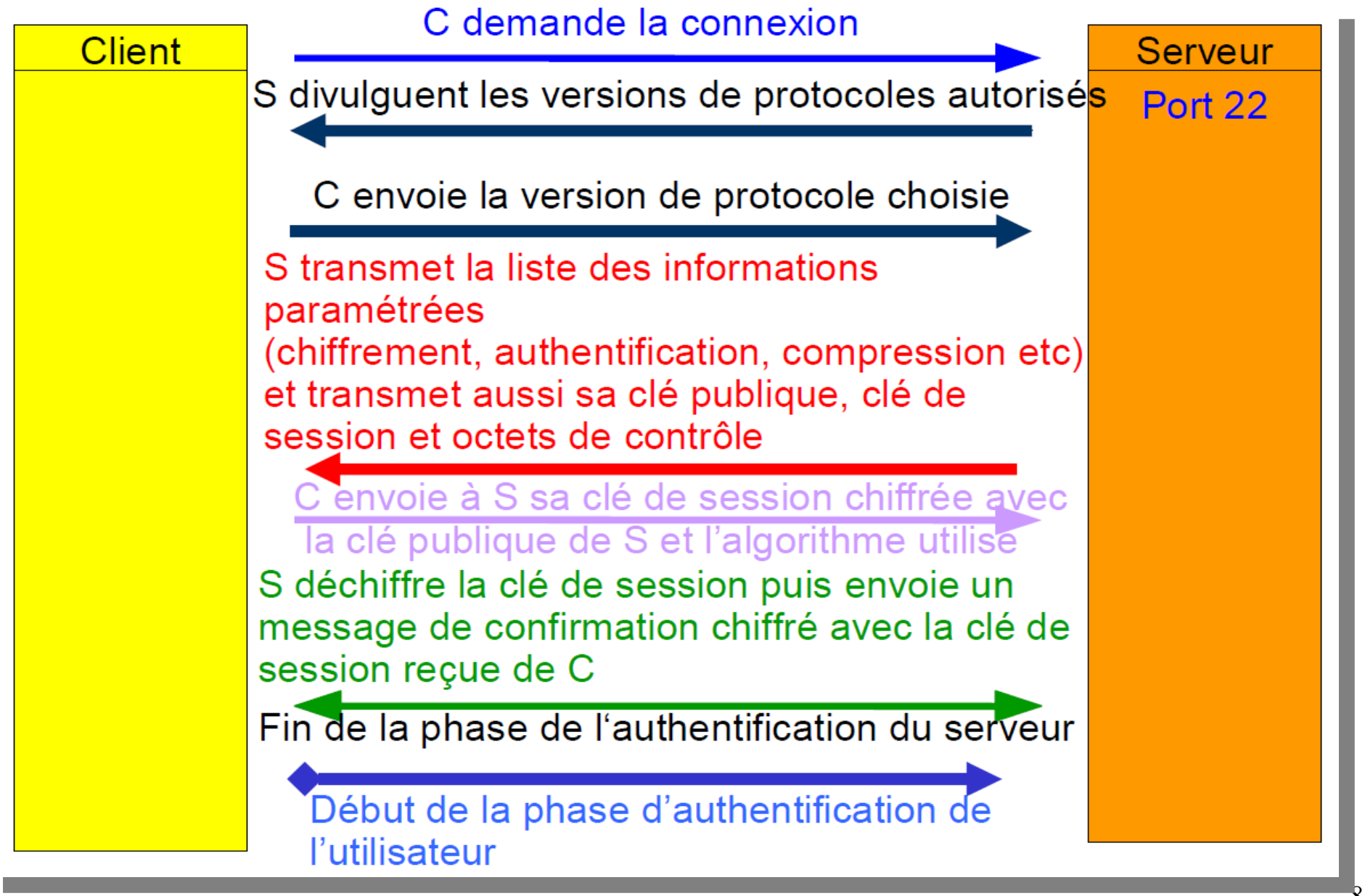
Les outils ssh

- La boîte à outils SSH est généralement composée de :
  - Serveur : sshd
  - Clients : ssh, scp, sftp (ssh = slogin)
  - Des outils de gestion: ssh-add, ssh-agent, ssh-keygen, sshkeyscan
  - Les fichiers de configuration (OpenSSH) sont souvent dans:
    - ✔ Pour le serveur : /etc/ssh
    - ✔ Pour les clients : /etc/ssh et $HOME/.ssh
- Fonctionnement sur le schéma d'un système client – serveur
  - Les clients ssh demandent une ouverture de connexion au serveur sshd

SSH sous Linux

- Cryptographie dans SSH
  - Repose sur les algorithmes d'OpenSSL
    - ✔ Algorithmes asymétriques:
      - RSA et DSA
    - ✔ Algorithmes symétriques:
      - 3DES, Blowfish, AES, Arcfour ...
- Linux/Unix : PAM et SSH
  - Gestion centralisée des services:
    - ✔ Possibilité d'activer le support de l'interface PAM (Pluggable Authentication Modules) au niveau de sshd

Fonctionnement du protocole SSH



L'authentification des serveurs

- Le principe d'authentification du serveur se fait par chiffrement à clé publique du protocole SSH.
- Le client doit donc connaître la clé publique du serveur sur lequel il veut se connecter avant toute connexion. Ainsi, il existe un mécanisme pour la machine cliente pour stocker les clés d'un serveur afin de les réutiliser ensuite;
- Il pourra ainsi vérifier la clé d'un serveur à chaque nouvelle connexion avec celle enregistrée lors de la première connexion;
- Cela permet d'éviter les attaques du type man-in-the-middle

L'authentification des utilisateurs

Une fois que la connexion sécurisée est mise en place entre le client et le serveur, le client doit s'identifier sur le serveur afin d'obtenir un droit d'accès.

- **Par mot de passe**: le client envoie un nom d'utilisateur et un mot de passe au serveur au travers de la communication sécurisé et le serveur vérifie si l'utilisateur concerné a accès à la machine et si le mot de passe fourni est valide
- **Par clés publiques** : si l'authentification par clé est choisie par le client, le serveur va créer un challenge et donner un accès au client si ce dernier parvient à déchiffrer le challenge avec sa clé privée
- **Par hôte de confiance** : système équivalent aux systèmes utilisant rhost ou hosts.equiv en utilisant les clés publiques des serveurs
- **Par Kerberos, SmartCard, PAM**

VPN (Virtual Private Network) avec SSH

- Le support d'un protocole de VPN est une fonctionnalité apparue récemment dans OpenSSH.
- Le principe est le même que pour OpenVPN mais il est basé sur le protocole SSH (différents des tunnels TCP SSH)
- Supporte les interfaces réseaux virtuelles Tun/Tap (niveau 3/niveau 2)
- Option à activer du côté serveur : ***PermitTunnel***

Séparation de privilèges du serveur SSH (chroot)

Chaque processus lors de son exécution ne possède que les privilèges nécessaires, et n'a alors accès qu'aux éléments nécessaires à son exécution

- cette mise en place augmente la sécurité des applications

chroot est une commande des systèmes d'exploitation UNIX permettant de changer pour un programme le répertoire racine de la machine hôte

Cette commande permet d'isoler l'exécution d'un programme et d'éviter ainsi certaines malveillances

- exemple: l'exploitation d'un dépassement de tampon, pour ensuite accéder au répertoire racine de la machine hôte.

Séparation de privilèges du serveur SSH (chroot)

sshd utilise deux processus :

- le processus père privilégié contrôle les progrès du processus fils non privilégié (effectue le minimum nécessaire : la réussite de l'authentification est déterminée par le processus père)
- Le processus fils est non privilégié. Ceci est atteint en changeant ses uid/gid vers un utilisateur non privilégié

```
root    20702  0.0  0.1   8188  2692 ?        Ss   10:42   0:00 sshd: victor [priv]
victor  20733  0.0  0.0   8188  1624 ?        S    10:43   0:00 sshd: victor@pts/2
```

Séparation de privilèges du serveur SSH (chroot)

Mise en cage : chroot

- OpenSSH intègre la directive ChrootDirectory permettant de chrooter dans un répertoire après la connexion.
- Cette fonctionnalité est principalement utilisée pour du SFTP
- On peu parfaitement configurer l'ensemble pour les utilisateurs SSH
- Il faut construire l'environnement nécessaire

Clients/serveurs ssh

Plate-forme Windows (clients gratuits) :

- SSH Tectia
  - SSH Secure Shell for Workstation (nom du produit)
  - http://www.ssh.com/support/downloads/secureshellwks/non-commercial.html
  - Seul le client Tectia 3 (ex-ssh.com) est gratuit, les versions Tectia sont payantes.
  - Inconvénient de la version gratuite : le module de connexion à base de certificats X509 est désactivé ainsi que le transfert d'agent
- Putty
  - Clients regroupant toutes les commandes connues sous OpenSSH
  - http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
- Winscp (v3)
  - Outil graphique de transfert de fichiers (scp/sftp, donc pas de ssh) très performant
  - – http://winscp.net/eng/docs/lang:fr/
- Serveur gratuit pour windows via Cygwin
  - http://www.cygwin.com/
  - sshd aussi disponible seul sans l'environnement Cygwin complet (attention aux problèmes de sécurité liés à ce serveur moins robuste)

Clients/serveurs ssh

Plate-forme Mac

- MacOS 9 et inférieur :
  - client nifty-telnet (v1 uniquement)
  - http://www.lysator.liu.se/~jonasw/freeware/niftyssh/
- MacOS X :
  - clients/serveur natifs (OpenSSH)
  - fugu : outil graphique de transfert de fichiers
    - http://www.columbia.edu/acis/software/fugu/
    -
- Unix (tous)
  - OpenSSH
    - http://www.openssh.org/
    - Proposé en standard dans la plupart des distributions Unix
  - Ssh.com (Tectia): existe aussi en version Unix

Plan du cours

- Présentation SSH
- SSH - coté client
- SSH - coté serveur
- Autorisation forte
- Conclusion

Client unix: ssh

- Utilisation simple : ssh
  - Connexion distante (alternative à telnet, rlogin) :

    ssh login@machine_distante

  - Ou bien avec l'option -l

    ssh -l test machine_distante

Client unix: ssh

- Première connexion distante :
  - message d'alerte lors d'une connexion vers une nouvelle machine distante

```
victor@victor-portable:~$  ssh victor@jupiter.dorsale.ifi
The authenticity of host 'lupiter.dorsale.ifit (192.168.100.2)' can't be established.
RSA key fingerprint is f9:be:e7:0a:a4:53:0f:b2:c2:1f:2d:8e:c4:34:f4:f1.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (RSA) to the list of known hosts.
victor@upiter.dorsale.ifi's password:
Linux jupiter 2.6.24-25-generic #1 SMP Tue Oct 20 07:31:10 UTC 2009 i68
```

  - La clé publique du serveur est ajoutée dans le fichier known_hosts

Client unix: ssh

Connexion distante (alternative à rsh) :utilisation d'une commande shell à distance (ls /tmp):

```
victor@victor-portable:~$ ssh 192.168.100.5 ls /tmp
victor@192.168.100.5's password:
acroread_1000_1000
atievntX.nMm2Fu
AtiXUEvent00001589_08078b40
gpg-6uxfZ0
keyring-tVjC9Q
....
```

- Utilisation comme un rlogin sécurise

Client unix: sftp

- Transfert de fichiers (une alternative sécurisée à ftp)

  sftp login@machine

- Les commandes sont les mêmes qu'avec ftp (put, get, mput etc…)

```
victor@victor-portable:~$ sftp jupiter.dorsale.ifi
Connecting to jupiter.dorsale.ifi...
victor@jupiter.dorsale.ifi's password:
sftp> ls -l tmp
-rw-r--r--     1 victor   victor      37896 Nov  7 09:10 htaccess.html
drwxr-xr-x    2 victor   victor       4096 Nov  7 09:10 htaccess_files
-rw-------    1 victor   victor          0 Nov  6 17:33 john.pot
-rw-------    1 victor   victor         77 Nov  6 18:04 restore
-rw-r-----    1 victor   victor       1029 Nov  6 17:33 shadow
```

Client unix: scp

Transfert de fichiers (alternative sécurisée à rcp): scp est une commande de copie de fichiers (ou répertoires) entre 2 machines à travers le réseau

Copier un fichier local sur une machine distante

scp abc.txt test@machine_distante:/tmp/test/

Copier un répertoire distant sur la machine locale

scp -r test@machine_distante:/tmp/test/ .

```
victor@victor-portable:~$ scp aaa.jpg victor@localhost:/home/victor/tmp/
victor@localhost's password:
aaa.jpg                                   100%   51KB  51.1KB/s   00:00
victor@victor-portable:~$ ls tmp
aaa.jpg   john.pot  restore.txt
```

Configuration du client ssh sous Linux

- Le reprtoire $HOME/.ssh
  - **known_hosts** : contient les clés publiques des serveurs sur lesquels l'utilisateur s'est connecté (vérifie ainsi si un serveur n'a pas été substitué ou changé)
  - **authorized_keys**: les clés publiques des utilisateurs distants pour authentification par clé

Plan du cours

- Présentation SSH
- SSH - coté client
- SSH - coté serveur
- Autorisation forte
- Conclusion

Configuration serveur sshd

Répertoire du serveur : /etc/ssh

- Les fichiers de configuration :
  - sshd_config , ssh_config, denyusers
    - ✔ sshd_config : paramétrages lors des connexions vers le serveur local, ce fichier s'applique au démon sshd
    - ✔ ssh_config : paramétrages base et général du client ssh, ce fichier s'applique donc pour les commandes ssh, scp, sftp
- Les fichiers des clés privées/publiques du serveur :
  - Clés compatibles V1 (ssh v1/ssf)
    - ✔ ssh_host_key (privée) , ssh_host_key.pub (publique)
  - Clés compatibles V2 (ssh v2)
    - ✔ ssh_host_dsa_key (privée) , ssh_host_dsa_key.pub (publique)
    - ✔ ssh_host_rsa_key (privée) , ssh_host_rsa_key.pub (publique)

Configuration serveur sshd: sshd_config

Protocol : choix du protocole à utiliser (initialiser depréférence à 2 ou si besoin d'une compatibilité SSH V1, laisser : 2,1)

PermitRootLogin : autorise le compte root à se connecter (de préférence à initialiser à no)

StrictModes yes : vérifie les permissions des fichiers et répertoires importants (accès au propriétaire uniquement)

RSAAuthentication yes : méthode d'authentification par RSA uniquement en V1 et V2

PubkeyAuthentication yes : méthode d'authentification forte (rsa ou dsa) en V2 uniquement

AuthorizedKeysFile .ssh/authorized_keys : nom et localisation du fichier de clés publiques individuelles sur les hôtes locaux.

Configuration serveur sshd: sshd_config

PasswordAuthentication yes : autorise la connexion par mot de passe

PermitEmptyPasswords no : interdit les connexions sans mot de passe

X11Forwarding yes : active le transfert X pour sshd

X11DisplayOffset 10 : réservation d'un numéro d'affichage X11 afin d'éviter les collisions entre sshd et le vrai serveur X.

X11UseLocalhost yes : bind sur l'interface de la boucle locale, permet d'éviter les problèmes de proxy

MaxStartups 10:30:60 : 10 - lle nombre de connexions acceptées sans qu'un utilisateur ait réussi à s'identifier, si cela passe au dessus de 10, il y a 30 % de chances que les suivantes soient bloquées, et ce pourcentage augmente linéairement jusqu'à 100 % lorsque on atteint, à 60 connexions

Subsystem sftp /usr/lib/ssh/sftp-server : active le système de transfert de fichiers via sftp

Configuration serveur sshd: sshd_config

Filtrage d'utilisateurs ou de groupes

Variables :

AllowUsers et DenyUsers

AllowGroups et DenyGroups

Permettre l'accès à certains utilisateurs et pas d'autres un accès ssh

Configuration pour autoriser les deux utilisateurs (test et admin) et aucun autre à se connecter en ssh:

DenyUsers

AllowUsers test admin

Configuration serveur sshd: ssh_config

- **Host** * : spécifie les hôtes concernés par la configuration qui suit (adresse ip ou nom DNS, * = toutes)
- **ForwardAgent** **yes** : indique à l'agent que l'agent d'authentification doit être renvoyé vers la machine distante
- **ForwardX11** **yes** : autorise la redirection du serveur graphique (possibilité de lancer des applications graphiques dans la session ssh)
- **ForwardX11Trusted** **yes**: n'oblige pas le client ssh à créer un « untrusted X cookie » de sorte que les attaques sur la retransmission X11 ne puissent pas devenir des attaques des clients X11 depuis une machine distante

Configuration serveur sshd: ssh_config

- **VerifyHostKeyDNS yes** : Le client ssh peut vérifier la clef publique RSA/DSA d'un serveur SSH à l'aide du DNS

- **StrictHostKeyChecking yes/ask/no**
  - **no** : automatise la gestion des clés d'hôtes dans known_hosts. Si la clé n'existe pas, la connexion ne sera pas refusée, et sera rajoutée sans le demander à l'utilisateur (l'utilisateur ne verra pas les changements de clés du serveur)
  - **ask** : demande à l'utilisateur s'il veut ajouter la clé dans le fichier known_hosts, puis permet la connexion, si la clé du serveur change, un message d'avertissement sera envoyé à l'utilisateur

Configuration serveur sshd: ssh_config

- **yes** : vérifie que la clé publique de l'hôte distant existe sur l'hôte qui cherche à se connecter et ensuite autorise la demande de connexion. Une non connaissance de la clé ou un changement de clé du serveur implique alors un échec de la connexion ssh
  - ✔ Il faut donc fournir la clé publique du serveur distant à la configuration cliente qui cherche à se connecter
  - ✔ /etc/ssh/ssh_known_hosts : base de données des clés d'hôte
  - ✔ On ne peut donc pas se connecter la première fois sans avoir la clé installée

Les clés du serveur sshd

Les fichiers de clés du serveur sshd

En fonction de la version de protocole et des algorithmes :

- **Les clés privées** (protégés en lecture pour root uniquement)
  - ✔ ssh_host_key : ssh v1, RSA1
  - ✔ ssh_host_key_rsa : v2, RSA
  - ✔ ssh_host_key_dsa : v2, DSA

- **Les clés publiques** (en lecture pour tous)
  - ✔ *.pub extension valable dans tous les cas

Les clés du serveur sshd

Changement de clés du serveur

Commande : ssh-keygen –t Algo –f files

- Options :
- -t : algorithme
- –f fichier cible
- -N : phrase d'identification vide

Conséquence d'un changement de clés du serveur

Alerte lors d'une re-connexion alors que la clé du serveur a changé :

- Alerte pour tous les utilisateurs qui chercheront à se connecter sur le serveur sshd
- Agir sur le fichier $HOME/.ssh/known_hosts pour autoriser le changement de la clé coté utilisateur

Lancement et arrêt du serveur sshd

Lancement du serveur :

- # /etc/init.d/sshd start

Arrêt du serveur :

- # /etc/init.d/sshd stop

Relancer le serveur après une modification de la configuration(sshd_config) :

- # /etc/init.d/sshd reload
- # /etc/init.d/sshd restart

Le redémarrage du serveur sshd n'interrompt généralement pas les connexions ouvertes

Plan du cours

- Présentation SSH
- SSH - coté client
- SSH - coté serveur
- Autorisation forte
- Conclusion

Autentification forte

- Basée sur une procédure d'identification plus complexe e tdifférente de celle des systèmes Unix classiques (nom et mot de passe)
- Cette procédure repose sur un principe d'une paire de clés publique/ privée dont la clé privée est protégée par une phrase d'identification
  - ATTENTION ! la sécurité de ssh par authentification forte repose alors sur la protection de la clé privée : il faut impérativement mettre une VRAI PHRASE D'AUTHENTIFICATION (au moins 14 caractères)qui est en fait plus qu'un simple mot de passe, mais une véritable phrase (moins de limitation)
    - ✔ Permet l'utilisation de caractères blancs (séparateur) et d'autres, maisattention aux caractères utilisés à cause des différents types de claviersafin de pouvoir taper la passe-phrase
- L'utilisateur s'identifiera alors sans utiliser le mot de passe de la connexion classique (mot de passe Unix), qui lui circule sur le réseau, mais par sa phrase d'identification sur l'hôte local

Autentification forte

- Utilise un algorithme très puissant pour le chiffrement (algorithme RSA/DSA)
- Ainsi chaque utilisateur possède son propre jeu de clés uniques (une clé privée = secrète, une clé publique = accessible par tous)
- une nouvelle connexion nécessite l'installation de la clé privée sur le client et de la clé publique sur le serveur
- le serveur va créer un challenge et donner un accès au client si ce dernier parvient à déchiffrer le challenge avec saclé privée

Gestion des clés

- Création des paires de clés :
  - **ssh-keygen**

    Options :

    ✔ -t algorithme : choix de l'algorithme (rsa1, rsa et dsa)

    ✔ -p changer sa phrase d'identification

    exemple. ssh-keygen -t rsa

    ✔ Definir la phrase d'identification pour protejer la clé privée

- Mise en mémoire des clés (évite la saisie répétée des phrases d'identification
  - **ssh-agent**

- Chargement des clés dans l'agent
  - **ssh-add**

Distribution des clés

- Problématique de l'installation de la clé privée(id_rsa.pub) sur le serveur distant dans le fichier des clés (authorized_keys) :
- Soit transférer la clé publique
  - On dispose de la clé publique sur la machine locale
  - Il faut la transférer et copier son contenu dans un fichier nommé authorized_keys sur la machine distante
  - Ce fichier va donc pouvoir contenir plusiers clés publiques des utilisateurs
- Soit envoyer le fichier (clé publique) à l'administrateurdu serveur distant qui l'installera sur la machine

Distribution des clés

- **Methode manuelle**
  - Sur le client:
    - ✔ La clé publiuqe est dans le reprtoire local
    - ✔ Transferer la clé publique sur le serveur dans le repertoire $HOME/.ssh
  - Sur le serveur
    - ✔ cat id_rsa.pub >> authorized_keys
    - ✔ chmod 600 authorized_keys
- **Methode automatique**
  - Sur le client:
    - ✔ ssh-copy-id serveur
      - Transferer la clé publique directement dans les fichier authorized_keys sur le serveur distant
  - Sur le serveur
    - ✔ Rien a faire

Transfert d'agent

- Relais des demandes d'authentification entre hôtes
- Permet de cascader les hôtes sans avoir à donner d'authentification supplémentaire
- Nécessite une configuration administrateur sur toutes les machines à cascader.
- Il faut démarrer un agent au départ de la machine cliente source, et activer le chargement des clés en mémoire
- Et activer le transfert d'agent au niveau de tous les serveurs concernés dans le fichier de configuration "client"

  /etc/ssh/ssh_config : **ForwardAgen**t **yes**

Tunneling

- C’est une méthode d'utilisation de SSH pour sécuriser les autres applications TCP sensibles.
- Son principe est de créer un tunnel chiffré entre deux machines et d'y faire passer les applications dedans. Ainsi, onpeut continuer à utiliser des outils comme FTP, IMAP, POP, smtp ... de manière sécurisé (pour FTP: cela dépend du paramétrage du serveur ftp)
- Au niveau sécurité, les tunnels posent quelques soucis car les ports locaux sur lesquels des transferts sont établis sont accessibles à tous les utilisateurs de la machine (notamment les passerelles)

- Position du problème
  - Il s'agit de rediriger des services souhaités et que l’on ne peut accéder normalement car ils sont filtrés; et d’utiliser la connexion ssh (seule acceptée) pour y faire passer ces services
  - Autres services inaccessibles directement
  - Service ssh accessible

Tunneling

Exemples

Tunneling : cas d'un intranet

- Créer le tunnel sur la machine locale:
  - ssh –L 18000:login@ServWeb:80 passerelle
- Puis lancer le navigateur Web avec l'URL : http://localhost:18000

Consultation de sa messagerie par IMAP

- Création du tunnel pour accéder au serveur Imap par le tunnel ssh
  - ssh –L 14300:localhost:143 machine2
  - ou : ssh –N –L 14300:localhost:143 machine2
- Puis configurer l'application client imap sur localhost:14300

Conclusions

- SSH est une boîte à outils complète
  - Les connexions sont sécurisées
- Améliore la sécurité mais il permet aussi de la contourner
- Existe différentes possibilités de connexions et de transferts de fichiers
- Les relais possibles des applications TCP permettentd'accéder à de nouvelles ressources
- Installé en standard sur tous les systèmes (client etserveur) Unix et MacOS X
- De très bons clients pour les plate-formes existantes