SUP'COM
Higher School of Communication of Tunis

Administration de la Sécurité sous Linux

Durcissement du SSH: Cas d'OpenSSH

Master Cybersécurité Opérationnelle : M2

02 Nov – 19 Déc 2020

Dr. Ing. Nizar Ben Neji
nizar.benneji@supcom.tn

2020 / 2021

Secure Shell

- SSH (Secure Shell) est un protocole applicatif (couche 7 du modèle de l'OSI) sécurisé de transfert conçu pour remplacer les protocoles non sécurisés d'échange de données: FTP, RCP, TELNET, RLOGIN et RSH qui doivent être désinstallés du système.
- SSH est couramment utilisé pour l'**administration distante** et pour le **transfert de fichiers**, il est important de **maîtriser sa configuration**, de **durcir son installation** et de respecter les **règles correctes de son exploitation**
- SSH est standardisé par une série de RFC (4250 à 4254) qui spécifient ses protocoles de communication et les mécanismes cryptographiques qu'il doit supporter
- SSH existe en 2 versions : SSHv1 et **SSHv2**. SSHv1 présente des vulnérabilités structurelles qui ont été corrigées dans la version suivante
- SSH garantit une communication sécurisée grâce aux trois sous protocoles:
  - **SSH-USERAUTH**, protocole d'authentification de la partie client – RFC 4252
  - **SSH-TRANS**, protocole de transfert sécurisé, permettant l'authentification du serveur et l'établissement d'un canal de communication sécurisé (confidentialité et intégrité) – RFC 4253
  - **SSH-CONNECT**, protocole de connexion permettant le multiplexage de canaux de communication (données et commandes) – RFC 4254

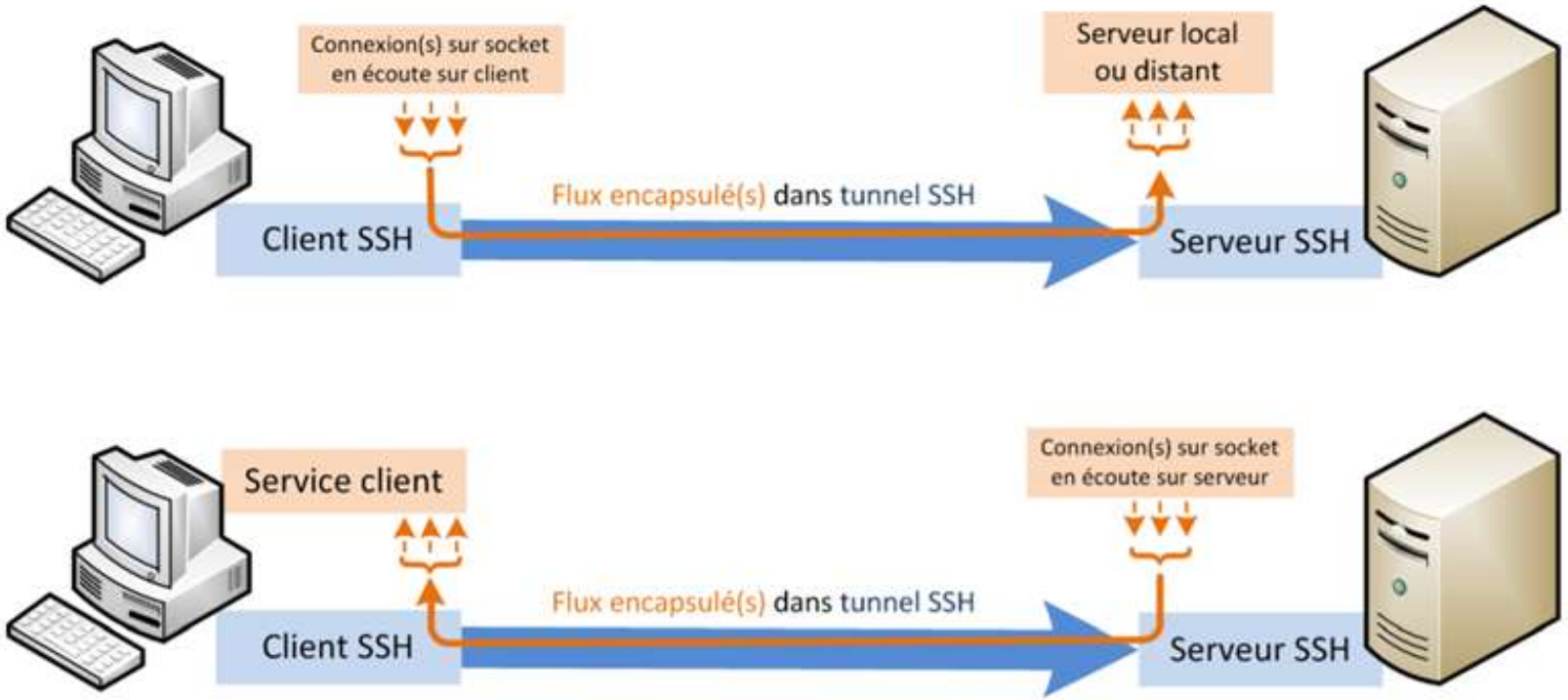
OpenSSH

- **OpenSSH** est l'implémentation de référence du protocole SSH que l'on retrouve sur un grand nombre de systèmes, aussi bien des serveurs que des postes clients ou des équipements réseau.
- Il inclue une suite d'outils offrant de nombreuses fonctionnalités. Cette suite logicielle est composée de nombreux outils :
  - un serveur **sshd**
  - plusieurs clients, suivant les usages :
    - connexion shell distante : **ssh**
    - transfert et téléchargement de fichiers : **scp**, **sftp**
  - un outil de génération de clés, **ssh-keygen**
  - un service de trousseau de clés, **ssh-agent** (conservation des informations sur clés et mots de passe pour session) et **ssh-add** (ajout des clés privées SSH au niveau de l'agent d'authentification SSH)
  - un scanner de clés publiques présentes sur les serveurs SSH, **ssh-keyscan**
- ssh remplace TELNET, RSH, RLOGIN
- scp et sftp remplacent RCP et FTP

Cas d'usage

- **Administration à distance en ligne de commande:** L'administration à distance en ligne de commande est le cas d'usage le plus répandu de SSH. Il consiste à se connecter à une machine distante et à lancer une session shell une fois les opérations d'authentification réussies.
- **Transfert et téléchargement de fichiers:** Le deuxième cas d'usage relativement fréquent de SSH est le transfert de fichier, à la fois dans le sens montant (client vers serveur) et descendant (serveur vers client). SSH propose deux mécanismes : SCP et SFTP. SFTP est plus élaboré que SCP, et permet une navigation dans une arborescence là où SCP se borne à permettre de transférer des données. SSH permettant de multiplexer le canal de données et celui de contrôle dans la même connexion, le flux ne requiert pas une ouverture dynamique de port à la différence d'un protocole comme FTP.
- **Redirection de flux**: La redirection de flux est une fonctionnalité couramment utilisée via SSH. Elle consiste à encapsuler des flux TCP/IP directement dans le tunnel SSH, pour permettre (entre autres) de sécuriser le transport d'un protocole non sécurisé, ou de donner accès à des services qui sont a priori protégés derrière une passerelle. Ces redirections peuvent intervenir sur le client SSH ou sur le serveur.

Cas d'usage


Connexion(s) sur socket
en écoute sur client
Serveur local
ou distant
Flux encapsulé(s) dans tunnel SSH
Client SSH
Serveur SSH
Service client
Connexion(s) sur socket
en écoute sur serveur
Flux encapsulé(s) dans tunnel SSH
Client SSH
Serveur SSH

Bonnes Pratiques

Authentification et Choix des Algorithmes Asymétriques

- SSH repose très largement sur la **cryptographie asymétrique** pour l'authentification donc Il faut s'assurer de la légitimité du serveur contacté avant de poursuivre l'accès. Cela passe par l'authentification préalable de la machine au travers de l'empreinte de sa **clé publique**, ou d'un **certificat** valide et vérifié.
- Avec OpenSSH, ce contrôle se fait de plusieurs façons :
  - en s'assurant que l'empreinte de la clé présentée par le serveur est la bonne (obtenue préalablement avec **ssh-keygen -l**)
  - en rajoutant la clé manuellement dans le fichier **known_hosts**
  - en vérifiant la signature du certificat présenté par le serveur avec une autorité de certification (AC) reconnue par le client
- OpenSSH adopte par défaut un modèle de sécurité **Trust On First Use (TOFU)** : lors de la première connexion et à défaut de pouvoir authentifier l'hôte, ssh demande confirmation à l'utilisateur qu'il s'agit bien de la bonne clé (via son empreinte). Si l'utilisateur confirme que l'empreinte est bonne, ssh procèdera à son enregistrement dans le fichier **known_hosts** afin de permettre sa vérification lors des visites suivantes.
- Génération et usage des clés d'authentification (taille et algorithmes):
  - Usage du DSA n'est pas recommandé (à désactiver)
  - Usage de RSA est recommandé avec une taille de clés minimale de 2048
  - Usage d'ECDSA est recommandé avec une taille de clés minimale de 256

Bonnes Pratiques

Clés d'authentification

- Afin de supprimer l'utilisation des clés DSA:
  - client ssh : supprimer les fichiers **~/.ssh/id_dsa** et **~/.ssh/id_dsa.pub** ;
  - serveur sshd : mettre en commentaires les lignes HostKey pointant vers une clé DSA (comme **/etc/ssh/ssh_host_dsa_key**).
- Des clés respectant ces exigences peuvent être générées par les commandes suivantes :

```
ssh-keygen   -t   rsa   -b   2048   -f
ssh-keygen   -t   ecdsa   -b   256   -f
```

ssh-keygen génèrera deux fichiers : clé privée et clé publique (nom de fichier se termine par un .pub). Ce sont ces fichiers qui sont ensuite utilisés comme clé d'identification hôte (attribut **HostKey** de sshd), ou utilisateur (attribut **IdentityFile** de ssh).

- Les clés privées générées par OpenSSL peuvent être directement utilisées par **ssh-keygen** et produire la clé publique correspondante :

```
openssl genrsa -aes128 -passout stdin -rand /dev/urandom 2048 > <clé privée>
ssh-keygen -y -f  « clé privée »   >   « clé publique »
```

Bonnes Pratiques

Encodage et Conversion des clés de chiffrement

- Depuis la version 6.1, OpenSSH est capable de lire différents types d'encodage de clés publiques. Cela permet notamment d'importer des clés distribuées par certificats X.509 sans nécessiter d'accès à la clé privée, ou d'utiliser des fichiers au format PKCS12:

Les clés publiques sont importées via ssh-keygen :

```
# Conversion d'une clé encodée en PEM vers une clé publique OpenSSH
openssl x509 -pubkey -noout -in  <certificat>| ssh-keygen -i -m PKCS8 -f  /dev/stdin

# Récupération clé publique d'un certificat X.509 et import OpenSSH
openssl x509 -pubkey -noout -in | ssh-keygen -i  -m PKCS8 -f /dev/stdin

# Récupération clé privée et obtention de la clé publique OpenSSH
# correspondante à partir d'un fichier PKCS12
openssl pkcs12 -in <fichier p12> -nocerts -aes128 > <clé privée>
openssl pkcs12 -in <fichier p12> -nocerts -nodes | ssh-keygen -y  -f /dev/stdin > <clé publique>
```

Bonnes Pratiques

Choix des algorithmes de chiffrement symétrique

- Les clés d'authentification SSH peuvent être regroupées selon deux rôles :
  - celles utilisées pour l'authentification d'utilisateurs ;
  - celles utilisées pour l'authentification d'un hôte/serveur
- Le serveur sshd doit vérifier la rectitude des modes et droits des fichiers de l'utilisateur avant de le laisser ouvrir une session. Ceci se configure via la directive **StrictModes** du fichier **sshd_config**:

```
StrictModes   yes
```

- Une fois les parties sont mutuellement authentifiées, le canal de communication est protégé en chiffrement et en intégrité. L'algorithme de chiffrement doit reposer sur l'une de ces algorithmes l'**AES 128**, **AES 192** ou **AES 256** en mode **CTR (Counter Mode)**. L'intégrité doit reposer sur du **HMAC SHA-1**, **SHA-256** ou **SHA-512**.

NB:

- Le mode CTR permet de s'affranchir de vulnérabilités connues du mode CBC
- Les HMAC SHA-256 et SHA-512 ne sont supportés que depuis la version 5.9 d'OpenSSH. Les systèmes d'exploitation utilisant des versions antérieures doivent alors reposer sur un HMAC SHA-1.

Bonnes Pratiques

Choix des algorithmes de chiffrement symétrique

- Sous OpenSSH, les directives suivantes doivent être rajoutées dans le **sshd_config** et le **ssh_config** :

```
Ciphers  aes256-ctr,aes192-ctr,aes128-ctr
# Pour les versions 6.3+, OpenSSH supporte le ETM (encrypt-then-mac),
# plus sûr que les anciennes implémentations (mac-then-encrypt)
MACs   hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com
# S'il s'agit d'une ancienne version, utilisez uniquement "hmac-sha1"
MACs   hmac-sha2-512,hmac-sha2-256,hmac-sha1
```

- Les droits d'un utilisateur doivent suivre le principe de moindre privilège. La restriction peut porter sur de nombreux paramètres comme l'adresse IP d'origine ou le forwarding ssh.
- Quand les droits ne peuvent être appliqués directement au niveau du serveur (via sshd_config), le fichier **authorized_keys** permet de spécifier un ensemble d'options qui restreignent les droits associés à une clé donnée :

```
# La clé n'autorise l'accès que si le client provient du réseau 192.168.15/24
from="192.168.15.*" ssh-ecdsa AAAAE2Vj...
```

Bonnes Pratiques

- L'accès aux comptes sans mot de passe doit être proscrit dans **sshd_config**. L'opération d'authentification doit être d'une durée relativement courte et le nombre de tentatives doit être limitée à une par connexion.

```
PermitEmptyPasswords no
# Attention, MaxAuthTries doit être strictement supérieur à 1.
# De plus, un chiffre trop faible peut poser souci.
MaxAuthTries 2
LoginGraceTime 30
```

- OpenSSH fournit un utilitaire (ssh-agent) permettant de garder en mémoire les clés privées d'un utilisateur lorsque celui-ci s'authentifie par des mécanismes cryptographiques asymétriques comme RSA ou ECDSA.
- Il est assez fréquent pour un utilisateur de rebondir successivement d'un hôte SSH vers un autre hôte SSH. La redirection de l'agent d'authentification (ou **Agent Forwarding**) (option -A de ssh) permet à un hôte relais de rediriger les requêtes d'authentification du second serveur vers le poste client où s'exécute l'agent qui se chargera de l'opération d'authentification. Ce mécanisme permet de conserver la clé privée utilisateur hors d'atteinte du serveur relais tout en bénéficiant des avantages de la cryptographie par clé publique. Le serveur hôte relais doit être un hôte de confiance.

Bonnes Pratiques

- Chaque utilisateur doit disposer de son propre compte, unique, incessible. Le compte root est un exemple de compte générique que l'on retrouve sur tous les systèmes Unix/Linux, et utilisé comme compte d'administration. Le compte root ne doit pas être accessible directement à un utilisateur :

```
PermitRootLogin no
```

- Le possession d'un compte dédié pour chaque utilisateur permet une gestion plus fine des accès et une meilleure traçabilité. La directive PrintLastLog du sshd_config permet de spécifier au serveur qu'il doit afficher les informations de dernière connexion à l'utilisateur lorsqu'il se connecte.

```
PrintLastLog  yes
```

- Les accès à un service doivent être restreints aux utilisateurs qui en ont un besoin justifié. Cette restriction doit s'appliquer en droits positifs : uniquement ceux explicitement autorisés ont le droit de se connecter en SSH sur un hôte, et éventuellement en provenance d'adresses IP spécifiées. Sous OpenSSH, la directive **AllowUsers** permet de spécifier la liste des utilisateurs autorisés à se connecter au service SSH. Dans le cas où un groupe d'utilisateurs est concerné (administrateurs, utilisateurs avec pouvoir. . .), utiliser **AllowGroups**.

Bonnes Pratiques

- Dans la mesure où sshd doit être exposé à des adresses IPs qui ne sont pas exclusives à un réseau d'administration, ces directives permettent de restreindre les accès utilisateurs suivant leur adresse IP :

```
# Restreint 'admin' aux adresses IPs d'un réseau d'administration
# (192.168.17/24) et 'user' aux adresses d'un réseau Intranet (10.127/16)
AllowUsers admin@192.168.17/24 user@10.127/16
# Même chose, mais avec un groupe 'administrateurs' et 'users'
AllowGroups administrateurs@192.168.17/24 users@10.127/16
```

- Le serveur SSH doit écouter uniquement sur une adresse d'administration et lorsque le serveur SSH est exposé à un réseau non maîtrisé, il est recommandé de lui mettre un port d'écoute différent du port par défaut (22). Il faut privilégier un port inférieur à 1024 afin d'empêcher les tentatives d'usurpation par des services non administrateur sur la machine distante. Sur un réseau maîtrisé, le serveur SSH doit écouter uniquement sur une interface du réseau d'administration, distinct du réseau opérationnel.

Avec OpenSSH, la spécification de l'adresse et du port d'écoute se fait au travers de la directive ListenAddress du sshd_config :

```
# Ecoute uniquement sur l'adresse A.B.C.D, port 26
ListenAddress A.B.C.D:26
```

Bonnes Pratiques

- Sauf besoin dûment justifié, toute fonctionnalité de redirections de flux doit être désactivée :

— au niveau de la configuration du serveur SSH
— au niveau du pare-feu local en bloquant les connexions

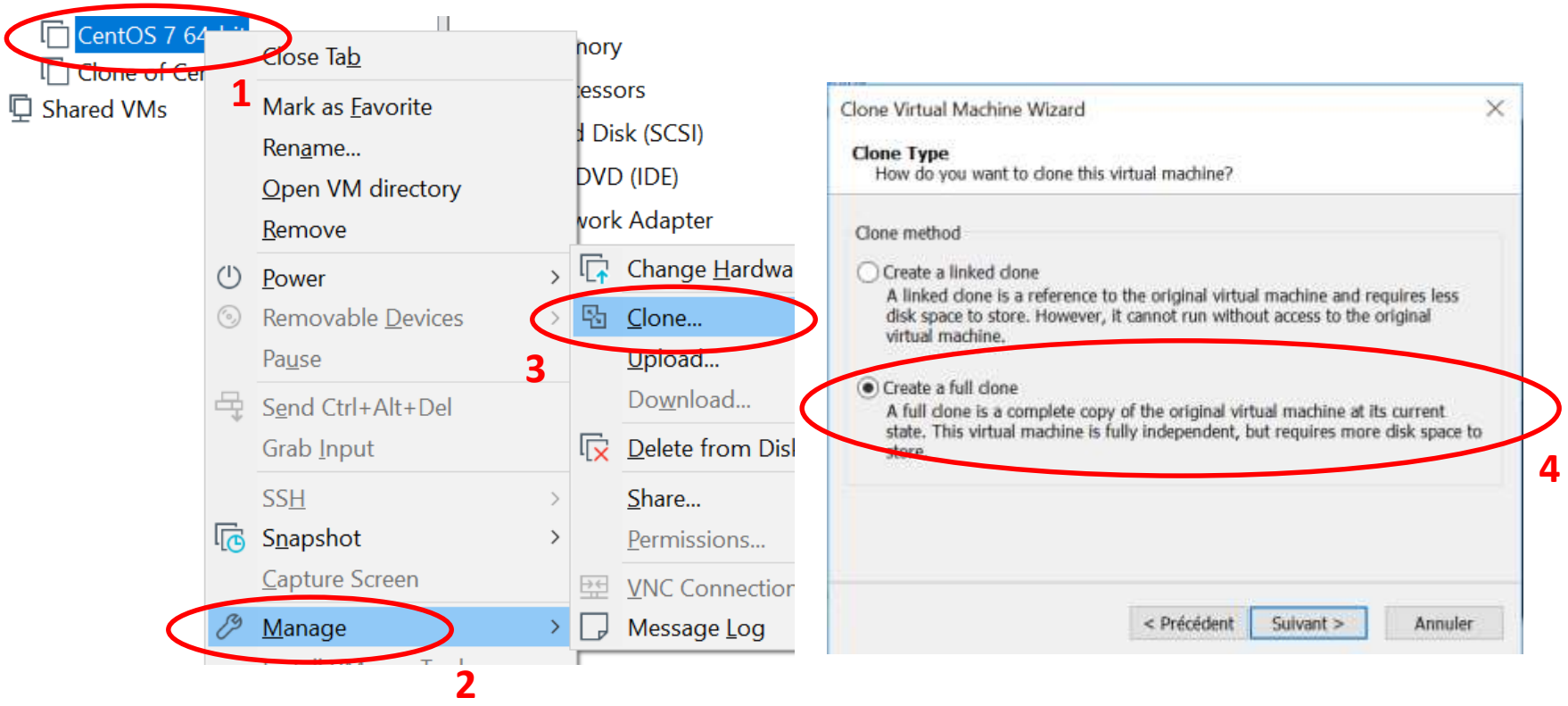
- Sous OpenSSH, les redirections côté serveur se désactivent au travers de la directive AllowTcpForwarding de sshd_config :

```
AllowTcpForwarding no
```

- Depuis la version 5.4, OpenSSH offre une PKI pour faciliter la gestion des clés

TP

- Créer une deuxième machine qui sera la machine cliente SSH: nouvelle installation ou bien juste cloner la machine déjà créé.



Préparation et lancement du SSH

- Vérifier la présence des paquetages d'OpenSSH Serveur et Client au niveau des deux machines comme suite:

```
# rpm  -qa | grep  openssh-server
# rpm -qa | grep openssh-clients
```

- Au niveau de chaque machine, installer ***OpenSSH***:

```
# sudo  yum  install openssh-server
# sudo  yum  install openssh-clients
```

- Au niveau de la **machine serveur**:
  - Démarrer le service SSH comme suite:
    - # sudo systemctl start sshd
  - Lancer SSHD au démarrage de la machine:
    - # sudo systemctl enable sshd
  - Vérifier puis ajouter le service SSH si nécessaire au niveau du parefeu puis recharger la configuration du parefeu:
    - # sudo firewall-cmd --permanent --list-services
    - # sudo firewall-cmd --permanent --add-service=ssh
    - # sudo firewall-cmd --reload

Accès du Client OpenSSH

- Au niveau serveur, créer aussi un utilisateur nommé **admin**:

```
# su  -
# adduser  admin
# passwd  admin
```

- A partir de la machine cliente, accéder en mode ssh avec l'utilisateur **admin** comme suite:

```
# ssh  admin@ADRESSE_IP_SERVEUR
The authenticity of host '192.168.1.4 (192.168.1.4)' can't be established.
ECDSA key fingerprint is SHA256:ZfXvICQz77t90GgUbyl9nTOk1mdSYtVXIh80ikWzzWY.
ECDSA key fingerprint is MD5:34:64:6c:1d:b5:e1:a0:f8:03:03:e0:db:e8:f1:c3:94.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.4' (ECDSA) to the list of known hosts.
admin@192.168.1.4's password:
Last login: Thu Oct  8 06:03:23 2020 from 192.168.1.5
```

- Pour faire une déconnexion :

```
# exit
```

- Une fois l'empreinte de la **clé publique du serveur** est vérifiée et acceptée, elle sera ajoutée en permanence au fichier **known_hosts** de l'utilisateur courant (supcom):

```
# cat  /home/supcom/.ssh/known_hosts
```

Accès du Client OpenSSH

- Au niveau de la **machine cliente** et avec l'utilisateur **supcom**, utiliser l'outil de génération de clés **ssh-keygen** pour la génération d'une paire de clés RSA comme suite:

```
# ssh-keygen -t rsa -b 4096 -f ~/.ssh/id_rsa
```

```
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/supcom/.ssh/id_rsa.
Your public key has been saved in /home/supcom/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:OiLNmyIIky5RZvSOiU4IJ3weGDXtgz3AEjRu1u9B9V0 supcom@localhost.localdomain
The key's randomart image is:
+---[RSA 4096]----+
|.++o.  .     E   |
|.o+o... . . .    |
|.=+o=.   . .     |
|=o=++=           |
|oOo+.oo S        |
|B.o+o ..         |
|*o. +.o          |
|+o.. + .         |
|.. .o            |
+----[SHA256]-----+
```

- Afficher la clé publique récemment crée comme suite:

```
# cat /home/supcom/.ssh/id_rsa.pub
```

Accès du Client OpenSSH

- Copier le contenu de la clé publique manuellement et coller le au niveau du fichier **authorized_keys** du coté serveur dans l'espace **.ssh** de l'utilisateur **admin**

```
# vi   /home/admin/.ssh/authorized_keys  # (copier/coller) ou
# scp ~/.ssh/id_rsa.pub    <serveur destination>
```

- A partir de la machine cliente, accéder en ssh de nouveau sur le serveur avec la nouvelle clè RSA (le mot de passe de protection de la clé privé sera demandé):

```
# ssh  admin@ADRESSE_IP_SERVEUR
  Enter passphrase for key '/home/supcom/.ssh/id_rsa':
```

- Au niveau la machine cliente, générer une deuxième clé avec OpenSSL puis extraire la clé publique SSH correspondante comme suite:

```
# openssl  genrsa  -out  /home/supcom/.ssh/id_rsa_4096.key  -des3  4096
# ssh-keygen  -y  -f  /home/supcom/.ssh/id_rsa_4096.key  > .ssh/id_rsa_4096.pub
```

- De la même manière, ajouter la nouvelle clé publique au niveau du fichier **authorized_keys** au niveau du serveur.
- Au niveau de la machine cliente, créer au niveau du répertoire **.ssh** de l'utilisateur **supcom**, un fichier config la ou on va indiquer au client la nouvelle clè avec laquelle il va dorénavant s'authentifier:

```
# vi  /home/supcom/.ssh/config
  IdentityFile /home/supcom/.ssh/id_rsa_4096.pub
```

Accès du Client OpenSSH

- Tester de nouveau l'accès SSH avec la nouvelle clè:

```
# ssh  admin@ADRESSE_IP_SERVEUR
  Enter passphrase for key '/home/supcom/.ssh/id_rsa_4096.pub':
  admin@192.168.1.4's password:
```