SSH – Authentification par clés

Génération des clés

Les clés seront générées et sauvegardées automatiquement dans le répertoire *$HOME/.ssh* de l’utilisateur courant.

Une passphrase (un mot de passe) sera demandée pendant la procédure de

génération : elle permet de protéger la clé privée au cas où un tiers s'en emparerait. La clé privée protégée ne pourra pas être utilisée sans la passphrase que vous avez renseignée.

Si vous ne souhaitez pas que la clé soit protégée, vous pouvez laisser le champ vide et valider.

Notez qu'on peut ajouter un commentaire, dans le fichier de la clé publique, avec la commande [redacted].

Vous obtenez donc 2 fichiers :

- « *id_rsa* » , la clé privée
- « *id_rsa.pub* » , la clé publique

Téléchargez la clé privée sur votre ordinateur local (le client), est sauvegardez-là en lieu sûr : elle ne doit en aucun cas être volée par un tiers, puisque c'est elle qui autorise les

connexions.

Une fois téléchargée, vous devez supprimer la clé privée du serveur :

N.B. : ici nous avons généré les clés sur un serveur distant, il aurait été plus convenable de les générer sur le poste client (on évite ainsi le transfert de la clé privée).

Conversion de la clé privée au format PuTTY

Pour pouvoir vous connecter avec PuTTY ou WinSCP, la clé privée doit être au format PuTTY, voici donc la procédure à suivre :

·téléchargez PuTTYgen (site

officiel),

- exécutez PuTTYgen et allez dans « Conversions » puis « Import key »,
- ensuite sélectionnez la clé privée « *id_rsa* » pour la charger dans le programme,
- cliquez sur le bouton « Save private key » et choisissez un fichier de destination pour sauvegarder la clé au format PuTTY « *.ppk*« .

Autoriser la clé publique

Nous devons autoriser les connexions par jeu de clés sur le serveur, pour cela il suffit d’ajouter la clé publique au fichier « *authorized_keys* » (ou « *authorized_keys2* » mais ce nom de

fichier est déprécié, même si les deux fonctionnent normalement) de l’utilisateur avec lequel on va se connecter.

Veuillez noter que le serveur peut gérer plusieurs clés publiques : il suffit d’ajouter une clé par ligne dans le fichier « *authorized_keys*« .

Activer l’authentification par clés

Vous devez ajouter/modifier les directives suivantes dans le fichier de configuration d’OpenSSH (*/etc/ssh/sshd_config*) :

Rechargez la configuration du serveur OpenSSH :

Testez la connexion en utilisant la clé privée avec PuTTY ou WinSCP.

Désactiver les mots de passe

Vous pouvez désactiver la connexion par mot de passe si le test a été concluant.

Vous devez ajouter/modifier les directives suivantes dans le fichier de configuration d’OpenSSH (*/etc/ssh/sshd_config*) :

Rechargez la configuration du serveur OpenSSH :

Limiter les connexions

Vous pouvez spécifier quels utilisateurs sont autorisés à se connecter via le serveur SSH, il suffit pour cela d’ajouter leur nom à la directive « *AllowUsers* » en les séparant par un espace, toujours dans le fichier de configuration d’OpenSSH (*/etc/ssh/sshd_config*) :

Rechargez la configuration du serveur OpenSSH :

Connexion

Pour se connecter avec le client SSH dans un terminal :

La spécification du port avec le paramètre [redacted] est facultative, par défaut le port 22 est utilisé.

On peut forcer l’utilisation d’une clé privée avec le paramètre [redacted], on peut l’omettre si les clés sont bien présentes dans le dossier [redacted] (le client tentera alors la connexion avec chaque clé privée trouvée), ou dans le cas où un fichier de configuration [redacted] existe et est correctement renseigné.

A des fins de debugging, pour savoir quelles clés sont proposées au serveur, on peut lancer une connexion en mode verbeux et chercher les lignes commençant par « Offering… »

Configuration de

connexion client

Le fichier [redacted] permet de s’affranchir du passage de plusieurs paramètres au client SSH. Voici le format utilisé :

On utilisera le client de la manière suivante :

Notes

Les utilisateurs créés avec la commande « *useradd* » doivent avoir un mot de passe valide, dans le cas contraire, les utilisateurs seront bloqués et ne pourront pas se connecter. On peut vérifier la présence d’un mot de passe dans le fichier « */etc/shadow* » : celui-ci contient un « *!* » après le nom d’un utilisateur

bloqué.

Voir aussi : https://wiki.debian.org/fr/SSH.

Installation de Memcached (à partir des sources)

Installation de Libevent

Voir : Installation de Libevent.

Compilation de Memcached

On teste

Si ça marche on a quelque chose de ce goût là en sortie

On arrête

On vérifie qu’un ou plusieurs serveurs

sont démarrés

Tuer les processus Memcached, si on les a démarré en tant que démon

Voir aussi : Installation de Memcached.

Installation de Libevent

Préliminaire

Vérifier si Libevent est déjà installé

On enlève l’ancienne version de Libevent le cas échéant (et Memcached)

On vérifie la version installée

Compilation

On rafraichit le cache des chemins vers les librairies partagées

Installation de Nginx, PHP 5.3 et PHP-FPM

Nginx

Configuration de Nginx

Il est possible que « fastcgi_split_path_info » ne soit pas supporté, car cette variable de configuration n’est disponible que dans les versions récentes de Nginx.

PHP, PHP-FPM et ses modules

On utilise le dépôt DotDeb

Et on installe PHP

Initialisation

Mémo MySQL

Connexions persistantes

Les connexions persistantes permettent de ré-utiliser une connexion déjà ouverte par PHP. Elles ne se terminent pas à la fin de l’exécution d’un script. L’intérêt est

de limiter le temps de latence dû à l'ouverture de la connexion, ce qui peut-être utile lorsque le serveur de base de données et HTTP ne se situent pas sur le même serveur physique.

Dans le cas d'Apache, une connexion par processus fils est ouverte, il faut donc en tenir compte lorsqu'on fixe la valeur « max_connections » du serveur MySQL. En effet, deux requêtes consécutives peuvent être gérées par des processus Apache différents, ce qui engendre l'ouverture d'une connexion persistante pour chaque processus ! La charge de la machine, en terme de mémoire utilisée notamment, va augmenter de façon proportionnelle aux nombres de connexions simultanées.

Attention, une transaction qui n'est pas terminée, restera active entre les requêtes Apache ! De même, quand vous bloquez (lock) une table, normalement elle est débloquée lorsque la connexion est fermée, mais comme les connexions persistantes ne se ferment pas (sic), les tables que vous avez quittées en état

bloquées resteront bloquées, et la seule façon de les déloquer sera d’attendre que la connexion atteigne le timeout (« wait_timeout ») ou de tuer le processus MySQL. Pour finir, les options de connexions restent actives entre les appels, par exemple :

Dans MySQL, les tables temporaires sont visibles uniquement par la connexion courante, mais si vous avez une connexion persistante, la table temporaire sera visible par tous les scripts qui partagent la même connexion persistante.