

Introduction à SSH

Lucas Nussbaum

lucas.nussbaum@univ-lorraine.fr

Traduction Française initiale : François Dupont (ASRALL 2015)

Licence professionnelle ASRALL

Administration de systèmes, réseaux et applications à base de logiciels libres



UNIVERSITÉ
DE LORRAINE



nancy Charlemagne
Département Informatique

Plan

- 1 Les bases de SSH
 - Introduction
 - Authentification par clé publique
 - Vérifier l'identité du serveur
 - Configurer SSH

- 2 Utilisation avancée
 - SSH : couche de communication générique
 - Accès à un filesystem distant : sshfs
 - SSH tunnels, X11 forwarding, and SOCKS proxy
 - VPN sur SSH
 - Passer d'hôtes en hôtes avec ProxyCommand
 - Déclencher des commandes distantes
 - Séquences d'échappement
 - Autres outils liés

- 3 Conclusion

Introduction

- ▶ SSH = Secure SHell
- ▶ Un protocole et un service réseau standard (port TCP 22)
- ▶ De nombreuses implémentations, dont :
 - ◆ OpenSSH : Linux/Unix, Mac OS X ← on parlera surtout de ça
 - ◆ Putty, MobaXterm : Windows, client seulement
 - ◆ Dropbear : systèmes légers (routers, embarqué)
- ▶ Commande Unix (ssh) ; coté serveur : sshd
- ▶ Établit une **communication sécurisée** entre deux machines
- ▶ S'appuie sur la cryptographie
- ▶ L'usage le plus simple : **l'accès shell** sur une machine distante
- ▶ De nombreux usages avancés :
 - ◆ Transférer de données (scp, sftp, rsync)
 - ◆ Se connecter à des services spécifiques (Git ou SVN)
 - ◆ Creuser des tunnels sécurisés à travers Internet
- ▶ Plusieurs systèmes d'authentification : mot de passe, clé publique

Utilisation basique

- ▶ Se connecter à un serveur distant :
`$ ssh login@remote-server`
↪ Fournit un shell sur *remote-server*
- ▶ Exécuter une commande sur un serveur distant :
`$ ssh login@remote-server ls /etc`
- ▶ Copier des données (avec `scp`, similaire à `cp`) :
`$ scp local-file login@remote-serv:remote-directory/`
`$ scp login@remote-serv:remote-dir/file local-dir/`
Les options habituelles de `cp` fonctionnent, comme `-r` (récursif)
- ▶ Copier des données (avec `rsync`, plus efficace que `scp` s'il y a de nombreux fichiers) :
`$ rsync -avzP localdir login@server:path-to-rem-dir/`

Note : le slash de fin est important avec `rsync` (pas avec `cp`)

- ◆ `rsync -a rep1 u@h:rep2` ↪ rep1 copié dans rep2
- ◆ `rsync -a rep1/ u@h:rep2` ↪ contenu de rep1 dans rep2

Authentification par clé publique

- ▶ Idée générale
 - ◆ Cryptographie asymétrique (ou cryptographie à clé publique)
 - ★ La clé publique est utilisée pour chiffrer quelque chose
 - ★ Seule la clé privée peut le déchiffrer
 - ◆ L'utilisateur possède une clé privée (secrète), stockée sur la machine locale
 - ◆ Le serveur a la clé publique correspondant à la clé privée
 - ◆ Authentification = *<server> prouve que tu possèdes cette clé privée!*
- ▶ Implémentation (*Authentification par challenge-réponse*) :
 - 1 Le serveur génère un nonce (une valeur aléatoire arbitraire)
 - 2 Le serveur chiffre ce nonce avec la clé publique du client
 - 3 Le serveur envoie le nonce chiffré (= le challenge) au client
 - 4 Le client utilise la clé privée pour déchiffrer le challenge
 - 5 Le client renvoie ce nonce (= la réponse) au serveur
 - 6 Le serveur compare le nonce avec la réponse

Authentification par clé publique (2)

- ▶ Avantages :
 - ◆ Les mots de passe ne sont pas envoyés par le réseau
 - ◆ La clé privée ne quitte JAMAIS le client
 - ◆ La même clé peut être utilisée pour plusieurs serveurs
 - ◆ Le procédé peut être automatisé
- ▶ Cependant, la clé privée doit être protégée (que se passerait-il si votre ordinateur portable était volé ?)
 - ◆ Habituellement avec une passphrase

Génération d'une paire de clé

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa): [ENTER]
Enter passphrase (empty for no passphrase): passphrase
Enter same passphrase again: passphrase
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
f6:35:53:71:2f:ff:00:73:59:78:ca:2c:7c:ff:89:7b user@my.hostname.net
The key's randomart image is:
+--[ RSA 2048]-----+
  ..o
 (... )
  .o
+-----+
$
```

- ▶ Crée une paire de clés :
 - ◆ ~/.ssh/id_rsa (clé privée)
 - ◆ ~/.ssh/id_rsa.pub (clé publique)

Copier la clé publique sur un serveur

- ▶ Exemple de clé publique :

```
ssh-rsa AAAAB3NX[. . .]hpoR3/PLlXgGcZS4oR user@my.hostname.net
```

- ▶ Sur le serveur, `~user/.ssh/authorized_keys` contient une liste des clés publiques autorisées à se connecter au compte user
- ▶ La clé peut y être copiée manuellement
- ▶ Ou `ssh-copy-id` peut être utilisé pour copier la clé :

```
client$ ssh-copy-id user@server
```
- ▶ Parfois la clé publique a besoin d'être fournie en utilisant une interface web (par exemple sur GitHub, FusionForge, Redmine, etc.)

Éviter de taper la passphrase

- ▶ Si la clé privée n'est pas protégée par une passphrase, la connexion est établie immédiatement :

```
*** login@laptop:~$ ssh rlogin@rhost [ENTER]
*** rlogin@rhost:~$
```

- ▶ Sinon, ssh demande la passphrase :

```
*** login@laptop:~$ ssh rlogin@rhost [ENTER]
Enter passphrase for key '/home/login/id_rsa': [passphrase+ENTER]
*** rlogin@rhost:~$
```

- ▶ Un **agent SSH** peut être utilisé pour stocker la clé déchiffrée
 - ◆ La plupart des environnements de bureau peuvent jouer le rôle d'agent SSH automatiquement
 - ◆ On peut lancer `ssh-agent` si besoin
 - ◆ Les clés peuvent être ajoutées manuellement avec `ssh-add`

Vérifier l'identité du serveur : known_hosts

- ▶ Objectif : détecter un serveur compromis
Et si quelqu'un se faisait passer pour un serveur pour voler des mots de passe ?
- ▶ Quand on se connecte à un serveur pour la première fois, ssh stocke la clé publique du serveur dans `~/.ssh/known_hosts`

```
*** login@laptop:~$ ssh rlogin@server [ENTER]
The authenticity of host 'server (10.1.6.2)' can't be established.
RSA key fingerprint is
94:48:62:18:4b:37:d2:96:67:c9:7f:2f:af:2e:54:a5.
Are you sure you want to continue connecting (yes/no)? yes [ENTER]
Warning: Permanently added 'server,10.1.6.2'(RSA) to the list of
known hosts.
rlogin@server's password:
```

Vérifier l'identité du serveur known_hosts (2)

- ▶ À chaque nouvelle connexion, ssh s'assure que la clé est toujours la même, ou avertit l'utilisateur dans le cas contraire

```
*** login@laptop:~$ ssh rlogin@server [ENTER]
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
e3:94:03:90:5d:81:ed:bb:d5:d2:f2:de:ba:31:18:d8.
Please contact your system administrator.
Add correct host key in /home/login/.ssh/known_hosts to get rid of this message.
Offending RSA key in /home/login/.ssh/known_hosts:12
RSA host key for server has changed and you have requested strict checking.
Host key verification failed.
*** login@laptop:~$
```

- ▶ Une clé véritablement périmée peut être supprimée avec `ssh-keygen -R server`

Configurer SSH

- ▶ SSH obtient les informations de configuration depuis :
 - 1 les options de la ligne de commande (-o ...)
 - 2 le fichier de configuration de l'utilisateur : `~/.ssh/config`
 - 3 le fichier de configuration système : `/etc/ssh/ssh_config`
- ▶ Ces options sont décrites dans la page de manuel `ssh_config(5)`
- ▶ `~/.ssh/config` contient une liste d'hôtes (avec wildcards)
- ▶ Pour chaque paramètre, la première valeur trouvée est utilisée
 - ◆ Les déclarations spécifiques à un hôte sont au début
 - ◆ Les paramètres par défaut sont à la fin

Exemple : ~/.ssh/config

```
Host mail.acme.com
    User root
```

```
Host foo # alias/raccourci. 'ssh foo' marche
    Hostname very-long-hostname.acme.net
    Port 2222
```

```
Host *.acme.com
    User jdoe
    Compression yes # default is no
    PasswordAuthentication no # only use public key
    ServerAliveInterval 60 # keep-alives for bad firewall
```

```
Host *
    User john
```

- ▶ Note : bash-completion auto-complète avec les hôtes de ssh_config

Plan

- 1 Les bases de SSH
 - Introduction
 - Authentification par clé publique
 - Vérifier l'identité du serveur
 - Configurer SSH

- 2 Utilisation avancée
 - SSH : couche de communication générique
 - Accès à un filesystem distant : sshfs
 - SSH tunnels, X11 forwarding, and SOCKS proxy
 - VPN sur SSH
 - Passer d'hôtes en hôtes avec ProxyCommand
 - Déclencher des commandes distantes
 - Séquences d'échappement
 - Autres outils liés

- 3 Conclusion

SSH : couche de communication générique

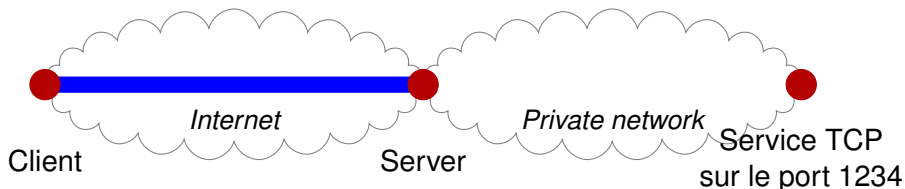
- ▶ Plusieurs applications utilisent SSH comme leur couche de communication et d'authentification
- ▶ scp, sftp, rsync (transfert de données)
 - ◆ lftp (CLI) et gftp (GUI) supportent le protocole SFTP
- ▶ unison (synchronisation)
- ▶ **Subversion** : `svn checkout svn+ssh://user@rhost/path/to/repo`
- ▶ **Git** : `git clone ssh://git@github.com/path-to/repository.git`
Ou : `git clone git@github.com:path-to/repository.git`

Accès à un filesystem distant : sshfs

- ▶ sshfs : Une solution basée sur FUSE pour accéder à des machines distantes
- ▶ Idéal pour éditer un fichier en GUI à distance, copier des petits fichiers, etc...
- ▶ Monter un répertoire distant :
sshfs root@server:/etc /tmp/local-mountpoint
Démonter : fusermount -u /tmp/local-mountpoint
- ▶ Combiné avec afuse pour monter automatiquement n'importe quelle machine :
afuse -o mount_template="sshfs %r:/ %m" -o \
unmount_template="fusermount -u -z %m" ~/.sshfs/
↪ cd ~/.sshfs/rhost/etc/ssh

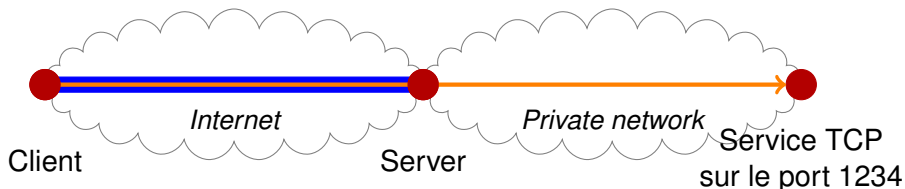
Tunnels SSH avec -L et -R

- ▶ Objectif : transporter le trafic dans une connexion sécurisée
 - ◆ Contourner le filtrage réseau (pare-feux)
 - ◆ Éviter d'envoyer des données en clair sur Internet
 - ◆ Mais fonctionne seulement pour les connexions TCP
- ▶ **-L** : accéder à un service distant derrière un pare-feu (serveur intranet)
 - ◆ `ssh -L 12345:service:1234 server`
 - ◆ Encore sur *Client* : `telnet localhost 12345`
 - ◆ *Server* établit une connexion TCP vers *Service*, port 1234
 - ◆ Le trafic est tunnelisé dans la connexion SSH vers *Server*



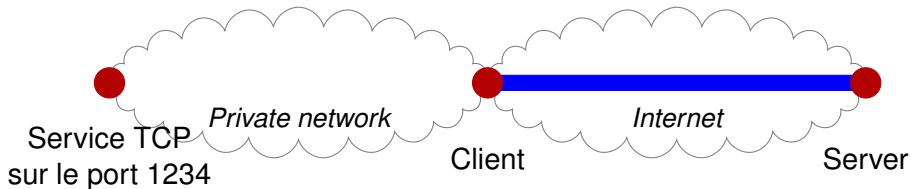
Tunnels SSH avec -L et -R

- ▶ Objectif : transporter le trafic dans une connexion sécurisée
 - ◆ Contourner le filtrage réseau (pare-feux)
 - ◆ Éviter d'envoyer des données en clair sur Internet
 - ◆ Mais fonctionne seulement pour les connexions TCP
- ▶ **-L** : accéder à un service distant derrière un pare-feu (serveur intranet)
 - ◆ `ssh -L 12345:service:1234 server`
 - ◆ Encore sur *Client* : `telnet localhost 12345`
 - ◆ *Server* établit une connexion TCP vers *Service*, port 1234
 - ◆ Le trafic est tunnelisé dans la connexion SSH vers *Server*



Tunnels SSH avec -L et -R

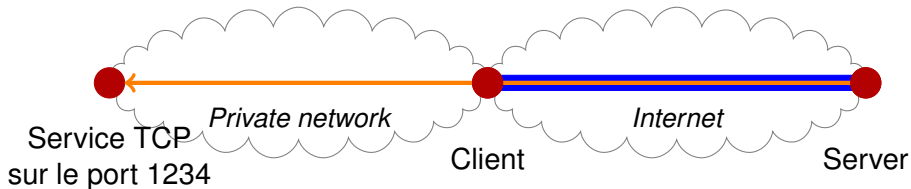
- ▶ **-R** : fournir un accès distant à un service local privé
 - ◆ `ssh -R 12345:service:1234 server`
 - ◆ Sur *Server* : `telnet localhost 12345`
 - ◆ *Client* établit une connexion TCP vers le *Service*, port 1234
 - ◆ Le trafic est tunnelisé dans la connexion SSH vers le *Client*



- ▶ Notes :
 - ◆ Note : les tunnels SSH marchent mal pour HTTP, car IP+port est insuffisant pour identifier un site web (en-tête Host :)
 - ◆ Les tunnels n'écoutent que localement. Pour permettre à des machines distantes de s'y connecter : `-g` (gateway)

Tunnels SSH avec -L et -R

- ▶ **-R** : fournir un accès distant à un service local privé
 - ◆ `ssh -R 12345:service:1234 server`
 - ◆ Sur *Server* : `telnet localhost 12345`
 - ◆ *Client* établit une connexion TCP vers le *Service*, port 1234
 - ◆ Le trafic est tunnelisé dans la connexion SSH vers le *Client*



- ▶ Notes :
 - ◆ Note : les tunnels SSH marchent mal pour HTTP, car IP+port est insuffisant pour identifier un site web (en-tête Host:)
 - ◆ Les tunnels n'écoutent que localement. Pour permettre à des machines distantes de s'y connecter : `-g` (gateway)

Applis GUI X11 à travers SSH : -X

- ▶ Lancer une application graphique sur une machine distante pour l'afficher localement
- ▶ Similaire à VNC mais fonctionne par applications
- ▶ `ssh -X server`
- ▶ `$DISPLAY` sera déclaré par SSH sur le serveur :
`$ echo $DISPLAY`
`localhost:10.0`
- ▶ Puis lancer les applications sur le serveur (par exemple `xeyes`)
- ▶ Problèmes courants :
 - ◆ `xauth` doit être installé sur la machine distante
 - ◆ Le serveur Xorg local doit autoriser les connexions TCP
 - ★ `pgrep -a Xorg` \rightsquigarrow `-noListen` ne doit pas être défini
 - ★ Configurable dans le gestionnaire de sessions utilisateur
 - ◆ Fonctionne mal sur des connexions lentes ou à forte latence

SOCKS proxy avec -D

- ▶ SOCKS : protocole pour proxifier des connexions TCP via une machine distante
- ▶ SSH peut agir en tant que serveur SOCKS : `ssh -D 1080 server`
- ▶ Similaire aux tunnels avec -L mais en plus flexible
 - ◆ Le proxy peut servir pour plusieurs connexions
- ▶ Utilisation :
 - ◆ Manuel : configurer les applications pour utiliser le proxy
 - ◆ Transparent : utiliser tsocks pour re-router les connexions

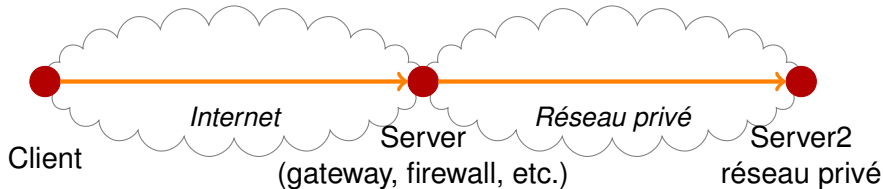
```
$ cat /etc/tsocks.conf
server = 127.0.0.1
server_type = 5
server_port = 1080 # puis, lancer ssh avec -D 1080
$ tsocks pidgin # proxifier l'application
```

- ◆ Autre proxy transparent : redsocks (utilise iptables pour rediriger vers un service local au lieu de LD_PRELOAD)

VPN sur SSH

- ▶ SSH intègre un VPN avec *devices tun*
 - ◆ nécessaire d'activer `PermitTunnel yes` côté serveur
 - ◆ `ssh -w 0:0 root@server` (0, 0 sont les numéros des tuns)
 - ◆ Puis configurer les adresses IP des deux côtés
- ▶ `sshuttle` : une autre solution de VPN sur SSH
 - ◆ Accès Root non nécessaire côté serveur
 - ◆ Idée similaire à `slirp`
 - ◆ Utilise des règles `iptables` pour rediriger le trafic vers le VPN
 - ◆ (as root :) `sshuttle -r user@server 0/0 -vv`
 - ◆ Limitation : ne permet pas de tunneler du trafic UDP ou ICMP

Passer d'hôtes en hôtes avec ProxyCommand



- ▶ Pour se connecter à Server2, il faut se connecter à Server
 - ◆ Comment le faire en une étape ? (requis pour les transferts de données, tunnels, X11 forwarding)
- ▶ Combine deux fonctionnalités de SSH
 - ◆ L'option `ProxyCommand` : commande utilisée pour se connecter à un hôte ; la connection doit être disponible sur les entrées et sorties standard
 - ◆ `ssh -W host:port ~` établit une connection TCP, la donne sur l'entrée/sortie standard (adapté pour `ProxyCommand`)

Passer d'hôtes en hôtes avec ProxyCommand (2)

- ▶ Exemple de configuration :

```
Host server2 # ssh server2 marche
  ProxyCommand ssh -W server2:22 server
```

- ▶ Fonctionne aussi avec les wildcards

```
Host *.priv # ssh host1.priv marche
  ProxyCommand ssh -W $(basename %h .priv):%p server
```

- ▶ -W est disponible depuis OpenSSH 5.4 (≈ 2010), mais peut être réalisé avec netcat :

```
Host *.priv
  ProxyCommand ssh serv nc -q 0 $(basename %h .priv) %p
```

- ▶ Version plus générique : <https://glandium.org/blog/?p=3631>

- ▶ Solution similaire pour se connecter via un proxy :

- ◆ SOCKS : connect-proxy -4 -S myproxy:1080 rhost 22
- ◆ HTTP (avec CONNECT) : corkscrew myproxy 1080 rhost 22
- ◆ Si les requêtes CONNECT sont interdites, activer httpunnel sur un serveur distant puis utiliser htc et hts

Passer d'hôtes en hôtes avec ProxyJump

- ▶ Disponible depuis OpenSSH 7.3
- ▶ Une manière plus simple de configurer ProxyCommand :

```
Host server2 # ssh server2 marche  
    ProxyJump server
```

- ▶ Ou sur la ligne de commande :
- ```
ssh -J server server2
```

## Déclencher des commandes distantes

- ▶ Objectif : notifier Server2 d'un évènement, depuis Server1
  - ◆ Mais Server1 ne doit pas avoir un accès shell sur Server2
- ▶ Méthode : limiter l'accès à une seule commande dans `authorized_keys`
  - ◆ Aussi connu sous le nom de *SSH triggers*
- ▶ Exemple d'`authorized_keys` sur Server2 :

```
from="server1.acme.com",command="tar czf - /home",no-pty,
no-port-forwarding ssh-rsa AAAA[...]oR user@my.host.net
```

# Séquences d'échappement

- ▶ Objectif : interagir avec une connexion SSH déjà établie
  - ◆ Ajouter des tunnels ou des proxy SOCKS, terminer les connexions qui ne répondent pas
- ▶ Les séquences démarrent avec un '~' au début de ligne
  - ◆ Presser [enter] puis ~ puis, par exemple '?'
- ▶ Séquences principales (autres documentées dans ssh(1)) :
  - ◆ ~. – déconnexion (si la connexion ne répond pas)
  - ◆ ~? – affiche la liste des séquences d'échappement
  - ◆ ~C – ligne de commande openSSH, par exemple ~C -D 1080
  - ◆ ~& – logout et mettre SSH en tâche de fond pendant que les tunnels ou les sessions X11 se terminent

## Autres outils liés

- ▶ `screen` et `tmux` : fournissent des terminaux virtuels sur les machines distantes, pour démarrer des commandes, s'en détacher, et s'y reconnecter plus tard
- ▶ `mosh`, une alternative à SSH adaptée aux connexions mobiles, longue distance et aux réseaux Wi-Fi
- ▶ `autossh` : vérifie qu'une session SSH est toujours active, et la relance si besoin  
`autossh -t server 'screen -RD'` : maintient une session `screen` ouverte malgré les déconnexions réseaux
- ▶ `concierge` : gérer votre fichier de configuration SSH avec un système de templates
- ▶ `clustershell` : SSH parallèle (exécuter la même commande sur plusieurs machines)

# Plan

- 1 Les bases de SSH
  - Introduction
  - Authentification par clé publique
  - Vérifier l'identité du serveur
  - Configurer SSH

- 2 Utilisation avancée
  - SSH : couche de communication générique
  - Accès à un filesystem distant : sshfs
  - SSH tunnels, X11 forwarding, and SOCKS proxy
  - VPN sur SSH
  - Passer d'hôtes en hôtes avec ProxyCommand
  - Déclencher des commandes distantes
  - Séquences d'échappement
  - Autres outils liés

- 3 Conclusion

# Conclusion

- ▶ Le couteau suisse de l'administration à distance
- ▶ Disposant de nombreuses fonctionnalités utiles et puissantes
- ▶ TP : tester tous les exemples mentionnés dans cette présentation
  - 1 scp, rsync
  - 2 Authentification par clé
  - 3 Utilisation d'un agent SSH
  - 4 Alias dans la configuration SSH
  - 5 sshfs, sftp
  - 6 Tunnels SSH
  - 7 X11 forwarding
  - 8 SOCKS proxy avec tsocks
  - 9 Passer d'hôtes en hôtes avec ProxyCommand
  - 10 Séquences d'échappement
  - 11 ...