Introduction à SSH

Lucas Nussbaum
lucas.nussbaum@univ-lorraine.fr

Licence professionnelle ASRALL

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

Plan

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 parle surtout de ca
  - Putty: Windows, client seulement
  - Dropbear: systèmes restreints (routers, embarqué)
- La commande Unix (`ssh`); coté serveur: `sshd`
- Établit une communication sur un canal sécurisé entre deux machines
- S'appuie sur la cryptogrphie
- L'usage le plus simple: obtenir un accès shell sur une machine distante
- De nombreux usages avancés:
  - Transfert de données (`scp`, `sftp`, `rsync`)
  - Connexion à des services specifiques (comme des serveurs Git ou SVN)
  - "Creuse" des tunnels sécurisés à travers l'internet
- Plusieurs systèmes d'authentification: mot de passe, clef publique

Utilisation basique

- Se connecter à un serveur distant:
  `$ ssh login@remote-server`
  $\rightsquigarrow$ Fournit un shell *remote-server*

- Éxecuter une commande sur un serveur distant:
  `$ ssh login@remote-server ls /etc`

- Copying data (with `scp`, similar to `cp`):
  `$ scp local-file login@remote-serv:remote-directory/`
  `$ scp login@remote-serv:remote-dir/file local-dir/`
  Usual `cp` options work, e.g. `-r` (recursive)

- Copier des données (avec `rsync`, est plus efficace qu'avec `scp` si il y a pleins de fichiers):
  `$ rsync -avzP localdir login@server:path-to-rem-dir/`

  Note: le slash de fin importe avec `rsync` (pas avec `cp`)
  - `rsync -a dir1 u@h:dir2` $\rightsquigarrow$ répertoire1 copié dans répertoire2

Authentification par clef publique

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

- Avantages
  - Les mots de passes ne sont pas envoyés par le réseau
  - La clef privée ne quitte JAMAIS le client
  - Le procédé peut être automatisé
- Cependant, la clef privée doit être protégée (que se passerait-il sivotre ordinateur portable était volé?)
  - Habituellement avec une passphrase

Génération d'une paire de clef

```
$ 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
+-----------------+
$
```

- Créer une paire de clef
  - `~/.ssh/id_rsa` (private key)
  - `~/.ssh/id_rsa.pub` (public key)

Copier la clef publique sur un serveur

- Exemple public key:
  `ssh-rsa AAAAB3NX[. . . ]hpoR3/PLlXgGcZS4oR user@my.hostname.net`
- Sur le serveur, **~user/.ssh/authorized_keys** contient une liste des clefs publiques autorisées au compte `user`
- La clef peut y être copiée manuellement
- Ou utiliser `ssh-copy-id` pour copier la clef automatiquement:
  `client$ ssh-copy-id user@server`
- Parfois la clef publique a besoin d'être fournie en utilisant une interface web(e.g. sur GitHub, FusionForge, Redmine, etc.)

Se souvenir de la passphrase

- Si la clef 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 se souvenir de la passphrase
  - La plupart des environnements de bureau peuvent jouer le rôle d'agent SSH automatiquement
  - On peut lancer `ssh-agent` si besoin
  - On ajoute les clefs manuellement avec `ssh-add`

Verifier l'identité du serveur : known_hosts

- Objectif: détecter un serveur contrefait
  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 clef 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 clef est toujours la même, ou averti 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:~$
```

- Supprimer une véritable clef périmée 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 documentées dans la page de manuel `ssh_config(5)`
- `~/.ssh/config` contient une liste d'hôtes (avec wildcards)
- Pour chaque paramètre, c'est la première valeur trouvée que l'on utilise
  - Les déclarations spécifiques à l'hôte sont données au début
  - Paramètres par défaut à la fin

Exemple: ~/.ssh/config

```
Host mail.acme.com
    User root

Host foo # alias/shortcut. 'ssh foo' works
    Hostname very-long-hostname.acme.net
    Port 2222

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

Host *
    User john
```

- Note: `bash-completion` peut auto-compléter en utilisant les hôtes de `ssh_config`

Plan

SSH, une couche communication pour les applicatio

- Plusieurs applications utilisent SSH comme leur couche communication
  - Parfois aussi comme couche d'authentification
- `scp`, `sftp`, `rsync` (transfert de donnée)
- `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 à travers SSH: sshfs

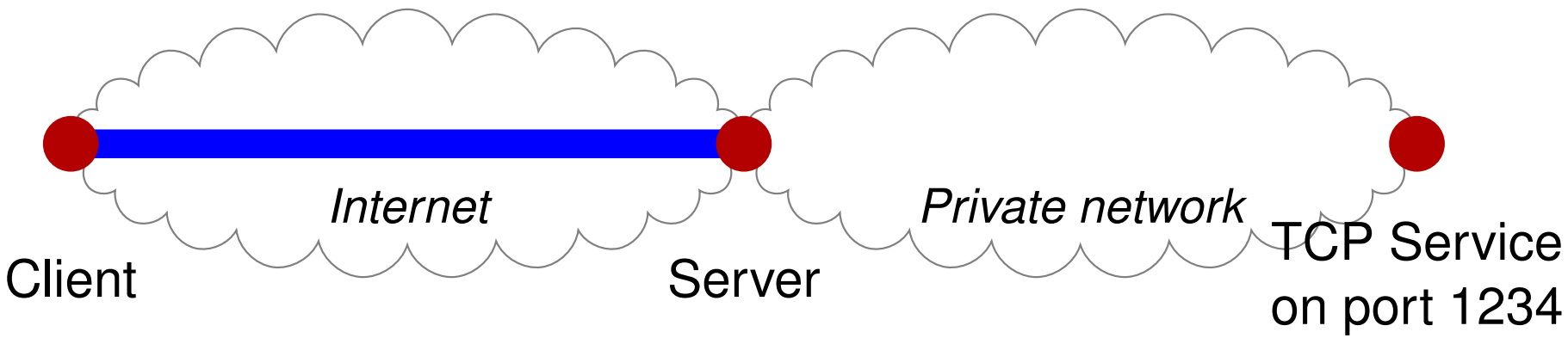
- `sshfs`: Des solution basées sur FUSE pour accéder à des machines distantes
- Idéal pour editer un fichier en GUI à distance, copier des petites quantitées de données, etc...
- Monter un répertoire distant:
  `sshfs root@server:/etc /tmp/local-mountpoint`
  Démonter: `fusermount -u /tmp/local-mountpoint`
- 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/
```

  $\rightsquigarrow$ `cd ~/.sshfs/rhost/etc/ssh`

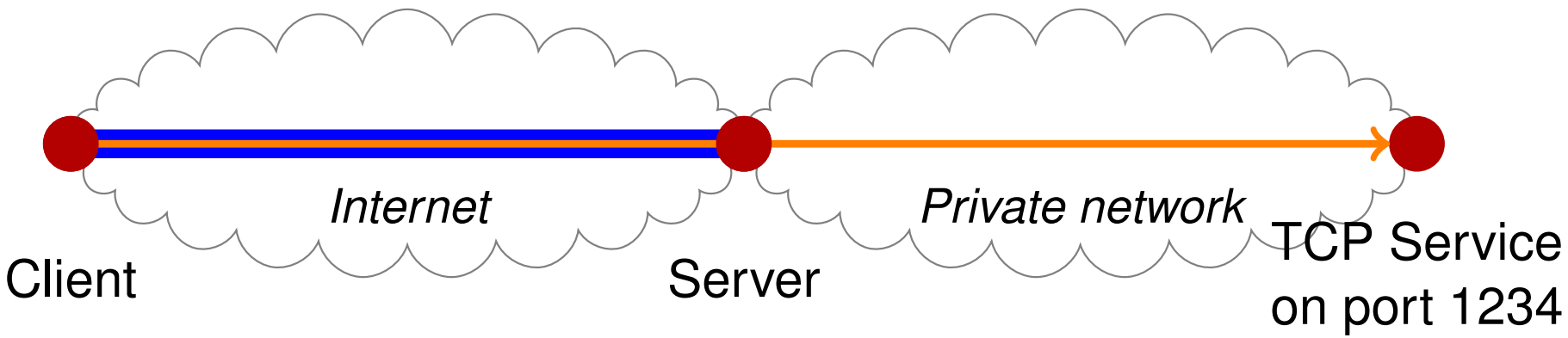
Tunnels SSH avec -L et -R

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



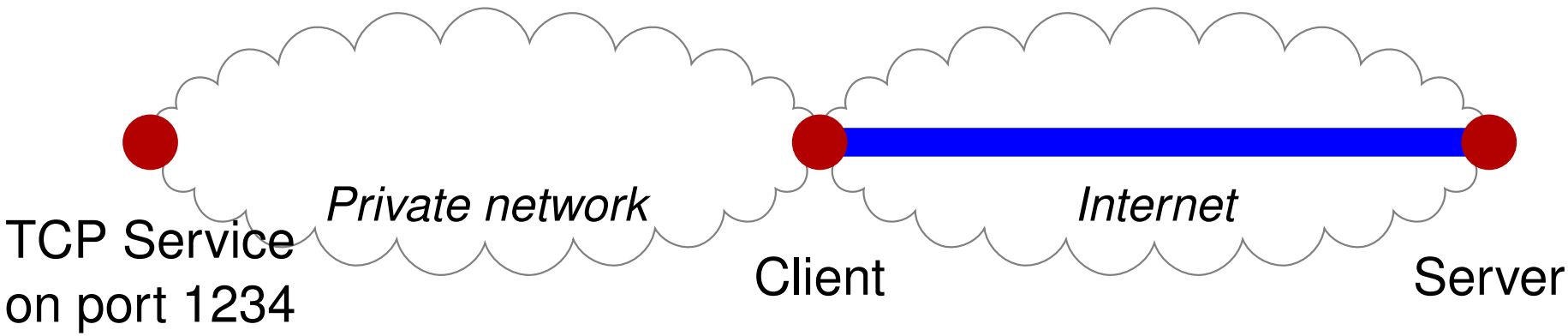
Tunnels SSH avec -L et -R

- Objectif: transporter le traffic dans une connection sécurisée
  - Contourner le filtrage réseau (pare-feux)
  - Éviter d'envoyer des données en clair sur Internet
  - Mais fonctionne seulement pour les connections TCP
- **-L**: accèder à un service distant derrière un pare-feux (serveur intranet)
  - `ssh -L 12345:service:1234 server`
  - Toujours 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

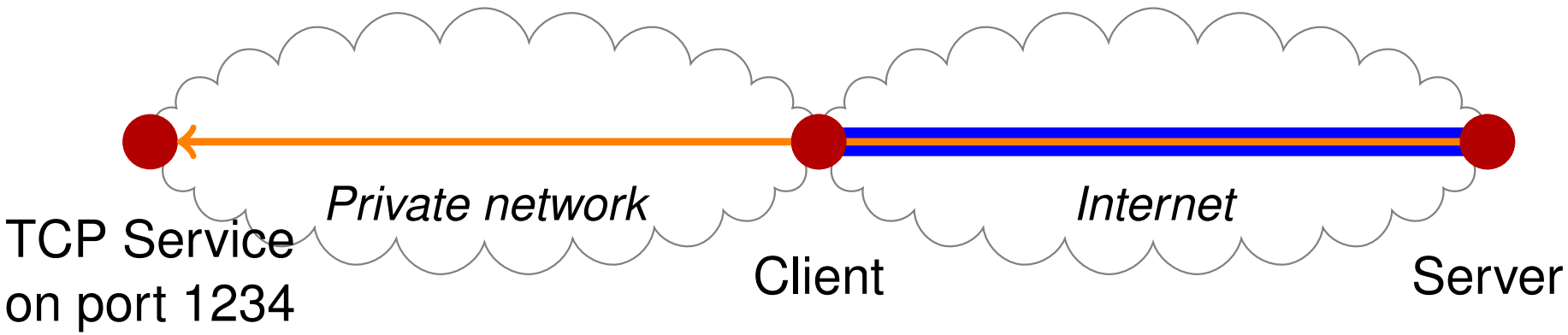
- **-R**: fournit un accès distant à un service local privé
  - `ssh -R 12345:service:1234 server`
  - Sur *Server*: `telnet localhost 12345`
  - *Client* établit une connexion TCP vers le *Service*, port 1234
  - Le trafic est tunnellisé dans la connexion SSH vers le *Client*



- Note: les tunnels SSH ne fonctionnent pas très bien pour HTTP, parce que IP+port est insufisant pour identifier un site web (`Host:` HTTP header)

Tunnels SSH avec -L et -R

- **-R**: fournit 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 tunnellisé dans la connexion SSH vers le *Client*



- Note: les tunnels SSH ne fonctionnent pas très bien pour HTTP, parce que IP+port est insufisant pour identifier un site web (`Host:` HTTP header)

X11 transféré à l'aide de -X: applications GUI à trave

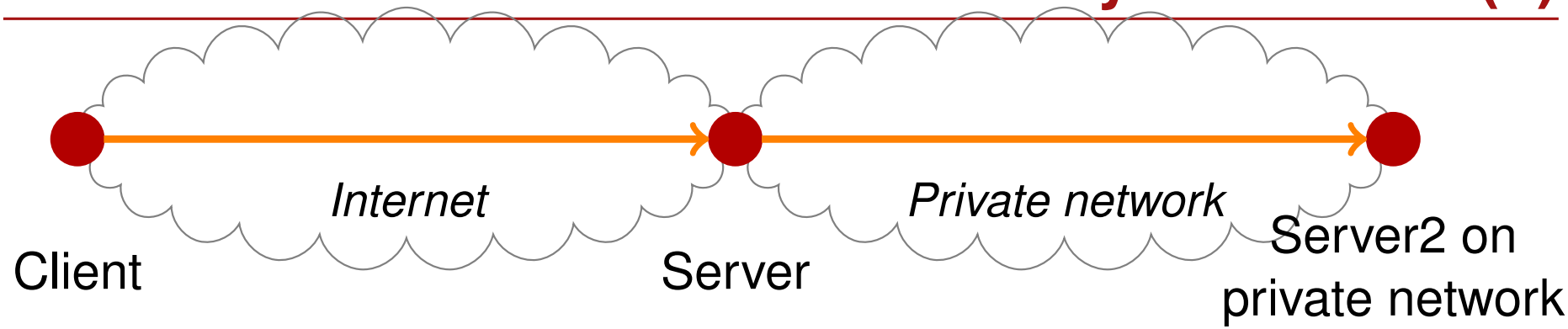
- Lancer une application graphique sur une machine distante pour l'afficher localement
- Similaire à VNC mais fonctionne par applications
- `ssh -X server`
- `$DISPLAY` sera déclaré par SSH sur le serveur:
  ```
  $ echo $DISPLAY
  localhost:10.0
  ```
- Puis lancer les applications sur le serveur (e.g. `xeyes`)
- Diagnostic:
  - `xauth` doit être installé sur la machine distante
  - Le serveur Xorg local doit autoriser les connections TCP
    - `pgrep -a Xorg` ⇝ `-nolisten` ne doit pas être inclus
    - Peut être configuré dans le gestionnaire de sessions utilisateur

SOCKS proxy avec -D

- SOCKS: protocole des connexions proxy TCP via une machine distance
- SSH peut agir en tant que serveur SOCKS: `ssh -D 1080 server`
- Cas d'utilisation similaires aux tunnels avec -L mais en plus flexible
  - Mettre en place le proxy pour plusieurs connexions
- Utilisation:
  - Manuel: configurer les applications pour uiliser le proxy SOCKS
  - Transparent: utiliser `tsocks` pour re-router les connections via SOCKS

```
$ cat /etc/tsocks.conf
server = 127.0.0.1
server_type = 5
server_port = 1080 # puis, lancer ssh avec -D 1080
$ tsocks pidgin # tunneliser l'application \‘{a} travers
```

Passer d'hôtes en hôtes avec ProxyCommand (2)



- Problème: pour se connecter à Server2, il faut se connecter à Server1
  - Pouvez-vous le faire en une étape? (requis pour les transferts de données, tunnels, X11 forwarding)
- Combine deux fonctionnalités de SSH
  - l'option `ProxyCommand`: connexion vers l'hôte disponible sur l'entrée/sortie standard
  - `ssh -W host:port` $\rightsquigarrow$ établit une connection TCP sur l'entrée/sortie standard (adaptée pour `ProxyCommand`)

Passer d'hôtes en hôtes avec ProxyCommand (2)

- Exemple de configuration

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

- Fonctionne aussi avec les wildcards

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

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

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

- Solution similaire de connexion via un proxy:
  - SOCKS: `connect-proxy -4 -S myproxy:1080 rhost 22`
  - HTTP (with CONNECT): `corkscrew myproxy 1080 rhost 22`
  - Si les requêtes CONNECT sont interdites, activer `httptunnel` sur un serveur distant puis utiliser `htc` et `hts`

Déclancher l'exécution de commandes distantes en

- Objectif: notifier Server2 que quelque chose s'est terminé sur Server1
  - Mais Server1 ne doit pas avoir un accès shell total 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 connections qui ne répondent pas
- Les séquences d'échappement avec '~' au début de ligne
  - Presser `[enter]` puis {texttt~ puis, par exemple '?'
- Séquences principales (autres documéntées dans `ssh(1)`):
  - `~.` – déconnexion (si elle ne répond pas)
  - `~?` – affiche la liste des séquences d'échappement
  - `~C` – ligne de commande openSSH, par exemple `~C -D1080`
  - `~&` – déconnecter et mettre SSH en tâche de fond pendant que les transmissions ou les sessions X11 se terminent

Plan

Conclusions

- Le couteau Suisse de l'administration à distance
- Disposant de nombreuses fonctionnalités utiles et puissantes
- Travaux pratique: tester tous les examples mentionnés dans cette présentation
- Autres sujets n'étant pas abordés dans cette présentation
  - Support du VPN intégré
  - Les autres méthodes d'authentification (certificats)
  - Gestion des sessions distantes avec `screen` ou `tmux`
  - Mosh, une alternative SSH adaptée pour les connections mobiles, longue distance et les réseaux Wi-Fi