SSH

Secure SHell

Anaïs Gantet, Benoît Camredon
TLS-SEC

Plan

Introduction

Administration sécurisée

`SSH` est un protocole qui vise à fournir

- Moyen simple d'accéder/administrer une machine
  - Obtenir un shell
  - Exécuter une commande
- Authentification forte
- Chiffrement des échanges
- Encapsulation d'autres flux

A l'origine, remplacement d'outils existants

- Outils d'administration : `rlogin`, `rsh`, `telnet`...
- Outils de transfert de fichiers : `rcp`, `FTP`...

Un peu d'histoire...

- `SSHv1` en Finlande en 1995
- `SSHv2` en 2006 par l'IETF

Plusieurs RFC: SSH Authentication protocol (RFC 4252), SSH Transport Layer protocol (RFC 4253), SSH Connection protocol (RFC 4254)

Quelques implémentations...

Côté serveur

- `openssh`
- `Dropbear`
- `Paramiko`

Côté client

- `openssh`
- `putty`
- `winscp`
- `Paramiko`

OpenSSH configuration

Configuration du serveur : `/etc/ssh/sshd_config`

- Type d'authentification : clé, mot de passe, ...
- Protocole autorisé : `SSHv1`, `SSHv2`...
- Services disponibles : `sftp`...
- Port, adresse d'écoute
- Utilisateurs autorisés
- Algorithmes de chiffrement
- ...

Configuration du client : `/etc/ssh/ssh_config`

- Peu de configuration nécessaire
- Peut être surchargé par le fichier personnel `~/.ssh/config`

Recommandation pour un usage sécurisé d'OpenSSH par l'ANSSI : `https://www.ssi.gouv.fr/uploads/2014/01/NT_OpenSSH.pdf`

Authentification

Authentification des hôtes

TOFU : Trust On First Use

- Lors du premier échange, on stocke l'empreinte (fingerprint) de la clé de l'hôte

Clés du serveur

- `/etc/ssh/ssh_host_rsa_key.pub`
- `/etc/ssh/ssh_host_dsa_key.pub`

Fingerprint de clés serveurs connus par le client

- `/etc/ssh/ssh_known_hosts`
- `~/.ssh/known_hosts`

Comportement par défaut du client

- Hôte absent de `known_hosts` : confirmation
- Hôte présent dans `known_hosts` mais clé différente : refus
- Hôte présent dans `known_hosts` et clé identique : OK

Authentification des hôtes

Lors de la première connexion

```
>> ssh host
The authenticity of host 'host (192.168.0.64)' can't be established.
RSA key fingerprint is SHA256:AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.
Are you sure you want to continue connecting (yes/no)?
```

En cas de mauvaise clé publique

```
>> ssh host
Warning: the RSA host key for 'host' differs from the key for the IP address '192.168.0.64'
Offending key for IP in /home/user/.ssh/known_hosts:3
Are you sure you want to continue connecting (yes/no)?
```

Configuration

```
StrictHostKeyChecking ask
```

ssh_config

Authentification utilisateur

Par défaut...

- Authentification par login/mot de passe

```
PasswordAuthentication yes
```
sshd_config

- Attaque par dictionnaire/bruteforce
- Difficile à utiliser avec des scripts

Mais aussi...

- Authentification à clé publique

```
PubkeyAuthentication yes
```
sshd_config

Possibilité de mixer plusieurs types d'authentification

Authentification utilisateur par clé : côté client

Générer sa clé

- Génération via l'outil `ssh-keygen`

```
ssh-keygen -t [dsa|ecdsa|ed25519|rsa] -b bits
```

- Possibilité de la protéger avec une passphrase
- Possibilité de stocker la clé privée sur un support physique (`yubikey...`)

Fichiers de clé privées/publiques

- `~/.ssh/id_ecdsa[.pub]`
- `~/.ssh/id_rsa[.pub]`
- `~/.ssh/id_dsa[.pub]`

Authentification par clé : côté serveur

Configurer le serveur pour l'authentification par clé

```
PubkeyAuthentication yes
AuthorizedKeysFile   %h/.ssh/authorized_keys
PasswordAuthentication no
```

sshd_config

Ajouter sa clé sur le serveur

- Copier sa clé dans le répertoire de l'utilisateur avec qui on va se connecter (`~/.ssh/authorized_keys`)
- Attention aux droits du fichier !

Fichier ~/.ssh/authorized_keys

Syntaxe

- Chaque ligne contient une clé publique
- Chaque ligne est décomposée en : `[options] keytype base64-encoded key comment`

Quelques options

- `from="pattern-list"` : restriction par adresse
- `command="command"` : seule la commande listée pourra être appelée par cette clé
- `no-pty` : interdit l'utilisation d'un tty, et donc d'un shell

```
command="dump /home",no-pty
↪  ssh-dss AAAAC3...51R==
↪  foobarteam.net
```

~/.ssh/authorized_keys

```
Match User user Address 1.2.3.4
  ForceCommand  /path/to/script.sh
```

sshd_config

Signature de clés

Problèmes

- Maintenance difficile quand le nombre d'utilisateurs/machines devient important
- Croyance aveugle dans un fingerprint

Solution

- Signer les clés des machines
- Signer les clés des utilisateurs
- Utiliser un certificat `ssh`

Signature de clés machines

1. Créer une clé qui permettra de signer

```
ssh-keygen -f server_ca
```

2. Signer la clé d'une machine

```
ssh-keygen -s server_ca -I host_auth_server -h -n auth.example.com -V +52w ssh_host_rsa_key.pub
```

   - `-s` : Clé à utiliser pour signer
   - `-I` : Nom de la clé à signer
   - `-h` : Signature d'une clé d'une machine et pas d'un utilisateur
   - `-n` : L'ensemble des noms de domaine de la machine
   - `-V` : La durée de validité de la signature

3. Modifier le fichier `/etc/ssh/sshd_config` du serveur

```
HostCertificate /etc/ssh/ssh_host_rsa_key-cert.pub
                                                    sshd_config
```

4. Modifier le fichier `~/.ssh/known_hosts` des clients

```
>> cat ~/.ssh/known_hosts
@cert-authority *.example.com ecdsa-sha2-nistp521 AAAAA...
```

Signature de clés utilisateurs

Etapes

1. Créer une clé qui permettra de signer

```
ssh-keygen -f user_ca
```

2. Signer la clé de l'utilisateur

```
ssh-keygen -s user_ca -I user_str -n user -V +52w id_ecdsa.pub
```

   - `-s` : Clé à utiliser pour signer
   - `-I` : Nom de la clé à signer
   - `-n` : L'identifiant de l'utilisateur sur le serveur
   - `-V` : La durée de validité de la signature

3. Copier le fichier produit `id_ecdsa-cert.pub` sur la machine de *user*
4. Modifier le fichier `/etc/ssh/sshd_config` du serveur

```
TrustedUserCAKeys /etc/ssh/user_ca.pub
```

sshd_config

5. On peut supprimer la clé de `/home/user/.ssh/authorized_keys` sur le serveur

Boite à outils

SSH : couteau suisse

Boite à outils

- Shell distant (fonction de base)
- Transfert de fichiers
- Redirection de ports

Plusieurs binaires

- `ssh` : Connexion vers un serveur
- `scp` : Envoi ou réception d'un fichier depuis un serveur
- `sftp` : Commande similaire à `ftp`
- `ssh-keygen` : Génération de clé, signature...
- `ssh-agent`,`ssh-add` : Service de trousseau de clés
- `ssh-keyscan` : Vérification des clés d'hôtes

Administration à distance

Shell distant

- `ssh monlogin@server`
- `ssh monlogin@server command`

Sécurité

- Authentifié sur le serveur
- Communications chiffrées

Transfert de fichiers

scp

```
>> scp fichier monlogin@server:fichier_distant
>> scp -r monlogin@server:repertoire_distant repertoire
```

sftp[1]

```
>> sftp monlogin@server
Password:
sftp>
```

Activation de `sftp`

- `sftp` doit être activé dans `/etc/ssh/sshd_config`

```
Subsystem sftp /usr/lib/openssh/sftp-server
```

sshd_config

[1] Ou `lftp` en plus puissant

Transfert de fichiers sans scp/sftp

Pas de scp ou sftp ?

- Utilisation de l'entrée/sortie standard

Téléchargement d'un fichier du serveur

```
>> ssh monlogin@server cat file > /tmp/file
```

Envoi d'un fichier sur le serveur

```
>> ssh monlogin@server < file "cat > /tmp/file"
```

Quelques commandes utiles...

```
>> ssh server "tar cz /etc" | tar xz
>> tar cz | ssh server "cat > tutu.tgz"
>> ssh server cat /path/to/remotefile | diff /path/to/localfile -
```

SSH Agent

Motivations

- *Passphrase* fastidieuse à taper
- Impossible de déplacer sa clé privée sur une autre machine

Objectifs de `ssh-agent`

- Définir un *agent*, sorte de coffre-fort qui conserve les clés en cours d'utilisation
- L'agent évite de retaper sa passphrase à chaque fois
- L'agent peut être utilisé par une machine distante sur laquelle on est connecté
- Possibilité de transférer son agent de machine en machine
- La clé ne sort jamais de la machine initiale

SSH Agent : utilisation

Etapes

1. Ajout de la (ou des) clé privée dans l'agent

```
>> ssh-add
Enter passphrase for /home/user/.ssh/id_ecdsa: *****************
Identity added: /home/user/.ssh/id_ecdsa (/home/user/.ssh/id_ecdsa)
```

2. Une socket unix est créée permettant d'interroger l'agent pour utiliser la clé privée
3. `ssh` utilise ensuite la variable d'environnement `SSH_AUTH_SOCK` pointant sur la socket
4. La commande `ssh-agent` permet de savoir quelle socket est utilisée

```
>> eval `ssh-agent`
```

5. On se connecter directement sans avoir à retaper sa *passphrase*

SSH Agent

Attention

- Si la machine distante est compromise, possibilité d'utiliser l'agent à distance...

```
>> export SSH_AUTH_SOCK=/tmp/ssh-DEADBEEF/agent.1337
```

- Possibilité d'utiliser `ssh-add` avec l'option `-c` pour forcer une demande de confirmation

Objectif : Amélioration des performances

- Principe : faire passer tous les flux TCP dans la même connexion
- Utilise la notion de Master/Slave

```
Host GetinMeForFree
    ControlMaster auto
    ControlPath ~/.ssh/currents/\%r@\%h:\%p
```

ssh_config

Problème

- Dès que la connexion est établie, *tous le le monde*[2] peut s'en servir sans remettre de mdp / passphrase

[2] Droits nécessaires sur la socket unix

Idée

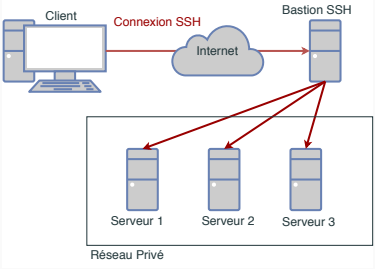
- Se connecter sur une machine en passant automatiquement par une ou plusieurs autres

```
>> ssh -J intermediaire1,intermediaire2 serveur
```

Pratique en cas de Bastion SSH

```
Host serveur1
  ProxyJump bastion
  Hostname ip_interne
```

~/.ssh/config



Meta-commandes SSH

<enter>~ donne accès à des meta-commandes

```
~.   - terminate connection (and any multiplexed sessions)
~B   - send a BREAK to the remote system
~C   - open a command line
~R   - request rekey
~V/v - decrease/increase verbosity (LogLevel)
~^Z  - suspend ssh
~#   - list forwarded connections
~&   - background ssh (when waiting for connections to terminate)
~?   - this message
~~   - send the escape character by typing it twice
```

Commande utile

- `<enter>~.` pour couper une connexion `ssh` qui ne répond plus

Redirection de flux

Redirection de flux

Mise en place de redirection

- Statique sur le client (redirection locale)
- Statique sur le serveur (redirection remote)
- Pour les flux X
- Dynamique

Mise en place de VPN

- Niveau 2
- Niveau 3

Redirection de port sur le client

Redirection d'un port en local

```
>> ssh -L <localport>:<target>:<remoteport> relay
```

Comment ça marche ?

- Ouvre une connexion entre votre machine `server`
- Chaque connexion à destination de `localhost:localport` est redirigée dans le tunnel ssh vers `target:remoteport`
- Seule la connexion entre `localhost` et `relay` est chiffrée. Pour la 2ème connexion, ça dépend du protocole.

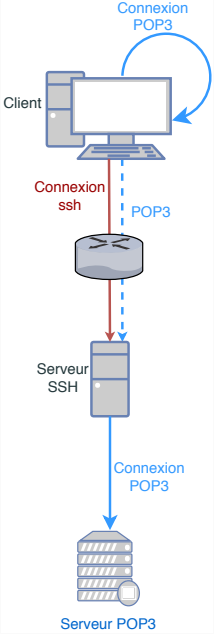
Redirection de port sur le client

Redirection d'un port en local

```
>> ssh -L 1110:pop3:110 monlogin@ssh_server
```

Intérêts

- Sécuriser des protocoles faibles
- Accéder à un service qui n'est accessible que depuis le serveur SSH



Redirection de port sur le serveur

Redirection d'un port sur une machine distante

```
>> ssh -R [bind_address:]port:host:hostport server
```

Comment ça marche ?

- Ouvre une connexion vers `server` sur le port 22
- Utilise `bind_address` (ou le loopback si ce n'est pas spécifié) pour se mettre en écoute sur `server`
  - Fonctionne avec autre chose que le loopback ssi le serveur est configuré pour l'autoriser (option `GatewayPorts`)
- Les connexions vers `[bind_address:]port` sont redirigées vers le client, qui relaie ensuite vers `host:hostport`
  - Le client se place en homme du milieu
- Seule la connexion entre le client et `server` est chiffrée. Pour la 2ème connexion, ça dépend du protocole.

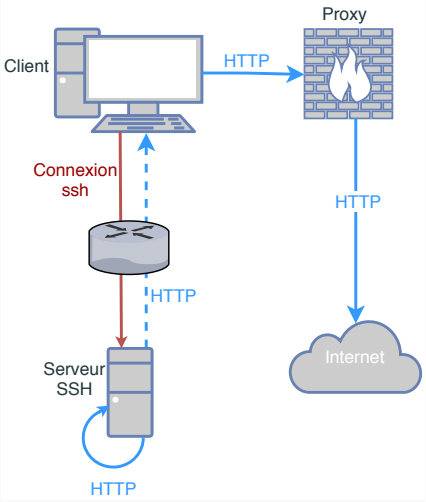
Redirection de port sur le serveur

Redirection d'un port sur une machine distante

```
>> ssh -R 8080:proxy:3128 monlogin@server
server>> export http_proxy=http://localhost:8080/
server>> sudo apt update
```

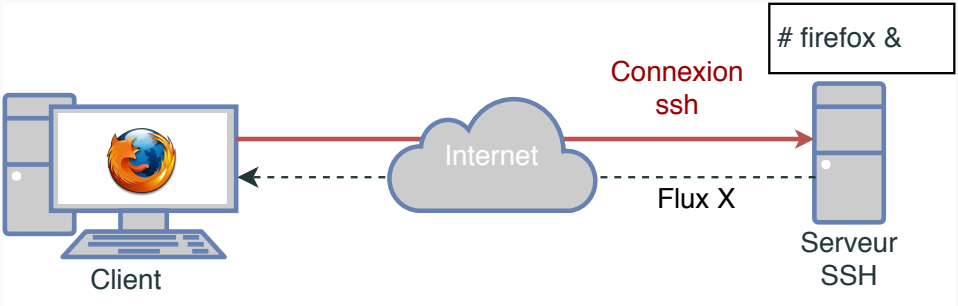
Intérêts

- Protéger et rediriger les communications externes d'un serveur
- Rendre accessible un service à une machine



Redirection de flux X

Les applications X du serveur sont redirigées à travers la connexion SSH vers le serveur X du client




```
>> ssh -X monlogin@server
server>> firefox
```

Utile pour administrer graphiquement un serveur sans installer/lancer un serveur X dessus (juste besoin des bibliothèques) ... sauf que ...

Redirection de flux X : Risque de sécurité

ssh et `X11Forwarding`

- Création d'un cookie dans `~/.XAuthority`
- Si un attaquant peut accéder au cookie, il peut interagir avec le serveur X...

Exemple d'actions possibles

- Faire un screenshot du bureau X du client

```
/* smile for the screenshot */
>> xwd -display localhost:10.0 -root -out test.xwd
/* display it */
>> xwud -in test.xwd
```

- Lister les clients X : `xlsclients`
- Surveiller les événements X : `xev`
- Transformer ssh en keylogger : `DISPLAY=localhost:10.0 xmacrorec2 -k 0`

Protocole SOCKS

- Ajout d'une en-tête spécifiant une IP et un port (`SOCKS4`)

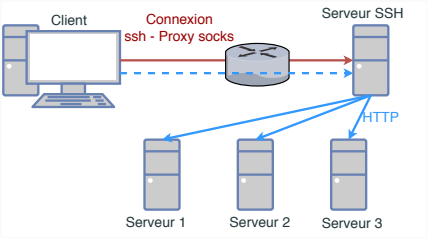
Proxy SOCKS

- Proche de la redirection d'un port en local (`-L`)
- Encapsulation dans le protocole `SOCKS`

```
>> ssh -ND 1080 monlogin@server
```

Utilisation de SOCKS

- L'application doit être prévue pour
- Possibilité d'utiliser `tsocks`



VPN SSH

Objectif

- Relier 2 réseaux distants en utilisant ssh

Etapes

1. Lancer la commande ssh => création d'interface `tun` sur le client et le serveur ssh
2. Configuration des interfaces en mode point à point
3. Configuration du routage

Commandes

```
>> sudo ssh -N -w 0:0 root@machine2
>> ip addr add 10.0.0.1 peer 10.0.0.2 dev tun0
>> ip link set up dev tun0
```

Inconvénients

- Besoin d'être root sur les deux machines
- Peu optimal en terme de données utiles par paquet