Tunnel SSH

Relayage de ports avec ssh

Dans cette fiche pratique, nous montrons au travers de quelques exemples comment ssh fourni un moyen simp pour créer un tunnel parfaitement sûr. Les deux options dont nous détaillerons les effets sont `-L` (port Local) et (port distant - *Remote*).

Le relayage de ports (*port forwarding*) est souvent troublant car de nombreux éléments entrent en compte : un client et un serveur ssh, un client et un serveur TCP dont la connexion sera encapsulée dans le trafic ssh, soit quatre intervenants.

La différence entre relayage local et distant vient du « sens de la connexion ». Le client ssh se connecte sur le serveur ssh. En revanche, selon l'emplacement du client et du serveur TCP par rapport au client et au serveur s le relayage est local ou distant :

- local : le client TCP et le client ssh sont du même côté
- distant : le serveur TCP et le client ssh sont du même côté.

Dans tous les exemples ci-dessous, on suppose qu'un serveur ssh tourne sur la machine distante. Il s'agit d'un openssh-3.5p1. Nous utilisons uniquement le protocole ssh2.

Redirection locale

Syntaxe :

```
ssh -L <port-local-à-relayer>:<machine-distante>:<port-distant> machine-distante
```

Exemple :

```
batman$ ssh -L 1234:<hostname>:25 robin
```

Cette commande ouvre une connexion sur batman vers robin. En fait, un tunnel est ouvert depuis le port 1234 d batman, à destination du port 25 de *hostname*. Ce tunnel part de localhost:1234 vers robin:22 pour terminer en *hostname*:25. Après cette commande, une connexion sur le port 1234 de batman est donc dirigées vers différen endroits, en fonction de qui est *hostname*.

- **Si *hostname* est localhost**

  ```
  batman$ ssh -L 1234:localhost:25 robin
  ```

  localhost (127.0.0.1) fait référence à l'adresse de loopback de robin. Cette connexion conduit donc au serveur de mails de robin :

  ```
  batman$ telnet localhost 1234
  Trying 127.0.0.1...
  Connected to localhost.
  Escape character is '^]'.
  220 robin ESMTP Sendmail 8.11.6/8.7.3; Tue, 7 May 2002 12:08:10 +0200
  ```

- **Si *hostname* est robin**

```
batman$ ssh -L 1234:robin:25 robin
```

Il s'agit de presque la même chose que la commande précédente, sauf que la connexion au port distant sera sur robin et non localhost, ce qui permet de s'adapter selon les règles de filtrage (firewall ou wrapp

La commande `netstat` lancée sur robin montre cette différence :

```
# avec ssh -L 1234:localhost:25 robin
tcp   0   0 127.0.0.1:33471      127.0.0.1:25         ESTABLISHED -
# avec ssh -L 1234:robin:25 robin
tcp   0   0 128.93.24.10:33469   128.93.24.10:25      ESTABLISHED -
```

- **Si *hostname* est batman**

```
batman$ ssh -L 1234:batman:25 robin
```

Peu d'intérêt puisque cela connecte robin au port 25 de batman : on aurait plus vite fait de se connexion port 25 directement.

```
batman$ telnet localhost 1234
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 batman ESMTP Postfix
```

- **Si *hostname* est poisonivy**

```
batman$ ssh -L 1234:poisonivy:25 robin
```

Une connexion est ouverte entre batman et robin, mais le relayage de ports n'est pas entre ces deux hô mais entre robin:1234 et poisonivy:25 :

```
batman$ telnet localhost 1234
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 poisonivy ESMTP Sendmail 8.11.0/8.7.3; Sun, 8 Dec 2002 12:33:11 +0200
```

Attention, dans cette configuration, *le trafic entre robin et poisonivy n'est pas chiffré* !!! Lancer la commande `tcpdump -X host poisonivy` sur robin pour vous en rendre compte.

Il est possible de chiffrer cette connexion en enchaînant les tunnels :

```
batman$ ssh -L 1234:localhost:5678 robin
Last login: Sun Dec  8 12:38:12 2002 from batman
robin$ ssh -L 5678:localhost:25 poisonivy
...
```

En guise de mesure de sécurité, ssh n'autorise le relayage de ports que depuis localhost (127.0.0.1) de sorte à q que les personnes se connectant sur le port 1234 de batman ne soient pas emmenées où elles ne devraient pas Néanmoins, l'option `-g` permet de changer cela :

```
batman$ ssh -g -L 1234:<hostname>:25 robin
```

Reprenons le dernier exemple pour voir ce que cela change :

```
batman$ ssh -g -L 1234:poisonivy:25 robin
...
batcave$ telnet batman 1234
Trying 192.168.1.1...
Connected to batman.
Escape character is '^]'.
220 poisonivy ESMTP Sendmail 8.11.0/8.7.3; Sun, 8 Dec 2002 12:33:11 +0200
```

Redirection distante

Syntaxe :

```
ssh -R <port-distant-à-relayer>:<machine-distante>:<port-local-du-serveur>  machine-distar
```

Exemple :

```
batman$ ssh -R 1234:<hostname>:25 robin
```

L'hôte depuis lequel est lancé cette commande sert alors de relais entre robin:1234 et *hostname:25*. Cela sert p exemple lorsque l'administrateur de ce relais (batman dans notre exemple) souhaite donner une autorisation de connexion à une machine externe vers une machine interne à un réseau. Ici, batman se retrouve en position d'homme du milieu, entre robin et *hostname*. Le seul canal chiffré se situe toutefois entre batman et robin, puis la connexion ssh lie ces deux machines. En revache, il s'agit d'une connexion TCP tout à fait standard entre batman et *hostname*. Son chiffrement dépend alors du protocole qui entre en jeu.

- **Si *hostname* est localhost**

  ```
  batman$ ssh -R 1234:localhost:25 robin
  ```

  Un utilisateur sur robin qui ouvre une connexion sur localhost:1234 est redirigé vers batman:25, puisque localhost fait ici référence à cette machine :

  ```
  robin$ telnet localhost 1234
  Trying 127.0.0.1...
  Connected to localhost.localdomain (127.0.0.1).
  Escape character is '^]'.
  220 batman ESMTP Postfix
  ```

- **Si *hostname* est poisonivy**

  ```
  batman$ ssh -R 1234:poisonivy:25 robin
  ```

  Une connexion est ouverte entre batman et robin, mais le relayage se fait entre robin:1234 et poisonivy

  ```
  robin$ telnet localhost 1234
  Trying 127.0.0.1...
  Connected to localhost.
  Escape character is '^]'.
  220 poisonivy ESMTP Sendmail 8.11.0/8.7.3; Sun, 8 Dec 2002 12:33:11 +0200
  ```

  Le trafic est donc chiffré entre robin et batman, mais passe en clair entre batman et poisonivy.

Conclusion

Ces options permettent de mettre en place des tunnels très facilement. De cette manière, il est possible de protéger des flux qui ne sont pas naturellement chiffrés (pop3 ou imap par exemple). Voud devriez maintenant être suffisamment familier avec ssh pour voir que ces deux commandes sont équivalentes :

```
HOST1$ ssh -L p1:HOST3:p2 HOST2
HOST2$ ssh -R p1:HOST3:p2 HOST1
```

Dans nos exemples, il n'était à aucun moment nécessaire d'être root sur une machine pour mettre en place le tunnel. En revanche, en reprenant la notation ci-dessus, cela est impératif si P1<1024.

Dernier point, regardez les options de la commande ssh. Il en existe plusieurs qui peuvent s'avérer utiles. Par exemple, si vous ne souhaitez pas ouvrir de shell sur la machine distante, un `-N` fera l'affaire. Si vous souhaitez que la connexion distante n'existe que pour une connexion à un moment donné, l'option -f et faite pour vous :

```
batman$ ssh -f -R 1234:poisonivy:25 robin sleep 10
```

La connexion ssh est initialisée entre batman et robin. Un utilisateur sur robin dispose alors de 10 secondes pou se connecter. Une fois ce délai expiré, la connexion ssh entre batman et robin est close. Toutefois, si pendant ce intervalle, l'utilisateur sur robin s'est connecté au port local 1234, il est transporté comme prévu vers poisonivy moment de clore la connexion ssh entre batman et robin, comme il reste une connexion active (le port forwardi la connexion ssh entre batman et robin ne peut pas se fermer. Toutefois, le relayage de ports n'est plus actif.

Bibliographie

[PERROT 01] Sécuriser ses connexions avec ssh
Bernard Perrot
HS Linux Magazine 8 / MISC 0
Disponible sur http://www.miscmag.com/articles/index.php3?page=105

Frédéric Raynal - *pappy@miscmag.com* - *http://www.security-labs.org*

Last modified: Mon Dec 9 10:51:17 CET 2002

Last modified : Monday May 10 2004 -- 1:37