Utilisation de ssh-agent pour SSH1 et OpenSSH

Linux Gazette numéro 67

Jose Nazario

jose@cwru.edu

Jérôme Fenal

jerome@fenal.org

1. Introduction

Je discutais récemment avec un ami à propos de la façon d'utiliser SSH afin d'arriver à une façon sûre de faire de l'authentification sans mot de passe. Il recherchait un moyen d'automatiser des transferts de fichiers et voulait le faire en utilisant un script **expect** (pour injecter le mot de passe lors de sa demande) afin d'automatiser le processus. Je lui ai suggéré « **ssh-agent** », mais je ne savais pas comment le faire fonctionner à ce moment-là. Depuis, j'ai appris, et c'est relativement facile.

L'utilisation de l'agent pour l'authentification par clés et une méthode pour *faciliter* les communications. Vous pouvez utiliser l'authentification sans agent, vous avez juste à déverrouiller la clé à chaque fois que vous voulez l'utiliser. Notez que par défaut, le

client SSH essaiera de vous authentifier en utilisant les clés avant le mot de passe. L'agent simplifie largement la gestion de tout ça.

Il existe plusieurs mises en oeuvre du protocole SSH, chacune avec ses particularités d'usage et de comportement. Les deux les plus communes sont celles de openssh.org (/http://www.openssh.org) et de ssh.com (http://ssh.com). OpenSSH a été écrit pour OpenBSD, et est donc un logiciel libre. Le produit ssh de ssh.com est un produit commercial gratuit pour les systèmes d'exploitation libres (et pour les utilisations non-commerciales, ou à titre d'essai ou d'enseignement sur les autres systèmes d'exploitation). Chacune de ces mises en oeuvre a ses petites particularités de comportement et à l'utilisation.

Comme si de multiples logiciels n'étaient pas suffisants, il existe aussi deux protocoles SSH, SSH1 et SSH2. Cet article se focalisera uniquement sur l'utilisation du protocole SSH1, qui diffère légèrement du protocole SSH2. Plusieurs articles précédemment parus la *Linux Gazette* ont présenté l'utilisation de **ssh-agent** pour SSH2 (cf. plus bas). Notez que par défaut, SSH2 utilise des clés DSA, ainsi que des noms de répertoires et de fichiers différents de SSH1, bien qu'une certaine compatibilité puisse être conservée. Comme la plupart des gens utilisent le protocole SSH1 (données issues d'une récente scrutation de l'Internet avec **scan-ssh** par l'Université de l'Alberta), nous nous concentrerons donc sur cette version. OpenSSH suit, quasiment parfaitement, la syntaxe du programme **ssh1** de ssh.com en ce qui concerne la gestion des clés par l'agent. Notez bien que cela diffère de la gestion de SSH2 (non couvert ici).

Les avantages de l'authentfication RSA sont nombreuses, brutes de fonderie :

Authentification mutuelle

> Dans l'authentification RSA, chaque côté doit vérifier qu'ils sont bien ceux qu'ils disent qu'ils sont. Le client vérifie que le serveur est bien celui qu'il doit être (par rapport à leur clé publique, stockée dans le fichier `~/.ssh/known_hosts`), et le serveur vérifie l'authenticité de l'identité du client via une clé RSA. Cela est utilisé pour se protéger d'une attaque du type « homme du milieu », grâce à la véracité des clés des serveurs.

Protection par phrase de passe plus forte

> Les clés RSA peuvent être protégées par une phrase de passe, et non un mot de passe, ce qui ouvre un plus grand espace de recherche pour des méthodes d'attaque par la force brute. Ainsi, au lieu de « p@55w0rd », vous pouvez utiliser « Toby Betts est le cothurne de David Monk et drague F0xT4il. » (Vous devriez utiliser quelque chose de plus complexe que l'un ou l'autre de ces deux exemples.)

Authentification plus forte

> La force d'une authentification signifie, dans ce cas une paire de clés RSA, est relativement forte. Le chiffrement RSA est connu pour son coût et l'infaisabilité d'une attaque de type force brute. On ne peut pas dire la même chose des mots de passe.

Facilité accrue pour l'utilisateur

> Vous n'aimez pas taper des mots de passe souvent, non ? Moi aussi. Après quelques moments passés à tout mettre en place (ce qui fait à peu près autant de frappe au clavier qu'un mot de passe d'authentification de session), ce qui se fait maintenant sans effort, connectez-vous simplement à l'hôte distant et votre authentification est prise en charge.

Ainsi, je ne vois aucune raison (autre que ne pas savoir comment faire, ce que ce document essaie de vous enseigner) pour ne pas l'utiliser.

2. Composants

Tout d'abord, la distribution des rôles. Ces acteurs jouent tous un rôle particulier, il va nous falloir apprendre à les connaître :

ssh

> Le client SSH. Dans notre cas, nous allons travailler uniquement avec les programmes ssh1 de ssh.com (à savoir ssh-1.2.30) et OpenSSH (openssh-2.5.2)

sshd

Le serveur, soit dans sa version 1, soit dans sa version OpenSSH.

ssh-agent

L'agent qui va gérer l'interaction entre nos clés publiques et le client ssh.

ssh-add

L'outil qui va nous permettre d'ajouter (et enlever) nos clés RSA au cache de l'agent. Ils communiquent par le biais d'un tube sur la machine cliente.

ssh-keygen

L'outil qui permet de créer les paires de clés publiques et privées RSA utilisées dans l'authentification.

~/.ssh/identity

Le fichier contenant votre clé privée. Protégez-le bien !

```
-rw-----   1 jose     users         530 Feb  8 12:14 identity
```

~/.ssh/identity.pub

Le fichier contenant la partie publique de votre paire de clés RSA.

```
-rw-----   1 jose     users         334 Feb  8 12:14 identity.pub
```

~/.ssh/authorized_keys

Le fichier contenant une liste de clés publiques correspondant à vos clés privées. C'est ce fichier qui est utilisé pour gérer les authentifications

3. Premiers pas dans l'authentification par l'agent

Bien, commençons. L'ordre des opérations est relativement simple : générer une paire de clés, distribuer les clés publiques sur les serveurs sur lesquels nous allons nous connecter, puis configurer notre agent. Avant de commencer, assurons-nous que le serveur cible sache utiliser l'authentification par clé RSA.

```
$ grep RSA /etc/sshd_config
RSAAuthentication yes
```

Si la configuration dit « no », alors tout ça devient un sujet de conversation. Demandez à votre administrateur si vous en avez besoin.

Nous utilisons **ssh-keygen** pour générer la paire de clés. Une session classique reseemble à ça :

```
$ ssh-keygen
Initializing random number generator...
Generating p: .........................++ (distance 446)
Generating q: ..............++ (distance 168)
Computing the keys...
Testing the keys...
Key generation complete.
Enter file in which to save the key (/home/jose/.ssh/identity):
Enter passphrase: (not echoed)
Enter the same passphrase again: (not echoed)
Your identification has been saved in /home/jose/.ssh/identity.
Your public key is:
1024 37 13817424072879097025507991426858228764125028777547883762896424 32\
59597585487623134987310300351071105712187641659384690637621876213570981 5\
81119645923186045356271883326851730641652865341406978001102074124496073 9\
34884375702474119206648694266058341743663093177942158569001735419539170 0\
10038598384219240371212301614841694440673809 79 jose@biocserver
Your public key has been saved in /home/jose/.ssh/identity.pub
```

Bien, nous avons maintenant les deux pièces manquantes, nos clés publique et privée. Il nous faut donc distribuer la clé publique. Franchement, c'est comme avec PGP, vous pouvez partager ceci avec n'importe qui, puis vous connecter sans incident. J'utiliserai **scp** pour la copier sur les autres machines :

```
$ scp .ssh/identity.pub jon2@li:~/.ssh/biocserver.pub
jon2@li's password:(not echoed)
identity.pub | 0 KB | 0.3 kB/s | ETA: 00:00:00 | 100%
```

L'ayant copié là-bas, je vais maintenant me connecter sur la machine cible (dans notre cas la machine li du SCL) et l'ajouter à la liste des clés authorisées :

```
li$ cat biocserver.pub » authorized_keys
```

Ainsi, la machine li est prête à m'authentifier en utilisant ma clé privée RSA, clé générée ci-dessus. Retournons à ma machine cliente, et paramétrons **ssh-agent**. Tout d'abord, avant de lancer l'agent, regardons le contenu de quelques variables d'environnement de mon interpréteur de commandes :

```
$ env | grep -i SSH
SSH_TTY=/dev/ttyp3
SSH_CLIENT=129.22.241.148 785 22
```

Lançons maintenant **ssh-agent** proprement. Il doit lancer un sous-interpréteur, donc nous devons lui préciser lequel pour qu'il puisse le paramétrer correctement :

```
$ ssh-agent /bin/bash
```

Mon environnement est maintenant correctement paramétré :

```
$ env | grep -i SSH
SSH_TTY=/dev/ttyp3
SSH_AGENT_PID=3012
SSH_AUTH_SOCK=/tmp/ssh-jose/ssh-3011-agent
SSH_CLIENT=129.22.241.148 785 22
```

Les deux nouvelles variables SSH_AGENT_PID et SSH_AUTH_SOCK vont permettre à l'agent et aux applications connexes (le client **ssh**, le chargeur de cache de clés **ssh-add**, et autres du genre). Les sockets sont de simples fichiers dans le répertoire `/tmp` :

```
$ ls -l /tmp/ssh-jose/
total 0
srwx----   1 jose     users           0 Apr 24 13:36 ssh-3012-agent
```

Maintenant que l'agent est paramétré correctement, chargez le cache avec votre clé privée. Souvenez-vous que l'agent communique avec le client pour lui fournir votre clé privée quand vous vous authentifiez. Le lancer sans arguments le fait charger le fichier de la clé par défaut :

```
$ ssh-add1
Need passphrase for /home/jose/.ssh/identity (jose@biocserver).
Enter passphrase:(not echoed)
Identity added: /home/jose/.ssh/identity (jose@biocserver)
```

La phrase de passe que vous utilisez ici est faite pour s'assurer que « oui, c'est moi, j'ai le droit d'utiliser cette clé », et que c'est la même phrase de passe utilisée lorsque vous avez lancé **ssh-keygen**. Maintenant que la clé est chargée, regardons le contenu du cache, en utilisant l'option `-l` (pour liste) de **ssh-add** :

```
$ ssh-add -l
1024 37 113755886569632845157118935469762164915013148487621292987199586 1\
```

```
553162729709874182866289762398712097874714486515746971439573611270055860\
187630540060660487199692328631713510202123260680797564262765311338987532\
521475739334862853313810363888071565945239125248209981354764262500250893\
713818101131541180033061253240131839257 7 jose@biocserver
```

Maintenant, quand vous utilisez ssh pour vous connecter à un autre serveur, la clé privée vous authentifiera grâce à **ssh-agent** !

```
$ ssh -l jon2 li
Last login: Tue Apr 24 14:53:39 2001 from biocserver.bioc.
You have mail.
bash-2.03$
```

Regarde, maman, sans la phrase de passe !

Notez que vous pouvez modifier cette façon de faire, pour y introduire de la flexibilité. Pour commencer, vous pouvez utiliser la sortie du programme **ssh-agent** (invoqué sans argument), pour modifier l'environnement de l'interpréteur de commandes courant et indiquer la socket de communication de l'agent :

```
$ eval ‘ssh-agent‘
Agent pid 19353;
```

Vous pouvez maintenant ajouter des clés comme décrit ci-dessus, alors que vous n'avez pas démarré de sous-interpréteur, ayant simplement modifié l'interpréteur courant que vous utilisez. La commande **eval** et les apostrophes inverses sont nécessaires pour utiliser ce que l'agent fournit sur la sortie standard pour paramétrer votre environnement. Cela parce que les processus fils ne peuvent modifier les paramètres de l'interpréteur père.

Une seconde modification que vous pouvez appliquer est de démarrer votre bureau X, tel que Gnome ou KDE, en tant qu'argument à **ssh-agent**. Cela permettra à chaque client X démarré localement d'être capable de communiquer avec l'agent, permettant

une plus grande facilité quand vous utilisez des émulateurs de terminaux pour vous connecter à d'autres serveurs.

4. Note importante

Avant que nous terminions cet article, précisons un point très important : le cache est chargé, et vous avez authentifié vos clés privées. Elles sont en mémoire. Que se passerait-il si vous vous éloigniez de votre poste de travail ? N'importe qui aurait accès aux serveurs qui vous laissent vous authentifier avec vos clés RSA.

Cela dit, vous pouvez décharger des clés spécifiques en l'utilisant l'option `-d` de **ssh-add**, ou vous pouvez toutes les décharger en utilisant l'option `-D` :

```
$ ssh-add -D
All identities removed.
```

C'est un bon réflexe à acquérir quand vous vous éloignez de votre poste de travail. Il serait agréable d'avoir une fonctionnalité automatique après un certain temps d'inactivité, ou de relier ceci avec votre économiseur d'écran, ou encore avec la suspension (surseoit) APM de votre ordinateur portable.

5. Comment mal le faire

Lancer ssh-agent sans aucun argument, ce qui, pour parler de façon impropre, lance un sous-interpréteur de commande, mais sans positionner les bons paramètres de votre environnement. Ceux-ci seront affichés, mais ne seront pas pris en compte :

```
$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-jose/ssh-3019-agent; export SSH_AUTH_SOCK;
SSH_AGENT_PID=3020; export SSH_AGENT_PID;
echo Agent pid 3020;
```

Jetons un coup d'oeil pour vérifier que les variables d'environnement sont bien définies dans notre interpréteur de commandes. Celles-ci sont nécessaires pour que l'agent fonctionne correctement, ainsi que nous l'avons vu plus haut :

```
$ env | grep -i ssh
SSH_TTY=/dev/ttyp3
SSH_CLIENT=129.22.241.148 785 22
```

Les conséquences de tout ça sont évidentes lorsque l'on essaie d'ajouter des clés dans le cache :

```
$ ssh-add
Need passphrase for /home/jose/.ssh/identity (jose@biocserver).
Enter passphrase: (not echoed)

Could not open a connection to your authentication agent.
```

(**ssh-add**) ne peut trouver la socket ou l'identifiant du processus de l'agent, qui est défini dans cette variable. Ainsi, aucune clé n'est disponible par le cache.

6. Conclusion

Cet article est la plus rudimentaire des introductions sur la façon d'utiliser ssh-agent pour des authentifications fortes. Vous devez expérimenter si vous voulez en apprendre un peu plus, comme ajouter des clés autres que celles par défaut, et lire l'excellente documentation de la distribution OpenSSH. Le livre à l'escargot de O'Reilly, « SSH: Secure Shell, The Definitive Guide », est une très bon livre de référence, et donc plus que recommandé.

7. Articles précédents à propos des outils SSH

- Utilisation de ssh
- SSH et compagnie : sftp, scp et ssh-agent

Paru dans le numéro 67 de la Linux Gazette, Juin 2001.

Traduction française par Jérôme Fenal <jerome@fenal.org>.