RFC 4251 : The Secure Shell (SSH) Protocol Architecture

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 27 février 2009

Date de publication du RFC : Janvier 2006

https://www.bortzmeyer.org/4251.html

Pendant longtemps, les connexions interactives à une machine distante, via l'Internet, se faisaient par un protocole (telnet ou rlogin) qui ne chiffrait pas les communications et notamment le mot de passe échangé pour s'authentifier. D'innombrables "*sniffers*" ont ainsi capturés des mots de passe et les ont ensuite transmis à Eve, pour qu'elle en fasse un mauvais usage. L'arrivée de SSH, en 1995 <http://groups.google.com/group/comp.security.unix/msg/67079d812a19f499>, a donc bouleversé la sécurité sur Internet. Sa simplicité d'usage, la possibilité de résoudre enfin la majorité des problèmes de sécurité de X11, et la sécurité qu'il offre lui ont permis de l'emporter facilement contre des concurrents compliqués à déployer et à utiliser comme Kerberos telnet.

SSH est désormais un des protocoles de base de l'infrastructure d'Internet. Il a plusieurs mises en œuvre dont une en logiciel libre, OpenSSH. Longtemps spécifié uniquement dans les documents liés à une mise en œuvre particulière, SSH est aujourd'hui normalisé dans une série de RFC dont ce RFC 4251 [1] est le point de départ. Il décrit l'architecture générale, le RFC 4252 décrit le protocole d'authentification, le RFC 4253 le protocole de transport, le RFC 4254 le protocole de connexion et le RFC 4250 les différents identificateurs utilisés par SSH.

Les différents protocoles sont résumés dans la section 1 :

— Tout en bas, le protocole de transport (RFC 4253), authentifie le serveur et chiffre les communications avec lui.
— Un cran au dessus, le protocole d'authentification (RFC 4252) permet au client de s'authentifier auprès du serveur.
— Enfin, tout en haut, le protocole de connexion (RFC 4254) multiplexe plusieurs canaux (par exemple une session interactive de type shell et une session X11) sur un seul transport chiffré.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc4251.txt

Quels sont les composants essentiels de SSH ? La section 4 les énumère. D'abord, il y a la notion de clé de machine (section 4.1). Il n'existe pas sur Internet de mécanisme général d'**identité** des machines. Ce manque est parfois gênant, par exemple dans un réseau pair-à-pair où on essaie de garder trace des services que rend un pair, ou bien dans des modèles de sécurité comme la "*Same Origin Policy*" de Javascript où le navigateur Web ne peut se connecter qu'à « la même machine » que celle d'où vient le code Javascript, sans que « la même machine » soit une notion définie rigoureusement, ce qui permet les changements d'adresse IP <https://www.bortzmeyer.org/dns-rebinding-pinning.html>.

SSH fonctionne donc en donnant à chaque machine une identité qui est sa clé publique (stockée, avec OpenSSH, en `/etc/ssh/ssh_host_dsa_key.pub`). Cette clé permettra de s'assurer qu'on parle bien à la machine attendue. Quel modèle de confiance utiliser ? Le RFC en cite deux, une base locale des clés publiques des machines connues, qui a l'avantage de ne pas nécessiter de lourdes et chères PKI, ou bien une PKI, avec une CA qui signe les clés. SSH n'impose pas un modèle particulier mais encourage la première méthode. En effet, le RFC insiste beaucoup sur la nécessité pour le protocole d'être facile à utiliser et facilement déployable. Les PKI en sont tout l'opposé et, si SSH avait imposé l'usage d'une PKI, il n'aurait probablement jamais connu de succès (le RFC 5218 discute ce point). (Depuis, le RFC 4255 a fourni un mécanisme permettant de récupérer la clé dans le DNS ; sans DNSSEC, ce mécanisme ne vaut pas grand'chose.)

Pour peupler cette base locale, la même section 4.1 conseille un modèle dit TOFU ("*Trust On First Use*") ou encore (mais ce terme est moins précis) "*leap of faith*" (acte de foi), où la clé est vérifiée systématiquement, sauf à la première connexion, où on demande juste à l'utilisateur son accord pour enregistrer la nouvelle machine, en lui indiquant l'empreinte de la clé publique :

```
% ssh susanna
The authenticity of host 'susanna (2001:660:3003:3::1:3)' can't be established.
DSA key fingerprint is 56:b7:62:30:11:d3:6d:b6:d5:ec:5d:1e:7e:3c:42:1e.
Are you sure you want to continue connecting (yes/no)?
```

Si ce modèle rencontre l'opposition de certains experts en sécurité (car il rend SSH vulnérables aux attaques de l'homme du milieu lors de la première connexion), c'est aussi lui qui a permis le succès du déploiement de SSH, d'autant plus qu'il n'existe aucune PKI largement déployée sur l'Internet. Ici, le mieux était clairement l'ennemi du bien si on avait attendu une telle PKI, on utiliserait toujours des connexions non chiffrées et bien plus vulnérables. (La section 9.3.4 revient en grand détail sur le risque d'attaque par l'homme du milieu et sur la vérification des clés des machines auxquelles on se connecte.)

(À noter qu'aujourd'hui, des techniques comme Perspectives <https://www.bortzmeyer.org/perspectives-ssh.html> tentent d'améliorer le système TOFU.)

Les clés SSH sont donc en général non signées. On peut aussi, outre les clés des machines, avoir des clés liées à un utilisateur. Celui-ci les génère, avec OpenSSH, en tapant `ssh-keygen`, et peut les utiliser pour s'authentifier, ce qui est plus sûr qu'un mot de passe, car cela évite de transmettre un secret réutilisable (le mot de passe) à une machine pas forcément fiable (le serveur). Les sections 9.4.4 et 9.4.5 discutent les forces et les faiblesses des deux mécanismes d'authentification : la clé publique ne nécessite pas de transmettre un secret au serveur mais sa compromission sur la machine client est toujours possible, et le serveur ne peut pas contrôler si la machine client applique de bonnes politiques de sécurité.

Comme précisé dans la section 4.5, SSH peut utiliser plusieurs algorithmes. La cryptographie et la cryptanalyse évoluent en effet sans cesse et il peut être nécessaire de changer les algorithmes de cryptographie utilisés, sans changer le protocole (cf. section 9.3.1). Pour cela, chaque algorithme a un

nom (la section 6 détaille les règles de nommage) et peut être négocié à l'établissement de la connexion (avec OpenSSH, option `-c` en ligne de commande et mot-clé `Ciphers` dans `sshd_config`). Les algorithmes officiels sont enregistrés dans un registre IANA <https://www.iana.org/assignments/ssh-parameters> et les non-officiels ont un nom qui inclus un @, par exemple `mon-algo@leroidelacrypto.fr`.

La section 9, consacrée à l'étude détaillée des questions de sécurité est évidemment particulièrement longue. C'est une lecture indispensable pour qui veut évaluer la sécurité de SSH. 9.1 rappelle l'importance cruciale de bons algorithmes de génération de nombres aléatoires, une faiblesse classique en cryptographie (bien illustrée, par exemple, par la faille Debian/OpenSSL <http://wiki.debian.org/SSLkeys>, qui a affecté OpenSSH sur Debian). Il est donc nécessaire de suivre les consignes du RFC 4086.

La section 9.3.1 discute le choix des algorithmes de cryptographie, en recommandant de s'appuyer sur l'état actuel de la science, en utilisant des algorithmes reconnus comme AES.

Certaines fonctions de SSH sont débrayables. La section 9.3.2 rappelle que le contrôle d'intégrité par MAC peut être ainsi coupé (option `-m` de OpenSSH) et insiste que cela doit être fait uniquement pour le débogage.

Tous les systèmes utilisant la cryptographie sont vulnérables à des attaques DoS où l'attaquant va déclencher chez sa victime de lourds calculs... sans aller jusqu'à s'authentifier. La section 9.3.5 recommande donc de permettre de limiter les tentatives depuis certaines machines (avec OpenSSH, on utilise en général les " *tcpwrappers*" pour cela et on met quelque chose comme `sshd: [2001:DB8:0:1::]/64` dans `/etc/hosts.allow` et pour arrêter les machines en dehors de `2001:DB8:0:1::/64`, on a `sshd: ALL` dans `/etc/hosts.deny`.

Pour illustrer un certain nombre de points de SSH, voici le résultat d'un `ssh -v` depuis une machine OpenSSH (Debian) depuis une autre (Gentoo) :

```
OpenSSH_5.1p1 Debian-5, OpenSSL 0.9.8g 19 Oct 2007
...
debug1: Connecting to munzer.bortzmeyer.org [10.75.84.80] port 6622.
debug1: Connection established.
```

La connexion TCP a été établie.

```
...
debug1: Checking blacklist file /usr/share/ssh/blacklist.DSA-1024
debug1: Checking blacklist file /etc/ssh/blacklist.DSA-1024
```

Le test ci-dessus est spécifique à Debian et correspond à la faille Debian 1571 <http://www.debian.org/security/2008/dsa-1571>.

```
debug1: Remote protocol version 2.0, remote software version OpenSSH_5.1
...
debug1: kex: server->client aes128-cbc hmac-md5 none
debug1: kex: client->server aes128-cbc hmac-md5 none
```

Ici, les algorithmes de cryptographie acceptés ont été transmis (section 4.5 du RFC). Le premier est AES.

```
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
debug1: checking without port identifier
debug1: Host 'munzer.bortzmeyer.org' is known and matches the RSA host key.
debug1: Found key in /home/stephane/.ssh/known_hosts:24
debug1: ssh_rsa_verify: signature correct
```

Ici, la vérification de la machine distante, `munzer.bortzmeyer.org`, a été faite. Déjà connue, sa clé a été acceptée.

```
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,keyboard-interactive
debug1: Next authentication method: publickey
debug1: Offering public key: /home/stephane/.ssh/id_dsa
debug1: Server accepts key: pkalg ssh-dss blen 433
debug1: Authentication succeeded (publickey).
```

Une authentification de l'utilisateur par la clé publique DSA qu'il a présentée a marché.

```
debug1: Entering interactive session.
```

Et le reste n'est plus que formalité...

Si, par contre, une machine s'était glissée au milieu, devant le serveur attendu, on aurait obtenu le fameux message :

```
The authenticity of host '[munzer.bortzmeyer.org]:6622 ([10.75.84.80]:6622)' can't be established.
RSA key fingerprint is ac:73:5e:34:59:29:7f:4a:a0:9f:56:d4:00:21:fe:c6.
Are you sure you want to continue connecting (yes/no)?
```

SSH est aujourd'hui présent sur pratiquement, toutes les machines, du téléphone portable au commutateur Ethernet. Malgré cela, les mauvaises habitudes ont la vie dure et le rapport 2008 d'Arbor Network sur la sécurité <http://www.arbornetworks.com/en/docman/worldwide-infrastructure-securi
download.html> remarquait que 24 % des opérateurs utilisaient toujours telnet (et ce rapport est basé uniquement sur les déclarations des acteurs, qui n'ont pas intérêt à avouer de telles faiblesses ; le chiffre réel est donc probablement supérieur.).