Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE
PREMIER MINISTRE

S . G . D . S . N
Agence nationale de la sécurité des systèmes d'information
CERT-FR

Paris, le 29 mai 2017
N° CERTFR-2017-ACT-022

Affaire suivie par :
CERT-FR

BULLETIN D'ACTUALITÉ

Objet : Bulletin d'actualité CERTFR-2017-ACT-022

1 - Authentification avec OpenSSH

Présentation rapide du projet et du protocole

OpenSSH, dont la version actuelle est la 7.58, est un projet soutenu par certains développeurs d'OpenBSD, qui vise à fournir des outils de communication s'appuyant sur le protocole SSH défini par plusieurs RFCs(voir Documentation). Grâce à ces outils, OpenSSH permet la transmission de commande, de fichier ou la création de tunnels entre différents noeuds connectés. Certaines RFCs, spécifiques à l'utilisation d'algorithmes particuliers au sein du protocole SSH, ne sont pas mentionnées dans cet article, mais sont consultables au travers de la spécification du protocole. Pour mémoire, un guide de bonne pratique pour "un usage sécurisé" d'OpenSSH est disponible sur le site de l'ANSSI depuis 2014.

Rappel sur les méthodes d'authentification

Lors de la première connexion SSH, afin de vérifier par la suite l'identité du serveur, un condensat de la clef SSH publique du serveur est présenté à l'utilisateur, puis stocké dans un fichier nommé par défaut known_hosts. Ce fichier associe un identifiant (FQDN, IP, etc.) avec une clef publique et l'algorithme associé. Une fois la connexion SSH établie, l'utilisateur peut être authentifié suivant différentes méthodes :

- mot de passe (password) ;
- clef SSH (publickey) ;
- noeud source (hostbased) ;
- GSSAPI ;
- interactive (keyboard-interactive).

La méthode keyboard-interactive, permet aussi d'authentifier l'utilisateur en s'appuyant sur l'utilisation de jeton de sécurité ou de personnaliser les modalités d'authentification (par exemple via un module PAM ou un serveur Kerberos). Les différentes méthodes d'authentifications proposées ci-dessus peuvent être combinées et ordonnées (en utilisant l'option AuthenticationMethods) afin d'imposer, par exemple, la possession de plusieurs secrets à l'utilisateur. Pour plus d'information sur les différentes méthodes d'authentification et leurs combinaisons, vous pourrez lire la présentation de Facebook lors de la conférence security @scale 2014.

Rappel sur les algorithmes utilisés

OpenSSH s'appuie sur différentes méthodes et algorithmes pour chiffrer, authentifier et garantir l'intégrité des échanges. Régulièrement, OpenSSH étend ses capacités par l'ajout de nouveaux algorithmes pouvant s'utiliser dans différentes parties des sessions (échange de clef, authentification, chiffrement, intégrité). Le choix des algorithmes et leurs paramétrages doivent être étudiés avec soin et en cohérence avec les éléments utilisés et leurs conditions d'usage (se référer au RGS pour plus d'information).

Certificats OpenSSH

L'authentification des utilisateurs est généralement réalisée grâce à une clef publique SSH. Ceci implique de copier la clef de l'ensemble des utilisateurs sur l'ensemble des serveurs ce qui en complexifie le déploiement et la mise à jour. Pour pallier cette difficulté, le projet OpenSSH (à partir de la version 5.4) est en capacité de produire et d'utiliser des certificats OpenSSH (cette fonctionnalité est spécifique au projet OpenSSH). Les certificats OpenSSH, comme les certificats x509, permettent d'associer une identité (nom d'utilisateur, FQDN, etc.) à une clef publique en s'appuyant sur une Autorité de Certification(CA). Dans ce cadre, il suffit donc d'installer les clefs publiques des autorités de certifications sur les serveurs pour permettre l'authentification de l'ensemble des utilisateurs. De la même façon, il devient possible de signer les clefs publiques SSH utilisées par les serveurs (démon sshd) et permettre ainsi l'authentification du serveur par l'ensemble des postes clients.

Ce fonctionnement nécessite la configuration des clients et des serveurs pour activer l'utilisation des certificats. Côté serveur, l'option de configuration TrustedUserCAKeys permet de lister l'ensemble des clefs publiques d'AC de confiance pour identifier les utilisateurs. Côté client, la configuration se fait par l'ajout de ligne commençant par @cert-authority dans le fichier known_hosts ce qui permet de lister les clefs publiques des AC ayant autorité pour authentifier les serveurs suivant leur domaine.

Cependant, les certificats OpenSSH sont légèrement différents des x509 que ce soit en format, mais aussi dans l'architecture de l'infrastructure de gestion de clefs (IGC). Un certificat OpenSSH contient :

- Une identité (nom d'utilisateur, FQDN);
- Une clef publique et son algorithme ;
- Une période de validité ;
- Un numéro de série ;
- Un identifiant de clef (key_id) qui sera utilisé lors des processus de journalisation ;
- Un NONCE ;
- Éventuellement des limitations qui s'appliqueront sur les actions possibles après authentification ;
- La signature et clef publique de l'AC.

Comme souligné précédemment, un certificat OpenSSH intègre des options qui lui permettent aussi d'être extensible pour de nouveaux usages dans le futur. On remarque premièrement que les extensions doivent être marquées comme "critique" ou "non-critique", la violation d'une option critique devra engendrer un échec de l'authentification. Certaines options non critiques permettent une certaine granularité sur l'utilité du certificat (permit-X11-forwarding, permit-port-forwarding, etc.). Enfin, comme toute IGC il est possible pour une AC de publier une liste de révocation permettant de refuser l'usage de certaines clefs.

Cas pratique

Création et usage de certificat OpenSSH (certificat utilisateur)

Dans un premier temps, nous allons créer une paire de clefs qui sera utilisée par notre AC (cette clef privée sera utilisée pour signer les certificats utilisateurs):

```
$ mkdir -p -m 0700 ca  $ ssh-keygen -t rsa -b 4096 -C 'clef AC' -f ./ca/ca.ssh.key
```

Il est important de faire ces actions dans de bonnes conditions (entropie suffisante, noeud déconnecté, protection de la clef par une passphrase, etc.).

```
$ ssh-keygen -l -f ./ca/ca.ssh.key
4096 SHA256:1SNo9dfE3au8YRIxMO2ryb4vPs49mHUzvDYTVMhjVX4 clef AC  (RSA)
```

Nous allons maintenant créer une paire de clef OpenSSH pour l'utilisateur "user1" :

```
$ mkdir -p -m 0700 ./user1 $ ssh-keygen -t rsa -b 4096 -C 'user1'
  -f ./user1/user1.ssh.key $ ssh-keygen -l -f ./user1/user1.ssh.key
4096 SHA256:+gRpX211mR/GEu93MtgnHBR20ZCGn8cM6V0Fn4tCB4E user1 (RSA)
```

Il convient ensuite pour notre AC de signer cette clef et de lui donner une validité adaptée. Le paramètre principal devra contenir le nom de l'utilisateur avec lequel il devra s'authentifier :

```
$ ssh-keygen -V +1D -s ./ca/ca.ssh.key -I 'user1 authentication SSH cert'
  -n user1 ./user1/user1.ssh.key.pub
Signed user key ./user1/user1.ssh.key-cert.pub: id "user1 authentication SSH cert"
serial 0 for user1 valid from 2017-05-15T12:27:00 to 2017-05-16T12:28:56
```

Un nouveau fichier sera donc créé et portera l'extension -cert.pub (./user1/user1.ssh.key-cert.pub) :

```
 ssh-rsa-cert-v01@openssh.com AAAAHHNzaC1yc2EtY2VydC12MDFAb3BlbnNzaC5jb20AAA
Agh6bLWl9I2VKXdG7EuN3IUQN1OWjemWjCxLLZwZXhPKsAAAADAQABAAACAQDgUQg5qoqBg6h2IR
597T1nMhtPHPXMtB7HAqemFsFGKwA3DFUTX0fYj6T7a3BsAryQcbNmerRbYubkahknrZH5Hum0U0
lh1svalrMCBu62Irv03mZYt7vTP7OTkb/dWJ4TDDvfY+O5XsTCz0MOj0QY+g+u2Bx9sObxSN0tTO
bRbcP3Oh0usxlrxiweou41mpiS2teLXWa66M3zvNyZZ9F8JZTpLxpEJoUFoljeZ4zYRM1iRQLULB
XUFEuXQPFxJE0s4bbEIPXAcjnq2QuUNR60i5Mvl68bamAxaw97enJPTOOgHBu9WSlIhLV80hsDme
m05W9AcvyZSWp3P3iird80bgh6bxsgkAl4UycbASqlQ5TKDqqZSJZKJ6ozl2BEpkOvw9gq7nnVxi
oHnougitSmyyGldqapWBkqWdhJwOcmOdR7Mj25jcOGRTZuBqOWOcSB6hEPmWEGejZDEVE34FvBKv
eWOpadIis96dLpLVYxIw+Kn51EoG0cqtbUjosW/EhqMDDCQ97C8FHgqvmeteLUuOtm2TLlJSpPHH
Vqzoyg4Hf0w2SphpSnsIp0z05JK+LnL4ZICEM6wARYmGYCBpUoG8fpyu22DPYYhP23abDFb5CLTZ
Mn2Mfxb73KcRTpoLisVGAVL/gxs/TYH+f8syH3zyOcb8qsgjqjcqWI5RwAzQAAAAAAAAAAAAAAAQ
AAAB11c2VyMSBhdXRoZW50aWNhdGlvbiBTU0ggY2VydAAAAAkAAAAFdXNlcjEAAAAAWRmelAAAAA
BZGvCIAAAAAAAAAIIAAAAVcGVybWl0LVgxMS1mb3J3YXJkaW5nAAAAAAAAABdwZXJtaXQtYWdlbn
QtZm9yd2FyZGluZwAAAAAAAAAWcGVybWl0LXBvcnQtZm9yd2FyZGluZwAAAAAAAAAKcGVybWl0LX
B0eQAAAAAAAAAOcGVybWl0LXVzZXItcmMAAAAAAAAAAAAAAAAhcAAAAHc3NoLXJzYQAAAAMBAAEAAA
IBAOY3dtgnofupWNiECBaJkt+VGrziYiq5QfRX3vDefAAXmc41DVakpw4yFW2EWRln0jQeQSQoH7
ZABd47tqkp2WG1VGFkFeXtC6YUKyPj5YLFdUY1kO+Qj4nEgC63Nzl4N2nVdSGuilvBg6n5ouIHKA
8eb2br6vy0TM1ojez/a+C5vye3LxH/m1M57afexgU5eKSJBiaN3k4peOyHPONOeo1bsNdfOnGD6D
PuKS3UwnI7RhS8EpxhnkQKyKgp0i2IwK8yOGXvRvb3HtDWfPFLSvN+dCzaedW4Pwj4b/50S7tSI0
7MRewgDpnqeduQZ1VU4YDmW7WcBdb2BkrepiFEbPU/eGfp5OwQDHipw6a/Juv8nvcdizHY/A6KvW
aCyrYsY+8AHDo14fnvQ5W48LLM54JQaHfseM5Q3ndCFFYU+8cuvE3fSjdrlvTWw2rdt9kh/gTvyl
X8zgRP9OR+/OA8wCEQRTutQ66s1O9lk6pibcE7a2XDSDcgcg2e60FJNKxRCOrnjwhZlwdZHk0qdl
OeLyF3NRU3RGp8jqZU1G6c0NAPQ5uQd4dE8onffAoFal/0R7Owzxu29GWqi9sjUDcpz0K2VlY6l1
jwCwOBGlZxENrDud4Z3957d2W7wnfqS7Nr6Ih4yCe7tA//BHpVnZGpQwI7pAbwrK6NgfZrRDdiZg
sHAAACDwAAAAdzc2gtcnNhAAACALwCXCYo8F2CS3JA6ooQ0WAojbPfk5kZ3WI1rSNKMBAJy19vtX
3aeaqWpOV/FvNcr1cgOdwsZ4tVFmlqRIGIjij9uouyHTeubUHMBp/wkOR8WifZRtD9uE3Jyt2sFZ
jGEjLK/frpaf4S58W6wwnZQGAFVla73jYLQqEx9PYc3gwNltqgUNhW8cLS1z9GW6YjoGfohz3h/l
jtFePW4KyrHyz7XOan+t0uKr7z3iuunQEbeRRpfb18hsjOTHtRcY0+pMcwBVRn9inUSUplnnAkAn
uoPxCg0s94TcKAtwYwGeI+hw7XczRGTWJzO99RPSXBpITxyEA9nauVAFW9A4wL2vSH7FwyonF0LC
FUexXpYLlRGXcatNHOfrqOq7E83uTMG9023hdJ5kfGmajxmbQDbdv1EIr32kARN2TKntPllEfBWS
5zXvrH/9e2x0TA9eTDtsgbasqbhkirslp/iD8HIuNI4Dox5/b2qwJzSzxYy6NIzy4n4pugObTQJC
X7m9erYqo4b7jfDsln+nxLSG5b+c2LG9WSKAvTeXF85etkxhM5lZlTBIbChnopCaRnyPPmXJcXrR
O0Lq9gK2Xh01MK5PUtk+xvcTFrmEVYl54C4ueAbVudFMEeKuoFPKSClmoYZaD4PfSfAxd931G05U
g6ShCupVwn0RADj+BYEEhhLiEM user1
```

L'encodage du certificat utilise les mêmes formats que pour l'encodage des clefs. Enfin, il est possible de lister le certificat sous une forme plus verbeuse :

```
 $ ssh-keygen -L -f ./user1/user1.ssh.key-cert.pub  ./user1/user1.ssh.key-cert.pub:
Type: ssh-rsa-cert-v01@openssh.com user certificate
Public key: RSA-CERT SHA256:+gRpX211mR/GEu93MtgnHBR20ZCGn8cM6V0Fn4tCB4E
Signing CA: RSA SHA256:1SNo9dfE3au8YRIxMO2ryb4vPs49mHUzvDYTVMhjVX4
Key ID: "user1 authentication SSH cert"
Serial: 0
Valid: from 2017-05-15T12:27:00 to 2017-05-16T12:28:56
Principals:                 user1
Critical Options: (none)
Extensions:
                permit-X11-forwarding
                permit-agent-forwarding
                permit-port-forwarding
                permit-pty
                permit-user-rc
```

On remarque bien que le champ "SHA256" contient bien l'empreinte de la clef de notre utilisateur et de notre AC. Plus d'options sont disponibles dans le manuel ssh-keygen(5). Le fichier de configuration du serveur SSH doit ensuite être modifié pour lister l'ensemble des AC autorisées à signer des certificats utilisateur, ainsi que la liste des utilisateurs autorisés à s'authentifier avec un certificat.

Le fichier de configuration du démon SSH (/etc/ssh/sshd_config):

```
TrustedUserCAKeys /etc/ssh/ssh_ca_user
AuthorizedPrincipalsFile /etc/ssh/ssh_principals
```

Le fichier contenant les clefs publiques des AC de confiances (/etc/ssh/ssh_ca_user):

```
 ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDmN3bYJ6H7qVjYhAgWiZLflRq84mIquUH0V97w3
nwAF5nONQ1WpKcOMhVthFkZZ9I0HkEkKB+2QAXeO7apKdlhtVRhZBXl7QumFCsj4+WCxXVGNZDvkI+
JxIAutzc5eDdp1XUhropbwYOp+aLiBygPHm9m6+r8tEzNaI3s/2vgub8nty8R/5tTOe2n3sYFOXiki
QYmjd5OKXjshzzjTnqNW7DXXzpxg+gz7ikt1MJyO0YUvBKcYZ5ECsioKdItiMCvMjhl70b29x7Q1nz
xS0rzfnQs2nnVuD8I+G/+dEu7UiNOzEXsIA6Z6nnbkGdVVOGA5lu1nAXW9gZK3qYhRGz1P3hn6eTsE
Ax4qcOmvybr/J73HYsx2PwOir1mgsq2LGPvABw6NeH570OVuPCyzOeCUGh37HjOUN53QhRWFPvHLrx
N30o3a5b01sNq3bfZIf4E78pV/M4ET/TkfvzgPMAhEEU7rUOurNTvZZOqYm3BO2tlw0g3IHINnutBS
TSsUQjq548IWZcHWR5NKnZTni8hdzUVN0RqfI6mVNRunNDQD0ObkHeHRPKJ33wKBWpf9EezsM8btvR
lqovbI1A3Kc9CtlZWOpdY8AsDgRpWcRDaw7neGd/ee3dlu8J36kuza+iIeMgnu7QP/wR6VZ2RqUMCO
6QG8KyujYH2a0Q3YmYLBw== clef AC
```

La liste des utilisateurs locaux pouvant s'authentifier avec un certificat (/etc/ssh/ssh_principals):

```
 user1
```

On note que l'utilisateur unix "user1" doit être localement présent sur la machine du serveur SSH :

```
 $ getent passwd user1 user1:x:1001:1001::/home/user1:/bin/sh
```

On peut maintenant faire un test de connexion :

```
 $ ssh -l user1 -i ./user1/user1.ssh.key 127.0.0.1
```

Ce qui apparait dans les journaux sous la forme suivante :

```
Accepted publickey for user1 from 127.0.0.1 port 52226 ssh2: RSA-CERT ID user1
authentication SSH cert (serial 0) CA RSA
SHA256:1SNo9dfE3au8YRIxMO2ryb4vPs49mHUzvDYTVMhjVX4
```

L'exemple ne sera pas illustré ici, mais la présentation d'un certificat valide, mais signé par une AC non présente dans le fichier /etc/ssh/ssh_ca_user ou pour un principal non présent dans /etc/ssh/ssh_principals conduira à une erreur.

Création et usage de certificat OpenSSH (certificat serveur)

Dans cet exemple, nous allons garder la même hiérarchie que précédemment. Dans un premier temps sur le client, on s'assure que nous n'avons aucune information présente dans nos fichiers known_hosts :

```
 $ ssh-keygen -F 127.0.0.1
```

Puis on signe la clef utilisée par notre serveur ssh

```
 $ ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key.pub
2048 SHA256:5LIB5jDi07opJgG452hEOnR0LFqCH8SoM2V24NvDkW8 root@totorong (RSA)
$ ssh-keygen -V +52w -s ./ca/ca.ssh.key -h -I 'local serveur cert'
   -n 127.0.0.1 /etc/ssh/ssh_host_rsa_key.pub
Signed host key /etc/ssh/ssh_host_rsa_key-cert.pub: id "local serveur cert"
serial 0 for 127.0.0.1 valid from 2017-05-15T12:36:00 to 2018-05-14T12:37:03
```

Il faudra maintenant configurer le serveur SSH pour ne plus utiliser les clefs, mais uniquement les certificats associés à ses clefs (/etc/ssh/sshd_config):

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

Puis signifier que notre AC est de confiance pour signer des certificats pour 127.0.0.1, le client doit éditer le fichier /etc/ssh/ssh_known_hosts :

```
 @cert-authority 127.0.0.1 ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDmN3bYJ6H7qVjYh
AgWiZLflRq84mIquUH0V97w3nwAF5nONQ1WpKcOMhVthFkZZ9I0HkEkKB+2QAXeO7apKdlhtVRhZBXl7
QumFCsj4+WCxXVGNZDvkI+JxIAutzc5eDdp1XUhropbwYOp+aLiBygPHm9m6+r8tEzNaI3s/2vgub8nt
y8R/5tTOe2n3sYFOXikiQYmjd5OKXjshzzjTnqNW7DXXzpxg+gz7ikt1MJyO0YUvBKcYZ5ECsioKdIti
MCvMjhl70b29x7Q1nzxS0rzfnQs2nnVuD8I+G/+dEu7UiNOzEXsIA6Z6nnbkGdVVOGA5lu1nAXW9gZK3
qYhRGz1P3hn6eTsEAx4qcOmvybr/J73HYsx2PwOir1mgsq2LGPvABw6NeH570OVuPCyzOeCUGh37HjOU
N53QhRWFPvHLrxN30o3a5b01sNq3bfZIf4E78pV/M4ET/TkfvzgPMAhEEU7rUOurNTvZZOqYm3BO2tlw
0g3IHINnutBSTSsUQjq548IWZcHWR5NKnZTni8hdzUVN0RqfI6mVNRunNDQD0ObkHeHRPKJ33wKBWpf9
EezsM8btvRlqovbI1A3Kc9CtlZWOpdY8AsDgRpWcRDaw7neGd/ee3dlu8J36kuza+iIeMgnu7QP/wR6V
Z2RqUMCO6QG8KyujYH2a0Q3YmYLBw== clef AC
```

```
 $ ssh -v -l user1 -i ./user1/user1.ssh.key 127.0.0.1
debug1: Server host certificate: ssh-rsa-cert-v01@openssh.com
 SHA256:5LIB5jDi07opJgG452hEOnR0LFqCH8SoM2V24NvDkW8, serial 0 ID "local serveur cert"
 CA ssh-rsa SHA256:1SNo9dfE3au8YRIxMO2ryb4vPs49mHUzvDYTVMhjVX4
 valid from 2017-05-15T12:36:00 to 2018-05-14T12:37:03
debug1: Host '127.0.0.1' is known and matches the RSA-CERT host certificate.
debug1: Found CA key in /etc/ssh/ssh_known_hosts:1
```

Création d'une Key Revocation List

Cette liste contiendra l'ensemble des condensats des clefs ne devant plus être utilisées.

```
# ssh-keygen -k -s ./ca/ca.ssh.key -f /etc/ssh/krl ./user1/user1.ssh.key.pub
Revoking from ./user1/user1.ssh.key.pub
```

Il sera encore nécessaire de faire une modification dans la configuration du serveur SSH pour prendre en compte cette liste de révocation (/etc/ssh/sshd_config).

```
RevokedKeys /etc/ssh/krl
```

L'utilisation d'un certificat révoqué fera échouer l'authentification.

```
 $ ssh -v -l user1 -i ./user1/user1.ssh.key 127.0.0.1
Received disconnect from 127.0.0.1 port 22:2: Too many authentication failures
Authentication failed.
```

Les journaux nous donnent plus d'information sur la raison de l'échec:

```
error: Authentication key RSA SHA256:+gRpX211mR/GEu93MtgnHBR20ZCGn8cM6V0Fn4tCB4E
       revoked by file /etc/ssh/krl
error: Authentication key RSA-CERT SHA256:+gRpX211mR/GEu93MtgnHBR20ZCGn8cM6V0Fn4tCB4E
       revoked by file /etc/ssh/krl
```

On note qu'il est possible de révoquer une clef sans avoir le fichier de clef publique, la seule connaissance de l'ID/serial du certificat, ou du hash de la clef est suffisante.

Utilisation de jeton Yubikey - certificat serveur

Il est intéressant de pouvoir stocker ses clefs secrètes sur un périphérique par exemple une clef Yubikey. Cette méthode n'est pas incompatible avec l'utilisation de certificat, il sera simplement nécessaire de déplacer le certificat signé sur un autre périphérique. Nous allons donc utiliser la clef privée stockée dans le slot PIV sur une Yubikey 4. Pour plus d'information, un article explique le format des clefs SSH11 pour mieux comprendre leur représentation et la possibilité de stockage dans les Yubikeys.

Dans un premier temps il faut extraire la partie publique de la clef stockée sur la Yubikey :

```
 $ ssh-keygen -D /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so -e > ca_yubi.ssh.key.pub
```

On signe la clef du serveur SSH local en utilisant la clef stockée sur le jeton PKCS11 :

```
# ssh-keygen -V +52w -D /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
  -s ./ca/ca_yubi.ssh.key.pub -h -I 'local serveur cert'
  -n 127.0.0.1 /etc/ssh/ssh_host_rsa_key.pub
Enter PIN for 'PIV_II (PIV Card Holder pin)':
Signed host key /etc/ssh/ssh_host_rsa_key-cert.pub: id "local serveur cert"
serial 0 for 127.0.0.1  valid from 2017-04-20T15:07:00 to 2018-04-19T15:08:40
```

Puis, on effectue les modifications dans le fichier /etc/ssh/ssh_known_hosts.

```
 $ ssh -v -l user1 -i ./user1/user1.ssh.key 127.0.0.1
debug1: Server host certificate: ssh-rsa-cert-v01@openssh.com
 SHA256:5LIB5jDi07opJgG452hEOnR0LFqCH8SoM2V24NvDkW8,
 serial 0 ID "local serveur cert" CA ssh-rsa
 SHA256:1SNo9dfE3au8YRIxMO2ryb4vPs49mHUzvDYTVMhjVX4
 valid from 2017-05-15T12:36:00 to 2018-05-14T12:37:03
debug1: Host '127.0.0.1' is known and matches the RSA-CERT host certificate.
debug1: Found CA key in /etc/ssh/ssh_known_hosts:1
```

Utilisation de jeton Yubikey - certificat client

Dans un premier temps, il faut extraire la partie publique de la clef stockée sur une Yubikey:

```
$ ssh-keygen -D /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so -e
  > ./user3/user3.ssh.key.pub
```

Cette clef est utilisée pour générer et signer un certificat :

```
$ ssh-keygen -V +1D -s ./ca/ca.ssh.key -I 'user3 authentication SSH cert'
  -n user3 ./user3/user3.ssh.key.pub
```

Nous ajoutons user3 à la liste des principal pouvant s'authentifier avec certificat SSH, puis nous utilisons le certificat généré en combinaison avec la clef privée stockée sur la Yubikey pour s'authentifier au serveur :

```
 $ ssh -l user3 -I /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so 127.0.0.1
Enter PIN for 'PIV_II (PIV Card Holder pin)'~:
```

Coté serveur, nous avons la preuve que l'authentification s'est faite via certificat :

```
Accepted publickey for user3 from 127.0.0.1 port 36994 ssh2: RSA-CERT ID
user3 authentication SSH cert (serial 0) CA ED25519
SHA256:nvpSZANW1GE1Lq5v+Jxs7G0rmovZKjN4sAcqvGZXOEA
```

Retour sur l'usage des certificats OpenSSH

Depuis l'an dernier, plusieurs entités ont décidé de mettre à jour une partie de leurs infrastructures basées sur une authentification par certificat OpenSSH. En premier lieu, Facebook a publié un article sur leur usage interne des certificats comme alternative lors de crash de l'infrastructure LDAP/kerberos ainsi que la séparation de leur infrastructure en différent domaine de sécurité. De leur côté les ingénieurs de Netflix ont créé un projet nommé Bless qui permet de signer automatiquement des certificats SSH (dans la même idée que Let's encrypt).

Conclusion

Cet article a permis d'aborder différentes méthodes supportées par OpenSSH pour authentifier les utilisateurs et les serveurs, et en particulier l'authentification par certificat (avec usage de périphérique externe ou non).

Le protocole SSHFP (qui permet la publication des clefs publiques des CA), a été volontairement écarté en raison du faible niveau de sécurité du DNS, protocole sur lequel il s'appuie. Bien évidemment, dans tous les cas, il convient d'utiliser des algorithmes de confiance, avec des clefs de tailles adaptées ainsi que des formats sûrs (typiquement Encrypt-Then-Mac).

Documentation

- https://www.openssh.com/
- https://www.openbsd.com/
- https://www.ietf.org/rfc/rfc4250.txt
- https://www.ietf.org/rfc/rfc4251.txt
- https://www.ietf.org/rfc/rfc4252.txt
- https://www.ietf.org/rfc/rfc4253.txt
- https://www.ietf.org/rfc/rfc4254.txt
- https://www.openssh.com/specs.html
- https://www.openssh.com/txt/release-7.5
- https://www.ssi.gouv.fr/administration/guide/recommandations-pour-un-usage-securise-dopenssh/
- https://www.youtube.com/watch?v=pY4FBGI7bHM
- https://kyleisom.net/articles/ssh_keys.html
- https://code.facebook.com/posts/365787980419535/scalable-and-secure-access-with-ssh/
- https://github.com/Netflix/bless

2 - Rappel des avis émis

Dans la période du 22 au 28 mai 2017, le CERT-FR a émis les publications suivantes :

- CERTFR-2017-AVI-162 : Multiples vulnérabilités dans le noyau Linux de SUSE
- CERTFR-2017-AVI-163 : Multiples vulnérabilités dans Cisco Integrated Management Controller
- CERTFR-2017-AVI-164 : Vulnérabilité dans F5 BIG-IP Azure cloud
- CERTFR-2017-AVI-165 : Vulnérabilité dans Samba

Gestion détaillée du document

29 mai 2017 version initiale.