Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

PREMIER MINISTRE

Secrétariat général
de la défense
et de la sécurité nationale

*Agence nationale de la sécurité
des systèmes d'information*

Paris, le 17 août 2015

N° DAT-NT-007/ANSSI/SDE/NP

Nombre de pages du document
(y compris cette page) : 22

Note technique

Recommandations pour un usage sécurisé d'(Open)SSH

Public visé :

Développeur	
Administrateur	✓
RSSI	✓
DSI	
Utilisateur	

Informations

Avertissement

Ce document rédigé par l'ANSSI présente les **« Recommandations pour un usage sécurisé d'(Open)SSH »**. Il est téléchargeable sur le site www.ssi.gouv.fr/nt-ssh. Il constitue une production originale de l'ANSSI. Il est à ce titre placé sous le régime de la « Licence ouverte » publiée par la mission Etalab (www.etalab.gouv.fr). Il est par conséquent diffusable sans restriction.

Ces recommandations sont livrées en l'état et adaptées aux menaces au jour de leur publication. Au regard de la diversité des systèmes d'information, l'ANSSI ne peut garantir que ces informations puissent être reprises sans adaptation sur les systèmes d'information cibles. Dans tous les cas, la pertinence de l'implémentation des éléments proposés par l'ANSSI doit être soumise, au préalable, à la validation de l'administrateur du système et/ou des personnes en charge de la sécurité des systèmes d'information.

Personnes ayant contribué à la rédaction de ce document :

Contributeurs	Rédigé par	Approuvé par	Date
BSS, BAS, LAM	BSS	SDE	17 août 2015

Évolutions du document :

Version	Date	Nature des modifications
1.0	6 avril 2013	Version initiale
1.1	15 avril 2013	Corrections de forme
1.2	21 janvier 2014	Compléments
1.3	17 août 2015	Mise à jour charte graphique

Pour toute remarque :

Contact	Adresse	@mél	Téléphone
Bureau Communication de l'ANSSI	51 bd de La Tour-Maubourg 75700 Paris Cedex 07 SP	communication@ssi.gouv.fr	01 71 75 84 04

Table des matières

1 Préambule

1.1 Quelques rappels historiques.

L'apparition des premiers Unix et systèmes d'information communicants s'est accompagnée de l'émergence de piles protocolaires visant l'échange de données entre machines comme FTP, TELNET ou encore RSH.

Bien qu'encore largement utilisés aujourd'hui, ces protocoles n'ont pas été conçus pour être sécurisés ; leurs fonctionnalités sont particulièrement pauvres lorsqu'il s'agit d'authentifier la source ou l'émetteur, ou encore de garantir l'intégrité et la confidentialité des flux.

Leur usage est même devenu problématique d'un point de vue filtrage. FTP nécessite par exemple une ouverture dynamique de port sur une passerelle utilisant du NAT.

Pour ces raisons, le besoin d'un protocole applicatif sécurisé capable de remplacer ces briques logicielles s'est fait rapidement sentir : SSH est né.

1.2 Objectifs de cette note :

Couramment utilisé pour l'administration distante, le transfert de fichiers, les redirections et encapsulations de flux sensibles, OpenSSH est devenu un élément incontournable d'un grand nombre de systèmes d'information.

Il est donc vital d'en maîtriser sa configuration, de durcir son installation et d'appliquer des règles d'hygiène strictes pour son exploitation. Tout cela se traduit par les recommandations présentées dans ce document.

1.3 À qui s'adresse ce document ?

Cette note technique s'adresse aux intégrateurs et administrateurs système et réseaux, soucieux d'installer et d'administrer correctement un parc grâce au protocole SSH et à son implémentation de référence : OpenSSH.

2 Présentation de SSH

SSH, ou Secure SHell, est un protocole applicatif (couche 7 du modèle de l'OSI) qui vise à corriger les déficiences connues dans les protocoles FTP, RSH, RCP et TELNET, au travers de 3 sous-protocoles :

- SSH-USERAUTH, protocole d'authentification de la partie client – RFC 4252 ;
- SSH-TRANS, protocole de transfert sécurisé, permettant l'authentification du serveur et l'établissement d'un canal de communication sécurisé (confidentialité et intégrité) – RFC 4253 ;
- SSH-CONNECT, protocole de connexion permettant le multiplexage de canaux de communication (données et commandes) – RFC 4254.

2.1 Le protocole SSH

SSH est standardisé par une série de RFC (4250 à 4254) qui spécifient ses protocoles de communication et les mécanismes cryptographiques qu'il doit supporter.

SSH existe aujourd'hui en 2 versions : SSHv1 et SSHv2. SSHv1 présente des vulnérabilités structurelles qui ont été corrigées dans la version suivante. La version 1 du protocole est maintenant obsolète.

R1

Seule la version 2 du protocole SSH doit être autorisée.

SSH est avant tout un protocole de communication. Ce n'est pas un shell Unix, ni un terminal, ni un interpréteur de commandes.

Son implémentation la plus couramment rencontrée aujourd'hui est OpenSSH. Plus qu'un protocole, OpenSSH est une suite d'outils offrant de nombreuses fonctionnalités.

L'obligation d'utiliser la version 2 du protocole avec OpenSSH se fait avec la directive `Protocol` de `sshd_config` :

```
Protocol 2
```

2.2 Présentation d'OpenSSH

OpenSSH est développé et maintenu par le projet OpenBSD. C'est à ce jour l'implémentation de référence du protocole SSH que l'on retrouve sur un grand nombre de systèmes, aussi bien des serveurs (Unix, GNU/Linux, Windows) que des postes clients ou des équipements réseau.

Cette suite logicielle est composée de nombreux outils :

- un serveur, `sshd`.
- plusieurs clients, suivant les usages :
  - connexion shell distante : `ssh` ;
  - transfert et téléchargement de fichiers : `scp`, `sftp` ;
- un outil de génération de clés, `ssh-keygen` ;
- un service de trousseau de clés, `ssh-agent` et `ssh-add` ;
- un scanner de clés publiques présentes sur les serveurs SSH, `ssh-keyscan`.

3 Cas d'usage

3.1 Administration à distance en ligne de commande

L'administration à distance en ligne de commande est le cas d'usage le plus répandu de SSH. Il consiste à se connecter à une machine distante et à lancer une session shell une fois les opérations d'authentification réussies.

L'avantage évident apporté par SSH est sa sécurité. Là où `telnet` n'apporte ni authentification du serveur ni création d'un canal chiffré et authentifié, SSH va permettre de le faire dès lors que quelques règles d'hygiène simples sont appliquées.

R2

SSH doit être utilisé en lieu et place de protocoles historiques (TELNET, RSH, RLOGIN) pour des accès shell distants.

R3

Les serveurs d'accès distants TELNET, RSH, RLOGIN doivent être desinstallés du système.

3.2 Transfert, téléchargement de fichiers

Le deuxième cas d'usage relativement fréquent de SSH est le transfert de fichier, à la fois dans le sens montant (client vers serveur) et descendant (serveur vers client).

Pour ce faire, SSH propose deux mécanismes : SCP et SFTP. SFTP est plus élaboré que SCP, et permet une navigation dans une arborescence là où SCP se borne à permettre de transférer des données. Dans les deux cas, la sécurité repose essentiellement sur SSH qui fournit le canal de communication.

D'un point de vue réseau, l'usage de SCP/SFTP facilite l'établissement des règles de filtrage au travers de passerelles. SSH permettant de multiplexer le canal de données et celui de contrôle dans la même connexion, le flux ne requiert pas une ouverture dynamique de port à la différence d'un protocole comme FTP.

R4

SCP ou SFTP doivent être utilisés en lieu et place de protocoles historiques (RCP, FTP) pour du transfert ou du téléchargement de fichiers.

3.3 Redirection de flux

La redirection de flux est une fonctionnalité couramment utilisée via SSH. Elle consiste à encapsuler des flux TCP/IP directement dans le tunnel SSH, pour permettre (entre autres) de sécuriser le transport d'un protocole non sécurisé, ou de donner accès à des services qui sont *a priori* protégés derrière une passerelle.

Ces redirections peuvent intervenir sur le client SSH (Figure 1) ou sur le serveur (Figure 2).

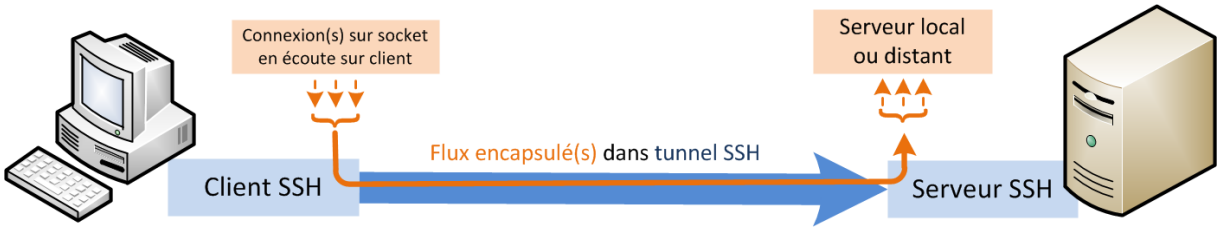

FIGURE 1 – Schéma d'une redirection sur le client SSH

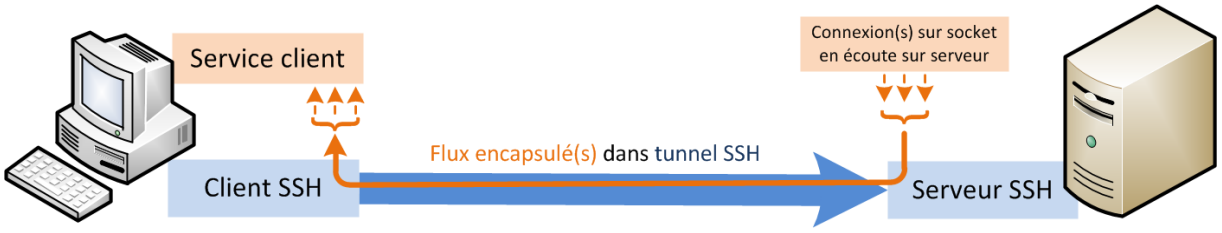

FIGURE 2 – Schéma d'une redirection sur le serveur SSH

R5

La mise en place de tunnels SSH doit être strictement réservée aux protocoles n'offrant pas de mécanismes de sécurité robustes et pouvant en tirer un bénéfice relatif (par exemple : X11, VNC).
Cette recommandation n'exclut pas l'usage de protocoles de sécurité plus bas niveau supplémentaires comme IPSEC (cf. "Recommandations de sécurité relatives à IPsec" sur `www.ssi.gouv.fr`).

4 Bonnes pratiques d'exploitation OpenSSH

Les recommandations suivantes s'appliquent aux outils et services OpenSSH.

4.1 Cryptographie

4.1.1 Authentification

SSH repose très largement sur la cryptographie asymétrique pour l'authentification.
Ne pas s'assurer de l'authenticité du serveur peut avoir de nombreux impacts sur la sécurité :

- impossibilité de vérifier que l'on communique bien avec le bon serveur (risque d'usurpation) ;
- exposition aux attaques de "l'homme du milieu", permettant de récupérer l'ensemble des données du flux (frappes clavier, éléments affichés, logins, mots de passe, fichiers lus ou édités...).

R6

Il faut s'assurer de la légitimité du serveur contacté avant de poursuivre l'accès. Cela passe par l'authentification préalable de la machine au travers de l'empreinte de sa clé publique, ou d'un certificat valide et vérifié.

Avec OpenSSH, ce contrôle se fait de plusieurs façons :
- en s'assurant que l'empreinte de la clé présentée par le serveur est la bonne (obtenue **préalablement** avec `ssh-keygen -l`) ;
- en rajoutant la clé manuellement dans le fichier *known_hosts* ;
- en vérifiant la signature du certificat présenté par le serveur avec une autorité de certification (AC) reconnue par le client (voir IGC, § 4.5).

La validation explicite par l'utilisateur de la clé hôte se spécifie via l'attribut `StrictHostKeyChecking` dans `ssh_config` :

```
StrictHostKeyChecking  ask
```

OpenSSH adopte par défaut un modèle de sécurité *Trust On First Use* (TOFU) : lors de la première connexion et à défaut de pouvoir authentifier l'hôte, `ssh` demande confirmation à l'utilisateur qu'il s'agit bien de la bonne clé (via son empreinte). Si l'utilisateur confirme que l'empreinte est bonne, `ssh` procèdera à son enregistrement dans le fichier *known_hosts* afin de permettre sa vérification lors des visites suivantes.

4.1.2 Génération de clés - tailles et algorithmes

Il existe plusieurs algorithmes et tailles de clés exploitables par SSH. Tous ne sont pas en mesure aujourd'hui de répondre aux critères de sécurité du RGS.

Dans la pratique, notons que les clés SSH «hôte» (celles permettant d'authentifier un serveur SSH) sont peu renouvelées. Il est donc important de choisir dès le départ des clés de taille suffisamment grande.

L'implémentation DSA présente dans OpenSSH est limitée à une taille de clé de 1024 bits, qui est de taille insuffisante.

R7

L'usage de clés DSA n'est pas recommandé.

Afin de supprimer l'utilisation de clés DSA :
- client `ssh` : supprimer les fichiers `~/.ssh/id_dsa` et `~/.ssh/id_dsa.pub` ;
- serveur `sshd` : mettre en commentaires les lignes `HostKey` pointant vers une clé DSA (comme `/etc/ssh/ssh_host_dsa_key`).

Les recommandations suivantes s'appliquent pour une durée de vie estimée de 3 ans de l'hôte [1].

R8

La taille de clé minimale doit être de 2048 bits pour RSA.

R9

La taille de clé minimale doit être de 256 bits pour ECDSA.

Ces tailles suivent les directives données dans le chapitre 2.2 de l'annexe B1 du RGS, «Cryptographie asymétrique».

1. le RGS fixe la durée de vie à 3 ans pour les certificats électroniques de services applicatifs dans son Annexe A.

Des clés respectant ces exigences peuvent être générées par les commandes suivantes :

```
ssh-keygen -t rsa   -b 2048 -f <fichier de clé RSA>
ssh-keygen -t ecdsa -b 256  -f <fichier de clé ECDSA>
```

`ssh-keygen` génèrera deux fichiers : clé privée et clé publique (nom de fichier terminé en `.pub`).

Ce sont ces fichiers qui sont ensuite utilisés comme clé d'identification hôte (attribut `HostKey` de `sshd`), ou utilisateur (attribut `IdentityFile` de `ssh`).

R10

Lorsque les clients et les serveurs SSH supportent ECDSA, son usage doit être préféré à RSA.

4.1.3 Génération de clés - qualité de l'aléa

La qualité des clés est un facteur important de robustesse. Celle-ci dépendant directement de la source d'aléa utilisée lors de l'étape de génération, fournir une source d'aléa correcte est donc cruciale. Cela fait défaut à bon nombre d'équipements et de machines, notamment sur des composants industriels sommaires (SCADA, terminaux...) ou des machines virtuelles.

R11

Les clés doivent être générées dans un contexte où la source d'aléa est fiable, ou à défaut dans un environnement où suffisamment d'entropie a été accumulée.

Cette recommandation est d'autant plus importante qu'il est commun de générer les clés hôtes `sshd` par des scripts lors du premier démarrage du service. Dans ce cas, il faut prévoir un mécanisme de changement de clés et les remplacer.

R12

Quelques règles permettent de s'assurer que le réservoir d'entropie est correctement rempli :

- la machine de génération de clés doit être une machine physique ;
- elle doit disposer de plusieurs sources d'entropie **indépendantes** ;
- l'aléa ne doit être obtenu qu'après une période d'activité suffisamment importante (plusieurs minutes voire heures).

Les machines virtuelles reposant essentiellement sur des périphériques virtuels, leurs interruptions sont elles aussi virtualisées. Vu que celles-ci peuvent être utilisées comme source d'entropie, le fait de les virtualiser conduit à un déterminisme dans leurs déclenchements, d'où une baisse d'entropie.

Le choix de sources indépendantes permet de s'assurer que le biais d'une des sources ne pourra affecter les autres.

Pour des raisons évidentes, les systèmes nouvellement démarrés ne peuvent avoir accumulés suffisamment d'entropie. Il est donc recommandé d'obtenir de l'aléa une fois qu'une période d'activité suffisante s'est écoulée.

Dans le cas où le système d'information dispose d'une IGC, celle-ci peut être utilisée pour obtenir des clés compatibles avec OpenSSH du moment que le format des clés privées est celui manipulé par OpenSSL.

Les clés privées générées par OpenSSL peuvent être directement utilisées par `ssh-keygen` et produire la clé publique correspondante :

```
# Exemple: génération d'une clé privée protégée par mot de passe
openssl genrsa -aes128 -passout stdin -rand /dev/urandom 2048 > <clé privée>

# Import dans ssh-keygen pour obtenir la clé publique compatible OpenSSH
ssh-keygen -y -f <clé privée> > <clé publique>
```

Depuis la version 6.1, OpenSSH est capable de lire différents types d'encodage de clés publiques. Cela permet notamment d'importer des clés distribuées par certificats X.509 sans nécessiter d'accès à la clé privée, ou d'utiliser des fichiers au format PKCS12 comme facteur d'authentification.

Les clés publiques sont importées via `ssh-keygen` :

```
# Conversion d'une clé encodée en PEM vers une clé publique OpenSSH
ssh-keygen -i -m PKCS8 -f <clé publique>

# Récupération clé publique d'un certificat X.509 et import OpenSSH
openssl x509 -pubkey -noout -in <certificat> | ssh-keygen -i \
    -m PKCS8 -f /dev/stdin

# Récupération clé privée et obtention de la clé publique OpenSSH
# correspondante à partir d'un fichier PKCS12
openssl pkcs12 -in <fichier PKCS12> -nocerts -aes128 > <clé privée>
openssl pkcs12 -in <fichier PKCS12> -nocerts -nodes | ssh-keygen -y \
    -f /dev/stdin > <clé publique>
```

4.1.4 Droits d'accès et diffusion des clés

Les clés d'authentification SSH peuvent être regroupées selon deux rôles :
- celles utilisées pour l'authentification d'utilisateurs ;
- celles utilisées pour l'authentification d'un hôte/serveur.

Le service d'authentification repose sur de la cryptographie asymétrique. À ce titre, la règle d'usage suivante s'y applique :

R13

La clé privée ne doit être connue que de l'entité qui cherche à prouver son identité à un tiers et éventuellement d'une autorité de confiance. Cette clé privée doit être dûment protégée pour en éviter la diffusion à une personne non autorisée.

Les mesures de protection applicables vont légèrement varier suivant qu'il s'agisse d'une clé hôte ou d'une clé utilisateur.

Une clé privée d'authentification hôte ne doit être lisible que par le service `sshd`. Sous Unix/Linux, cela signifie qu'elle n'est lisible que par `root` :

```
-rw------- root:root /etc/ssh/ssh_host_rsa_key
-rw------- root:root /etc/ssh/ssh_host_ecdsa_key
```

Une clé privée d'authentification utilisateur ne doit être lisible que par l'utilisateur auquel elle est associée. Sous Unix/Linux, cela signifie :
- que la clé privée n'est lisible que par l'utilisateur ;
- qu'elle est protégée par un mot de passe connu seulement de l'utilisateur (voir l'option `-p` de `ssh-keygen`).

Le risque d'une utilisation frauduleuse de la clé privée générée pour un utilisateur étant non négligeable, le fichier texte utilisé pour la stocker doit si possible être chiffré afin de la protéger. En cas de perte ou de vol, cela laisse suffisamment de temps aux équipes d'administration pour révoquer l'usage de la biclé associée.

R14

L'AES-128 mode CBC doit être utilisé comme algorithme de chiffrement pour la protection de la clé privée utilisateur par mot de passe.

Notons que dans les versions précédant la 5.3, OpenSSH protège les clés par passphrase via du 3DES (mode CBC), tandis que les versions 5.3 et supérieures utilise l'AES-128 (toujours en mode CBC).

Pour les clés privées protégées via du 3DES, le passage vers de l'AES peut se faire en utilisant une version suffisamment récente de OpenSSH et en demandant une opération de changement de mot de passe (option `-p` de `ssh-keygen`).

Une fois le mot de passe modifié, il faut s'assurer que l'ancien fichier chiffré en 3DES a été correctement brouillé puis effacé.

Côté serveur, les accès au répertoire `~/.ssh/` et son contenu doivent aussi être protégés. Seul l'utilisateur du compte doit posséder les droits d'écriture à cette arborescence.

Le serveur `sshd` doit vérifier la rectitude des modes et droits des fichiers de l'utilisateur avant de le laisser ouvrir une session. Ceci se configure via la directive `StrictModes` du fichier `sshd_config`.

```
StrictModes yes
```

4.1.5 Choix des algorithmes symétriques

Une fois les parties mutuellement authentifiées, le canal de communication est protégé en chiffrement et en intégrité.

R15

L'algorithme de chiffrement doit reposer sur de l'AES 128, AES 192 ou AES 256, en mode CTR.
L'intégrité doit reposer sur du HMAC SHA-1, SHA-256 ou SHA-512.

Le choix des algorithmes étant établi suivant une négociation entre le client et le serveur, la liste des algorithmes doit être fixée des deux côtés. Les algorithmes considérés comme faibles doivent être retirés de la suite.

Le mode CTR permet de s'affranchir de vulnérabilités connues du mode CBC tel qu'utilisé par SSH [2].

Les HMAC SHA-256 et SHA-512 ne sont supportés que depuis la version 5.9 d'OpenSSH. Les systèmes d'exploitation utilisant des versions antérieures doivent alors reposer sur un HMAC SHA-1.

Sous OpenSSH, les directives suivantes doivent être rajoutées dans le `sshd_config` et le `ssh_config` :

```
Ciphers aes256-ctr,aes192-ctr,aes128-ctr
# Pour les versions 6.3+, OpenSSH supporte le ETM (encrypt-then-mac),
# plus sûr que les anciennes implémentations (mac-then-encrypt)
MACs    hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com

# S'il s'agit d'une ancienne version, utilisez uniquement "hmac-sha1"
MACs    hmac-sha2-512,hmac-sha2-256,hmac-sha1
```

4.2 Durcissement système

4.2.1 Durcissement à la compilation

Le service `sshd` est très souvent exécuté sous l'utilisateur `root` car il nécessite des privilèges de super-utilisateur pour pouvoir changer de propriétaire lorsqu'un utilisateur se connecte.

Il convient donc d'en durcir le code en vue de retarder, voire d'empêcher sa compromission.

R16

Une étape préliminaire à ce durcissement est donc d'user de drapeaux de compilation adéquats. Se rapporter par exemple aux "Recommandations de sécurité relatives à un système GNU/Linux".

4.2.2 Séparation de privilèges

La séparation de privilèges permet de limiter les impacts d'une faille en cherchant à respecter le principe de moindre privilège : le serveur va réduire ses droits au strict nécessaire.

Depuis la version 5.9, OpenSSH supporte des mécanismes de confinement plus avancés tels SECCOMP (Linux) ou systrace (OpenBSD). Il est vivement recommandé de les utiliser s'ils sont disponibles.

La séparation de privilèges doit être activée dans le fichier de configuration `sshd_config`. Préférez utiliser le sandboxing lorsque celui-ci est disponible :

```
# Si "sandbox" n'est pas utilisable, mettre "yes" en remplacement
UsePrivilegeSeparation sandbox
```

2. «Plaintext Recovery Attacks Against SSH», Martin R. Albrecht, Kenneth G. Paterson et Gaven J. Watson.

4.2.3 SFTP et Chroot

Comme déjà indiqué, `sshd` permet aussi de fournir un serveur SFTP qui peut être utilisé pour le téléchargement ou l'envoi de fichiers. Ce service ne requiert pas d'accès shell.

Tous les utilisateurs (ou groupe) n'exploitant que le service SFTP doivent être chrootés dans une partition à part du système, au travers de l'option `ChrootDirectory` :

```
# Pour un utilisateur: "Match user <login/utilisateur>", ou
# Pour un groupe: "Match group <groupe>"
Match group sftp-users
    ForceCommand internal-sftp
    ChrootDirectory /sftp-home/\%u
    ...
```

Si l'option `group` est utilisée, les utilisateurs doivent être membre du groupe système en question (ici, `sftp-users`).

L'utilisation d'une arborescence indépendante permet de s'assurer que :

- les utilisateurs du service SFTP sont isolés dans une arborescence fichiers et n'ont pas de visibilité ou d'accès sur le reste ;
- la volumétrie n'est pas partagée avec d'autres partitions et évite ainsi les risques de saturation.

Lorsque le partitionnement le permet, la partition associée à cette cage chroot doit être montée en `nodev`, `noexec`, `nosuid`. Pour que la cage chroot soit efficace, la séparation de privilège (voir 4.2.2) doit être activée.

4.3 Authentification et contrôle d'accès utilisateur

4.3.1 Authentification d'un utilisateur

Plusieurs mécanismes peuvent être utilisés pour permettre l'authentification d'un utilisateur.

R17

L'authentification d'un utilisateur doit se faire à l'aide d'un des mécanismes suivants, par ordre de préférence :

- par cryptographie asymétrique ECDSA ;
- par cryptographie asymétrique RSA ;
- par cryptographie symétrique (tickets Kerberos par la GSSAPI) ;
- PAM (ou BSD Auth) permettant de faire appel à des modules d'authentification tiers n'exposant pas le mot de passe utilisateur ou son condensat (OTP) ;
- par mot de passe vis à vis d'une base de données comme `passwd/shadow` ou annuaire.

Note : Dans tous les cas, si l'utilisateur a des privilèges élevés sur le système, l'usage du mot de passe est à proscrire.

Les mécanismes de cryptographie asymétrique (ECDSA, RSA) sont directement fournis par la suite d'outils d'OpenSSH. Un utilisateur dont la clé publique est renseignée dans le fichier `authorized_keys` d'un compte aura accès à celui-ci.

Les comptes utilisateurs possèdent toujours des droits sur le système, aussi minimes soient-ils. Il est donc important de protéger leurs accès par un mécanisme d'authentification et d'empêcher autant

que possible le passage en force brute.

R18

Les droits d'un utilisateur doivent suivre le principe de moindre privilège. La restriction peut porter sur de nombreux paramètres : commandes accessibles, adresses IPs d'origine, droit de redirections ou de forwarding. . ..

Quand les droits ne peuvent être appliqués directement au niveau du serveur (via `sshd_config`), le fichier `authorized_keys` permet de spécifier un ensemble d'options qui restreignent les droits associés à une clé donnée :

```
# La clé n'autorise l'accès que si le client provient du réseau 192.168.15/24
from="192.168.15.*" ssh-ecdsa AAAAE2Vj...
# La clé autorise l'accès pour un client venant du domaine ssi.gouv.fr sauf
# pour le domaine public.ssi.gouv.fr
from="!*.public.ssi.gouv.fr, *.ssi.gouv.fr" ssh-ecdsa AAAAE2Vj...
# Bloque l'usage de l'agent forwarding pour cette clé
no-agent-forwarding ssh-ecdsa AAAAE2Vj...
```

Certains modules PAM, BSD Auth. . . requièrent l'envoi en clair vers le serveur du mot de passe. Seul son transport sur le réseau est protégé par le canal créé par SSH. En cas de compromission du serveur, le mot de passe peut ainsi être réutilisé vers d'autres services.

Bien que la GSSAPI et le module PAM `pam_krb5` reposent sur les mêmes mécanismes, utiliser le module `pam_krb5` ne permet pas de profiter des avantages de Kerberos (protection du mot de passe vis-à-vis d'un service tiers) : la GSSAPI exploite directement les tickets cryptographiques Kerberos, alors que le module PAM se contente de soumettre le mot de passe de l'utilisateur auprès du KDC une fois que celui-ci a été reçu, en clair, par le serveur. `pam_krb5` ne doit donc pas être utilisé.

L'accès aux comptes sans mot de passe doit être proscrit dans `sshd_config`. L'opération d'authentification doit être d'une durée relativement courte et le nombre de tentatives doit être limitée à une par connexion.

```
PermitEmptyPasswords no
# Attention, MaxAuthTries doit être strictement supérieur à 1.
# De plus, un chiffre trop faible peut poser souci.
MaxAuthTries 2
LoginGraceTime 30
```

4.3.2 Agent d'authentification

Il est assez fréquent pour un utilisateur de rebondir successivement d'un hôte SSH vers un autre hôte SSH.

L'utilisateur va se heurter à un problème d'authentification lors des rebonds après le premier : la machine hôte relais n'ayant pas connaissance des éléments secrets de l'utilisateur, celui-ci ne pourra plus s'authentifier aux autres hôtes à moins de fournir le secret lui-même directement (exemple : saisie directe de mot de passe).

OpenSSH fournit un utilitaire (`ssh-agent`) permettant de garder en mémoire les clés privées d'un utilisateur lorsque celui-ci s'authentifie par des mécanismes cryptographiques asymétriques comme RSA ou ECDSA.

La redirection de l'agent d'authentification (ou *Agent Forwarding*) permet à un hôte relais de rediriger les requêtes d'authentification du second serveur vers le poste client où s'exécute l'agent qui se chargera de l'opération d'authentification.

R19

L'usage du mécanisme d'agent forwarding (option `-A` de `ssh`) est recommandé dans les cas où un rebond SSH est nécessaire au travers d'un serveur hôte relais.

Ce mécanisme permet de conserver la clé privée utilisateur hors d'atteinte du serveur relais tout en bénéficiant des avantages de la cryptographie par clé publique.

R20

Le serveur hôte relais doit être un hôte de confiance.

Bien que la clé privée ne soit jamais exposée au serveur hôte relais, celui-ci a la possibilité de communiquer directement avec l'agent afin de lancer des opérations d'authentification.

La compromission de l'hôte relais expose donc l'agent à des requêtes d'authentification non contrôlées par l'utilisateur.

Afin de minimiser l'exposition des clés à des requêtes d'authentification non légitimes, celles-ci peuvent être chargées avec les options suivantes dans le trousseau (via `ssh-add`) :

`-c` demande la confirmation de l'usage d'une clé du trousseau auprès de l'utilisateur par un programme graphique externe (comme `ssh-askpass`) ;

`-t <s>` la clé sera automatiquement retirée du trousseau après `<s>` secondes.

4.3.3 Imputabilité des accès

L'imputabilité des accès à un serveur est essentielle pour la traçabilité.

R21

Chaque utilisateur doit disposer de son propre compte, unique, incessible.

`root` est un exemple de compte générique que l'on retrouve sur tous les systèmes Unix/Linux, et utilisé comme compte d'administration.
Le compte `root` ne doit pas être accessible directement à un utilisateur :

```
PermitRootLogin no
```

Le possession d'un compte dédié pour chaque utilisateur permet une gestion plus fine des accès et une meilleure traçabilité. Un utilisateur ayant connaissance que son compte lui est dédié peut être plus vigilant sur son usage.

La directive `PrintLastLog` du `sshd_config` permet de spécifier au serveur qu'il doit afficher les informations de dernière connexion à l'utilisateur lorsqu'il se connecte.

```
PrintLastLog yes
```

4.3.4 AllowUsers, AllowGroups

R22

Les accès à un service doivent être restreints aux utilisateurs qui en ont un besoin justifié. Cette restriction doit s'appliquer en droits positifs : uniquement ceux explicitement autorisés ont le droit de se connecter en SSH sur un hôte, et éventuellement en provenance d'adresses IP spécifiées.

Sous OpenSSH, la directive `AllowUsers` permet de spécifier la liste des utilisateurs autorisés à se connecter au service SSH. Dans le cas où un groupe d'utilisateurs est concerné (administrateurs, utilisateurs avec pouvoir...), utiliser `AllowGroups`.

OpenSSH offre beaucoup de fonctionnalités qui peuvent poser des problèmes de sécurité lorsqu'elles sont exploitées à des fins malveillantes : accès shell, redirections, envoi/téléchargement de binaires... Il faut donc restreindre les accès comptes au strict nécessaire.

Dans la mesure où `sshd` doit être exposé à des adresses IPs qui ne sont pas exclusives à un réseau d'administration, ces directives permettent de restreindre les accès utilisateurs suivant leur adresse IP :

```
# Restreint 'admin' aux adresses IPs d'un réseau d'administration
# (192.168.17/24) et 'user' aux adresses d'un réseau Intranet (10.127/16)
AllowUsers admin@192.168.17/24 user@10.127/16
# Même chose, mais avec un groupe 'wheel' et 'users'
AllowGroups wheel@192.168.17/24 users@10.127/16
```

4.3.5 Restriction de l'environnement utilisateur

Les variables d'environnement peuvent grandement modifier le fonctionnement attendu de programmes. Certaines peuvent avoir des répercussions sur la sécurité, notamment quand elles altèrent la configuration ou l'exécution (`LD_PRELOAD`, `LD_LIBRARY_PATH`...).

R23

L'altération de l'environnement par un utilisateur doit être bloquée par défaut. Les variables autorisées à la modification par le client doivent être sélectionnées au cas par cas.

Bloquer la modification de l'environnement par le service `sshd` se configure via la directive `PermitUserEnvironment` de `sshd_config` :

```
PermitUserEnvironment no
```

L'autorisation à la modification d'une variable par un client est quant à elle faite au travers de la directive `AcceptEnv`.

Une autre mesure efficace de restriction d'un environnement utilisateur est la directive `ForceCommand` qui, comme son nom l'indique, *force* l'exécution d'une commande lorsqu'un utilisateur se connecte au système sans que celui-ci puisse la contourner.

R24

Les commandes lancées par un utilisateur au travers d'une session SSH doivent être réduites au strict nécessaire ; cette restriction peut être mise en place par l'utilisation :
- de la directive `ForceCommand` pour un utilisateur donné, dans le fichier `sshd_config` ;
- en spécifiant des options dans le fichier `authorized_keys`. Voir section 4.3.1 ;
- de binaires dûment protégés comme `sudo` ou `su`.

La directive `ForceCommand` est notamment commode dans le cadre de commandes SSH appelant des exécutables à distance et que l'on souhaite éviter que la compromission du compte conduise à un accès shell trivial. Elle se destine donc à des activités de monitoring, l'exécution de commandes d'administration simples (redémarrage de services, obtention du statut des services), récupération de résultats de scripts...

Soit un utilisateur *monproc* dédié à un service de monitoring et ne devant permettre que la récupération des processus en cours d'execution sur une machine donnée. Son sshd_config pourra contenir :

```
Match user monproc
ForceCommand /bin/ps -elf
```

A chaque connexion de cet utilisateur, seule la commande `/bin/ps -elf` sera ainsi exécutée.

4.4 Protocole et accès réseau

4.4.1 ListenAddress et Ports

L'usage de SSH pour des opérations d'administration est courant. À ce titre, le service SSH ne doit pas pouvoir être contacté ailleurs qu'au travers du réseau d'administration, idéalement séparé physiquement du réseau opérationnel.

R25

Le serveur SSH doit écouter uniquement sur une adresse d'administration.

Lorsque des contraintes opérationnelles font que le serveur est exposé à un réseau non maîtrisé, des attaques par force brute sur le port par défaut (22) sont fréquentes.

Ces attaques polluent les journaux en plus de présenter un risque pour les comptes avec un mot

de passe faible.

R26

Lorsque le serveur SSH est exposé à un réseau non maîtrisé, il est recommandé de lui mettre un port d'écoute différent du port par défaut (22).
Il faut privilégier un port inférieur à 1024 afin d'empêcher les tentatives d'usurpation par des services non administrateur sur la machine distante.
Sur un réseau maîtrisé, le serveur SSH doit écouter uniquement sur une interface du réseau d'administration, distinct du réseau opérationnel.

Avec OpenSSH, la spécification de l'adresse et du port d'écoute se fait au travers de la directive `ListenAddress` du `sshd_config` :

```
# Ecoute uniquement sur l'adresse A.B.C.D, port 26
ListenAddress A.B.C.D:26
```

4.4.2 AllowTCPForwarding

Comme évoqué précédemment, OpenSSH permet à un utilisateur de rediriger des flux au travers du serveur, aussi bien en provenance du réseau local que du réseau distant.

Cette fonctionnalité peut être utilisée dans le cadre d'attaques par rebonds qui cherchent à accéder à des services a priori inaccessibles (bloqués par une passerelle filtrante, en écoute sur un réseau différent...). En outre, la redirection de flux locaux risque d'exposer un réseau interne à des flux non maitrisés car provenant de l'extérieur.

R27

Sauf besoin dûment justifié, toute fonctionnalité de redirections de flux doit être désactivée :
- au niveau de la configuration du serveur SSH ;
- au niveau du pare-feu local en bloquant les connexions.

Sous OpenSSH, les redirections côté serveur se désactivent au travers de la directive `AllowTcpForwarding` de `sshd_config` :

```
AllowTcpForwarding no
```

4.4.3 X11Forwarding

Dans ses fondements, une redirection X11 ressemble de très près à une redirection de flux distante (Figure 3). C'est une fonctionnalité qui doit donc être désactivée autant que possible.

De façon générale, il est fortement déconseillé d'installer un serveur X sur un hôte ne nécessitant pas d'affichage graphique, de par les nombreuses failles de sécurité du X Window System.

R28

La redirection X11 doit être désactivée sur le serveur.

FIGURE 3 – Schéma d'une redirection X11 via SSH

La desactivation de la redirection X11 se fait via la directive `X11Forwarding` de `sshd_config` :

```
X11Forwarding no
```

Quand une redirection X11 est active (exécution distante d'un client X11), la compromission de l'hôte distant peut conduire à une fuite d'information non maitrisée :

- capture de frappes clavier de l'utilisateur ;
- envoi de signaux non légitimes à des applications [3] ;
- redirection d'affichage...

Dans le cas où une redirection X11 est nécessaire, les clients X11 distants doivent être traités avec méfiance et disposer du moins de privilèges possibles.

le client `ssh` doit utiliser l'option `-X`, et désactiver l'option `ForwardX11Trusted` du fichier de configuration client, `ssh_config` :

```
ForwardX11Trusted no
```

4.5 IGC OpenSSH

4.5.1 Autorité de certification

Depuis la version 5.4, OpenSSH offre une IGC pour faciliter la gestion des clés en simplifiant le nombre de relations à maintenir entre différentes entités (hôtes et utilisateurs).

Toutefois, l'implémentation de cette IGC est spécifique à OpenSSH et ne repose pas sur des standards tels X.509. Sa mise en place consiste à créer une (ou plusieurs) AC, puis à utiliser sa clé privée pour signer des certificats et sa clé publique pour en vérifier la signature.

À titre d'exemple, une AC peut être utilisée afin d'authentifier les hôtes du réseau : une fois la clé publique de l'AC connue du client, il peut l'utiliser pour vérifier la validité de la signature d'un certificat hôte. Cette validation présente l'avantage de ne pas nécessiter d'intervention lourde sur tous les postes clients à chaque nouveau serveur installé (exemple : mise à jour du fichier `known_hosts`).

Il faut noter que les choix de conception d'OpenSSH ne permettent pas d'obtenir une IGC conforme au Référentiel Général de Sécurité (format des certificats non X.509, pas de chaîne de certification

3. par exemple le vol de focus via l'évènement `WM_TAKE_FOCUS`.

possible...). Elle doit donc servir uniquement du MCO/MCS d'un parc de machines "OpenSSH".

R29

Il est recommandé de créer des ACs distinctes lorsque leurs rôles diffèrent. On aura par exemple :
- une qui jouera le rôle d'AC « hôte » ;
- une qui jouera le rôle d'AC «utilisateurs».

Chaque clé privée d'une AC doit être protégée par un mot de passe unique, robuste.

L'IGC OpenSSH ne supporte pas l'établissement de chaîne de certification, avec une autorité racine et plusieurs ACs déléguées. Chaque AC doit se cantonner à un rôle bien particulier.

Une AC OpenSSH se crée comme une entité «utilisateur» : la génération d'une biclé se fait via `ssh-keygen`. Voir section 4.1.2.

Une fois l'AC créée, celle-ci peut être utilisée pour générer et signer des certificats via sa clé privée. Sa clé publique doit être diffusée auprès des entités (hôtes ou utilisateurs) supposées lui faire confiance.

Pour un hôte `sshd`, la clé publique d'une AC signant des certificats utilisateurs est déclarée dans le fichier spécifié via l'attribut `TrustedUserCAKeys` du `sshd_config`.

Pour un utilisateur client `ssh`, la clé publique d'une AC signant des certificats hôtes est déclarée par le marqueur `@cert-authority` dans le fichier `known_hosts`. La ligne sera donc de la forme :

```
# Déclaration de la clé CA valable pour tout hôte
@cert-authority * AAAAB4Nz1...1P4==
```

Il est important de noter que l'AC signe une clé publique qui peut être obtenue par différents moyens de communication. Étant donné que cette clé sera associée à une identité et pourra être accompagnée d'éventuelles options, le lien entre cette identité et la clé doit être validé. La remise seule de la clé ne suffit généralement pas à cette vérification. Par exemple une rencontre face à face, un message avec une signature cryptographique ou une remise d'empreinte par un canal de communication tiers sécurisé sont des mesures permettant de conserver un lien entre la clé et l'identité de son propriétaire pour sa certification.

4.5.2 Certificats

Un certificat est constitué de divers éléments dont l'intégrité est garantie par la signature de l'AC. La documentation de `ssh-keygen` en donne une description exhaustive.

Pour un **certificat utilisateur**, les paramètres seront ainsi :
- **identité :** login de l'utilisateur tel qu'il est utilisé pour accéder au serveur ;
- **identifiant :** une chaîne de caractère arbitraire, unique, permettant de stocker des données associée au certificat (numéro de version, nom complet utilisateur...) ;
- **clé publique :** la clé publique de l'utilisateur ;
- **durée de validité :** 3 ans.

S'agissant du **certificat hôte**, nous aurons :

- **identité :** le ou les noms d'hôtes utilisés pour la connexion, séparés par des virgules ;
- **identifiant :** chaîne de caractère arbitraire, unique ;
- **clé publique :** la clé publique de l'hôte ;
- **durée de validité :** 3 ans.

La durée de validité des «certificats électroniques de personnes» et des «certificats électroniques de services applicatifs» est fixée par l'annexe A du RGS.

L'étape de génération puis de signature d'un certificat se fait via `ssh-keygen` :

```
ssh-keygen -s <clé privée CA> -I <identifiant> \
     -n <identité1,identité2,...> -V +156w <fichier clé publique>
```

S'il s'agit d'un hôte, y ajouter l'option `-h`.

Une fois généré et signé, le certificat est enregistré dans un fichier se terminant par `-cert.pub`. Ce fichier peut être transféré :
- pour un hôte : chemin spécifié via l'attribut `HostCertificate` de `sshd_config` ;
- pour un utilisateur : dans son répertoire `~/.ssh/`.

4.5.3 Révocation

La révocation est un processus qui peut concerner aussi bien des clés hôtes que des clés utilisateurs, et des ACs. Une fois une clé révoquée, celle-ci ne peut plus être utilisée pour authentifier des entités.

R30

Dans le cas où une clé ne peut plus être considérée comme sûre, l'usage de celle-ci doit être rapidement révoqué au niveau de SSH.

Le mécanisme de révocation choisi par OpenSSH repose sur des fichiers textes à plat. La suite logicielle ne fournit pas d'outils ou de protocoles permettant de synchroniser de listes de révocation, les opérateurs sont tenus de mettre en place les procédures nécessaires au maintien de ces fichiers.

Au niveau d'un hôte `sshd`, les clés utilisateur révoquées sont contenues dans le fichier pointé par l'attribut `RevokedKeys` de `sshd_config`.
Au niveau d'un utilisateur client `ssh`, les clés hôtes révoquées sont déclarées avec le marqueur `@revoked` dans le fichier `known_hosts`.

4.6 Enregistrements DNS

Les zones DNS peuvent stocker des empreintes d'hôtes SSH via les enregistrements `SSHFP`.

Les enregistrer dans le DNS permet de s'appuyer sur un système tiers qui ajoute une validation *a priori* de l'empreinte : le client `ssh` cherchera à obtenir l'empreinte de la clé hôte par une requête auprès du serveur de nom.

DNS n'étant pas un protocole sécurisé (sauf à déployer DNSSEC), l'enregistrement de l'empreinte obtenu n'est qu'un indicateur. Cette méthode de vérification est bien plus fragile qu'une méthode reposant sur des mécanismes cryptographiques fiables, mais présente l'avantage d'être plus simple à déployer.

Il permet de s'assurer que deux entités, a priori distinctes, fournissent les mêmes identifiants de l'hôte : un décalage entre l'empreinte de la clé fournie par l'hôte `sshd` et celle du DNS doit être vu comme un incident de sécurité critique.

OpenSSH implémente la RFC 4255 spécifiant la publication d'empreintes SSH au travers des enregistrements DNS.
L'enregistrement peut s'obtenir via `ssh-keygen` :

```
ssh-keygen -r <hostname> -f <fichier clé publique>
```

La vérification de l'empreinte doit ensuite être activée auprès du client `ssh` via l'attribut `VerifyHostKeyDNS` du `ssh_config` :

```
VerifyHostKeyDNS  ask
```

R31

La validation des empreintes SSH hôte envoyées par le serveur DNS ne doit pas se faire sans une vérification complémentaire.