Sécuriser ses connexions avec OpenSSH

Olivier Hoarau (olivier.hoarau@funix.org)

V2.8 du 24 décembre 2016

1 Historique

V2.8 24.12.16 passage à OpenSSL 1.1.0c et OpenSSH 7.4p1

V2.7 15.10.15 passage à OpenSSL 1.0.2d et OpenSSH 7.1p1, modifications suite passage à Mageia5 et systemd, suppression des clients windows (section obsolète)

V2.6 03.11.08 Passage à OpenSSL 0.9.8i et OpenSSH 5.1p1

V2.5 01.04.07 Passage à OpenSSL 0.9.8e et OpenSSH 4.6p1

V2.4 02.11.06 Passage à OpenSSL 0.9.8d et OpenSSH 4.4p1

V2.3 26.07.06 Installation sous ubuntu, passage à OpenSSL 0.9.8b

V2.2 30.04.06 Passage à OpenSSH 4.3p2 et OpenSSL 0.9.8

V1.9 27.11.04 Passage à OpenSSH 3.9p1 et OpenSSL 0.9.7e

V1.7 06.01.04 Passage à OpenSSH 3.7.1p2 et OpenSSL 0.9.7c

V1.6 15.09.02 Passage à OpenSSH 3.4p1

V1.5 07.07.02 Passage à OpenSSH 3.2.3p1, OpenSSL 0.9.6d

V1.4 16.12.01 passage à 3.0.1p1, modification de certains paramètres des fichiers de conf, mise à jour des clients windows

V1.3 11.3.01 Passage à la version 2.5.1pl1, rajout paragraphe client SSH2 pour windows

V1.2 10.12.00 Correction du problème de Xforwarding

V1.1 3.12.00 Passage à 2.3.0pl1

V1.0 20.8.00 Création du document

2 Préambule

Ce document présente **OpenSSH** qui est une alternative sécurisée à des outils de connexion comme **telne**t.
La dernière version de ce document est téléchargeable à l'URL http://www.funix.org. Ce document peut être reproduit et distribué librement dès lors qu'il n'est pas modifié et qu'il soit toujours fait mention de son origine et de son auteur, si vous avez l'intention de le modifier ou d'y apporter des rajouts, contactez l'auteur pour en faire profiter tout le monde.
Ce document ne peut pas être utilisé dans un but commercial sans le consentement de son auteur. Ce document vous est fourni "dans l'état" sans aucune garantie de toute sorte, l'auteur ne saurait être tenu responsable des quelconques misères qui pourraient vous arriver lors des manipulations décrites dans ce document.

3 Présentation

OpenSSH est une alternative sécurisée à des outils de connexion comme **telnet** (pour lequel le mot de passe circule en clair sur le réseau), mais c'est bien plus que cela puisqu'il permet aussi de lancer des commandes à distance (comme **rsh**, ou **remsh**), mais aussi de transférer des fichiers ou des répertoires entiers (comme **rcp**).
OpenSSH se présente sous la forme d'un daemon et d'un client, le daemon tourne sur un serveur et attend les requêtes des clients **SSH** pour que les utilisateurs distants puissent se connecter sur le serveur.

4 Comment ça marche

On prend comme exemple **olivier** sur **obelix** et **veronique** sur **asterix**. **OpenSSH** repose sur le protocole **SSH**, le protocole a évolué au fil du temps, et on fait la différence entre **SSH1** et **SSH2**, le dernier étant plus récent et quelque peu différent du premier. Avec **OpenSSH** vous pouvez combiner les caractéristiques des deux protocoles. Voilà les différences entre les deux versions.

SSH1

Il y a quatre manières de s'authentifier.

La première méthode va sans doute rappeler à certains les commandes "r" (**rlogin**, **rcp**, **remsh** ou rsh). Si **asterix** est listé dans le fichier /**etc/hosts.equiv** ou **/etc/shots.equiv** d'**obelix**, et qu'il existe un compte **veronique** sur **obelix**, **veronique** pourra se connecter sans problème avec **ssh** sur **obelix**.

Si l'utilisateur **olivier** possède un fichier **~/.shosts** ou **~/.rhosts** contenant:

asterix veronique

veronique pourra se connecter sur le compte d'**olivier** avec **ssh** (en donnant néanmoins le mot de passe d'**olivier**).

Ces deux formes d'authentification ne sont absolument pas sécurisées, Le problème avec cette méthode est que n'importe quelle machine peut se faire passer pour **asterix** (spoofing).

La deuxième méthode est l'authentification toujours sur les fameux fichiers **rhosts** et **hosts.equiv** mais combinée avec une authentification en utilisant les clés **RSA** des machines. C'est à dire qu'on vérifie d'abord **/etc/hosts.equiv**, **/etc/shosts.equiv**,**~/.rhosts** et **~/.shosts** puis ensuite on vérifier si la clé publique du de la machine qui cherche à se connecter se trouve dans le fichier **/etc/ssh_known_hosts** ou **~/.ssh/known_hosts**. Cette méthode est plus

sécurisée car elle évite qu'une machine se fasse passer pour une autre, car à la connexion il va y avoir vérification de la clé publique par rapport à la clé privée de la machine cliente.

La troisième méthode est basée sur l'échange des clés utilisateurs en utilisant le protocole de gestion des clés **RSA**, **veronique** donne sa clé publique à **olivier** , quand elle essaye de se connecter sur le compte d'**olivier**, **SSH** vérifie que la clé publique que détient **olivier** correspond bien à la clé privée de **véronique** . Plus précisément du côté d'**olivier** le serveur **SSH** crypte un nombre aléatoire (un "challenge" dans la doc en anglais) en utilisant la clé publique de **véronique** détenue par **olivier**, seule la clé privée de **veronique** pourra décrypter ce "challenge", celui-ci est envoyé au client **SSH** qui va le décrypter avec la clé privée de **véronique** ce qui va authentifier cette dernière complètement.

Et la dernière méthode consiste à donner le mot de passe du compte sur lequel on se connecte, le mot de passe circule crypté sur le réseau.

Voyons le fonctionnement en détail de l'établissement d'une connexion:

Chaque machine possède un couple de clé (publique et privée par défaut de 1024 bits) de type **RSA**, quand le daemon se lance il génére une clé serveur (**server key**) de type **RSA** (par défaut 768bits). Cette clé serveur est normalement détruite et reconstruite à intervalle régulier (toutes les heures par défaut) et n'est jamais stockée sur le disque.

Quand un client se connecte, le daemon répond en envoyant la clé publique de la machine serveur et la clé serveur. La machine cliente va comparer la clé publique de la machine serveur par rapport à celle qui possède (s'il l'a) pour voir si elle n'a pas changée. Le client va générer alors un nombre aléatoire de 256 bits. Il crypte ce nombre en se servant de la clé publique et de la clé serveur de la machine serveur. Ce nombre aléatoire va servir pour générer une clé de session, on l'appelle clé de session car elle est spécifique à la session de connexion, cette clé de session va servir à chiffrer tout ce qui va transiter pendant la connexion. Les algo qui sont choisis pour crypter les données avec la clé de session qui vont circuler sont **Blowfish**, **tripleDES**.

SSH2

Avec **SSH2** on utilise aussi l'authentification par clé privée-publique au niveau utilisateur, mais cette fois-ci en utilisant l'algorithme de gestion des clés **DSA** plutôt que **RSA** (tombé dans le domaine public en septembre 2000)
Avec ce protocole on a en plus des moyens supplémentaires pour assurer la confidentialité des données, on peut chiffrer les données qui transitent en utilisant, entre autres, le **tripleDES**, **Blowfish**, **AES** ou **Arcfour** et l'intégrité avec des algorithmes de hachage comme **SHA1** ou **MD5** ce que ne possède pas la version 1 de SSH.

Fonctionnement en détail:

C'est similaire à SSH1, chaque machine possède un couple de clé (publique et privée) de type **DSA** par contre. Quand le serveur se lance, aucune clé serveur n'est générée, on utilise une gestion de clé de type **Diffie-Hellman**. Cette gestion débouche sur la création d'une clé de

session spécifique à la session de connexion qui servira à chiffrer les données qui vont transiter.

ATTENTION : A noter que les clés **DSA** sont considérées maintenant comme vulnérables et ne sont pas prises en compte par défaut par les dernières versions d'**OpenSSH** (à moins de le préciser dans le fichier de configuration), il faudra prendre les clés **ECDSA** ou **ed25519**, cette dernière étant considérée comme la plus performante.

En pratique

Un utilisateur **olivier** sur la machine **obelix** peut autoriser certaines personnes venant de certaines machines à se connecter sous **obelix** en utilisant son compte. Pour cela il doit récupérer la clé publique de chacune de ces personnes. Lors de l'établissement de la communication **SSH** va vérifier que la clé publique d'un utilisateur que possède **olivier** correspond bien à clé privé de l'utilisateur en question. En conséquence si **olivier** ne possède pas votre clé publique, il sera impossible de vous connecter sous son compte, le seul moyen est de "voler" la clé privée d'une personne habilitée (dont **olivier** possède la clé publique).

Si vous n'avez toujours pas compris, voici les différentes étapes pour établir une connexion via **SSH** entre les utilisateurs **veronique** sous **asterix** et **olivier** sous **obelix**.

veronique génère une clé publique et privée sur **asterix**
olivier génère clé publique et privée sur **obelix**
veronique donne sa clé publique à **olivier**

veronique veut se connecter sous le compte d'**olivier** sous **obelix** en lançant un client **SSH** sous **asterix**
le daemon **SSH** sous **obelix** vérifie si **olivier** possède bien la clé publique de **veronique** (autorisation)
Le daemon **SSH** sous **obelix** et le client **SSH** sous **asterix** rentrent en communication pour savoir si la clé publique de **veronique** que possède **olivier** correspond bien à la clé privée de **veronique**
veronique doit rentrer une phrase password pour s'identifier auprès du client **SSH** tournant sur **asterix**
veronique doit maintenant rentrer le mot de passe d'**olivier** (éventuellement)

Ca y est **veronique** est connectée sous le compte d'**olivier** après être passé par quatre phases d'identification.

Dans cette page on présentera **OpenSSH** pour un fonctionnement uniquement en utilisant **SSH2**, on passera sous silence tout ce qui concerne **SSH1**, notamment pour l'authentification en utilisant les fichiers **.rhosts**, .**shosts**, **hosts.equiv** et **shosts.equiv**.

5 Installation

5.1 Installation OpenSSL

OpenSSL est un projet qui a pour but de proposer des outils logiciels contenus dans une bibliothèque qui implémente les protocoles Secure Sockets Layer (**SSL** v2 et v3) et Transport Layer Security (**TLS** v1).

Assurez vous d'abord qu'**OpenSSL** n'est pas déjà installé sur votre système, en tapant:

rpm -qa | grep -i openssl

Vous pouvez éventuellement supprimer les packages obtenus pour installer la version tarball généralement plus récente, avec la commande (**rpm -e nom-du-package**).

S'il y a une tonne de dépendances, ce n'est pas grave vous n'êtes pas obligé de supprimer le package, ça marchera très bien en faisant cohabiter la version mdk et la version tarball. Par contre pour pouvoir utilisé la dernière version d'**OpenSSH** vous devez utiliser aussi la dernière version d'**OpenSSL** qu'on peut récupérer à l 'URL www.openssl.org sous la forme d'une archive tarball **openssl-1.1.0c.tar.gz**, qu'on va décompresser en tapant:

tar xvfz openssl-1.1.0c.tar.gz

Attention la dernière version ne compile pas avec un certain d'autres outils, il est encore préférable d'utiliser une ancienne version (comme la 1.0.2d). Cela va nous créer un répertoire **openssl-1.1.0c** dans lequel vous taperez

./config

A noter que pour une version 1.0.2d, sur une configuration 64 bits, on modifiera le fichier **Makefile** au niveau du paramètre **CFLAG** en rajoutant un **-fPIC** comme ceci

CFLAG= -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -Wa,--noexecstack -m64 -DL_ENDIAN -fPIC -O3 -Wall -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DMD5_ASM -DAES_ASM -DVPAES_ASM -DBSAES_ASM -DWHIRLPOOL_ASM -DGHASH_ASM -DECP_NISTZ256_ASM

Ce n'est pas nécessaire pour une version 1.1.0c. Puis on tape

make

Il est nécessaire avant d'aller plus loin de tester la biblio, pour cela préalablement si vous ne l'avez pas vous devez installer la commande **bc** (calculatrice en ligne), contenue dans une mandrake dans le package **bc**.

Tapons à présent:

make test

Vous ne devriez pas avoir d'erreur. Et enfin, en tant que root

make install

Cela va installer les fichiers (librairies, binaires, man, doc, ...) de **SSL** dans le répertoire **/usr/local/ssl**. On rajoutera maintenant dans le fichier **/etc/ld.so.conf** la ligne suivante

/usr/local/ssl/lib

on tape ensuite **ldconfig**

5.2 Installation d'OpenSSH

OpenSSH est intégré dans la Mageia, on va utiliser la dernière version disponible sur le site officiel d'**OpenSSH**. Vous devez d'abord vérifier qu'**OpenSSH** n'est pas déjà installé sur votre système :

rpm -qa | grep -i openssh

Vous supprimez avec la commande **rpm -e nomdupackage**.

Vous pouvez récuperer la dernière version de **OpenSSH** sur le site www.openssh.com. C'est une archive tarball **openssh-7.4p1.tar.gz** . La décompression de l'archive se fait dans un répertoire de travail en tapant:

tar xvfz openssh-7.4p1.tar.gz

Cela va créer le répertoire **openssh-7.4p1**. Vous devez maintenant installer le package **lib64x11-devel** pour pouvoir exporter X, ainsi que les packages **policycoreutils** et **zlib1-devel**. Maintenant dans le répertoire d'**OpenSSH**, vous devez maintenant taper:

./configure --with-ssl-dir=/usr/local/linux/securite/openssl-1.0.2d

Vous devez indiquer le chemin des sources **openssl** conforme à votre système. Attention la version 7.4p1 ne compile pas avec la version 1.1.0c d'**OpenSSL** d'où l'emploi de la version 1.0.2d. Dans le cas où vous utilisez le package **openssl** de la Mageia il faudra installer le package **openssl-devel** et tapez simplement **./configure**

On obtient en fin de commande un récapitulatif des options de compilation

OpenSSH has been configured with the following options:
User binaries: /usr/local/bin
System binaries: /usr/local/sbin
Configuration files: /usr/local/etc
Askpass program: /usr/local/libexec/ssh-askpass
Manual pages: /usr/local/share/man/manX
PID file: /var/run
Privilege separation chroot path: /var/empty
sshd default user PATH: /usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin
Manpage format: doc
PAM support: no
OSF SIA support: no
KerberosV support: no
SELinux support: no
Smartcard support:
S/KEY support: no
MD5 password support: no
libedit support: no
Solaris process contract support: no
Solaris project support: no
Solaris privilege support: no
IP address in $DISPLAY hack: no

```
Translate v4 in v6 hack: yes
     BSD Auth support: no
  Random number source: OpenSSL internal ONLY
 Privsep sandbox style: seccomp_filter

     Host: x86_64-pc-linux-gnu
 Compiler: gcc
Compiler flags: -g -O2 -Wall -Wpointer-arith -Wuninitialized -Wsign-compare -Wformat-security -Wsizeof-pointer-memaccess -Wno-pointer-sign -Wno-unused-result -fno-strict-aliasing -D_FORTIFY_SOURCE=2 -ftrapv -fno-builtin-memset -fstack-protector-strong -fPIE
Preprocessor flags: -I/usr/local/linux/securite/openssl-1.1.0c/include
Linker flags: -L/usr/local/linux/securite/openssl-1.1.0c  -Wl,-z,relro -Wl,-z,now -Wl,-z,noexecstack -fstack-protector-strong -pie
Libraries: -lcrypto -ldl -lutil -lz  -lcrypt -lresolv
```

Puis

make

On doit maintenant créer un utilisateur **sshd**, on tapera les commandes suivantes en tant que root

```
mkdir /var/empty
chown root:sys /var/empty
chmod 755 /var/empty
groupadd sshd
useradd -g sshd -c 'sshd privsep' -d /var/empty -s /bin/false sshd
```

/var/empty ne doit contenir aucun fichier. Ceci permet de réaliser certaines opérations sans être root.

make install

A la fin des commandes les clés de votre machine sont générées dans le cas où celles ci n'existent pas sous **/usr/local/etc**

```
Generating public/private rsa1 key pair.
Saving key "/usr/local/etc/ssh_host_key" failed: unknown or unsupported key type
Generating public/private dsa key pair.
Your identification has been saved in /usr/local/etc/ssh_host_dsa_key.
Your public key has been saved in /usr/local/etc/ssh_host_dsa_key.pub.
The key fingerprint is:
SHA256:+4AlELhvZA4mBUvrU/+WO4WV1BlFNAvnkUbAg3eQw0Q
root@obelix.kervao.fr
The key's randomart image is:
+---[DSA 1024]----+
|.o ..     BE@O.  |
|..+  .   o X+++  |
|.o o.   . o =o   |
|o = +.   o       |
| = * .. S        |
|  . + .=.o       |
|   .  .++        |
|      ...o       |
|       .. .      |
+----[SHA256]-----+
```

```
Generating public/private rsa key pair.
Your identification has been saved in /usr/local/etc/ssh_host_rsa_key.
Your public key has been saved in /usr/local/etc/ssh_host_rsa_key.pub.
The key fingerprint is:
SHA256:qGBnrwQXbfQK8/o092bCiQGU8ygc7NI65AzFYWxwelQ
root@obelix.kervao.fr
The key's randomart image is:
+---[RSA 2048]----+
|.==oE..          |
| =B +o .         |
|.*.o++o .        |
|oo= o*.o         |
|=o+.+.+ S        |
|o+ * +.          |
| .  + ++..       |
|   . +.o+.o      |
|    . .  +.      |
+----[SHA256]-----+
Generating public/private ed25519 key pair.
Your identification has been saved in /usr/local/etc/ssh_host_ed25519_key.
Your public key has been saved in /usr/local/etc/ssh_host_ed25519_key.pub.
The key fingerprint is:
SHA256:hzpRRV+XrggYbmwdw2lbmG1HTVuss4cSRLgAJ1fAazk
root@obelix.kervao.fr
The key's randomart image is:
+--[ED25519 256]--+
|      oo=+Ooo.+o+|
|       =oX.=.o ++|
|      o *+*oo .o |
|       BE=. . o. |
|      +.S.o ...+ |
|       o . ...o .|
|      o      . . |
|       .         |
|                 |
+----[SHA256]-----+
Generating public/private ecdsa key pair.
Your identification has been saved in /usr/local/etc/ssh_host_ecdsa_key.
Your public key has been saved in /usr/local/etc/ssh_host_ecdsa_key.pub.
The key fingerprint is:
SHA256:2S/VH7hJz7YEgYwH/ZfkTYPxsWZiXHH4mvIAxcIEX2I
root@obelix.kervao.fr
The key's randomart image is:
+---[ECDSA 256]---+
|        .=E...o=o|
|         o*+=.+o=|
|         ..*.=oB+|
|         oo .o*+o|
|        S ...+o+ |
|          o+ O..|
|         . .* =.|
|          .  + .|
```

| .|
+----[SHA256]-----+
/usr/local/sbin/sshd -t -f /usr/local/etc/sshd_config

Cela va créer les clés publiques et privées de la machine qui seront placées par défaut dans **/usr/local/etc** . Plus précisément 4 types de clés différentes sont créées on trouve la clé privée **ssh_host_rsa_key** si on utilise la gestion de clés type **RSA** (**SSH1**) et la publique associée **ssh_host_rsa_key.pub** et de même les paires de clé **DSA**, **ECDSA** et **ed25519** pour le **SSH2**.

NOTES - En cas d'upgrade les clés de la machine ne seront pas écrasées de même que les fichiers de conf.

- Dans le cas où les clés de la machine sont réinitialisées, pensez à supprimer les fichiers **~/.ssh/known_hosts** qui contiennent les identifiants des anciennes versions, sous peine d'obtenir le warning suivant

@@@
@   WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!   @
@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the DSA host key has just been changed.
The fingerprint for the DSA key sent by the remote host is
0e:6f:19:83:12:96:a6:d1:4d:2b:d6:40:93:39:05:98.
Please contact your system administrator.
Add correct host key in /home/olivier/.ssh/known_hosts to get rid of this message.
Offending key in /home/olivier/.ssh/known_hosts:1
Password authentication is disabled to avoid man-in-the-middle attacks.
Keyboard-interactive authentication is disabled to avoid man-in-the-middle attacks.
X11 forwarding is disabled to avoid man-in-the-middle attacks.

L'installation va installer les binaires utilisateurs sous **/usr/local/bin**, le daemon sous **/usr/local/sbin** et le serveur ftp sécurisé sous **/usr/local/libexec**

Les fichiers de config sont sous **/usr/local/etc** (**ssh_config** pour le client **sshd_config** pour le serveur). Si ces chemins ne vous plaisent pas, lancer **configure** avec les arguments qui vont bien donnés par:

configure --help

A présent si vous utilisez **PAM** vous devez prendre le fichier **sshd.pam** se trouvant sous **./openssh-7.4p1/contrib/redhat** et le placer sous **/etc/pam.d** en le renommant **sshd** tout court. En vous plaçant donc dans le répertoire de travail d'**OpenSSH**:

cp ./contrib/redhat/sshd.pam /etc/pam.d/sshd

NOTE Comment savoir si vous utilisez **PAM**, tapez **journalctl -f** si vous voyez des mentions à **PAM** (**PAM_pwdb** par exemple quand vous vous loguez ou faites un **su**), c'est qu'il est actif sur votre système.

6 Configuration

6.1 Configuration du serveur

Le daemon **SSH** se configure avec le fichier **sshd_config** se trouvant sous **/usr/local/etc** voici son contenu :

```
# This is ssh server systemwide configuration file.

#Port 22
#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::

# The default requires explicit activation of protocol 1
#Protocol 2

# HostKey for protocol version 1
#HostKey /etc/ssh/ssh_host_key
# HostKeys for protocol version 2
#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_dsa_key
#HostKey /etc/ssh/ssh_host_ecdsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key

# Lifetime and size of ephemeral version 1 server key
#KeyRegenerationInterval 1h
#ServerKeyBits 1024

# Ciphers and keying
#RekeyLimit default none

# Logging
# obsoletes QuietMode and FascistLogging
#SyslogFacility AUTH
#LogLevel INFO

# Authentication:

#LoginGraceTime 2m
#PermitRootLogin prohibit-password
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

#RSAAuthentication yes
#PubkeyAuthentication yes

# The default is to check both .ssh/authorized_keys and .ssh/authorized_keys2
# but this is overridden so installations will only check .ssh/authorized_keys
AuthorizedKeysFile      .ssh/authorized_keys
```

```
#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody

# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#RhostsRSAAuthentication no
# similar for protocol version 2
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# RhostsRSAAuthentication and HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
#PasswordAuthentication yes
#PermitEmptyPasswords no

# Change to no to disable s/key passwords
#ChallengeResponseAuthentication yes

# Kerberos options
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#KerberosTicketCleanup yes
#KerberosGetAFSToken no

# GSSAPI options
#GSSAPIAuthentication no
#GSSAPICleanupCredentials yes

# Set this to 'yes' to enable PAM authentication, account processing,
# and session processing. If this is enabled, PAM authentication will
# be allowed through the ChallengeResponseAuthentication and
# PasswordAuthentication.  Depending on your PAM configuration,
# PAM authentication via ChallengeResponseAuthentication may bypass
# the setting of "PermitRootLogin without-password".
# If you just want the PAM account and session checks to run without
# PAM authentication, then enable this but set PasswordAuthentication
# and ChallengeResponseAuthentication to 'no'.
#UsePAM no

#AllowAgentForwarding yes
#AllowTcpForwarding yes
#GatewayPorts no
X11Forwarding no
X11DisplayOffset 10
X11UseLocalhost yes
#PermitTTY yes
#PrintMotd yes
```

```
#PrintLastLog yes
#TCPKeepAlive yes
#UseLogin no
UsePrivilegeSeparation sandbox          # Default for new installations.
#PermitUserEnvironment no
#Compression delayed
#ClientAliveInterval 0
#ClientAliveCountMax 3
#UseDNS no
#PidFile /var/run/sshd.pid
#MaxStartups 10:30:100
#PermitTunnel no
#ChrootDirectory none
#VersionAddendum none

# no default banner path
#Banner none

# override default of no subsystems
Subsystem       sftp    /usr/libexec/sftp-server

# Example of overriding settings on a per-user basis
#Match User anoncvs
#       X11Forwarding no
#       AllowTcpForwarding no
#       PermitTTY no
#       ForceCommand cvs server
```

6.2 Configuration du client

Le fichier de config du client **SSH** (se trouvant sur la machine distante et non pas sur le serveur) se trouve sous **/usr/local/etc** et a pour nom **ssh_config**

```
#       $OpenBSD: ssh_config,v 1.28 2013/09/16 11:35:43 sthen Exp $

# This is the ssh client system-wide configuration file.  See
# ssh_config(5) for more information.  This file provides defaults for
# users, and the values can be changed in per-user configuration files
# or on the command line.

# Configuration data is parsed as follows:
#  1. command line options
#  2. user-specific file
#  3. system-wide file
# Any configuration value is only changed the first time it is set.
# Thus, host-specific definitions should be at the beginning of the
# configuration file, and defaults at the end.

# Site-wide defaults for some commonly used options.  For a comprehensive
# list of available options, their meanings and defaults, please see the
# ssh_config(5) man page.
```

```
# Host *
    ForwardAgent yes
    ForwardX11 yes
#   RhostsRSAAuthentication no
#   RSAAuthentication yes
#   PasswordAuthentication yes
#   HostbasedAuthentication no
#   GSSAPIAuthentication no
#   GSSAPIDelegateCredentials no
#   BatchMode no
#   CheckHostIP yes
#   AddressFamily any
#   ConnectTimeout 0
#   StrictHostKeyChecking ask
#   IdentityFile ~/.ssh/identity
#   IdentityFile ~/.ssh/id_rsa
#   IdentityFile ~/.ssh/id_dsa
#   Port 22
#   Protocol 2,1
#   Cipher 3des
#   Ciphers aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,aes128-cbc,3des-cbc
#   MACs hmac-md5,hmac-sha1,umac-64@openssh.com,hmac-ripemd160
#   EscapeChar ~
#   Tunnel no
#   TunnelDevice any:any
#   PermitLocalCommand no
#   VisualHostKey no
#   ProxyCommand ssh -q -W %h:%p gateway.example.com
#   RekeyLimit 1G 1h

# If this option is set to yes then remote X11 clients will have full access
# to the original X11 display. As virtually no X11 client supports the untrusted
# mode correctly we set this to yes.
ForwardX11Trusted yes

# Send locale-related environment variables
#SendEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY LC_MESSAGES
#SendEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
#SendEnv LC_IDENTIFICATION LC_ALL
StrictHostKeyChecking no
```

Pour la variable **Host** vous pouvez spécifier des paramètres différents suivant le serveur, exemple avec le serveur **tetepa** (utilise SSH2) et **atopa** (utilise SSH1)**.**

```
Host tetepa
   port 22
   Protocol 2
   Pubkeyauthentication yes
   PasswordAuthentication yes
```

Host atopa
port 22
Protocol 1
RSAAuthentication yes
PasswordAuthentication yes

7 Lancement du daemon sshd

La configuration du lancement sous **systemd** est la suivante. On crée un fichier **sshd.service** qu'on place sous **/usr/lib/systemd/system/** voilà son contenu

[Unit]
Description=OpenSSH server daemon
After=syslog.target network.target auditd.service

[Service]
ExecStart=/usr/local/sbin/sshd -D -e
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s

[Install]
WantedBy=multi-user.target

maintenant pour que le service soit lancé à chaque boot de la machine il faudra taper

systemctl enable sshd.service

voillà le résultat

Created symlink from /etc/systemd/system/multi-user.target.wants/sshd.service to /usr/lib/systemd/system/sshd.service.

8 Fonctionnement en mode debug

On va prendre le cas simple du même utilisateur **olivier** qui se connecte sur le serveur **obelix** sous son propre compte à partir d'**asterix**, l'utilisateur n'a généré aucune clé propre. En tapant

ssh -v obelix

voilà le résultat

OpenSSH_7.4p1, OpenSSL 1.0.2d 9 Jul 2015
debug1: Reading configuration data /usr/local/etc/ssh_config
debug1: Connecting to obelix [192.168.13.11] port 22.
debug1: Connection established.

vérification de l'existence de clé

debug1: identity file /home/olivier/.ssh/id_rsa type 1
debug1: key_load_public: No such file or directory
debug1: identity file /home/olivier/.ssh/id_rsa-cert type -1
debug1: identity file /home/olivier/.ssh/id_dsa type 2
debug1: key_load_public: No such file or directory
debug1: identity file /home/olivier/.ssh/id_dsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/olivier/.ssh/id_ecdsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/olivier/.ssh/id_ecdsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/olivier/.ssh/id_ed25519 type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/olivier/.ssh/id_ed25519-cert type -1

mise en place du protocole d'échange entre le client **asterix** et serveur **obelix**

debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_7.1
debug1: Remote protocol version 2.0, remote software version OpenSSH_7.1
debug1: match: OpenSSH_7.1 pat OpenSSH* compat 0x04000000
debug1: Authenticating to obelix:22 as 'olivier'
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client chacha20-poly1305@openssh.com <implicit> none
debug1: kex: client->server chacha20-poly1305@openssh.com <implicit> none
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
debug1: Server host key: ecdsa-sha2-nistp256 SHA256:2S/VH7hJz7YEgYwH/ZfkTYPxsWZiXHH4mvIAxcIEX2I
The authenticity of host 'obelix (192.168.13.11)' can't be established.
ECDSA key fingerprint is SHA256:2S/VH7hJz7YEgYwH/ZfkTYPxsWZiXHH4mvIAxcIEX2I.

le serveur **obelix** n'est pas connu, à défaut de posséder une clé publique du serveur on demande si on certifie qu'on a bien affaire à **obelix**, ensuite la clé publique d'**obelix** se retrouvera dans la liste des hôtes connus.

Are you sure you want to continue connecting (yes/no)?
Warning: Permanently added 'obelix,192.168.13.11' (ECDSA) to the list of known hosts.
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: Roaming not allowed by server
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received

maintenant que le serveur est authentifié, on passe à l'authentifcation de l'utilisateur, on a le choix entre l'échange de clé, le mot de passe et le "**keyboard-interactive**" qui correspond plus ou moins à l'authentification via **PAM**.

debug1: Authentications that can continue: publickey,password,keyboard-interactive

debug1: Next authentication method: publickey
debug1: Offering RSA public key: /home/olivier/.ssh/id_rsa
debug1: Authentications that can continue: publickey,password,keyboard-interactive

on teste d'abord l'échange de clé en zappant la clé **DSA** par défaut

debug1: Skipping ssh-dss key /home/olivier/.ssh/id_dsa for not in PubkeyAcceptedKeyTypes
debug1: Trying private key: /home/olivier/.ssh/id_ecdsa
debug1: Trying private key: /home/olivier/.ssh/id_ed25519
debug1: Next authentication method: keyboard-interactive
debug1: Authentications that can continue: publickey,password,keyboard-interactive

on passe à l'authentification par mot de passe

debug1: Next authentication method: password
obelix@olivier's password:

et voilà on est connecté

9 Création et échange des clé

Considérons toujours l'utilisateur **olivier** sur la machine **obelix** qui veut autoriser l'utilisateur **veronique** de la machine **asterix** à se connecter sur son compte. L'utilisateur **olivier** doit d'abord créer son couple de clé, on zappera donc **DSA** et on utilisera **ECDSA**

ssh-keygen -t ecdsa

voilà le résultat

Generating public/private ecdsa key pair.
Enter file in which to save the key (/home/olivier/.ssh/id_ecdsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/olivier/.ssh/id_ecdsa.
Your public key has been saved in /home/olivier/.ssh/id_ecdsa.pub.
The key fingerprint is:
SHA256:diY6BD8vKyM2C3LCODNcwd7YK0fcTI9EfTglM6YBLSM
olivier@asterix.kervao.fr
The key's randomart image is:
+---[ECDSA 256]---+
| .oo.=o. |
| .E o..+++. |
| oo o+ o |
| . *o= o |
| + =++S.o |
|+ . ...= + |
|Oo.. o+ . |
|.B+ = + |

| ..+ o. |
+----[SHA256]-----+

Les clés sera sauvegardée par défaut dans **~/.ssh** , la clé privée a pour nom **id_ecdsa** et la clé publique **id_ecdsa.pub**

Il est nécessaire de rentrer un mot de passe (passphrase), à noter que ce mot de passe peut être une phrase avec des blancs.

Maintenant **veronique** doit faire de même sous **asterix** , elle doit ensuite par un moyen ou un autre faire parvenir sa clé publique (**id_ecdsa.pub**) à **olivier**, ce dernier va mettre le contenu de ce fichier dans le fichier (éventuellement à créer) **~/.ssh/authorized_keys**.

ATTENTION il est primordial que la clé publique de **veronique** tienne sur **UNE** seule ligne dans le fichier **authorized_keys**, il ne doit pas y avoir de retour chariot.

NOTE Si **olivier** autorise plusieurs personnes à se connecter sur son compte par **ssh**, la syntaxe de **authorized_keys** deviendra:

 # utilisateur veronique
contenu de id_ecdsa.pub de veronique sur une seule ligne

 # utilisateur lambda
contenu de id_ecdsa.pub de lamba sur une seule ligne

 # ainsi de suite

est le début de commentaires, et les lignes vides ne sont pas prises en compte.

NOTE Pour changer de pass-phrase au niveau de **ssh-keygen** , il suffit de taper:

ssh-keygen -p

10Utilisation

10.1 Connexion simple avec ssh

Une fois l'échange des clés effectuées, **veronique** sur **asterix** va se connecter sur **obelix** en utilisant le compte d'**olivier**:

[veronique@asterix .ssh]$ ssh -l olivier obelix
The authenticity of host 'obelix (192.168.13.11)' can't be established.
DSA key fingerprint is bb:aa:0a:92:66:a8:6c:08:ab:6b:35:18:4d:ab:19:13.
Are you sure you want to continue connecting (yes/no)?yes
Warning: Permanently added 'obelix,192.168.13.11' (DSA) to the list of known hosts.
Enter passphrase for key '/home/veronique/.ssh/id_dsa':
Last login: Sat Jun 22 14:11:43 2002
[olivier@obelix olivier]$

C'est bon on est connecté, à noter:

Warning: Permanently added 'obelix,192.168.13.11' (DSA) to the list of known hosts.

La machine **obelix** va être rajouté à la liste des machines connues, y aura plus ce message par la suite, si vous voulez que la machine **obelix** ne soit pas rajoutée de manière permanente mettez le paramètre **StrictHostKeyChecking** à **yes** dans **ssh_config** mais dans ce cas il faudra récupérer la clé publique d'**obelix** pour la mettre manuellement dans le fichier **~/.ssh/known_hosts**

A noter aussi que la prochaine fois que vous essayerez de vous connecter sur **obelix**, il ne sera plus nécessaire de rajouter **-l olivier** , ça sera ajouté automatiquement par défaut.

NOTE Les infos sur la machine seront sauvegardées dans le fichier **~/.ssh/known_hosts**

10.2 Lancement une commande à distance

Imaginons que **veronique** veuille lancer la commande **df** sur **obelix**, il suffit de taper sur **asterix**:

```
[veronique@asterix .ssh]$ ssh obelix df
Enter passphrase for DSA key '/home/veronique/.ssh/id_dsa':
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/sdb1              149712     75053     66929  53% /
/dev/sda1              991995    806267    134478  86% /alphonse
/dev/sdb3              239979     77414    150175  34% /roger
/dev/sdb4               99136     42591     51425  45% /home
/dev/sdc5              938296    497756    392876  56% /usr
/dev/sdb5              496627    334344    136633  71% /usr/local
[veronique@asterix .ssh]$
```

Après exécution de la commande, la connexion est automatiquement coupée.

10.3 Lancement d'une commande X à distance

Voici ce qu'on devrait avoir pour un fonctionnement normal :

```
[veronique@asterix .ssh]$ ssh obelix
Enter passphrase for DSA key '/home/veronique/.ssh/id_dsa':
Last login: Fri May 26 12:14:52 2000 from asterix.armoric.bz
[olivier@obelix olivier]$ env | grep DISPLAY
DISPLAY=localhost:10.0
[olivier@obelix olivier]$ xauth list
obelix.armoric.bz:0  MIT-MAGIC-COOKIE-1  1f2c455765476a6d5c0a1225383d4f52
obelix.armoric.bz/unix:0  MIT-MAGIC-COOKIE-1
1f2c455765476a6d5c0a1225383d4f52
obelix.armoric.bz:10  MIT-MAGIC-COOKIE-1  599555deac54a6b3f338147f9bb56eb8
obelix.armoric.bz:1  MIT-MAGIC-COOKIE-1  545a095522c5fbe32bbb1d357ab52227
obelix.armoric.bz/unix:1  MIT-MAGIC-COOKIE-1
545a095522c5fbe32bbb1d357ab52227
obelix.armoric.bz/unix:10  MIT-MAGIC-COOKIE-1
```

9434717920d84aa2287b76974f723fe0
[olivier@obelix olivier]$

Remarquer bien que sur le client **SSH**, **DISPLAY** est positionné à localhost qui correspond au serveur c'est tout à fait normal, et là si vous lancez une commande **X** elle s'affichera bien à l'écran d'**asterix** et non pas d'**obelix**.

xauth permet d'ajouter, éditer les autorisations pour se connecter à un serveur X, concrètement **sshd** va appeler **xauth** pour que les commandes **X** puissent s'afficher sur le client.

Bon par contre, si vous avez l'erreur suivante

[olivier@obelix olivier]$ xmms &
[1] 1036
[olivier@obelix olivier]$ Gdk-ERROR **: X connection to obelix.armoric.bz:10.0 broken (explicit kill or server shutdown).

[1]+ Exit 1 xmms

Il faudra modifier le fichier **.bashrc** de l'utilisateur **olivier** au lieu de:

export XAUTHORITY=$HOME/.Xauthority

On mettra

[-z $XAUTHORITY] && export XAUTHORITY=$HOME/.Xauthority

Et là magique ça marche !!

10.4 Copier des fichiers

On retrouve une syntaxe complètement équivalente à la commande UNIX **rcp**. Admettons que **veronique** veuille copier le fichier **toto** se trouvant dans sa home directory dans le répertoire **/tmp** d'**obelix**. Il suffit de taper:

[veronique@asterix veronique]$ scp toto obelix:/tmp
Enter passphrase for DSA key '/home/veronique/.ssh/id_dsa':
toto 100% |***| 27 00:00**

Maintenant on veut récupérer le fichier **titi** se trouvant dans la home directory d'**olivier**, pour le mettre sous le**/tmp** d'**asterix**

```
[veronique@asterix veronique]$ scp obelix:/home/olivier/titi /tmp
Enter passphrase for DSA key '/home/veronique/.ssh/id_dsa':
olivier@obelix's password:
titi                  100% |*****************************************|    15       00:00
```

Maintenant on va récupérer tout le répertoire **php3** d'**olivier** pour le placer dans le répertoire courant (.)

```
[veronique@asterix veronique]$ scp -r obelix:/home/olivier/php3 .
Enter passphrase for DSA key '/home/veronique/.ssh/id_dsa':
fichier1              100% |*****************************************|     5       00:00
repertoire1           100% |*****************************************|     5       00:00
fichier2              100% |*****************************************|     5       00:00
```

A noter que l'option **-r** n'est ni documentée dans le man, ni en faisant un **scp -help** (oubli ?). Autres options intéressantes de **scp**:

-v verbose pour avoir un max de commentaires, notamment pour l'établissement de la connexion et l'identification.
-p pour préserver les droits (conseillé)
-c pour changer le type de cryptage
-B mode batch pour mettre **scp** dans un script (plus besoin de rentrer le passphrase et le mot de passe)
-C pour compresser

10.5 Utilisation d'un agent

L'agent permet de ne pas avoir à saisir la passphrase c'est utile quand vous mettez **ssh** dans un script. Pour activer l'agent il suffit de taper

```
ssh-agent
```

Voilà le résultat

```
SSH_AUTH_SOCK=/tmp/ssh-tikvJb1434/agent.1434; export SSH_AUTH_SOCK;
SSH_AGENT_PID=1435; export SSH_AGENT_PID;
echo Agent pid 1435
```

Tapez explicitement ces trois commandes dans un shell (il est bien évident que sur votre configuration vous devriez avoir des valeurs différentes !). On va maintenant donner à **ssh-agent** le passphrase, comme cela

```
ssh-add
```

Enter passphrase for /export/home/olivier/.ssh/id_dsa:
Identity added: /export/home/olivier/.ssh/id_dsa (/export/home/olivier/.ssh/id_dsa)

Tentez une connexion, vous verrez que ce ne sera plus nécessaire de saisir le passphrase.