Servei d’accés remot: SSH

Curs 2018-2019

Descripció dels aprenentatges

1) Instal·lació de la suite SSH. Components de què es compon. Estructura de directòris.
    a) Els paquets i els seus components. L'estructura de directoris. Els fitxers de configuració i claus.
    b) El model de funcionament i els mètodes de validació: passwd, public Key. Altres mètodes.
    c) Concepte de tunnel SSH (tractat al ISX-M11)
    d) Concepte de X-Forwarding.
2) Establiment d'una connexió SSH.
    a) Establiment de la clau pública del host destí al Known_hosts.
    b) Examinar Known_hots. Identificar visualment la clau pública de host remot.
    c) Sessió ssh remota.
    d) Ordres remotes. Fluxe d'informació (stdin / stdout / stderr) entre client i remot.
3) Accés desatès.
    a) Generació de la parell de claus púbica/privada.
    b) Transferència de la clau pública als comptes en els hosts remots.
    c) Accés remot desatès. Al mateix compte o a altres comptes configurats amb la mateixa clau pública.
4) Utilitats SSH Clients
    a) Utilitats clàssiques SCP i SFTP.
    b) Altres utilitats: slogin, ssh-agent, ssh-add, ssh-keyscan, ssh-copy-id.
5) Configuració SSH: client i servidor
    a) Nivells de configuració client: comanda, personalitzat d'usuari, global.
    b) Opcions de configuració client.
    c) Configuració global del servidor. Opcions més comunes de port, protocol, autenticació acceptada, accés de root, etc. sshd.conf.
6) Restriccions d'accés
    a) Restricció d'accés segons directives del fitxer de configuració del servidor sshd_config.
    b) Restricció d'accés usant PAM.
    c) Restricció de les comendes permeses en una connexió, especificat al authorized_keys.

Consulteu també:

La Documentació de Fedora (F24)
System Administrators Guide, Chapter 8 "Open SSH"

La suite SSH

Descripció dels components / paquets

La suite SSH es composa usualment dels paquets openssh, openssh-clients i openssh-servers. Les eines principals són el servidor sshd i les ordres ssh, scp i sftp.

```
$ ssh
ssh      ssh-add         ssh-agent       ssh-copy-id  sshd       sshd-keygen  ssh-keygen
ssh-keyscan

$ rpm -qa | grep ssh
openssh-clients-6.4p1-5.fc20.x86_64
libssh-0.6.3-1.fc20.x86_64
openssh-server-6.4p1-5.fc20.x86_64
openssh-6.4p1-5.fc20.x86_64
libssh2-1.4.3-9.fc20.x86_64

$ rpm -ql openssh
/etc/ssh
/etc/ssh/moduli
/usr/bin/ssh-keygen
/usr/libexec/openssh
/usr/libexec/openssh/ctr-cavstest
/usr/libexec/openssh/ssh-keysign
/usr/share/doc/openssh
…
/usr/share/man/man1/ssh-keygen.1.gz
/usr/share/man/man8/ssh-keysign.8.gz

$ rpm -ql openssh-clients
/etc/ssh/ssh_config
/usr/bin/scp
/usr/bin/sftp
/usr/bin/slogin
/usr/bin/ssh
/usr/bin/ssh-add
/usr/bin/ssh-agent
/usr/bin/ssh-copy-id
/usr/bin/ssh-keyscan
/usr/lib64/fipscheck/ssh.hmac
```

```
/usr/libexec/openssh/ssh-pkcs11-helper
/usr/share/man/man1/scp.1.gz
...
/usr/share/man/man8/ssh-pkcs11-helper.8.gz


# rpm -ql openssh-server
/etc/pam.d/sshd
/etc/ssh/sshd_config
/etc/sysconfig/sshd
/usr/lib/systemd/system/sshd-keygen.service
/usr/lib/systemd/system/sshd.service
/usr/lib/systemd/system/sshd.socket
/usr/lib/systemd/system/sshd@.service
/usr/lib64/fipscheck/sshd.hmac
/usr/libexec/openssh/sftp-server
/usr/sbin/sshd
/usr/sbin/sshd-keygen
/usr/share/man/man5/moduli.5.gz
/usr/share/man/man5/sshd_config.5.gz
/usr/share/man/man8/sftp-server.8.gz
/usr/share/man/man8/sshd.8.gz
/var/empty/sshd


# tree /etc/ssh/
/etc/ssh/
├── moduli
├── ssh_config
├── sshd_config
├── ssh_host_ecdsa_key
├── ssh_host_ecdsa_key.pub
├── ssh_host_rsa_key
└── ssh_host_rsa_key.pub

# tree .ssh/
.ssh/
└── known_hosts
```

Funcionament del client SSH

Tipus d'autenticació:

- Verificació del host remot.
- Authenticació d'usuari: password, publicKey, gssapi.
- Authenticació desatesa basada en host (deprecated? Per ser insegura.)

Verificació del host remot:

SSH genera automàticament en instal·lar-se un parell de claus (privada-pública) que identifiquen el host. Pot generar diversos tipus diferents de parells de claus: DSA i RSA. Usualment anomenades */etc/ssh/ssh_host_rsa_key* la clau privada RSA i *ssh_host_rsa_key.pub* la clau pública.

En conectar el client al servidor es realitza la transmissió de la clau pública del servidor al client, que si l'accepta passa a incorporar-la al *.ssh/knoun_hosts* de l'usuari client. Al ***.ssh/knnown_hosts*** hi ha les claus públiques que identifiquen els hosts als que aquest usuari-client s'ha connectat.

En una sessió SSH un cop establerta la connexió xifrada es realitza sempre la identificació del host servidor, la seva clau pública ha de coincidir amb la emmagatzemada (sinó pot ser que es produeixi un atac de suplantació o man-in-the-middle).
El fingerprint que es mostra en la primera connexió no és més que un HASH de la clau pública (un 'resum' perquè sigui més fàcil verificar-la). En aquesta primera connexió cladria sempre realitzar aquesta verificació.

Autenticació d'usuari:
Tip: consulteu a l'apèndix l'apartat Authentication

Password:

L'autenticació d'usuari clàssica que es realitza és la de password. Consisteix en que el servidor sol·licita a l'usuari client que s'identifiqui introduïnt el password apropiat de l'usuari al que es vol conectar.

Gssapi:

Authenticació basada en Kerberos.

PublicKey:

Autentica l'usuari sense password a través de la utilització de claus (parella de claus privada-pública). Serà a través de la verificació de seva clau pública que l'usuari podrà accedir al seu compte en el servidor sense necessitat de passwd. Això permet connexions desates.

L'usuari client ha de disposar de la parella de claus privada-pública que pot generar amb l'ordre **ssh-keygen**. Es poden generar claus RSA, DSA, etc. Les claus usualment s'anomenent *.ssh/id_rsa* la clau privada RSA i *.ssh/id_rsa.pub* per la clau pública RSA. Són al seu directori ssh de personalització i la clau privada ha de estar absolutament protegida (600).

La clau privada es manté sempre <u>absolutament</u> privada. La clau pública té la finalitat de fer-se <u>pública</u>, de propagar-se.

L'autenticació desatesa consisteix en que l'usuari client es connecti a un compte seu en un servidor remot (el mateix nom compte o fins i tot un altre) . Cal propagar la clau pública de l'usuari client al directori de personalització ssh del mateix usuari en el servidor remot.

És a dir, cal copiar la clau pública *.ssh/id_rsa.pub* del client dins del fitxer **authorized_keys** del servidor. Aquest fitxer conté totes aquelles claus públiques que tenen autorització per connectar-se a aquest compte d'usuari en el servidor remot sense necessitat d'indicar el password.

Les claus es concatenen en aquest fitxer de text, de manera que és tasca del client propagar la seva clau pública del seu compte client al compte en el servidor remot afegint-la al .ssh/authorized_keys.

Fixeu-vos que l'usuari pere del hostClient pot automatitzar l'entrada desatesa als comptes pere de hostRemot, al compte superman del hostRemot, al compte batman del hostRemot2 i al compte hulk del hostRemot3 (per exemple). Únicament li cal propagar la seva clau pública a cada un d'aquests comptes i visitar de tant en tant el psicòleg.

Authenticació per host:

L'autenticació per host permet que el servidor accepti connexions desateses (sense password) de qualsevol usuari client que provingui d'un '**trusted host**', un host amb qui confii.
Evidentment això pot representar un problema de seguretat important si la seguretat del host client és vulnerada automàticament compromet la seguretat del servidor remot.

Un compte d'usuari en el servidor remot confiarà amb tots els usuaris d'un host client si disposa en el seu **authorized_keys** de la clau pública del host client. Per tant cal concatenar la clau */etc/ssh/ssh_host_key.pub* (o la RSA o DSA) al *.ssh/authorized_keys* del compte de l'usuari en el servidor remot.

Ordre SSH

```
NAME
        ssh — OpenSSH SSH client (remote login program)

SYNOPSIS
        ssh [-1246AaCfgKkMNnqsTtVvXxYy] [-b bind_address] [-c cipher_spec]
             [-D [bind_address:]port] [-E log_file] [-e escape_char] [-F configfile] [-I pkcs11]
             [-i identity_file] [-L [bind_address:]port:host:hostport] [-l login_name]
             [-m mac_spec] [-O ctl_cmd] [-o option] [-p port]
             [-R [bind_address:]port:host:hostport] [-S ctl_path] [-W host:port]
             [-w local_tun[:remote_tun]]
             [user@]hostname [command] ssh -Q protocol_feature
```

DESCRIPTION

ssh (SSH client) is a program for logging into a remote machine and for executing commands on a remote machine. It is intended to replace rlogin and rsh, and provide secure encrypted communications between two untrusted hosts over an insecure network. X11 connections and arbitrary TCP ports can also be forwarded over the secure channel.

ssh connects and logs into the specified hostname (with optional user name). The user must prove his/her identity to the remote machine using one of several methods depending on the protocol version used (see below).

If command is specified, it is executed on the remote host instead of a login shell.

Verifying host keys

En realitzar-se la instal·lació del servei SSHD s'han creat al host les claus públiques/privades de host que identifiquen al host, a la màquina. Poden existir diversos tipus de claus com per exemple RSA, DSA, etc. Es genera per a cada tipus un parell de fitxers, un contenint la clau pública i l'altre la clau privada.

VERIFYING HOST KEYS

When connecting to a server for the first time, a fingerprint of the server's public key is presented to the user (unless the option StrictHostKeyChecking has been disabled). Fingerprints can be determined using ssh-keygen(1):

```
$ ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key
```

If the fingerprint is already known, it can be matched and the key can be accepted or rejected. Because of the difficulty of comparing host keys just by looking at hex strings, there is also support to compare host keys visually, using random art. By setting the VisualHostKey option to “yes”, a small ASCII graphic gets displayed on every login to a server, no matter if the session itself is interactive or not.

By learning the pattern a known server produces, a user can easily find out that the host key has changed when a completely different pattern is displayed. Because these patterns are not unambiguous however, a pattern that looks similar to the pattern remembered only gives a good probability that the host key is the same, not guaranteed proof.

To get a listing of the fingerprints along with their random art for all known hosts, the following command line can be used:

```
$ ssh-keygen -lv -f ~/.ssh/known_hosts
```

If the fingerprint is unknown, an alternative method of verification is available: SSH fingerprints verified by DNS. An additional resource record (RR), SSHFP, is added to a zonefile and the connecting client is able to match the fingerprint with that of the key presented.

In this example, we are connecting a client to a server, “host.example.com”. The SSHFP resource records should first be added to the zonefile for host.example.com:

```
$ ssh-keygen -r host.example.com.
```

The output lines will have to be added to the zonefile. To check that the zone is answering fingerprint queries:

```
$ dig -t SSHFP host.example.com
```

Finally the client connects:

```
$ ssh -o "VerifyHostKeyDNS ask" host.example.com
[...]
Matching host key fingerprint found in DNS.
Are you sure you want to continue connecting (yes/no)?
```

See the *VerifyHostKeyDNS* option in ssh_config(5) for more information.

```
# tree /etc/ssh/
/etc/ssh/
├── moduli
├── ssh_config
├── sshd_config
├── ssh_host_ecdsa_key
├── ssh_host_ecdsa_key.pub
├── ssh_host_rsa_key
└── ssh_host_rsa_key.pub

# ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key
2048 d9:0d:d0:de:06:43:74:cc:70:81:a8:c6:c6:67:4e:ed   (RSA)

# ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key.pub
```

```
2048 d9:0d:d0:de:06:43:74:cc:70:81:a8:c6:c6:67:4e:ed   (RSA)

# cat /etc/ssh/ssh_host_rsa_key
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEArYxD03dhJ1eSfJps1YLXr4fHI5ewYu/F8995eQXXvTWTtNus
…
mf6Jz29EAU5WsU12lPhIT61w2/ytB00VBErixbVtEXcxJr/MCOCygg==
-----END RSA PRIVATE KEY-----

# cat /etc/ssh/ssh_host_rsa_key.pub
ssh-rsaAAAB3NzaC1yc2EAAAADAQABAAABAQCtjEPTd2EnV5J8mmzVgtevh8cjI7Bi78Xz33
I5Bde9NefpifmkIxcvpoer7zxj,cnXb/es5puCg6FRxRKYD8sKgd9mJAIind7ZJI+JosmUxneDt4xU
JPgjnbf4yrTL75TZ3FAQXKUpobKJwpVGuEVXtfB4PhfH9KizBZepmhPnGaYVi+Zr0YrrmS6Pa
FwSZl8+T3aXKfedgz1uuiiwCWHhrIakDCCU7GRBREHDquVuKxmyiHFEqjHM+mDmi9PfWsk
baYFEd+bAPr2CrDxhaJYscHnJTBJjWzUWlxuZKW9lKpsJZm
```

```
# ssh root@j01
The authenticity of host 'j01 (192.168.3.1)' can't be established.
RSA key fingerprint is c9:43:e3:fa:c4:e3:29:85:bd:a8:a5:bc:85:86:e2:85.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'j01,192.168.3.1' (RSA) to the list of known hosts.
root@j01's password:
Last login: Wed Oct 22 11:31:14 2014
[root@j01 ~]#

# cat .ssh/known_hosts
gandhi,192.168.0.10ecdsa-sha2-nistp256AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbml
zdHAyNTYAAABBBP+FoXB3XW0BzxIgmjVWhrMXKv5cXj9CwFi3PAGmufB6/Rez8CDCIDBV
wxjeyELJy8cnL7vkXRpaVaQWtNjsLTY=
j01,192.168.3.1ssh-rsaAAAAB3NzaC1yc2EAAAADAQABAAABAQC9fvpZNYiIYa1iremp6OLd
oe7x1yI62oevtAYMHFOdGgx2hrtcftnbWKHXKeXDHDlkV9lVOOFAIYZiQP6bGy0P2W6Lo/Hn
6D2yJovxP6tD55lvnU+7dfXsxOFtDIrZD0znkMltM+NxLsS1hqoNnCxjh1DoUsnQHnmBpywozu
z3PF8pGW3I1Wm+/qpnORJhTDARKt2xQO36DX3eCiRCIjeXQLn6yaiOa43/zmLcEg6ZT3pB8
091NnM4y7Q9SpPY2UpEILIh/hrBf9Zfh8zbewFlyvJHULWTpR2EWFbBVoAnVdCSoXhXJ04R
Rb5e1nz8k0dZmw600d3bMrOWqiYn2dEj

# ssh-keygen -lv -f .ssh/known_hosts
256 70:ad:34:d9:17:c7:e7:e3:06:90:e1:8d:c2:fe:cc:57 gandhi,192.168.0.10 (ECDSA)
+--[ECDSA  256 ]---+
|          .+..         |
|          .+.ooo. . |
|          . =oooo. o |
|          +.o.. . ..    |
|          S.        oE.|
|          +   .o        |
|          + ..       |
|        .        |
|                 |
+-----------------+
2048 c9:43:e3:fa:c4:e3:29:85:bd:a8:a5:bc:85:86:e2:85 j01,192.168.3.1 (RSA)
+--[ RSA 2048]----+
|                 |
|                 |
|        o        |
|       + o       |
|        oS       |
|  .. ..oo.       |
|.E..o +o+.       |
|...o +o+.o       |
| .  =o .+        |
+-----------------+

# ssh -o "VisualHostKey yes" root@j01
Host key fingerprint is c9:43:e3:fa:c4:e3:29:85:bd:a8:a5:bc:85:86:e2:85
+--[ RSA 2048]----+
|                 |
|                 |
|        o        |
|       + o       |
|        oS       |
|  .. ..oo.       |
|.E..o +o+.       |
|...o +o+.o       |
| .  =o .+        |
+-----------------+
root@j01's password:
```

Man-in-the.middle Attack

Si la clau pública d'un host remot es modifica, quan el client hi connecta no coincieix la clau del client emmagatzemana al *.ssh/known-hosts* amb la que rep del host remot. Això pot ser simplement perquè s'ha canviat el host remot per un de nou, s'ha reinstal·lat el sistema… o simplement s'han generat unes noves claus.

Però també pot ser que es produeixi un atac de man-in-the-middle, és a dir, algun altre host està impostant l'identitat del host remot real. Si el sistema està en mode "**StrictMode yes**" no permetrà connexions quan no coincideixi la clau pública remota rebuda amb la emmagatzemanda al known_hosts.

Un exemple és editar manualment el fitxer .ssh/known_hosts i fer trampa i variar les ips entre dues entrades que hi hagi. En connectar amb alguna d'aquestes entrades 'petarà'.

```
$ ssh pere@n1
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@       WARNING: POSSIBLE DNS SPOOFING DETECTED!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The RSA host key for n1 has changed,
and the key for the corresponding IP address 192.168.1.15
is unchanged. This could either mean that
DNS SPOOFING is happening or the IP address for the host
and its host key have changed at the same time.
Offending key for IP in /home/pere/.ssh/known_hosts:5
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@       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 a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
c9:c6:24:c9:e1:f9:21:00:3d:4f:74:2c:7c:dc:97:d9.
Please contact your system administrator.
Add correct host key in /home/pere/.ssh/known_hosts to get rid of this message.
Offending RSA key in /home/pere/.ssh/known_hosts:7
RSA host key for n1 has changed and you have requested strict checking.
Host key verification failed.
```

Accés SSH desatès

Existeixen dos tipus d'accés desatès:

A. **Deprecated! Host Authentiction.** Autenticar el host client propagant la seva clau pública als hosts remots. Això permet que qualsevol usuari del host client tingui accés automàtic (sense password) al host remot. Evidentment és molt perillós, una maquina compromesa per un assaltant es pot fer amb el control de les altres màquines.

B. **User publicKey Authentication**. Propagar la clau pública d'un usuari als comptes remots als que vol accedir sense necessitat d'autenticar-se usant password.

PublicKey Authentication

Descripció del procediment per realitzar autenticació d'usuari utilitzant publickey en lloc de password. Això permet fer autenticació desatesa (automatitzada sense entrar el password) i per tant realitzar comandes remotes per exemple en scripts.

- Crear les claus d'usuari amb l'ordre *ssh-keygen*.
- Copiar la clau publica de l'usuari client al fitxer *.ssh/authorized_keys* del compte de l'usuari/host remot al que es vol connectar de manera desatesa. Usar l'ordre *ssh-copy-id*.

ssh-keygen:

```
SSH-KEYGEN(1)                          BSD General Commands Manual

NAME
        ssh-keygen — authentication key generation, management and conversion

DESCRIPTION
        ssh-keygen generates, manages and converts authentication keys for ssh(1).
ssh-keygen can create RSA keys for use by SSH protocol version 1 and DSA, ECDSA or
RSA keys for use by SSH protocol version 2.  The type of key to be generated is specified
with the -t option. If invoked without any arguments, ssh-keygen will generate an RSA key for
use in SSH protocol 2 connections.

        ssh-keygen is also used to generate groups for use in Diffie-Hellman group exchange
(DH-GEX).  See the MODULI GENERATION section for details.
```

Finally, ssh-keygen can be used to generate and update Key Revocation Lists, and to test whether given keys have been revoked by one. See the KEY REVOCATION LISTS section for details.

Normally each user wishing to use SSH with public key authentication runs this once to create the authentication key in ~/.ssh/identity, ~/.ssh/id_ecdsa, ~/.ssh/id_dsa or *~/.ssh/id_rsa*. Additionally, the system administrator may use this to generate host keys, as seen in /etc/rc.

Normally this program generates the key and asks for a file in which to store the private key. The public key is stored in a file with the same name but "*.pub*" appended. The program also asks for a *passphrase*. The passphrase may be empty to indicate no passphrase (host
keys must have an empty passphrase), or it may be a string of arbitrary length. A passphrase is similar to a password, except it can be a phrase with a series of words, punctuation, numbers, whitespace, or any string of characters you want. Good passphrases are 10-30 characters long, are not simple sentences or otherwise easily guessable (English prose has only 1-2 bits of entropy per character, and provides very bad passphrases), and contain a mix of upper and lowercase letters, numbers, and non-alphanumeric characters. The passphrase can be changed later by using the -p option.

There is no way to recover a lost passphrase. If the passphrase is lost or forgotten, a new key must be generated and the corresponding public key copied to other machines.

For RSA1 keys, there is also a comment field in the key file that is only for convenience to the user to help identify the key. The comment can tell what the key is for, or whatever is useful. The comment is initialized to "user@host" when the key is created, but can be
changed using the -c option.

After a key is generated, instructions below detail where the keys should be placed to be activated.

ssh-keygen supports two types of certificates: user and host. User certificates authenticate users to servers, whereas host certificates authenticate server hosts to users. To generate a user certificate:

```
$ ssh-keygen -s /path/to/ca_key -I key_id /path/to/user_key.pub
```

The resultant certificate will be placed in /path/to/user_key-cert.pub. A host certificate requires the -h option:

```
$ ssh-keygen -s /path/to/ca_key -I key_id -h /path/to/host_key.pub
```

The host certificate will be output to /path/to/host_key-cert.pub.

It is possible to sign using a CA key stored in a PKCS#11 token by providing the token library using -D and identifying the CA key by providing its public half as an argument to -s:

```
$ ssh-keygen -s ca_key.pub -D libpkcs11.so -I key_id host_key.pub
```

In all cases, key_id is a "key identifier" that is logged by the server when the certificate is used for authentication.

Certificates may be limited to be valid for a set of principal (user/host) names. By default, generated certificates are valid for all users or hosts. To generate a certificate for a specified set of principals:

```
$ ssh-keygen -s ca_key -I key_id -n user1,user2 user_key.pub
$ ssh-keygen -s ca_key -I key_id -h -n host.domain user_key.pub
```

Additional limitations on the validity and use of user certificates may be specified through certificate options. A certificate option may disable features of the SSH session, may be valid only when presented from particular source addresses or may force the use of a specific command. For a list of valid certificate options, see the documentation for the -O option above.

FILES

~/.ssh/identity

Contains the protocol version 1 RSA authentication identity of the user. This file should not be readable by anyone but the user. It is possible to specify a passphrase when generating the key; that passphrase will be used to encrypt the private part of this file using 3DES. This file is not automatically accessed by ssh-keygen but it is offered as the default file for the private key. ssh(1) will read this file when a login attempt is made.

~/.ssh/identity.pub

Contains the protocol version 1 RSA public key for authentication. The contents of this file should be added to ~/.ssh/authorized_keys on all machines where the user wishes to log in using RSA authentication. There is no need to keep the contents of this file secret.

~/.ssh/id_dsa
~/.ssh/id_ecdsa
~/.ssh/id_rsa

Contains the protocol version 2 DSA, ECDSA or RSA authentication identity of the user. This file should not be readable by anyone but the user. It is possible to specify a passphrase when generating the key; that passphrase will be used to encrypt the private part of this file using 128-bit AES. This file is not automatically accessed by ssh-keygen but it is offered as the default file for the private key. ssh(1) will read this file when a login attempt is made.

~/.ssh/id_dsa.pub
~/.ssh/id_ecdsa.pub
~/.ssh/id_rsa.pub

Contains the protocol version 2 DSA, ECDSA or RSA public key for authentication. The contents of this file should be added to *~/.ssh/authorized_keys* on all machines where the user wishes to log in using public key authentication. There is no need to keep the contents of this file secret.

/etc/ssh/moduli
Contains Diffie-Hellman groups used for DH-GEX. The file format is described in moduli(5).

… man ssh(1)

Public key authentication works as follows: The scheme is based on public-key cryptography, using cryptosystems where encryption and decryption are done using separate keys, and it is unfeasible to derive the decryption key from the encryption key. The idea is that each user creates a public/private key pair for authentication purposes. The server knows the public key, and only the user knows the private key. *ssh implements public key authentication protocol automatically*, using one of the DSA, ECDSA or RSA algorithms. Protocol 1 is restricted to using only RSA keys, but protocol 2 may use any. The HISTORY section of ssl(8)
contains a brief discussion of the DSA and RSA algorithms.

The file ~/.ssh/authorized_keys lists the public keys that are permitted for logging in. When the user logs in, the ssh program tells the server which key pair it would like to use for authentication. The client proves that it has access to the private key and the server checks that the corresponding public key is authorized to accept the account.

The user creates his/her key pair by running ssh-keygen(1). This stores the private key in *~/.ssh/identity* (protocol 1), *~/.ssh/id_dsa* (protocol 2 DSA), *~/.ssh/id_ecdsa* (protocol 2 ECDSA), or *~/.ssh/id_rsa* (protocol 2 RSA) and stores the public key in ~/.ssh/identity.pub (protocol 1), ~/.ssh/id_dsa.pub (protocol 2 DSA), ~/.ssh/id_ecdsa.pub (protocol 2 ECDSA), or *~/.ssh/id_rsa.pub* (protocol 2 RSA) in the user's home directory. The user should then copy the public key to *~/.ssh/authorized_keys* in his/her home directory on the remote machine. The authorized_keys file corresponds to the conventional ~/.rhosts file, and has one key per line, though the lines can be very long. After this, the user can log in without giving the password.

A variation on public key authentication is available in the form of certificate authentication: instead of a set of public/private keys, signed certificates are used. This has the advantage that a single trusted certification authority can be used in place of many public/private keys. See the CERTIFICATES section of ssh-keygen(1) for more information.

The most convenient way to use public key or certificate authentication may be with an *authentication agent*. See ssh-agent(1) for more information.

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/pere/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/pere/.ssh/id_rsa.
Your public key has been saved in /home/pere/.ssh/id_rsa.pub.
The key fingerprint is:
b4:63:b4:05:3d:97:05:d2:13:a6:d8:49:05:da:a8:00 pere@portatil.localdomain
The key's randomart image is:
+--[ RSA 2048]----+
|  E      ..++==.  |
|   .       Oo==   |
|           .   * *o .   |
|           . + +     |
|           . S       |c
|           . .       |
|                     |
|                     |
|                     |
+-----------------+

[pere@portatil ~]$$ tree .ssh
.ssh
|-- id_rsa
|-- id_rsa.pub
`-- known_hosts

[pere@portatil ~]$ scp .ssh/id_rsa.pub pere@n1:.ssh/authorized_keys
pere@n1's password:
id_rsa.pub                                        100%  407      0.4KB/s   00:00

[pere@netbook1 ~]$ tree .ssh/
.ssh/
`-- authorized_keys

q:q[pere@portatil ~]$ ssh pere@n1
Last login: Sat Oct 25 00:18:15 2014 from as1
```

Exemple del procés seguit amb la connexió desatesa SSH:

```
[pau@portatil ~]$ ssh -v pau@n1
OpenSSH_5.9p1, OpenSSL 1.0.0k-fips 5 Feb 2013
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: /etc/ssh/ssh_config line 50: Applying options for *
debug1: Connecting to n1 [192.168.1.36] port 22.
debug1: Connection established.
```

```
debug1: identity file /home/pau/.ssh/id_rsa type 1
debug1: identity file /home/pau/.ssh/id_rsa-cert type -1
debug1: identity file /home/pau/.ssh/id_dsa type -1
debug1: identity file /home/pau/.ssh/id_dsa-cert type -1
debug1: Remote protocol version 2.0, remote software version OpenSSH_5.4
debug1: match: OpenSSH_5.4 pat OpenSSH*
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_5.9
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-ctr hmac-md5 none
debug1: kex: client->server aes128-ctr hmac-md5 none
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
debug1: Server host key: RSA c9:c6:24:c9:e1:f9:12:00:2d:4f:74:2c:7c:dc:97:d9
debug1: Host 'n1' is known and matches the RSA host key.
debug1: Found key in /home/pau/.ssh/known_hosts:1
debug1: ssh_rsa_verify: signature correct
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
debug1: Authentications that can continue: publickey,gssapi-keyex,gssapi-with-mic,password
debug1: Next authentication method: gssapi-keyex
debug1: No valid Key exchange context
debug1: Next authentication method: gssapi-with-mic
debug1: Unspecified GSS failure.  Minor code may provide more information
Credentials cache file '/tmp/krb5cc_1003' not found
debug1: Unspecified GSS failure.  Minor code may provide more information
Credentials cache file '/tmp/krb5cc_1003' not found
debug1: Unspecified GSS failure.  Minor code may provide more information
debug1: Unspecified GSS failure.  Minor code may provide more information
Credentials cache file '/tmp/krb5cc_1003' not found
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /home/pau/.ssh/id_rsa
debug1: Server accepts key: pkalg ssh-rsa blen 279
debug1: read PEM private key done: type RSA
debug1: Authentication succeeded (publickey).
Authenticated to n1 ([192.168.1.36]:22).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: Sending environment.
```

```
debug1: Sending env LANG = ca_ES.utf8
Last login: Sat Oct 25 00:19:02 2014 from as1
[pau@netbook1 ~]$
```

Exemple de passos d’una connexió fallida perquè al servidor no s’ha configurat apropiadament la directiva *PasswordAuthentication=no*:

```
[pere@portatil ~]$ ssh -v -o PasswordAuthentication=no pere@n1
OpenSSH_5.9p1, OpenSSL 1.0.0k-fips 5 Feb 2013
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: /etc/ssh/ssh_config line 50: Applying options for *
debug1: Connecting to n1 [192.168.1.36] port 22.
debug1: Connection established.
debug1: identity file /home/pere/.ssh/id_rsa type 1
debug1: identity file /home/pere/.ssh/id_rsa-cert type -1
debug1: identity file /home/pere/.ssh/id_dsa type -1
debug1: identity file /home/pere/.ssh/id_dsa-cert type -1
debug1: Remote protocol version 2.0, remote software version OpenSSH_5.4
debug1: match: OpenSSH_5.4 pat OpenSSH*
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_5.9
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-ctr hmac-md5 none
debug1: kex: client->server aes128-ctr hmac-md5 none
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
debug1: Server host key: RSA c9:c6:24:c9:e1:f9:12:00:2d:4f:74:2c:7c:dc:97:d9
debug1: Host 'n1' is known and matches the RSA host key.
debug1: Found key in /home/pere/.ssh/known_hosts:1
debug1: ssh_rsa_verify: signature correct
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
debug1: Authentications that can continue: publickey,gssapi-keyex,gssapi-with-mic,password
debug1: Next authentication method: gssapi-keyex
debug1: No valid Key exchange context
debug1: Next authentication method: gssapi-with-mic
debug1: Unspecified GSS failure.  Minor code may provide more information
Credentials cache file '/tmp/krb5cc_1001' not found
```

```
debug1: Unspecified GSS failure.  Minor code may provide more information
Credentials cache file '/tmp/krb5cc_1001' not found

debug1: Unspecified GSS failure.  Minor code may provide more information


debug1: Unspecified GSS failure.  Minor code may provide more information
Credentials cache file '/tmp/krb5cc_1001' not found

debug1: Next authentication method: publickey
debug1: Offering RSA public key: /home/pere/.ssh/id_rsa
debug1: Authentications that can continue: publickey,gssapi-keyex,gssapi-with-mic,password
debug1: Trying private key: /home/pere/.ssh/id_dsa
debug1: No more authentication methods to try.
Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).
```

HostKey Authentication (Deprecated)

Autenticar el host client propagant la seva clau pública als hosts remots. Això permet que qualsevol usuari del host client tingui accés automàtic (sense password) al host remot. Evidentment és molt perillós, una maquina compromesa per un assaltant es pot fer amb el control de les altres màquines.

```
Suppose you want to allow the account nocnoc@supplicant.foo.net access to
whosthere@server.foo.net.
Then:
1. Make sure hostbased authentication enabled in on server.foo.net:
/etc/ssh/sshd_config:
HostbasedAuthentication yes
IgnoreRhosts no
and optionally (see "Discussion"):
HostbasedUsesNameFromPacketOnly yes
and restart sshd.
2. Ensure that the ssh-keysign program is setuid root on the client machine. The file is usually
located
in /usr/libexec or /usr/libexec/openssh:
$ ls -lo /usr/libexec/openssh/ssh-keysign
-rwsr-xr-x
1 root
222936 Mar 7 16:09 /usr/libexec/openssh/ssh-keysign
3. Enable trusted host authentication in your system's client configuration file: [Recipe 6.12]
/etc/ssh/ssh_config:
Host remotehost
```

HostName remotehost
HostbasedAuthentication yes
4. Insert the client machine's host keys, /etc/ssh/ssh_host_dsa_key.pub and /etc/ssh/ ssh_host_rsa_key.pub, into the server's known hosts database, /etc/ssh/ssh_known_hosts , using
the client host's canonical name (supplicant.foo.net here; see "Discussion"):
/etc/ssh/ssh_known_hosts on server.foo.net:
supplicant.foo.net ssh-dss ...key...
5. Authorize the client account to log into the server, by creating the file ~/.shosts:
~whosthere/.shosts on server.foo.net:
supplicant.foo.net nocnoc
If the account names on the client and server hosts happen to be the same, you can omit the username. (But in this case the usernames are different, nocnoc and whosthere.)
6. Make sure your home directory and .shosts files have acceptable permissions:
$ chmod go-w ~
$ chmod go-w ~/.shosts
7. Log in from supplicant.foo.net:
$ ssh -l whosthere server.foo.net

Utilitats SSH Client

Utilitats ssh client (*openldap-clients*):

- SSH
- SCP
- SFTP
- SSHFS

scp

```
NAME
        scp — secure copy (remote file copy program)

SYNOPSIS
        scp [-12346BCpqrv] [-c cipher] [-F ssh_config] [-i identity_file] [-l limit] [-o ssh_option]
              [-P port] [-S program] [[user@]host1:]file1 ... [[user@]host2:]file2

DESCRIPTION
        scp copies files between hosts on a network.  It uses ssh(1) for data transfer, and uses
the same authentication and provides the same security as ssh(1).  Unlike rcp(1), scp will ask
for passwords or passphrases if they are needed for authentication.

        File names may contain a user and host specification to indicate that the file is to be
copied to/from that host.  Local file names can be made explicit using absolute or relative
pathnames to avoid scp treating file names containing ‘:’ as host specifiers.

        Copies between two remote hosts are also permitted.

        When copying a source file to a target file which already exists, scp will replace the
contents of the target file (keeping the inode).

        If the target file does not yet exist, an empty file with the target file name is created,
then filled with the source file contents.  No attempt is made at "near-atomic" transfer using
temporary files.
```

sftp

```
NAME
        sftp — secure file transfer program

SYNOPSIS
```

```
          sftp [-1246Cpqrv] [-B buffer_size] [-b batchfile] [-c cipher] [-D sftp_server_path]
               [-F ssh_config] [-i identity_file] [-l limit] [-o ssh_option] [-P port] [-R num_requests]
               [-S program] [-s subsystem | sftp_server] host
          sftp [user@]host[:file ...]
          sftp [user@]host[:dir[/]]
          sftp -b batchfile [user@]host

DESCRIPTION
          sftp is an interactive file transfer program, similar to ftp(1), which performs all
operations over an encrypted ssh(1) transport.  It may also use many features of ssh, such as
public key authentication and compression.  sftp connects and logs into the specified host,
then enters an interactive command mode.

          The second usage format will retrieve files automatically if a non-interactive
authentication method is used; otherwise it will do so after successful interactive
authentication.

          The third usage format allows sftp to start in a remote directory.

          The final usage format allows for automated sessions using the -b option.  In such
cases, it is necessary to configure non-interactive authentication to obviate the need to enter
a password at connection time (see sshd(8) and ssh-keygen(1) for details).

          Since some usage formats use colon characters to delimit host names from path
names, IPv6 addresses must be enclosed in square brackets to avoid ambiguity.
```

sshfs

```
NAME
          SSHFS - filesystem client based on ssh

SYNOPSIS
mounting
          sshfs [user@]host:[dir] mountpoint [options]

unmounting
          fusermount -u mountpoint

DESCRIPTION
SSHFS  (Secure  SHell  FileSystem)  is a file system for Linux (and other operating systems
with a FUSE implementation, such as Mac OS X or FreeBSD) capable of operating on files on
a remote computer using just a secure shell login on the remote computer. On the  local
computer where  the SSHFS is mounted, the implementation makes use of the FUSE
(Filesystem in Userspace) kernel module. The practical effect of this is that the end user can
```

seamlessly interact with remote files being securely served over SSH just as if they were local files on his/her computer. On the remote computer the SFTP subsystem of SSH is used.

-o allow_root
allow access to root

-o nonempty
allow mounts over non-empty file/dir

```
[user@hp01 ~]$ whereis sshfs
sshfs: /usr/bin/sshfs /usr/share/man/man1/sshfs.1.gz

[user@hp01 ~]$ rpm -qf /usr/bin/sshfs
Fuse-sshfs-2.5-1.fc21.x86_64

[user@hp01 ~]$ rpm -ql fuse-sshfs
/usr/bin/sshfs
/usr/share/doc/fuse-sshfs
/usr/share/doc/fuse-sshfs/AUTHORS
/usr/share/doc/fuse-sshfs/COPYING
/usr/share/doc/fuse-sshfs/ChangeLog
/usr/share/doc/fuse-sshfs/FAQ.txt
/usr/share/doc/fuse-sshfs/NEWS
/usr/share/doc/fuse-sshfs/README
/usr/share/man/man1/sshfs.1.gz
```

```
[user1]$ sshfs pere@localhost:. /tmp/dir1/

[user1]$ mount
…
pere@localhost:. on /tmp/dir1 type fuse.sshfs
(rw,nosuid,nodev,relatime,user_id=1000,group_id=1000)

[user1]$ fusermount -u /tmp/dir1
```

Altres utilitats SSH Client

Altres utilitats SSH client:

- slogin
- ssh-add
- ssh-agent
- ssh-copy-id
- ssh-key-scan

slogin

```
$ ls -l /usr/bin/slogin
lrwxrwxrwx 1 root root 5 sep 22 15:41 /usr/bin/slogin -> ./ssh
```

ssh-agent

```
NAME
        ssh-agent — authentication agent

SYNOPSIS
        ssh-agent [-c | -s] [-d] [-a bind_address] [-t life] [command [arg ...]]
        ssh-agent [-c | -s] -k

DESCRIPTION
        ssh-agent is a program to hold private keys used for public key authentication (RSA,
DSA, ECDSA).  The idea is that ssh-agent is started in the beginning of an X-session or a
login session, and all other windows or programs are started as clients to the ssh-agent
program.  Through use of environment variables the agent can be located and automatically
used for authentication when logging in to other machines using ssh(1).
```

```
[pere@hp01 ~]$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-cs1MLiuO8ipD/agent.3409; export SSH_AUTH_SOCK;
SSH_AGENT_PID=3410; export SSH_AGENT_PID;
echo Agent pid 3410;

[pere@hp01 ~]$ SSH_AUTH_SOCK=/tmp/ssh-cs1MLiuO8ipD/agent.3409; export
SSH_AUTH_SOCK;
[pere@hp01 ~]$ SSH_AGENT_PID=3410; export SSH_AGENT_PID;
```

```
[pere@hp01 ~]$ ll /tmp/ssh-cs1MLiuO8ipD/agent.3409
srw------- 1 pere pere 0 14 nov 17:48 /tmp/ssh-cs1MLiuO8ipD/agent.3409

[pere@hp01 ~]$ ssh-agent -k
unset SSH_AUTH_SOCK;
unset SSH_AGENT_PID;
echo Agent pid 3410 killed;
```

```
[pere@portatil ~]$ ssh-agent gnome-terminal &

En el gnome-terminal:
[pere@portatil ~]$ ssh-add
Identity added: /home/pere/.ssh/id_rsa (/home/pere/.ssh/id_rsa)

[pere@portatil ~]$ ssh-add -l
2048 b3:ba:cd:46:9b:ae:ca:07:a9:5e:b8:97:ed:54:ce:d2 /home/pere/.ssh/id_rsa (RSA)

[pere@portatil ~]$ env | grep SSH
SSH_AGENT_PID=4898
SSH_AUTH_SOCK=/tmp/ssh-KuoVdMIn4897/agent.4897
```

ssh-add

```
NAME
        ssh-add — adds private key identities to the authentication agent

SYNOPSIS
        ssh-add [-cDdkLlXx] [-t life] [file ...]
        ssh-add -s pkcs11
        ssh-add -e pkcs11

DESCRIPTION
        ssh-add adds private key identities to the authentication agent, ssh-agent(1).  When
run without arguments,
        it adds the files ~/.ssh/id_rsa, ~/.ssh/id_dsa, ~/.ssh/id_ecdsa and ~/.ssh/identity.  After
loading a private key, ssh-add will try to load corresponding certificate information from the
filename obtained by oppending -cert.pub to the name of the private key file.  Alternative file
names can be given on the command line.
        If any file requires a passphrase, ssh-add asks for the passphrase from the user.  The
passphrase is read from the user's tty.  ssh-add retries the last passphrase if multiple identity
files are given.
```

The authentication agent must be running and the SSH_AUTH_SOCK environment variable must contain the name of its socket for ssh-add to work.

```
[pere@hp01 ~]$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-yLbqiQ2Up7qI/agent.3476; export SSH_AUTH_SOCK;
SSH_AGENT_PID=3477; export SSH_AGENT_PID;
echo Agent pid 3477;
[pere@hp01 ~]$ SSH_AUTH_SOCK=/tmp/ssh-yLbqiQ2Up7qI/agent.3476; export
SSH_AUTH_SOCK;
[pere@hp01 ~]$ SSH_AGENT_PID=3477; export SSH_AGENT_PID;

# pere te una clau privada protegida amb passfrase
[pere@hp01 ~]$ ssh-add
Enter passphrase for /home/pere/.ssh/id_rsa:
Identity added: /home/pere/.ssh/id_rsa (/home/pere/.ssh/id_rsa)

[pere@hp01 ~]$ ssh-copy-id marta@localhost
/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are
already installed
/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install
the new keys
marta@localhost's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'marta@localhost'"
and check to make sure that only the key(s) you wanted were added.

[pere@hp01 ~]$ ssh marta@localhost
Last login: Mon Nov 14 17:52:21 2016 from localhost.localdomain
message of the day!
```

```
[marta@hp01 ~]$ logout
Connection to localhost closed.

[pere@hp01 ~]$ ssh-agent -k
unset SSH_AUTH_SOCK;
unset SSH_AGENT_PID;
echo Agent pid 3477 killed;

# un cop no està la passfrase en el ssh-agent, cal escriure-la per poder usar la
# clau privada
[pere@hp01 ~]$ ssh marta@localhost
Enter passphrase for key '/home/pere/.ssh/id_rsa':
Last login: Mon Nov 14 17:54:00 2016 from localhost.localdomain
```

```
message of the day!
```

ssh-copy-id

```
NAME
        ssh-copy-id — use locally available keys to authorise logins on a remote machine

SYNOPSIS
        ssh-copy-id [-n] [-i [identity_file]] [-p port] [-o ssh_option] [user@]hostname
        ssh-copy-id -h | -?

DESCRIPTION
        ssh-copy-id is a script that uses ssh(1) to log into a remote machine (presumably
using a login password, so password authentication should be enabled, unless you've done
some clever use of multiple identities).
        It assembles a list of one or more fingerprints (as described below) and tries to log in
with each key, to see if any of them are already installed (of course, if you are not using
ssh-agent(1) this may result in you being repeatedly prompted for pass-phrases).  It then
assembles a list of those that failed to log in, and using ssh, enables logins with those keys on
the remote server.  By default it adds the keys by appending them to the remote user's
~/.ssh/authorized_keys (creating the file, and directory, if necessary).  It is also capable of
detecting if the remote system is a NetScreen, and using its ‘set ssh pka-dsa key ...’
command instead.
```

```
# afegir la clau pública de pere (totes les que tingui) al compte de pere a j01
[pere@portatil]$ ssh-copy-id pere@j01

# afegir la clau pública de pere (totes les que tingui) al compte de marta a j01
[pere@portatil]$ ssh-copy-id marta@j01

# afegir la clau pública DSA de pere al compte de pere a j01
[pere@portatil]$ ssh-copy-id -i .ssh/id_dsa.pub pere@j01
```

ssh-keyscan

```
NAME
        ssh-keyscan — gather ssh public keys

SYNOPSIS
        ssh-keyscan [-46Hv] [-f file] [-p port] [-T timeout] [-t type] [host | addrlist namelist] ...
```

DESCRIPTION

ssh-keyscan is a utility for gathering the public ssh host keys of a number of hosts. It was designed to aid in building and verifying ssh_known_hosts files. ssh-keyscan provides a minimal interface suitable for use by shell and perl scripts.

ssh-keyscan uses non-blocking socket I/O to contact as many hosts as possible in parallel, so it is very efficient. The keys from a domain of 1,000 hosts can be collected in tens of seconds, even when some of those hosts are down or do not run ssh. For scanning, one does not need login access to the machines that are being scanned, nor does the scanning process involve any encryption.

```
# ssh-keyscan j01
# j01 SSH-2.0-OpenSSH_6.3
no hostkey alg
# j01 SSH-2.0-OpenSSH_6.3
j01 ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC9fvpZNYiIYa1iremp6OLdoe7x1yI62oevtAYMHF
OdGgx2hrtcftnbWKHXKeXDHDlkV9lVOOFAIYZiQP6bGy0P2W6Lo/Hn6D2yJovxP6tD55IvnU
+7dfXsxOFtDIrZD0znkMItM+NxLsS1hqoNnCxjh1DoUsnQHnmBpywozuz3PF8pGW3I1Wm+/q
pnORJhTDARKt2xQO36DX3eCiRCIjeXQLn6yaiOa43/zmLcEg6ZT3pB8091NnM4y7Q9SpPY
2UpEILIh/hrBf9Zfh8zbewFIyvJHULWTpR2EWFbBVoAnVdCSoXhXJ04RRb5e1nz8k0dZmw6
00d3bMrOWqiYn2dEj

# ssh-keyscan localhost
# localhost SSH-2.0-OpenSSH_6.4
localhost ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCtjEPTd2EnV5J8mmzVgtevh8cjI7Bi78Xz33I5Bde
9NZO026wsm+fdXzFTnXbnstFY/pe4GmMrUs6Hhz5+5Q9tWMaahL9swWXOpUFnXb/es5pu
Cg6FRxRKYD8sKgd9mJAIind7ZJI+JosmUxneDt4xUJPgjnbf4yrTL75TZ3FAQXKUpobKJwpV
GuEVXtfB4PhfH9KizBZepmhPnGaYVi+Zr0YrrmS6PaFwSZI8+T3aXKfedgz1uuiiwCWHhrIakD
CCU7GRBREHDquVuKxmyiHFEqjHM+mDmi9PfWskbaYFEd+bAPr2CrDxhaJYscHnJTBJjWz
UWlxuZKW9lKpsJZmp
# localhost SSH-2.0-OpenSSH_6.4
localhost ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBFlDyAROZjhNkldbyT
Edly//FZclH76de6sXtQZNv8g4buaNlcXanijFzeBGzpxD519XsqkiYaQMMYZhJTE0apY=

# ssh-keyscan j08
# j08 SSH-2.0-OpenSSH_6.4
j08 ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDZYDQesdI8IxSq1cp7SfBL2uKg7/RDibLpuNcJE
cPqI9K8V7sO5/FveuFMhq036t0Rxse1bssWCH/8kFaAFBfGKyrurIIIFK/XA0T71jYnI8dn1SGm
5KtgKxKZK4mMYaqdG6XtGO43/71yp+T2JVN+gIdK83tgy4NOEhCShnntybctcJ5JA1HBAMkF
3DRHY2kvI/iiiLXIzZYw8mBO42d8H8wFUCYtjFxC7fmKcfEkxPfkRYXCiP1JVr0uysf+vqehQro
UcI/i2zTNhvthm1mtWjv9mSDqw7FQm1IO1KUT4IT51QMTFp1VpI0ckx66A7xR7dswJWtTobn
QvJtbc+E7
```

```
# j08 SSH-2.0-OpenSSH_6.4
j08 ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBIVhIHgOy6A4ycdtBzi
OmbesIY7Z2S3YIq4+hVIdtbhVHkGm7soMUYmFmIY5uBwbhYqkLPH42ZQzal2X/Dm/HWE=

# ssh root@j08 "cat /etc/ssh/ssh_host_rsa_key.pub"
root@j08's password:
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDZYDQesdI8IxSq1cp7SfBL2uKg7/RDibLpuNcJE
cPql9K8V7sO5/FveuFMhq036t0Rxse1bssWCH/8kFaAFBfGKyrurIIIFK/XA0T71jYnI8dn1SGm
5KtgKxKZK4mMYaqdG6XtGO43/71yp+T2JVN+gldK83tgy4NOEhCShnntybctcJ5JA1HBAMkF
3DRHY2kvI/iiiLXIzZYw8mBO42d8H8wFUCYtjFxC7fmKcfEkxPfkRYXCiP1JVr0uysf+vqehQro
UcI/i2zTNhvthm1mtWjv9mSDqw7FQm1IO1KUT4IT51QMTFp1VpI0ckx66A7xR7dswJWtTobn
QvJtbc+E7
```

Forwarding

En aquest apartat es mostra l'informació de:

- TCP Forwarding, que es tracta al mòdul M11 a l'apartat de túnnels ssh i VPNs.
- X11 Forwarding: visualitzar localment aplicacions grafiques que s'executen remotament.

TCP Forwarding:

Tip: TCP fowrarding s'explica al modul M11 de ASIX conjuntament amb túnels SSH i VPN.

TCP FORWARDING

Forwarding of arbitrary TCP connections over the secure channel can be specified either on the command line or in a configuration file. One possible application of TCP forwarding is a secure connection to a mail server; another is going through firewalls.

In the example below, we look at encrypting communication between an IRC client and server, even though the IRC server does not directly support encrypted communications. This works as follows: the user connects to the remote host using ssh, specifying a port to be used to forward connections to the remote server. After that it is possible to start the service which is to be encrypted on the client machine, connecting to the same local port, and ssh will encrypt and forward the connection.

The following example tunnels an IRC session from client machine "127.0.0.1" (localhost) to remote server "server.example.com":

```
$ ssh -f -L 1234:localhost:6667 server.example.com sleep 10
$ irc -c '#users' -p 1234 pinky 127.0.0.1
```

This tunnels a connection to IRC server "server.example.com", joining channel "#users", nickname "pinky", using port 1234. It doesn't matter which port is used, as long as it's greater than 1023 (remember, only root can open sockets on privileged ports) and doesn't conflict with any ports already in use. The connection is forwarded to port 6667 on the remote server, since that's the standard port for IRC services.

The -f option backgrounds ssh and the remote command "sleep 10" is specified to allow an amount of time (10 seconds, in the example) to start the service which is to be tunnelled. If no connections are made within the time specified, ssh will exit.

X11 Forwarding

X11 FORWARDING

If the *ForwardX11* variable is set to “yes” (or see the description of the -X, -x, and -Y options above) and the user is using X11 (the DISPLAY environment variable is set), the connection to the X11 display is automatically forwarded to the remote side in such a way that any X11 programs started from the shell (or command) will go through the encrypted channel, and the connection to the real X server will be made from the local machine. The user should not manually set DISPLAY. Forwarding of X11 connections can be configured on the command line or in configuration files.

The DISPLAY value set by ssh will point to the server machine, but with a display number greater than zero. This is normal, and happens because ssh creates a “proxy” X server on the server machine for forwarding the connections over the encrypted channel. ssh will also automatically set up Xauthority data on the server machine. For this purpose, it will generate a random authorization cookie, store it in Xauthority on the server, and verify that any forwarded connections carry this cookie and replace it by the real cookie when the connection is opened. The real authentication cookie is never sent to the server machine (and no cookies are sent in the plain).

If the *ForwardAgent* variable is set to “yes” (or see the description of the -A and -a options above) and the user is using an authentication agent, the connection to the agent is automatically forwarded to the remote side.

```
[pere@hp01 ~]$ ssh -X marta@localhost
Last login: Mon Nov 14 17:55:45 2016 from localhost.localdomain
message of the day!

# l’aplicacio geany s’executa en el host remot però es visualitza en el client
[marta@hp01 ~]$ geany &
[1] 4997
[marta@hp01 ~]$ jobs
[1]+  S'est? executant           geany &

[marta@hp01 ~]$ echo $DISPLAY
localhost:10.0
```

```
[pere@hp01 ~]$ ssh -x marta@localhost
Last login: Mon Nov 14 17:58:48 2016 from localhost.localdomain
message of the day!

[marta@hp01 ~]$ echo $DISPLAY

[marta@hp01 ~]$ geany &
[1] 5708
[marta@hp01 ~]$ Geany: cannot open display
```

```
[1]+  Fi d'execuci? amb l'estat 1   geany
```

```
[pere@hp01 ~]$ ssh -o "ForwardAgent yes" -o "ForwardX11 yes" marta@localhost
Last login: Mon Nov 14 18:01:21 2016 from localhost.localdomain
message of the day!
[marta@hp01 ~]$ echo $DISPLAY
localhost:11.0
[pere@hp01 ~]$ gedit &
[1] 6322
```

```
[pere@hp01 ~]$ ssh -X marta@localhost "echo $DISPLAY; geany &"
Enter passphrase for key '/home/pere/.ssh/id_rsa':
:0
```

Configuració client SSH

Tipus de configuració:

- configuració del client
- configuració del servidor

Configuració client SSH

La configuració del client es determina segons els tres nivells següents (de major a menor prioritat):

- Opció indicada en la línia de comandes.
- Opció indicada en el fitxer de configuració client d'usuari: .ssh/config
- Opció de configuració client globla: /etc/ssh/ssh_config

Exemple de configuració client:

```
[root@portatil ~]# cat /etc/ssh/ssh_config
#	$OpenBSD: ssh_config,v 1.26 2010/01/11 01:39:46 dtucker 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 no
#   ForwardX11 no
#   RhostsRSAAuthentication no
#   RSAAuthentication yes
#   PasswordAuthentication yes
#   HostbasedAuthentication no
#   GSSAPIAuthentication no
#   GSSAPIDelegateCredentials no
#   GSSAPIKeyExchange no
#   GSSAPITrustDNS 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
Host *
        GSSAPIAuthentication yes
# 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 LANGUAGE
        SendEnv XMODIFIERS
```

La configuració client usualment resideix en el fitxer /**etc/ssh/ssh_conf.** Aquest fitxer defineix les opcions d'àmbit global o system-wide. Però hi ha més nivells de defició d'opcions de configuració, que són les següents:

- # 1. command line options ($ssh -o opcio1,opcio2… user@host)
- # 2. user-specific file (.ssh/config)
- # 3. system-wide file (/etc/ssh/ssh_config)

- # 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.

Podem observar que les opcions indicades en la línia de comandes tenen més prioritat que les definides en el fitxer de configuració de l’usuari. Les de personalització de l’usuari prevalen sobre les globals.

Es pot consultar la llista de les **opcions de configuració,** el seu significat i els seus valors vàlids a la pàgina del man **ssh_config(5)**.

```
SSH_CONFIG(5)                BSD File Formats Manual               SSH_CONFIG(5)

NAME
    ssh_config — OpenSSH SSH client configuration files

SYNOPSIS
    ~/.ssh/config
    /etc/ssh/ssh_config

DESCRIPTION
    ssh(1) obtains configuration data from the following sources in the following order:

         1.   command-line options
         2.   user's configuration file (~/.ssh/config)
         3.   system-wide configuration file (/etc/ssh/ssh_config)

    For each parameter, the first obtained value will be used.  The configuration files con-
    tain sections separated by “Host” specifications, and that section is only applied for
    hosts that match one of the patterns given in the specification.  The matched host name
    is the one given on the command line.

    Since the first obtained value for each parameter is used, more host-specific declara-
    tions should be given near the beginning of the file, and general defaults at the end.

    The configuration file has the following format:

    Empzty lines and lines starting with ‘#’ are comments.  Otherwise a line is of the format
    “keyword arguments”.  Configuration options may be separated by whitespace or optional
    whitespace and exactly one ‘=’; the latter format is useful to avoid the need to quote
    whitespace when specifying configuration options using the ssh, scp, and sftp -o option.
    Arguments may optionally be enclosed in double quotes (") in order to represent
arguments                containing spaces.
```

```
The possible keywords and their meanings are as follows (note that keywords are case-
insensitive and arguments are case-sensitive):
```

Exemple de tres nivells de configuració

a) Configuracó global (system wide) ***/etc/ssh/ssh_config***

```
…
VisualHostKey yes
```

Usuari pere intenta accedir a remot:

```
[pere@hp01 ~]$ ssh root@remot
Host key fingerprint is 29:36:9a:f9:4e:62:09:64:2a:8c:76:da:32:8b:2a:43
+--[ECDSA  256]---+
|                 |
|                 |
|  o              |
|o+       .       |
|+o..  + S        |
|oE+. * o         |
|.+ .B .          |
|+ +. +           |
|=o   .o          |
+-----------------+
root@remot's password:
```

b) Configuració personal de l’usuari pere **.ssh/config**

```
[pere@hp01 ~]$ cat .ssh/config
VisualHostKey no
```

Usuari pere intenta accedir a remot, la configuració global indica usar VisualHostkey, però la seva configuració personal indica de no fer-ho:

```
[pere@hp01 ~]$ ssh root@remot
root@remot's password:
```

c) Configuració en la comanda ssh usant **-o option**

```
[pere@hp01 ~]$ ssh -o VisualHostKey=yes root@remot
Host key fingerprint is 29:36:9a:f9:4e:62:09:64:2a:8c:76:da:32:8b:2a:43
+--[ECDSA  256]---+
|                 |
|                 |
|  o              |
|o+      .        |
|+o..  + S        |
|oE+. * o         |
|.+ .B .          |
|+ +. +           |
|=o   .o          |
+-----------------+
root@remot's password:
```

Observem que la opció que preval és la de comanda, que ‘overrides’ la de personalització de l’usuari (que indicava no usar VisualHostKey)

Configuracions client personalitzades per a Hosts remots diferents:

És possible establir opcions de configuració diferents per a connexions a servidors remots diferents. És a dir, la configuració client no ha de ser la mateixa per a totes les connexions, es pot personalitzar per a tots, uns quants o hosts individuals (usant *wildcards*).

```
SSH_CONFIG(5)                BSD File Formats Manual               SSH_CONFIG(5)

   Host   Restricts the following declarations (up to the next Host keyword) to be only for
          those hosts that match one of the patterns given after the keyword.  If more than
          one pattern is provided, they should be separated by whitespace.  A single ‘*’ as
          a pattern can be used to provide global defaults for all hosts.  The host is the
          hostname argument given on the command line (i.e. the name is not converted to a
          canonicalized host name before matching).

          A pattern entry may be negated by prefixing it with an exclamation mark (‘!’).
          If a negated entry is matched, then the Host entry is ignored, regardless of
          whether any other patterns on the line match.  Negated matches are therefore use-
          ful to provide exceptions for wildcard matches.

          See PATTERNS for more information on patterns.

PATTERNS

    A pattern consists of zero or more non-whitespace characters, ‘*’ (a wildcard that
    matches zero or more characters), or ‘?’ (a wildcard that matches exactly one
character).
    For example, to specify a set of declarations for any host in the “.co.uk” set of
    domains, the following pattern could be used:
         Host *.co.uk
    The following pattern would match any host in the 192.168.0.[0-9] network range:
          Host 192.168.0.?

    A pattern-list is a comma-separated list of patterns.  Patterns within pattern-lists may
    be negated by preceding them with an exclamation mark ('!').  For example, to allow a key
    to be used from anywhere within an organisation except from the "dialup" pool, the fol-
    lowing entry (in authorized_keys) could be used:
          from="!*.dialup.example.com,*.example.com"

FILES
    ~/.ssh/config
            This is the per-user configuration file.  The format of this file is described
            above.  This file is used by the SSH client.  Because of the potential for abuse,
            this file must have strict permissions: read/write for the user, and not accessi-
            ble by others.

    /etc/ssh/ssh_config
            Systemwide configuration file.  This file provides defaults for those values that
            are not specified in the user's configuration file, and for those users who do
            not have a configuration file.  This file must be world-readable.
```

Exemple de personalització de la configuració segons host destí

```
Host i01
        VisualHostKey yes
        ForwardAgent yes
        ForwardX11 yes

Host localhost
        VisualHostKey yes
        ForwardAgent yes
        ForwardX11 yes
        RSAAuthentication no
        DSAAuthentication no

Host 192.168.2.30
        VisualHostKey no
        ForwardAgent no
        ForwardX11 no
        RSAAuthentication yes
        DSAAuthentication yes
        PasswordAuthentication no
```

Exemple de prova

Pere es connecta al localhost sense VisualHostKey, Es connecta a tothom amb VisualHostKey yes (exterior), i si es connecta a la ip 192.168.80.252 (la seva pública) no es mostra el VisualHostKey ni es pot connectar perquè ni permet autenticació per passwd (no n'hi ha d'altres).

```
[pere@hp01 ~]$ cat .ssh/config
Host localhost
  VisualHostKey no
Host *
  VisualHostKey yes
Host 192.168.80252
  VisualHostKey no
  PasswordAuthentication no
```

Observeu que les regles no s'apliquen per ordre tipus ACLs sinó per match, la que fa millor match.

Servidor/Configuració SSHD

El servei SSHD

Observeu que el servei sshd s'assegura de crear al host les claus de host que permetran identificar-lo en les connexions SSH. Al directori */etc/ssh* es generaràn vàries parelles de claus publica/privada (tipus RSA, DSA, etc) amb l'utilitat ***sshd_keygen***. Les claus de host tene nom tipus:

- Clau privada tipus RSA: */etc/ssh/ssh_host_rsa_key*
- Clau publica tipus RSA: */etc/ssh/ssh_host_rsa_key.pub*

També hi ha en el mateix directòri la configuració del servidor sshd i la del client ssh:

- Configuració del dimoni servidor SSHD: */etc/ssh/sshd_config*
- Condigfuració global del client SSH: */etc/ssh/ssh_config*

```
[root@portatil ~]# systemctl status sshd
sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled)
   Active: active (running) since Mon, 27 Oct 2014 20:01:58 +0100; 1h 49min ago
 Main PID: 842 (sshd)
   CGroup: name=systemd:/system/sshd.service
           └ 842 /usr/sbin/sshd -D

Oct 27 20:02:00 portatil.localdomain sshd[842]: Server listening on 0.0.0.0 port 22.
Oct 27 20:02:00 portatil.localdomain sshd[842]: Server listening on :: port 22.

[root@portatil ~]# nmap localhost
Starting Nmap 6.01 ( http://nmap.org ) at 2014-10-27 21:52 CET
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000025s latency).
Other addresses for localhost (not scanned): 127.0.0.1
Not shown: 993 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
111/tcp open  rpcbind
143/tcp open  imap
631/tcp open  ipp

[root@portatil ~]# ss -t | grep ssh
ESTAB      0      0      192.168.1.41:42762        192.168.1.36:ssh

[root@portatil ~]# netstat  -t | grep ssh
```

```
tcp        0         0 as1:42762              notebook1:ssh           ESTABLISHED

[root@portatil ~]# cat /var/run/sshd.pid
842
```

```
[root@hp01 ~]# ll /etc/ssh/
total 284
-rw-r--r-- 1 root root     242153 14 ago  2015 moduli
-rw-r--r-- 1 root root     2208 11 nov 20:28 ssh_config
-rw------- 1 root root     4344 11 nov 21:10 sshd_config
-rw-r----- 1 root ssh_keys       227   10 abr  2015 ssh_host_ecdsa_key
-rw-r--r-- 1 root root     162   10 abr  2015 ssh_host_ecdsa_key.pub
-rw-r----- 1 root ssh_keys       387   10 abr  2015 ssh_host_ed25519_key
-rw-r--r-- 1 root root              82     10 abr  2015 ssh_host_ed25519_key.pub
-rw-r----- 1 root ssh_keys      1679 10 abr  2015 ssh_host_rsa_key
-rw-r--r-- 1 root root              382   10 abr  2015 ssh_host_rsa_key.pub
```

SSH (Secure Shell) is a protocol which facilitates secure communications between two systems using a client/server architecture and allows users to log into server host systems remotely. Unlike other remote communication protocols, such as FTP or Telnet, SSH encrypts the login session, rendering the connection difficult for intruders to collect unencrypted passwords.

The **ssh** program is designed to replace older, less secure terminal applications used to log into remote hosts, such as telnet or rsh. A related program called scp replaces older programs designed to copy files between hosts, such as rcp. Because these older applications do not encrypt passwords transmitted between the client and the server, avoid them whenever possible. Using secure methods to log into remote systems decreases the risks for both the client system and the remote host.

Generació de les claus de host

Observant els fitxers del servei sshd i sshd-keygen es pot veure com ho fa el servei SSH per generar les claus de host.

```
[root@hp01 ~]# cat /usr/lib/systemd/system/sshd.service
[Unit]
Description=OpenSSH server daemon
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target sshd-keygen.service
Wants=sshd-keygen.service
```

```
[Service]
EnvironmentFile=/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s

[Install]
WantedBy=multi-user.target
```

```
[root@hp01 ~]# cat /usr/lib/systemd/system/sshd-keygen.service
[Unit]
Description=OpenSSH Server Key Generation
ConditionPathExists=|!/etc/ssh/ssh_host_rsa_key
ConditionPathExists=|!/etc/ssh/ssh_host_ecdsa_key
ConditionPathExists=|!/etc/ssh/ssh_host_ed25519_key
PartOf=sshd.service sshd.socket

[Service]
ExecStart=/usr/sbin/sshd-keygen
Type=oneshot
RemainAfterExit=yes
```

Configuració servidor SSH

Exemple de configuració servidor:

```
[root@portatil ~]# cat /etc/ssh/sshd_config
# $OpenBSD: sshd_config,v 1.84 2011/05/23 03:30:07 djm Exp $

# This is the sshd server system-wide configuration file.  See
# sshd_config(5) for more information.

# This sshd was compiled with PATH=/usr/local/bin:/usr/bin

# The strategy used for options in the default sshd_config shipped with
# OpenSSH is to specify options with their default value where
# possible, but leave them commented.  Uncommented options override the
# default value.

#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

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

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

# Authentication:

#LoginGraceTime 2m
#PermitRootLogin yes
#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

#AuthorizedKeysCommand none
#AuthorizedKeysCommandRunAs 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
PasswordAuthentication yes

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

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

# GSSAPI options
#GSSAPIAuthentication no
GSSAPIAuthentication yes
#GSSAPICleanupCredentials yes
```

```
GSSAPICleanupCredentials yes
#GSSAPIStrictAcceptorCheck yes
#GSSAPIKeyExchange no

# 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'.
# WARNING: 'UsePAM no' is not supported in Fedora and may cause several
# problems.
#UsePAM no
UsePAM yes

#AllowAgentForwarding yes
#AllowTcpForwarding yes
#GatewayPorts no
#X11Forwarding no
X11Forwarding yes
#X11DisplayOffset 10
#X11UseLocalhost yes
#PrintMotd yes
#PrintLastLog yes
#TCPKeepAlive yes
#UseLogin no
#UsePrivilegeSeparation yes
#PermitUserEnvironment no
#Compression delayed
#ClientAliveInterval 0
#ClientAliveCountMax 3
#ShowPatchLevel no
#UseDNS yes
#PidFile /var/run/sshd.pid
#MaxStartups 10:30:100
#PermitTunnel no
#ChrootDirectory none

# no default banner path
#Banner none

# Accept locale-related environment variables
AcceptEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY LC_MESSAGES
AcceptEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
AcceptEnv LC_IDENTIFICATION LC_ALL LANGUAGE
AcceptEnv XMODIFIERS

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

# Uncomment this if you want to use .local domain
#Host *.local
#	CheckHostIP no

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

El fitxer de configuració del servidor sshd s'anomena usualment /**etc/ssh/sshd_config.** Es poden consultar la llista d'opcions de configuració amb **man sshd_config(5)**. Podem agrupar/descriure les opcions de configuració més importants en:

- Confiuració de '*listener*' del servei, port, adreça per la que escolta, logs que genera, etc.
- Opcions de configuració de les condicions de les connexions: nº màxim de connexions permeses, intents, accés de root permès o no, mode estricte (paranoic amb el man-in-the-middle) o no, etc.
- Opcions generals de descripció d'on són els certificats, els fitxers de motd (message of the day), banners, etc.
- Són importants les opcions que indiquen si es permet X11 Forwarding, si s'utilitza PAM i les realitves a la autorització o no de les connexions en funció del host client o de l'usuari remot.

Opcions de configuració de servidor destacades

```
Port 22
AddressFamily any
ListenAddress 0.0.0.0
ListenAddress ::

HostKey /etc/ssh/ssh_host_rsa_key

SyslogFacility AUTHPRIV
LoginGraceTime 2m
PermitRootLogin yes
StrictModes yes
MaxAuthTries 6
MaxSessions 10

RSAAuthentication yes
PubkeyAuthentication yes

AuthorizedKeysFile    .ssh/authorized_keys

PasswordAuthentication yes

UsePAM yes

AllowAgentForwarding yes
AllowTcpForwarding yes
X11Forwarding yes

PrintMotd yes
```

Opcions de configuració del servidor personalitzades segons host/Usuari client

Similar al que passa en la configuració client, el servidor SSHD pot establir opcions de configuració personalitzades segons sigui el host client que es connecta al servidor. També pemet establir configuracions particulars segons sigui l'usuari destí a connectar.

Exemples:

```
PrintMotd yes
PrintLastLog no

Match Host i03.*
        PasswordAuthentication yes
        PermitEmptyPasswords no
        PermitRootLogin no

Match User ramon
        PasswordAuthentication no

Match user mireia
        PermitEmptyPasswords yes
        Banner /etc/banner
```

Exemple a l'aula

```
Match user pere
   banner /etc/banner
   PasswordAuthentication yes
Match host 192.168.88.252
   banner /etc/banner2
```

El primer match "**match user pere**" defineix les directives a aplicar si el client es vol conectar al servidor al compte "**pere**" del servidor (en cap cas es tracta del compte client).

Prova-1
Observei que amb independència de si el client és "pere" o "root" s'apliquen les regles quan fa match que es volen connectar a l'usuari "*pere del servidor*".

```
[pere@hp01 ~]$ ssh -X pere@remot
aaaaaaaaaaaaaa
aaaaaaaaaaaaaa
```

```
aaaaaaaaaaaaaa

pere@remot's password:
Last login: Fri Nov 23 19:29:26 2018 from 192.168.1.33
[pere@remot ~]$
```

```
[root@hp01 ~]# ssh root@remot
aaaaaaaaaaaaaa
aaaaaaaaaaaaaa
aaaaaaaaaaaaaa

root@remot's password:
Last login: Fri Nov 23 19:36:29 2018 from 192.168.1.33
[root@remot ~]#
```

Prova-2

Observeu que l'usuari client és indiferent quin és, es connecta al servidor a un usuari que NO és pere i es conneta des del host client 192.168.1.33, per tant fa match amb la regla "**Match host 192.168.1.33**".

```
[pere@hp01 ~]$ ssh -X root@remot
bbbbbbbbbbbbbb
bbbbbbbbbbbbbb
bbbbbbbbbbbbbb
root@remot's password:
[root@remot ~]#
```

```
[root@hp01 ~]# ssh root@remot
bbbbbbbbbbbbbb
bbbbbbbbbbbbbb
bbbbbbbbbbbbbb
root@remot's password:
Last login: Fri Nov 23 19:46:40 2018 from 192.168.1.33
[root@remot ~]#
```

Restriccions d'accés

Existeixen múltiples maneres de restringir l'accés al servidor SSH, algunes d'elles són globals als serveis de xarxes i d'altres específiques per al servei sshd.

Mecanismes de restricció:

- Configuració del servei sshd usant les opcions de restricció d'accés que permet el fitxer **sshd.config**.

- Configuració de l'accés usant regles de **PAM** a través del fitxer de /etc/pam.d corresponent al servei sshd.
- Utilitzar **TCP Wrappers** amb els fitxers hosts.allow i hosts.deny de configuració global del sistema per gestionar els accessos TCP.
- Creació de **firewalls** amb **iptables** que defineixin regles d'accés al servei sshd.

SSHD_CONFIG(5)   BSD File Formats Manual   SSHD_CONFIG(5)

NAME
    sshd_config — OpenSSH SSH daemon configuration file

SYNOPSIS
    /etc/ssh/sshd_config

DESCRIPTION
    sshd(8) reads configuration data from /etc/ssh/sshd_config (or the file specified with -f on the command line). The file contains keyword-argument pairs, one per line. Lines starting with '#' and empty lines are interpreted as comments. Arguments may optionally be enclosed in double quotes (") in order to represent arguments containing spaces.

    The possible keywords and their meanings are as follows (note that keywords are case-insensitive and arguments are case-sensitive):

AllowGroups
    This keyword can be followed by a list of group name patterns, separated by spaces. If specified, login is allowed only for users whose primary group or supplementary group list matches one of the patterns. Only group names are valid; a numerical group ID is not recognized. By default, login is allowed for all groups. The allow/deny directives are processed in the following order: DenyUsers, AllowUsers, DenyGroups, and finally AllowGroups. See PATTERNS in ssh_config(5) for more information on patterns.

AllowUsers
    This keyword can be followed by a list of user name patterns, separated by spaces. If specified, login is allowed only for user names that match one of the patterns. Only user names are valid; a numerical user ID is not recognized. By default, login is allowed for all users. If the pattern takes the form USER@HOST then USER and HOST are separately checked, restricting logins to particular users from particular hosts. The allow/deny directives are processed in the following order: DenyUsers, AllowUsers, DenyGroups, and finally AllowGroups. See PATTERNS in ssh_config(5) for more information on patterns.

Banner
The contents of the specified file are sent to the remote user before authentication is allowed. If the argument is "none" then no banner is displayed. This option is only available for protocol version 2. By default, no banner is displayed.

DenyGroups

```
This keyword can be followed by a list of group name patterns, separated by spaces.
Login is disallowed for users whose primary group or supplementary group list matches one
of the patterns.  Only group names are valid; a numerical group ID is not recognized.  By
default, login is allowed for all groups.  The allow/deny directives are processed in the
following order: DenyUsers, AllowUsers, DenyGroups, and finally AllowGroups. See
PATTERNS in ssh_config(5) for more information on patterns.

DenyUsers
This keyword can be followed by a list of user name patterns, separated by spaces.
Login is disallowed for user names that match one of the patterns.  Only user names are
valid; a numerical user ID is not recognized.  By default, login is allowed for all users.  If the
pattern takes the form USER@HOST then USER and HOST are separately checked,
restricting logins to particular users from particular hosts.  The allow/deny directives are
processed in the following order: DenyUsers, AllowUsers, DenyGroups, and finally
AllowGroups. See PATTERNS in ssh_config(5) for more information on patterns.

Match
Introduces a conditional block.  If all of the criteria on the Match line are satisfied, the
keywords on the following lines override those set in the global section of the config file, until
either another Match line or the end of the file.The arguments to Match are one or more
criteria-pattern pairs.  The available criteria are User, Group, Host, and Address.
The match patterns may consist of single entries or comma-separated lists and may
use the wildcard and negation operatorsdescribed in the PATTERNS section of ssh_config(5).

The patterns in an Address criteria may additionally contain addresses to match in
CIDR address/masklen format, e.g. “192.0.2.0/24” or “3ffe:ffff::/32”.  Note that the mask length
provided must be consistent with the address - it is an error to specify a mask length that is
too long for the address or one with bits set in this host portion of the address.
For example, “192.0.2.0/33” and “192.0.2.0/8” respectively.

HostbasedAuthentication
Specifies whether rhosts or /etc/hosts.equiv authentication together with successful
public key client host authentication is allowed (host-based authentication).  This option is
similar to RhostsRSAAuthentication and applies to protocol version 2 only.  The default is
“no”.

RhostsRSAAuthentication
Specifies whether rhosts or /etc/hosts.equiv authentication together with successful
RSA host authentication is allowed. The default is “no”.  This option applies to protocol
version 1 only.
```

Restricció al servidor SSH pel Compte d’usuari

```
AllowUsers pere marta
AllowUsers pere@pc01.edt.org marta jordi@m06.cat
```

AllowUsers permet indicar una llista d'usuaris en el servidor ssh als que es permet l'accés, qualsevol altre usuari del servidor no és accessible via ssh, té l'accés denegat. L'accés que es permet és el que estigui establert per a l'usuari (amb password o amb public key).

AllowUsers user@host permet indicar que al compte de l'usuari 'user' s'hi permet l'accés només des del host '**host**'. Atenció, **no indica usuari@hos**t.

Restricció via PAM

```
[root@portatil ~]# cat /etc/pam.d/sshd
#%PAM-1.0
auth       required     pam_sepermit.so
auth       substack     password-auth
auth       include      postlogin
account    required     pam_nologin.so
account    required     pam_access.so accessfile=/etc/security/acess-sshd.conf
account    include      password-auth
password   include      password-auth
# pam_selinux.so close should be the first session rule
session    required     pam_selinux.so close
session    required     pam_loginuid.so
# pam_selinux.so open should only be followed by sessions to be executed in the user
context
session    required     pam_selinux.so open env_params
session    optional     pam_keyinit.so force revoke
session    include      password-auth
session    include      postlogin
```

pam_access.so

```
PAM_ACCESS(8)              Linux-PAM Manual                    PAM_ACCESS(8)

NAME
       pam_access - PAM module for logdaemon style login access control

SYNOPSIS
       pam_access.so [debug] [nodefgroup] [noaudit] [accessfile=file] [fieldsep=sep] [listsep=sep]

DESCRIPTION
       The pam_access PAM module is mainly for access management. It provides logdaemon style login access
control based on login names, host or domain names, internet addresses or network numbers, or on terminal line
names in case of non-networked logins.
```

By default rules for access management are taken from config file /etc/security/access.conf if you don´t specify another file.

If Linux PAM is compiled with audit support the module will report when it denies access based on origin (host or tty).

OPTIONS

accessfile=/path/to/access.conf

Indicate an alternative access.conf style configuration file to override the default. This can be useful when different services need different access lists.

ACCESS.CONF(5) Linux-PAM Manual ACCESS.CONF(5)

NAME

access.conf - the login access control table file

DESCRIPTION

The /etc/security/access.conf file specifies (user/group, host), (user/group, network/netmask) or (user/group, tty) combinations for which a login will be either accepted or refused.

When someone logs in, the file access.conf is scanned for the first entry that matches the (user/group, host) or (user/group, network/netmask) combination, or, in case of non-networked logins, the first entry that matches the (user/group, tty) combination. The permissions field of that table entry determines whether the login will be accepted or refused.

Each line of the login access control table has three fields separated by a ":" character (colon):
permission:users/groups:origins

The first field, the permission field, can be either a "+" character (plus) for access granted or a "-" character (minus) for access denied.

The second field, the users/group field, should be a list of one or more login names, group names, or ALL (which always matches). To differentiate user entries from group entries, group entries should be written with brackets, e.g. (group).

The third field, the origins field, should be a list of one or more tty names (for non-networked logins), host names, domain names (begin with "."), host addresses, internet network numbers (end with "."), internet network addresses with network mask (where network mask can be a decimal number or an internet address also), ALL (which always matches) or LOCAL. LOCAL keyword matches if and only if the PAM_RHOST is not set and <origin> field is thus set from PAM_TTY or PAM_SERVICE". If supported by the system you can use @netgroupname in host or user patterns. The @@netgroupname syntax is supported in the user pattern only and it makes the local system hostname to be passed to the netgroup match call in addition to the user name. This might not work correctly on some libc implementations causing the match to always fail.

The EXCEPT operator makes it possible to write very compact rules.

If the nodefgroup is not set, the group file is searched when a name does not match that of the logged-in user. Only groups are matched in which users are explicitly listed. However the PAM module does not look at the primary group id of a user.

The "#" character at start of line (no space at front) can be used to mark this line as a comment line.

EXAMPLES

These are some example lines which might be specified in /etc/security/access.conf.

User root should be allowed to get access via cron, X11 terminal :0, tty1, ..., tty5, tty6.

```
+ : root : crond :0 tty1 tty2 tty3 tty4 tty5 tty6
```

User root should be allowed to get access from hosts which own the IPv4 addresses. This does not mean that the connection have to be a IPv4 one, a IPv6 connection from a host with one of this IPv4 addresses does work, too.

```
+ : root : 192.168.200.1 192.168.200.4 192.168.200.9

+ : root : 127.0.0.1
```

User root should get access from network 192.168.201. where the term will be evaluated by string matching. But it might be better to use network/netmask instead. The same meaning of 192.168.201. is 192.168.201.0/24 or 192.168.201.0/255.255.255.0.

```
+ : root : 192.168.201.
```

User root should be able to have access from hosts foo1.bar.org and foo2.bar.org (uses string matching also).

```
+ : root : foo1.bar.org foo2.bar.org
```

User root should be able to have access from domain foo.bar.org (uses string matching also).

```
+ : root : .foo.bar.org
```

User root should be denied to get access from all other sources.

```
- : root : ALL
```

User foo and members of netgroup admins should be allowed to get access from all sources. This will only work if netgroup service is available.

```
+ : @admins foo : ALL
```

User john and foo should get access from IPv6 host address.

```
+ : john foo : 2001:db8:0:101::1
```

User john should get access from IPv6 net/mask.

```
+ : john : 2001:db8:0:101::/64
```

Disallow console logins to all but the shutdown, sync and all other accounts, which are a member of the wheel group.

```
-:ALL EXCEPT (wheel) shutdown sync:LOCAL
```

All other users should be denied to get access from all sources.

```
- : ALL : ALL
```

MODULE TYPES PROVIDED
All module types (auth, account, password and session) are provided.

pam_listfile.so

```
NAME
      pam_listfile - deny or allow services based on an arbitrary
      file

SYNOPSIS
      pam_listfile.so item=[tty|user|rhost|ruser|group|shell]
                 sense=[allow|deny] file=/path/filename
                 onerr=[succeed|fail] [apply=[user|@group]]
                 [quiet]

DESCRIPTION
      pam_listfile is a PAM module which provides a way to deny or allow services based on an arbitrary file.

      The module gets the item of the type specified -- user
      specifies the username, PAM_USER; tty specifies the name of
      the terminal over which the request has been made, PAM_TTY;
      rhost specifies the name of the remote host (if any) from
      which the request was made, PAM_RHOST; and ruser specifies
      the name of the remote user (if available) who made the
      request, PAM_RUSER -- and looks for an instance of that item
      in the file=filename.  filename contains one line per item
      listed. If the item is found, then if sense=allow,
      PAM_SUCCESS is returned, causing the authorization request to
      succeed; else if sense=deny, PAM_AUTH_ERR is returned,
      causing the authorization request to fail.

      If an error is encountered (for instance, if filename does
      not exist, or a poorly-constructed argument is encountered),
      then if onerr=succeed, PAM_SUCCESS is returned, otherwise if
      onerr=fail, PAM_AUTH_ERR or PAM_SERVICE_ERR (as appropriate)
      will be returned.

      An additional argument, apply=, can be used to restrict the
      application of the above to a specific user (apply=username)
      or a given group (apply=@groupname). This added restriction
      is only meaningful when used with the tty, rhost and shell
      items.

      Besides this last one, all arguments should be specified; do
      not count on any default behavior.

      No credentials are awarded by this module.

OPTIONS
      item=[tty|user|rhost|ruser|group|shell]
         What is listed in the file and should be checked for.

      sense=[allow|deny]
         Action to take if found in file, if the item is NOT found
         in the file, then the opposite action is requested.

      file=/path/filename
         File containing one item per line. The file needs to be a
         plain file and not world writable.

      onerr=[succeed|fail]
         What to do if something weird happens like being unable
         to open the file.
```

```
    apply=[user|@group]
       Restrict the user class for which the restriction apply.
       Note that with item=[user|ruser|group] this does not make
       sense, but for item=[tty|rhost|shell] it have a meaning.

    quiet
       Do not treat service refusals or missing list files as
       errors that need to be logged.

MODULE TYPES PROVIDED
    All module types (auth, account, password and session) are
    provided.
```

Restricció de comandes a executar

Using SSH forced commands, you can limit which programs a user may run as root. For example, this key entry:

```
~root/.ssh/authorized_keys:
command="/sbin/dump -0 /local/data" ssh-dss key...
```

Apèndix

Authentication SSH

AUTHENTICATION

The OpenSSH SSH client supports SSH protocols 1 and 2. The default is to use protocol 2 only, though this can be changed via the Protocol option in ssh_config(5) or the -1 and -2 options (see above). Both protocols support similar authentication methods, but protocol 2 is the default since it provides additional mechanisms for confidentiality (the traffic is encrypted using AES, 3DES, Blowfish, CAST128, or Arcfour) and integrity (hmac-md5, hmac-sha1, hmac-sha2-256, hmac-sha2-512, umac-64, umac-128, hmac-ripemd160). Protocol 1 lacks a strong mechanism for ensuring the integrity of the connection.

The methods available for authentication are: GSSAPI-based authentication, host-based authentication, public key authentication, challenge-response authentication, and password authentication. Authentication methods are tried in the order specified above, though protocol 2 has a configuration option to change the default order: *PreferredAuthentications*.

Host-based authentication works as follows: If the machine the user logs in from is listed in */etc/hosts.equiv* or */etc/ssh/shosts.equiv* on the remote machine, and the user names are the same on both sides, or if the files *~/.rhosts* or *~/.shosts* exist in the user's home directory on the remote machine and contain a line containing the name of the client machine and the name of the user on that machine, the user is considered for login. Additionally, the server must be able to verify the client's host key (see the description of */etc/ssh/ssh_known_hosts* and *~/.ssh/known_hosts*, below) for login to be permitted. This authentication method closes security holes due to IP spoofing, DNS spoofing, and routing spoofing. [Note to the administrator: */etc/hosts.equiv*, *~/.rhosts*, and the *rlogin/rsh* protocol in general, are inherently insecure and should be disabled if security is desired.]

Public key authentication works as follows: The scheme is based on public-key cryptography, using cryptosystems where encryption and decryption are done using separate keys, and it is unfeasible to derive the decryption key from the encryption key. The idea is that each user creates a public/private key pair for authentication purposes. The server knows the public key, and only the user knows the private key. *ssh implements public key authentication protocol automatically*, using one of the DSA, ECDSA or RSA algorithms. Protocol 1 is restricted to using only RSA keys, but protocol 2 may use any. The HISTORY section of ssl(8)
contains a brief discussion of the DSA and RSA algorithms.

The file ~/.ssh/authorized_keys lists the public keys that are permitted for logging in.

When the user logs in, the ssh program tells the server which key pair it would like to use for authentication. The client proves that it has access to the private key and the server checks that the corresponding public key is authorized to accept the account.

The user creates his/her key pair by running ssh-keygen(1). This stores the private key in *~/.ssh/identity* (protocol 1), *~/.ssh/id_dsa* (protocol 2 DSA), *~/.ssh/id_ecdsa* (protocol 2 ECDSA), or *~/.ssh/id_rsa* (protocol 2 RSA) and stores the public key in ~/.ssh/identity.pub (protocol 1), ~/.ssh/id_dsa.pub (protocol 2 DSA), ~/.ssh/id_ecdsa.pub (protocol 2 ECDSA), or *~/.ssh/id_rsa.pub* (protocol 2 RSA) in the user's home directory. The user should then copy the public key to *~/.ssh/authorized_keys* in his/her home directory on the remote machine. The authorized_keys file corresponds to the conventional ~/.rhosts file, and has one key per line, though the lines can be very long. After this, the user can log in without giving the password.

A variation on public key authentication is available in the form of certificate authentication: instead of a set of public/private keys, signed certificates are used. This has the advantage that a single trusted certification authority can be used in place of many public/private keys. See the CERTIFICATES section of ssh-keygen(1) for more information.

The most convenient way to use public key or certificate authentication may be with an *authentication agent*. See ssh-agent(1) for more information.

Challenge-response authentication works as follows: The server sends an arbitrary "challenge" text, and prompts for a response. Protocol 2 allows multiple challenges and responses; protocol 1 is restricted to just one challenge/response. Examples of challenge-response authentication include BSD Authentication (see login.conf(5)) and PAM (some non-OpenBSD systems).

Finally, if other authentication methods fail, ssh prompts the user for a password. The password is sent to the remote host for checking; however, since all communications are encrypted, the password cannot be seen by someone listening on the network.

ssh automatically maintains and checks a database containing identification for all hosts it has ever been used with. Host keys are stored in *~/.ssh/known_hosts* in the user's home directory. Additionally, the file */etc/ssh/ssh_known_hosts* is automatically checked for known hosts. Any new hosts are automatically added to the user's file. If a host's identification ever changes, ssh warns about this and disables password authentication to prevent server spoofing or man-in-the-middle attacks, which could otherwise be used to circumvent the encryption. The *StrictHostKeyChecking* option can be used to control logins to machines whose host key is not known or has changed.

When the user's identity has been accepted by the server, the server either executes the given command, or logs into the machine and gives the user a normal shell on the remote machine. All communication with the remote command or shell will be automatically encrypted.

If a pseudo-terminal has been allocated (normal login session), the user may use the escape characters noted below. If no pseudo-tty has been allocated, the session is transparent and can be used to reliably transfer binary data. On most systems, setting the escape character to “none” will also make the session transparent even if a tty is used.

The session terminates when the command or shell on the remote machine exits and all X11 and TCP connections have been closed.

Protocol SSH

SSH (Secure Shell) is a protocol which facilitates secure communications between two systems using a client/server architecture and allows users to log into server host systems remotely. Unlike other remote communication protocols, such as FTP or Telnet, SSH encrypts the login session, rendering the connection difficult for intruders to collect unencrypted passwords.

The **ssh** program is designed to replace older, less secure terminal applications used to log into remote hosts, such as telnet or rsh. A related program called scp replaces older programs designed to copy files between hosts, such as rcp. Because these older applications do not encrypt passwords transmitted between the client and the server, avoid them whenever possible. Using secure methods to log into remote systems decreases the risks for both the client system and the remote host.

Fedora includes the general OpenSSH package (openssh) as well as the OpenSSH server (openssh-server) and client (openssh-clients) packages. Note that the OpenSSH packages require the OpenSSL package (openssl), which installs several important cryptographic libraries, enabling OpenSSH to provide encrypted communications.

11.1. The SSH Protocol

11.1.1. Why Use SSH?

Potential intruders have a variety of tools at their disposal enabling them to disrupt, intercept, and re-route network traffic in an effort to gain access to a system. In general terms, these threats can be categorized as follows:

Interception of communication between two systems

The attacker can be somewhere on the network between the communicating parties, copying any information passed between them. He may intercept and keep the information, or alter the information and send it on to the intended recipient.

This attack is usually performed using a *packet sniffer*, a rather common network utility that captures each packet flowing through the network, and analyzes its content.

Impersonation of a particular host

Attacker's system is configured to pose as the intended recipient of a transmission. If this strategy works, the user's system remains unaware that it is communicating with the wrong host.

This attack can be performed using a technique known as *DNS poisoning*, or via so-called *IP spoofing*. In the first case, the intruder uses a cracked DNS server to point client systems to a maliciously duplicated host. In the second case, the intruder sends falsified network packets that appear to be from a trusted host.

Both techniques intercept potentially sensitive information and, if the interception is made for hostile reasons, the results can be disastrous. If SSH is used for remote shell login and file copying, these security threats can be greatly diminished. This is because the SSH client and server use digital signatures to verify their identity. Additionally, all communication between the client and server systems is encrypted. Attempts to spoof the identity of either side of a communication does not work, since each packet is encrypted using a key known only by the local and remote systems.

11.1.2. Main Features

The SSH protocol provides the following safeguards:

No one can pose as the intended server

After an initial connection, the client can verify that it is connecting to the same server it had connected to previously.

No one can capture the authentication information

The client transmits its authentication information to the server using strong, 128-bit encryption.

No one can intercept the communication

All data sent and received during a session is transferred using 128-bit encryption, making intercepted transmissions extremely difficult to decrypt and read.

Additionally, it also offers the following options:

It provides secure means to use graphical applications over a network

Using a technique called *X11 forwarding*, the client can forward *X11* (*X Window System*) applications from the server.

It provides a way to secure otherwise insecure protocols

The SSH protocol encrypts everything it sends and receives. Using a technique called *port forwarding*, an SSH server can become a conduit to securing otherwise insecure protocols, like POP, and increasing overall system and data security.

It can be used to create a secure channel

The OpenSSH server and client can be configured to create a tunnel similar to a virtual private network for traffic between server and client machines.

It supports the Kerberos authentication

OpenSSH servers and clients can be configured to authenticate using the GSSAPI (Generic Security Services Application Program Interface) implementation of the Kerberos network authentication protocol.

11.1.3. Protocol Versions

Two varieties of SSH currently exist: version 1, and newer version 2. The OpenSSH suite under Fedora uses SSH version 2, which has an enhanced key exchange algorithm not vulnerable to the known exploit in version 1. However, for compatibility reasons, the OpenSSH suite does support version 1 connections as

11.1.4. Event Sequence of an SSH Connection

The following series of events help protect the integrity of SSH communication between two hosts.

1. A cryptographic handshake is made so that the client can verify that it is communicating with the correct server.
2. The transport layer of the connection between the client and remote host is encrypted using a symmetric cipher.
3. The client authenticates itself to the server.
4. The remote client interacts with the remote host over the encrypted connection.

11.1.4.1. Transport Layer

The primary role of the transport layer is to facilitate safe and secure communication between the two hosts at the time of authentication and during subsequent communication. The transport layer accomplishes this by handling the encryption and decryption of data, and by providing integrity protection of data packets as they are sent and received. The transport layer also provides compression, speeding the transfer of information.

Once an SSH client contacts a server, key information is exchanged so that the two systems can correctly construct the transport layer. The following steps occur during this exchange:

- Keys are exchanged
- The public key encryption algorithm is determined
- The symmetric encryption algorithm is determined
- The message authentication algorithm is determined
- The hash algorithm is determined

During the key exchange, the server identifies itself to the client with a unique *host key*. If the client has never communicated with this particular server before, the server's host key is unknown to the client and it does not connect. OpenSSH gets around this problem by accepting the server's host key. This is done after the user is notified and has both accepted and verified the new host key. In subsequent connections, the server's host key is checked against the saved version on the client, providing confidence that the client is indeed communicating with the intended server. If, in the future, the host key no longer matches, the user must remove the client's saved version before a connection can occur.

SSH is designed to work with almost any kind of public key algorithm or encoding format. After an initial key exchange creates a hash value used for exchanges and a shared secret value, the two systems immediately begin calculating new keys and algorithms to protect authentication and future data sent over the connection.

After a certain amount of data has been transmitted using a given key and algorithm (the exact amount depends on the SSH implementation), another key exchange occurs, generating another set of hash values and a new shared secret value. Even if an attacker is able to determine the hash and shared secret value, this information is only useful for a limited period of time.

11.1.4.2. Authentication

Once the transport layer has constructed a secure tunnel to pass information between the two systems, the server tells the client the different authentication methods supported, such as using a private key-encoded signature or typing a password. The client then tries to authenticate itself to the server using one of these supported methods.

SSH servers and clients can be configured to allow different types of authentication, which gives each side the optimal amount of control. The server can decide which encryption methods it supports based on its security model, and the client can choose the order of authentication methods to attempt from the available options.