72 - Secure Shell - ssh & sshd

Connecting to a remote host via secure shell:

- On the remote host:
  Make sure that the remote unix has the sshd daemon running or set correctly in its /etc/inetd.conf file.

- On the client host:
  Run the following command:
  ```
  ssh user@<hostname or IP#>

  eg. ssh michel@dozlinux.linux.local
  ```

1 - Description

Secure Shell (SSH) stands for **S**ecure **SH**ell and is a replacement for the insecure **telnet**, **rlogin,** and **rsh**. It uses the combination Asymetrical encryption method (Private and Public Keys) for session initialization, key negociation and user authentication, then Symetrical or 'block cipher' method for data transfer during the rest of the session.

2 - Encryption Methods

There are 2 types of encryption methods used in secure communications:

- Symetrical or 'Block Cipher' using a common key for encryption and decryption.
  The same key is used by both parties to encrypt and decrypt data.
  This means both must have that same key already stored in system.
- Asymetrical or PK crypto, using Private and Public keys.
  The private-key is unique and secret for each host.
  It can be used to encryp a signature or decrypt data.
  The public-key is unique fro each host and can be distributed at will.

3 - Principle of Asymetrical (PK crypto)

This PK crypto uses the following method:

- If (B) host wants to send data to (A) host then:
  - (A) host sends its public-key to (B) host.
  - (B) host encrypt a message using (A) host's public-key and send it to (A)
  - (A) host uses its private-key to decrypt it.

The same procedure but reverse is used if (A) host send data to (B) host.

4 - Encryption schemes used in PK crypto

There are 2 types of methods using Public and Private key (PK crypto).

- Version 1 - RSA - Uses RSA Data Security Inc. proprietary encryption library of which the patent has expired in year 2001. This is the regular SSH Package *in SuSE.*
- Version 2 - DSA - Uses GPL OpenSSH encryption library and is free to be used by anybody. Only Local Government Security Restrictions applies. DSA is the one proposed to be the Internet Standard to the IETF organisation.

5 - Sequence of events in a RSA - SSH (Version 1) connection.

Notes:

- The sshd runs as Daemon on server and generates a new server-session-key (768 bits) every hour if it has been used. This key has nothing to do with the host-public-key or host-private-key. It will be used to transport the session-key generated by the client.
- Client is (A) and Server is (B).
- Communication that is in normal font is using NO encryption (clear text)
- Communication that is in *Italic* is using Asymetric encrypted connection (RSA)
- Communication that is underlined is using symetrical encrypted connection (3DES(default) or blowfish)

- A requests an ssh session to B via port 22. (ssh servername)
- B sshd daemon sends its server-key 768 Bit (saved RAM) and public-key to A
- *A generates a 256 Bits long random number (session-key), encrypts it using the server's server-key and his public-key and sends it to the B*. 3DES or blowfish 'Block cipher'
- A sends its user-public-key to B along with its user login name.
- If user is a valid user then B checks in user's **`~/.ssh/authorized_keys`** if A's user-public-key is present.
- If so then B checks in **`~/.ssh/known_hosts`** if the A's user-public-key is present.
- If so then the ssh session is allowed to start.
- If not then B asks A for Name and Password as second identification proof
- If all ok the SSH session is allowed to start.

6 - Sequence of events in a DSA - SSH (Version 2) connection.

Note: - Client is (A) and Server is (B).
- Communication that is in normal font is using NO encryption (clear text)
- Communication that is in *Italic* is using Asymetric encrypted connection (DSA)
- Communication that is underlined is using symetrical encrypted connection (3DES, Blowfish, CAST128 or Arcfour)

- A requests an ssh session to B via port 22. (ssh servername)
- B responds with the SSH Daemon version for compatibility check
- A encrypt user's signature using user-private-key and sends it along with user-public-key to B
- B decrypts A's signature using A's user-public-key to verify identity of the user.
- If user is valid then B checks in user's **`~/.ssh/authorized_keys`** if A's user-public-key is present.
- If so then B checks in **`~/.ssh/known_hosts`** if the A's user-public-key is present.
- If not so then B asks A for Name and Password as second identification proof
- If all ok the SSH session is allowed to start.

Tunnelling using SSH

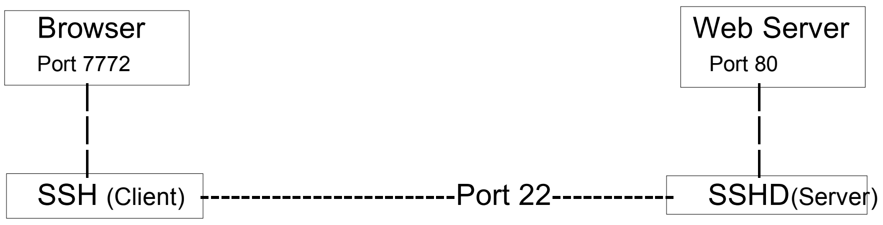
1. Start a terminal and make a connection ssh to remote web server.

   ```
   ssh -2 remoteIP/name -L secureport:remoteIP/name:serviceport
   ```

   eg. `ssh -2 sun.linux.local -L 7772:sun.linux.local:80`

   This will use the port 22 for the ssh connection and the port 7772 to tunnel the port 80 of the web server in `sun.linux.local`.

2. Start a web browser and give the address:
   `http://localhost:7772`
   This will use the local ssh client(port 22) as a tunnel to the remote web server.



Hosts Synchronization

`rsync` - Extra Program to remote sync 2 directories on different machines:

Description:
The most usefull application of `rsync` is to synchronize 2 directories together on different machines using compression and <u>encryption</u> (ssh).
The use is quite similar to `scp` but adds the <u>compression</u> and the <u>differencial copy</u> which adds enormous speed.

Requirements:
- Package `rsync` in serie 'n' on both source and destination machines.
- Package `ssh` in serie '`sec`' on both source and destination machines.

Syntax:
```
rsync -vazu -e ssh SourceDir RemoteDestDir
rsync -vazu -e ssh RemoteDestDir SourceDir
```

eg. `rsync -vazu -e ssh /home/public/ remote:/home/public`

This copies <u>recursively</u> (because the source dir. ends with a `/`) all the files and directories from local host `/home/public` to the remote host (*`remote:`*) in his `/home/public` directory.

Note: if the source directory doesn't end with a `/` then only the content of this directory is copied , therefore <u>not recursive</u>.

Windows SSH tools:

`Putty.exe`	Allows to connect a very good VT100 terminal via telnet or SSH protocol on a Unix/Linux machine having the Telnet or SSH Daemon running.
`Puttytel.exe`	Allows to connect a very good VT100 terminal via telnet only protocol on a Unix/Linux machine having the telnet Daemon running.
`Puttygen.exe`	Generates a Key set (Private and Public) for local Putty.
`Pscp.exe`	Copies files via SSH protocol to and from Unix/Linux hosts having SSH Daemon running.
`Plink.exe`	Allows to execute a shell command via SSH protocol on a remote Unix/Linux host having the SSH Daemon running. The syntax is: `plink.exe -ssh -P 22 -pw` *`password ShellCommand`*

eg. To start a Midnight Commander

```
plink -ssh -P 22 -pw mypasswd xterm -display 192.168.100.33:0 -e mc
```

- **To get to logon in a remote host via ssh without password:**
  1. Grenerate a key for the user on the local host that wants to logon to another host
     `ssh-keygen -t dsa` Generates a set of dsa keys or

     `ssh-keygen -t rsa` Generates a set of rsa keys
     Note: press <*`Enter`*> to all of he questions .
  2. Take the newly generated public key (`~/.ssh/id_dsa.pub`) and copy its content into the remote host in the home director of the target user in target host, into the file `~/.ssh/authorized_keys`.

     eg. From local root user to root user on remote host.
     ```
     cat ~/.ssh/*.pub | ssh root@remoteserver 'cat >>~/.ssh/authorized_keys'
     ```
  3. Now the local user should be able to login as the target user in the target host without being asked for a password.

- Some **proprietary implementations** of the ssh2 protocol generate public key files that start something like this:

  ```
  ---- BEGIN SSH2 PUBLIC KEY ----
  Subject: fred
  Comment: "my key"
  ```

  If you want to log in to an OpenSSH server using one of these keys, first copy the key to the OpenSSH server, then (assuming the file name is `identity.pub`), use `ssh-keygen` to import it with the **`-i`** option, and add it to your `authorized_keys`:

  ```
  ssh-keygen -i -f identity.pub >> ~/.ssh/authorized_keys
  ```

  If you still have trouble logging in, check the permissions for your `.ssh` directory and the files in it. If your ssh client doesn't have verbose mode, change

  `LogLevel INFO` to

  `LogLevel DEBUG`

in `/etc/sshd_config`, and restart sshd to get more information on the problem. See `man ssh-keygen` for more information on importing keys.

- **Problems with `Kerberos` and SSH** (SSH takes a very long time to login)

  This problem is known to be present in the SuSE 8.1. The solution is been know as to deinstall the package `heimdal-lib` or to update it from the 2 packages `heimdal` and `heimdal.lib` from SuSE in Internet. It has something to do with SSH waiting for some `kerberos` resolution by a DNS and times-out after a while.

- **Some other methods of remote SSH Loggin without password**
  (From Pro-Linux newsletter-Deutsch)

  Wer häufig per SSH auf anderen Rechnern arbeitet, muss viele Kennwörter im Kopf haben und in die Tastatur tippen. Mit der Zeit wächst der Wunsch nach Erleichterung...Abhilfe schafft die automatische Authentifizierung per SSH mittels öffentlichen Schlüsseln. So wird's gemacht:

  - Public SSH-Schlüssel

  Zunächst wird ein SSH-Schlüsselpaar erzeugt:

  ```
  % ssh-keygen -t rsa
  ```

  Nach der Eingabe eines längeren Kennwortes oder eines Satzes (engl. Passphrase) werden in `$HOME/.ssh` die Dateien id_rsa und id_rsa.pub erzeugt.

  Nun kopiert man die Datei `id_rsa.pub` mittels (aus Sicherheitsgründen) `scp` in das `$HOME/.ssh` des Rechners, auf den man sich automatisch einloggen lassen möchte, und nennt die Datei dort `authorized_keys2`.

  Wenn bereits eine solche Datei existiert, erweitert man diese Datei mit dem neuen Schlüssel. Beim nächsten Login wird nicht mehr das bisherige Login-Passwort abgefragt, dafür allerdings das SSH-Kennwort.

  - SSH-Agent

  Damit auch nicht mehr nach dem SSH-Kennwort gefragt wird, wird ein sogenannter SSH-Agent benötigt. Dieser wartet, nachdem das SSH-Kennwort beim Rechnerstart einmal abgefragt wurde, auf bevorstehende SSH-Logins und kümmert sich um die Authentifizierung, damit der Nutzer nichts mehr machen muss.

  Alle Anleitungen und HOWTOs, die ich im Netz zu diesem Thema gefunden habe, enthielten das Bearbeiten von `.xinitrc` und/oder `.xsession`. Benutzt man KDM/GDM/XDM, wird es nochmal ein Stück komplizierter. Außerdem müssen zwei Kennwörter eingegeben werden.

  Das Naheliegenste waere gewesen, die einschlaegigen man-Pages zu lesen (z.B. von `ssh-agent`), dort steht auch, was zu tun ist. Die `.xsession` ist sowieso meistens ein Link zur `.xinitrc`, ansonsten editiert man einfach beide (`.xsession` muss fuer die Nutzung mit Xdm editiert werden).

  Das Einzige, was hinzugefuegt werden muss (von dem Beispiel in diesem Kurztip ausgehend):

  `eval `ssh-agent`` (als ersten Befehl in der Datei)

  ```
  ssh-add ${HOME}/.ssh/id_rsa
  ```

(als zweiten Befehl, bewirkt auch die Abfrage der Passphrase)

```
ssh-agent -k
```

(als letzten Befehl, sorgt dafuer, dass der Agent am Ende der Sitzung beendet wird)

U.U. startet der Xdm den ssh-agent sogar selbst (z.B. bei Debian), sodass nur der Befehl "`ssh-add ${HOME}/.ssh/id_rsa`" noetig ist.

Ob KDM/GDM `.xsession` oder`.xinitrc` auslesen, weiss ich nicht, mit Xdm klappts auf jeden Fall so. Außerdem müssen zwei Kennwörter eingegeben werden.

Stimmt in diesem Beispiel _nicht_; es muss _nur_ die _eine_ Passphrase eingegeben werden, mit der der private Schluessel verschluesselt wurde.

• PAM-Modul

Einfacher dagegen ist es, einfach ein entsprechendes PAM-Modul zu verwenden. Ist dieses Modul installiert und konfiguriert, loggt man sich mit dem SSH-Kennwort am Rechner ein, nicht mehr mit dem normalen Nutzer-Kennwort. Der Vorteil liegt auf der Hand: statt zwei Kennwörtern (normaler Login + SSH-Kennwort) muss nur noch ein einziges eingegeben werden. Das Modul startet auch den SSH-Agenten.

Ich bin Gentoo-Nutzer, daher gehe ich jetzt nur auf diese Distribution ein. Bei anderen müsste es aber ähnlich funktionieren.

Als erstes wird das `ssh_pam`-Modul installiert. Als root eingeben:

```
% emerge ssh_pam
```

Danach muss noch PAM vom neuen Modul in Kenntnis gesetzt werden:

```
% zcat system-auth.example.gz | grep ssh >> /etc/pam.d/system-auth
```

(Evtl. die Pfade an reale Gegebenheiten anpassen!)

Dieses hängt der Datei `/etc/pam.d/system-auth` zwei neue Zeilen an. Die neue vorletzte Zeile mit "`auth`" und "`sufficient`" muss mit einem Texteditor an die zweite Stelle der Liste geschoben werden.

Achtung: Im Portage-Tree von Gentoo gibt es auch ein Paket namens ssh_pam_agent, das zusammen mit den Paketen Keychain und Expect läuft. Dieses Paket ist nicht gemeint! Bei einem kurzen Test funktionierte der automatische SSH-Login nur, wenn man sich einmal direkt über die Konsole eingeloggt und dann das SSH-Kennwort (die Passphrase des Schlüssels) eingegeben hatte. Per KDM konnte ich mich nicht mit dem SSH-Kennwort einloggen. Alles in allem ist `pam_ssh_agent` also nicht so komfortabel wie das `pam_ssh`-Modul.

Fertig! Das war es schon. Beim nächsten Login muss dann das SSH-Kennwort eingegeben werden, der SSH-Agent wird automatisch gestartet und kümmert sich um weitere SSH-Logins auf anderen Rechnern.

Secure SMTP via WiFi

The saga of "how can I securely send mail from my laptop when it's plugged in to someone else's network" continues. If you run a wireless network, you probably block port 25 to keep people from using your network for spamming. The problem is how do you make an outgoing SMTP connection from your laptop when you travel to other people's wireless networks?

Miek Gieben has an elegant answer based on Postfix and OpenSSH. Set up Postfix on the laptop to use localhost and a high port as its relayhost, then tunnel that high port to port 25 on your mail server. Details and config file entries for the basic method are here:

`http://www.miek.nl/linux/postfix.html`.

For security, though, you don't want to let people log in to your OpenSSH server as you without a passphrase. So, instead of using your main OpenSSH identity for the tunnel, add a user called tunnel to the OpenSSH server and make a tunnel-only SSH key on the laptop with this command:

```
ssh-keygen -t dsa -f tunnel
```

In the file `tunnel.pub`, add the following options to the beginning of the line:

```
command="/bin/false",no-X11-forwarding,no-agent-forwarding,no-pty
```

and then copy the contents of `tunnel.pub` to the end of the tunnel user's `.ssh/authorized_keys` file on the OpenSSH server.

If you can run only `/bin/false` and you can't forward **`X`** or the SSH agent, what's left? Port forwarding. Add `-i` tunnel to Miek's instructions to use the tunnel, and replace `miekg@` with `tunnel@`.

- **Using `mindterm` and `putty` with SuSE 9.x sshd**

  After installation SuSE has the password authentication deactivated. This prevents SSH clients programs to login. To remedy to that, change the following entry in the file: `/etc/ssh/sshd_config`

  ```
  PasswordAuthentication yes
  ```