SSH Secure Shell for UNIX Servers Administrator's Guide

December, 2000

© 1996 - 2000 SSH Communications Security Corp, Finland.

No part of this publication may be reproduced, published, stored in a electronic database, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, for any purpose, without the prior written permission of SSH Communications Security Corp. SSH and IPSEC Express are trademarks or registered trademarks of SSH Communications Security Corp. All brand and product names that are trademarks or registered trademarks are the property of their owners. US and other patents pending. This product includes software developed by the University of California, Berkeley and its contributors. THERE IS NO WARRANTY OF ANY KIND FOR THE ACCURACY OR USEFULNESS OF THIS INFORMATION EXCEPT AS REQUIRED BY APPLICABLE LAW OR EXPRESSLY AGREED IN WRITING.

SSH Communications Security Corp
Fredrikinkatu 42; FIN-00100 Helsinki; FINLAND

SSH Communications Security Inc.
1076 East Meadow Circle; Palo Alto, CA 94303; USA

SSH Communications Security K.K
2-7-1, House Hamamatsu-cho Bldg. 5F; Hamamatsu-cho; Minato-ku, Tokyo 105-0013; JAPAN

http://www.ssh.com/
e-mail: ssh-sales@ssh.com (sales), http://www.ssh.com/support/ssh/
Tel: +358 303 9870 (Finland), +1 650 251 2700 (USA), +81 3 3459 6830 (Japan)
Fax: +358 303 9871 (Finland), +1 650 251 2701 (USA), +81 3 3459 6825 (Japan)

Contents

 © 2000 SSH Communications Security

Chapter 1

Introduction to SSH Secure Shell

This chapter provides an introduction to the SSH Secure Shell software suite.

1.1 SSH Secure Shell

SSH Secure Shell is an application protocol and software suite that allows secure network services over an insecure network such as the public Internet. It replaces other, insecure protocols and services, including Telnet and FTP. It can be used for remote terminal connections, remote file copying, and forwarding X11 sessions (on Unix) as well as arbitrary TCP ports through a secure tunnel.

SSH Secure Shell is based on strong encryption and authentication. The software has been developed in Europe and can be used in any country that allows encryption.

 © 2000 SSH Communications Security

1.2 This Document

This document, the *SSH Secure Shell for UNIX Servers Administrator's Guide* is a brief introduction to the basic functions of SSH Secure Shell and the most critical administrator tasks. For more information about specific details of SSH Secure Shell usage, please consult the manual pages included in the distribution. For a comprehensive text about the various aspects of SSH Secure Shell, we recommend *UNIX Secure Shell* by Anne Carasik (McGraw-Hill, New York, 1999).

1.3 Supported Platforms

SSH Secure Shell software is available for a wide variety of hardware and software platforms. The Secure Shell concept originated on Unix as a replacement for the insecure "Berkeley services", that is, the `rsh`, `rlogin`, and `rcp` commands. SSH Secure Shell Server software (the software component that allows remote users to connect to your computer) is available for most Unix and Linux platforms and for Microsoft Windows NT4 (Service Pack 5 required) and Windows 2000. The associated client software (the component that remote users run on their computers) is also available for Microsoft Windows 95 and Windows 98.

Independent third parties have also ported Secure Shell to other platforms such as OS/2 and VMS. These independent software products are intended to be compatible with SSH Secure Shell products. However, SSH Communications Security can only provide support for its own software.

1.4 Different Versions of the SSH Protocol

The current version of the SSH protocol is version 2 (SSH2).

Several different versions of the Secure Shell client and server exist. Please note

that the different versions use different implementations of the SSH protocol, and therefore you cannot normally connect to an SSH version 1 server using SSH version 2 client software, or vice versa. However, SSH version 2.4 Unix server software includes support for fallback functionality if SSH1 is already installed. The Windows NT server software does not include this functionality.

For optimal results, upgrade all servers and clients to the newest available version of SSH Secure Shell.

1.5 Support

If the product documentation and the README files do not answer your questions, you can contact the SSH customer support by using the support Web pages at `http://www.ssh.com/support/ssh/`.

When reporting problems, please provide the following information to make it easier for the technical support personnel to help you:

- the version number of your SSH Secure Shell client and server
- list of any error messages you have received
- information about your computer hardware and software setup.

1.6 Legal Issues with Encryption

The encryption software included in SSH Communications Security products has been developed in Europe, and therefore these products are not subject to US export regulations.

The Secure Shell software can be used in any country that allows encryption, including the United States of America.

Chapter 2

Installing and Configuring Secure Shell

This chapter contains instructions on how to install and configure the Secure Shell server and client software.

2.1 Installation

The actual installation and system configuration of the Secure Shell software is dependent on the particular platform. For installation instructions on platforms not covered here, please consult the platform-specific documentation shipped with your software package.

 © 2000 SSH Communications Security

2.1.1 Installing on Linux Platforms

SSH Secure Shell products for Linux platforms are supplied in RPM (Red Hat Package Manager) binary packages.

Please note that the binary RPMs are intended for Red Hat and SuSE Linux distributions running on an Intel x86 platform. On other platforms that use the RPM package manager, the installation of the appropriate files will probably succeed, but the configuration phase might fail. In this case, you must do the configuration manually, as if you were installing directly from source files.

Installation

1. Change your working directory to the directory where you have copied the software to be installed and issue the following command with root privileges:

   ```
   rpm -ihv ssh-commercial-server-2.x.y-z.i386.rpm
   ```

   The command varies a bit according to the software and RPM release version. For example, `server` might be replaced with `workstation`, and the letters `x.y-z` should be replaced with the appropriate release number.

   If you have previous SSH Secure Shell RPMs installed, issue the following command with root privileges:

   ```
   rpm -Uhv ssh-commercial-server-2.x.y-z.i386.rpm
   ```

2. After issuing the command, the software will be installed. You might be asked to accept the License Agreement if you have not done so previously on the particular computer, or if the License has changed from the previous version.

3. The software should now be ready to use. If you already had the Secure Shell daemon running, you might want to restart it or reboot the computer.

Uninstallation

1. Uninstallation is accomplished by issuing the following command with root privileges:

   ```
   rpm -e ssh-commercial-server
   ```

   Once again, `server` may be replaced with something else, depending on the actual software version.

2. Please notice that even after a successful uninstallation, the Secure Shell daemon will be left running. You must kill it manually:

   ```
   kill ‘cat /var/run/sshd2_22.pid‘
   /etc/rc.d/init.d/sshd2 stop
   ```

2.1.2 Installing on Solaris SPARC Platforms

The package includes compiled binaries for Solaris 2.6, 7 and 8 on the SPARC architecture. For Solaris on the Intel x86 platform, no pre-compiled binaries are available.

Note: If you want to compile the source code yourself, we recommend the usage of Sun Microsystems' proprietary C compiler (Forte C, formerly Sun WorkShop Professional C, or its equivalent).

Installation with `pkgadd`

Unpack the distribution binary to some suitable place. The standard place is `/var/spool/pkg` in a Solaris environment.

```
gzip -dc ssh-2.x.y-server-solaris.tar.gz | tar xvf -
```

 © 2000 SSH Communications Security

The string `2.x.y` should be replaced with the appropriate version number.

After unpacking, install the package with the `pkgadd` tool.

```
pkgadd -d .
```

Installation without `pkgadd`

`Pkgadd` sets up a few variables which the installation script will use. If you are using the script without `pkgtool`, you have to set them up yourself:

```
VERSION=2.x.y               (x.y = the version number)
BASEDIR=/var/spool/pkg/SSHssh2/reloc
```

Remember also to export the variables.

Unpack the distribution to some suitable temporary space. The above `BASEDIR` applies only if you put the package under `/var/spool/pkg`. In this case, you have to do the installation manually, using the `postinstall` command.

```
gzip -dc ssh-2.x.y-server-solaris.tar.gz | tar xvf -
cd SSHssh2/reloc
tar cf - . | (cd /usr/local; tar xfBp -)
cd ../install
./postinstall
```

Uninstallation

1. Stop the Secure Shell daemon using the command

   ```
   kill ‘cat /etc/ssh2/sshd2_22.pid‘
   ```

2. Uninstall the package by issuing one of the following commands with root privileges:

   ```
   pkgrm SSHssh2
   ```

   or

   ```
   pkgrm SSHssh2.2
   ```

2.1.3 Installing on HP-UX

Installation

1. Unpack the package with `gunzip.`
2. Install the package by issuing the following command with root privileges:

   ```
   swinstall -s path_to/package ssh2
   ```

   The `/path_to/package` is the absolute path and name of the distribution file.

   The software will be installed in the `/opt/ssh2` directory, and the manual pages will be installed in the `/usr/man` directory. Symbolic links for binaries will be created in the `/usr/bin` and `/usr/sbin` directories.
3. Start the Secure Shell daemon using the command

   ```
   /sbin/init.d/sshd start
   ```

Uninstallation

1. Stop the Secure Shell daemon using the command

   ```
   /sbin/init.d/sshd stop
   ```

2. Uninstall the package by issuing the following command with root privileges:

   ```
   swremove ssh2
   ```

Please notice that even after a successful uninstallation, the Secure Shell daemon will be left running. You must kill it manually. Also, uninstallation does not remove any configuration files.

2.1.4 Installing on AIX 4.3.x

Note: If you want to compile the source code yourself, we recommend the usage of IBM's proprietary C compiler (IBM C for AIX or its equivalent).

Installation

1. Unpack the package:

   ```
   gzip -dc package | tar -xvf -
   ```

   The `package` is the name of the distribution file.

2. Install the package by issuing the following command with root privileges:

   ```
   installp -d . SSH.Secure.Shell
   ```

   If you only want to apply and not commit the package, you can use the `-a` flag with `installp`. Packages which are applied but not committed can be rejected later on. Please read the AIX manual pages for more information about the `installp` command.

Uninstallation

1. Stop the Secure Shell daemon using the command

   ```
   kill ‘cat /etc/ssh2/sshd2_22.pid‘
   ```

2. Uninstall the package by issuing the following command with root privileges:

   ```
   installp -u SSH.Secure.Shell
   ```

2.1.5 Installing on Other UNIX Platforms

If pre-compiled binaries from SSH Communications Security do not exist for your particular UNIX platform, you can compile the source yourself.

You need to have the following:

- An ANSI C compiler (`gcc` and `egcs` are available from the Free Software Foundation’s GNU project, `http://www.gnu.org`)
- Development libraries for your operating system

Then, login as `root,` and run the following commands.

```
gzip -dc ssh-2.x.y.tar.gz | tar -xvf -
cd ssh-2.x.y
./configure
make
make install
```

You can enable or disable certain functionality when you compile SSH Secure Shell. To use the optional functionality, just make sure you do it in the following syntax:

```
# ./configure --[option]
```

The most common options are listed below, but there are also additional options not listed here. Type `./configure --help` for more information.

`--disable-X11-forwarding`

Turns off X forwarding

`--bindir=DIR`

Install user binaries in `[bindir]/bin`

`--sbindir=DIR`

Install system binaries in `[sbindir]/bin`

`--enable-debug`

Enables debugging (recommended)

`--host=HOST`

Define the operating system to install on (listed in `config.sub`)

`--with-ssh-connection-limit=#`

Number of simultaneous connections allowed to *sshd2*

2.2 Basic Configuration

This section goes into some of the basic configuration options that many administrators like to have set up (or not set up, depending on the scenario). These include the basic files that Secure Shell uses, host key generation, X11 forwarding, and root logins.

2.2.1 Location of Secure Shell Files

The system files for SSH2 are by default in `/etc/ssh2.` The user and system binaries are stored in `/usr/local/bin` and `/usr/local/sbin,` respectively. In `/usr/local/sbin,` you'll find *sshd2*. All the other binaries are stored in `/usr/local/bin.`

The system-wide configuration files are the most important. They are in `/etc/ssh2/.`

The system public key pair (DSS only):

- `/etc/ssh2/hostkey`
- `/etc/ssh2/hostkey.pub`

The configuration files for the client and server, respectively:

- `/etc/ssh2/ssh2_config`
- `/etc/ssh2/sshd2_config`

Users can have their own configuration files (and other files as well). These are stored in `~/.ssh2.`

Host keys that are recognized for any users on the local system should be placed in the `/etc/ssh2/hostkeys` directory.

User-specific host keys should be in `~/.ssh2/hostkeys.`

If you are using host-based authentication, the system-wide file for recognized host keys is `/etc/ssh2/knownhosts.`

User-specific known-hosts keys should be in `~/.ssh2/knownhosts.`

2.2.2 Generating the Host Key

You only need to do this if you want to change your host key, or if your host key was not generated during the installation.

1. Login as `root`
2. Kill any instances of `sshd2` or `sshd`:

   ```
   killall sshd
   ```

3. Generate the host key with the following command:

   ```
   ssh-keygen2 -P /etc/ssh2/hostkey
   ```

4. Restart `sshd2`:

   ```
   sshd2
   ```

2.2.3 Permitting Root Logins

If you want to permit someone to login directly to the *root* login account via ssh, you can define three methods of control in the `/etc/ssh2/sshd2_config` file.

```
PermitRootLogin         no
```

This will disable all root logins. To enable root logins with any authentication method, use the following setting:

```
PermitRootLogin         yes
```

You can limit the authentication methods by using the following setting:

```
PermitRootLogin          nopwd
```

This will enable root logins only using host-based authentication or user public key authentication.

2.3 Authentication

There are four different methods to authenticate users in SSH Secure Shell: password, host-based, user public key and Kerberos. These authentication methods can be combined or used separately, depending on the level of functionality and security you want.

2.3.1 Password

This authentication method is the easiest to implement, as it is set up by default. Password authentication uses the `/etc/passwd` or `/etc/shadow` file on your UNIX system, depending on how your passwords are set up.

To make sure password authentication is enabled, the `AllowedAuthentications` field both in `/etc/ssh2/sshd2_config` and `/etc/ssh2/ssh2_config` files should contain the word `password`:

```
AllowedAuthentications   password
```

Other authentication methods can be listed in the configuration files as well.

 © 2000 SSH Communications Security

2.3.2 Host-Based Authentication

The following terms will be used in this example:

Remote is the SSH Secure Shell server into which you are trying to connect. *RemoteUser* is the user name on the server into which you would like to login. *Local* is the machine running a SSH Secure Shell client. *LocalUser* is the user name on the client machine that should be allowed to login to *Remote* as *RemoteUser.*

1. First, install SSH Secure Shell on the *Local* and *Remote* machines. Do not forget to generate a host key. If your installation did this, or if you already have a copy of your `/etc/ssh2/hostkey` and `/etc/ssh2/hostkey.pub`, you can skip the host key generation. Otherwise, do this:

   ```
   # ssh-keygen2 -P /etc/ssh2/hostkey
   ```

2. Copy the *Local* machine's `/etc/ssh2/hostkey.pub` file over to the *Remote* machine and name it

   `/etc/ssh2/knownhosts/`*`hostname.domain`*`.ssh-dss.pub`

   In the place of *`hostname.domain`* above, you must use the long host name of the *Local* machine (the fully qualified domain name). You will run into problems if the system does not recognize the host name as `hostname.domain.somewhere.com` but recognizes it only as `hostname`. You can find this out while running `sshd2` in verbose mode when trying to make connections.

   The *Remote* machine now has the *Local* machine's public key, so the *Remote* machine can verify the *Local* machine's identity based on a public key signature. By contrast, `rsh` only uses the IP address for host authentication.

3. To make sure that SSH Secure Shell finds your complete domain name, not just the host name, edit the following line in the `/etc/ssh2/ssh2_config` file on *Local*:

```
DefaultDomain   yourdomain.com
```

4. On the *Remote* machine, create a file in the home directory of *RemoteUser*, named `.shosts`. The contents of this file should be the long host name of *Local*, some tabs or spaces, and the user name of *LocalUser*.

   Contents of `~/.shosts`:

   ```
   localhostname.yourdomain.com   LocalUser
   ```

   Be sure to `chown` and `chmod` the `.shosts` file. The `.shosts` file must be owned by *RemoteUser* and should have mode 0400.

5. Check the files `/etc/ssh2/sshd2_config` on *Remote* and `/etc/ssh2/ssh2_config` on *Local*. Make sure that the `AllowedAuthentications` field contains the word `hostbased`. For example, it may read:

   ```
   AllowedAuthentications      hostbased,passwd
   ```

   It does not matter what else is in there. Just make sure that the `hostbased` keyword is first in the list.

6. Also check that `IgnoreRhosts` is set to `no` in the your `/etc/ssh2/sshd2_config` file on *Remote*.

   ```
   IgnoreRhosts      no
   ```

   If you had to modify the `sshd2_config` file, you will have to send a HUP signal to `sshd2` to make the change take effect.

   ```
   # kill -HUP ‘cat /var/run/sshd2_22.pid‘
   ```

   or

   ```
   # kill -HUP ‘cat /etc/ssh2/sshd2_22.pid‘
   ```

7. You should be all set.

   On *Local*, log in as *LocalUser* and give the command

   ```
   ssh RemoteUser@Remote uptime
   ```

   You should get back the results of `uptime` run on *Remote*.

   The first time you run `ssh` to that particular server, you will have to answer *yes* when asked if you want to connect to the server. This is because the local `ssh` does not yet have the remote server's SSH public key. This will only happen when connecting for the first time.

Troubleshooting

With SSH2, did you name the host key file appropriately? It should be `/etc/ssh2/knownhosts/HOSTNAME.ssh-dss.pub`, and `HOSTNAME` has to be the long host name (fully qualified domain name).

- Did you copy the host key properly?
- Did it get mangled when you copied it over?
- Check your spelling in the `.shosts` file.
- Make sure the `.shosts` file is owned by *RemoteUser*.
- Run the server with the `-v` flag (verbose). This is a good way to see if a host key file is missing, or if something is misconfigured.

2.3.3 User Public Key Authentication

Per-user configuration information and encryption keys are stored in the `.ssh2` subdirectory of each user's home directory.

In the following instructions, *Remote* is the SSH Secure Shell server machine into which you are trying to connect, and *Local* is the machine running an SSH Secure Shell client.

Keys generated with ssh-keygen

In order to set up user public key authentication, either use the Public Key Manager, `ssh-pubkeymgr`, or do a manual setup according to the following instructions. See Section 3.7 (Using Public Key Manager) if you want to know more about `ssh-pubkeymgr`.

1. To make sure that public key authentication is enabled, the `AllowedAuthentications` field both in `/etc/ssh2/sshd2_config` file on *Remote* and in `/etc/ssh2/ssh2_config` file on *Local* should contain the word `publickey`:

   ```
   AllowedAuthentications   publickey
   ```

   Other authentication methods can be listed in the configuration file as well.

2. Create a keypair by executing `ssh-keygen` (ssh-keygen2) on *Local*.

   ```
   Local> ssh-keygen
   Generating 1024-bit dsa key pair
      1 oOo.oOo.o
   Key generated.
   1024-bit dsa, user@Local, Wed Mar 22 2000 00:13:43
           +0200
   Passphrase :
   Again :
   Private key saved to
                 /home/user/.ssh2/id_dsa_1024_a
   Public key saved to
                 /home/user/.ssh2/id_dsa_1024_a.pub
   ```

 © 2000 SSH Communications Security

`Ssh-keygen` will ask you for a passphrase for the new key. Enter a sufficiently long (20 characters or so) sequence of any characters (white spaces are OK). `Ssh-keygen` creates a `.ssh2` directory in your home directory, and stores your new authentication key pair in two separate files. One is your private key which must NEVER be made available to anyone but yourself. The private key can only be used together with the passphrase. In the above example, the private key file is `id_dsa_1024_a`. The other file `id_dsa_1024_a.pub` is your public key, which can be distributed to other computers.

3. Create an `identification` file in your `~/.ssh2` directory on *Local*.

   ```
   Local> cd ~/.ssh2
   Local> echo "IdKey id_dsa_1024_a" > identification
   ```

   You now have an `identification` file which consists of one line that denotes the file containing your identification (your private key). For special applications, you can create multiple identifications by executing `ssh-keygen` again. This is, however, not needed in the most common cases.

4. Copy your public key (`id_dsa_1024_a.pub`) to the `~/.ssh2` directory on *Remote*.

5. Create an `authorization` file in your `~/.ssh2` directory on *Remote*. Add the following line to `authorization`:

   ```
   Key     id_dsa_1024_a.pub
   ```

   This directs the SSH server to use `id_dsa_1024_a.pub` as a valid public key when authorizing your login. If you want to login to `Remote` from other hosts, create authorization keys on the hosts (steps 1 and 2) and repeat steps 3 and 4 on Remote.

6. Now you should be able to login to *Remote* from *Local* using Secure Shell.

   Try to login:

```
Local>ssh Remote
Passphrase for key "/home/user/.ssh2/id_dsa_1024_a
with comment "1024-bit dsa, created by user@Local
Wed Mar 22 2000 00:13:43 +0200":
```

After you have entered the passphrase of your private key, a Secure Shell connection will be established.

PGP keys

The SSH Secure Shell only supports the OpenPGP standard and the PGP programs that use it. GnuPG is used in the following instructions. If you use PGP, the only difference is that the file extension is `pgp` instead of `gpg`.

1. To make sure that user public key authentication is enabled, the `AllowedAuthentications` field both in `/etc/ssh2/sshd2_config` file on *Remote* and `/etc/ssh2/ssh2_config` file on *Local* should contain the word `publickey`:

   ```
   AllowedAuthentications   publickey
   ```

   Other authentication methods can be listed in the configuration file as well.

2. Copy your private keyring (`secring.gpg`) to the `~/.ssh2` directory on *Local*.

3. Create an `identification` file in your `~/.ssh2` directory on *Local* if you don't already have one. Add the following lines to `identification`:

   ```
   PgpSecretKeyFile
     <the filename of the user's private keyring>
   IdPgpKeyName
   ```

 © 2000 SSH Communications Security

```
  <the name of the OpenPGP key
                              in PgpSecretKeyFile>
IdPgpKeyFingerprint
  <the fingerprint of the OpenPGP key
                              in PgpSecretKeyFile>
IdPgpKeyId
  <the id of the OpenPGP key in PgpSecretKeyFile>
```

Each keyword and the corresponding value should be written on the same line. In this document, they are shown on separate lines for typographical reasons.

4. Copy your public keyring (`pubring.gpg`) to the `~/.ssh2` directory on *Remote*

   ```
   scp2 pubring.gpg user@remote_host:.ssh2
   ```

5. Create an `authorization` file in your `~/.ssh2` directory on *Remote*. Add the following lines to `authorization`:

   ```
   PgpPublicKeyFile
     <the filename of the user's public keyring>
   PgpKeyName
     <the name of the OpenPGP key>
   PgpKeyFingerprint
     <the fingerprint the OpenPGP key>
   PgpKeyId
     <the id of the OpenPGP key>
   ```

6. Now you should be able to login to *Remote* from *Local* using Secure Shell.

   Try to login:

   ```
   Local>ssh Remote
   Passphrase for pgp key "user (comment)
                          <user@Local>":
   ```

   After you have entered the passphrase of your PGP key, a Secure Shell connection will be established.

2.3.4 Kerberos Authentication

The SSH Secure Shell only supports Kerberos5.

1. To make sure that Kerberos authentication is enabled, you should have the following line in your `/etc/ssh2/sshd2_config` file:

   ```
   AllowedAuthentications
            kerberos-1@ssh.com,kerberos-tgt-1@ssh.com
   ```

   Other authentication methods can be listed in the configuration file as well.

2. Also, make sure that you have the following line in your `/etc/ssh2/ssh2_config` file:

   ```
   AllowedAuthentications
            kerberos-1@ssh.com,kerberos-tgt-1@ssh.com
   ```

   Each keyword and the corresponding value should be written on the same line. In this document, they are shown on separate lines for typographical reasons.

After this, it is possible to authenticate using Kerberos credentials, forwardable TGT (ticket granting ticket) and passing TGT to remote host for single sign-on. It is also possible to use Kerberos password authentication.

2.3.5 Pluggable Authentication Modules (PAM)

When PAM is used, SSH Secure Shell transfers the control of authentication to the Linux-PAM library, which will then load the modules specified in the PAM configuration file. Finally, the Linux-PAM library tells SSH Secure Shell whether or not the authentication was successful. SSH Secure Shell neither knows or cares of the actual authentication method employed by Linux-PAM. Only the final result is of interest.

 © 2000 SSH Communications Security

1. In order to have PAM support, your need to compile the source:

   ```
   ./configure
   make
   make install
   ```

   By default, the PAM service name is `sshd2`. If you want to change it, you can add the configure flag `--with-daemon-pam-service-name=name`.

2. Make sure that you have the following lines in your `/etc/ssh2/sshd2_config` file:

   ```
   AllowedAuthentications  pam-1@ssh.com
   SshPamClientPath     /full/path/to/ssh-pam-client
   ```

   Note: By default, *SshPamClientPath* is `/usr/local/bin/ssh-pam-client`.

3. Edit your `/etc/ssh2/ssh2_config` file so that the *pam-1@ssh.com* authentication method is allowed.

The PAM configuration is either in `/etc/pam.conf` or in `/etc/pam.d/sshd2`. The modules are usually either in the `/lib/security` directory or in the `/usr/lib/security` directory. Currently, SSH Secure Shell supports PAM on Linux and on Solaris 2.6 or later.

There must be at least one `auth`, one `account` and one `session` module in the configuration file. Otherwise, the connection will be refused. Also, modules which require `PAM_TTY` will not work because TTY allocation is done in SSH Secure Shell after the authentication.

Examples

1. `/etc/pam.d/sshd2` file on Red Hat Linux:

```
auth      required  /lib/security/pam_pwdb.so
                                    shadow nullok
auth      required  /lib/security/pam_nologin.so
account   required  /lib/security/pam_pwdb.so
password  required  /lib/security/pam_cracklib.so
password  required  /lib/security/pam_pwdb.so
                        shadow nullok use_authtok
session   required  /lib/security/pam_pwdb.so
```

2. `/etc/pam.conf` entry on Solaris:

```
sshd2  auth      required
             /usr/lib/security/pam_sample.so debug
sshd2  account   required
             /usr/lib/security/pam_sample.so debug
sshd2  password  required
             /usr/lib/security/pam_sample.so debug
sshd2  session   required
             /usr/lib/security/pam_sample.so debug
```

Each keyword and the corresponding value should be written on the same line. In this document, they are shown on separate lines for typographical reasons.

Note: SSH Communications Security does not provide technical support on how to configure PAM. Our support covers only SSH Secure Shell applications and source code.

2.3.6 SecurID

Please familiarize yourself with the RSA ACE/Server documentation before reading further.

In the instructions below, the `/top` directory refers to the RSA ACE/Server top-level directory.

1. In order to have SecurID support, you need to compile the source:

   ```
   ./configure --with-serversecurid[=/PATH]
               --with-clientsecurid
   make
   make install
   ```

   Replace `/PATH` with the absolute PATH to the directory containing the following files:

   - `sdclient.a`
   - `sdacmvls.h`
   - `sdconf.h`
   - `sdi_authd.h`
   - `sdi_size.h`
   - `sdi_type.h`
   - `sdi_defs.h`

   The above files are normally in `/top/ace/examples`.

   Note: If you do not want to make the compilation as root, make sure that all the above files are readable.

2. Make sure that you have the following line both in your `/etc/ssh2/sshd2_config` file and in your `/etc/ssh2/ssh2_config` file:

   ```
   AllowedAuthentications  securid-1@ssh.com
   ```

3. Check that the user's shell is **not** `/top/ace/prog/sdshell`.

4. Start the RSA ACE/Server.

5. Check that the VAR_ACE environment variable is set. It has to be set before starting `sshd2`, and its value must be `/top/ace/data`.

6. Start `sshd2`.

Note: SSH Communications Security does not provide technical support on how to configure RSA ACE/Server. Our support covers only SSH Secure Shell applications and source code.

2.4 Forwarding

The SSH2 connection protocol provides channels that can be used for a wide range of purposes. All of these channels are multiplexed into a single encrypted tunnel and can be used for forwarding (”tunneling”) arbitrary TCP/IP ports and X11 connections.

2.4.1 X11 Forwarding

To enable X11 forwarding, make sure that the SSH Secure Shell software was compiled with X (you didn’t run `./configure` with any X disabling options). Also, make sure that you have this line in your `/etc/ssh2/sshd2_config` file:

```
ForwardX11                      yes
```

Log into the remote system and type `xclock &`. This starts a X clock program that can be used for testing the forwarding connection. If the X clock window is displayed properly, you have X11 forwarding working fine.

NOTE: Do *NOT* set the `DISPLAY` variable on the client. You will most likely disable encryption. (X connections forwarded through Secure Shell use a special local display setting.)

To forward X11 traffic on the SSH Secure Shell for Workstations Windows client:

1. Install an X server (X emulation) program on Windows (eXceed, Reflection, or the like)
2. Start the SSH Secure Shell for Workstations Windows client
3. Select *Edit -> Settings... -> Tunneling* and make sure that the *Forward X11 connections* checkbox is checked
4. Save your settings for the SSH Secure Shell for Workstations Windows client
5. Quit the Windows client, start it again and log into the remote host
6. Start the X server (X emulation) program
7. Run `xterm` or `xclock` from SSH Secure Shell, and it should work.

2.4.2 Port Forwarding

Port forwarding, in other words tunneling, is a way to forward otherwise insecure TCP traffic through SSH Secure Shell. For example, you can secure POP3, SMTP and HTTP connections that would otherwise be insecure. (Figure 2.1 (Making insecure TCP connections secure using channels inside the encrypted ssh2 tunnel).)

Figure 2.1: Making insecure TCP connections secure using channels inside the encrypted ssh2 tunnel

There are two kinds of port forwarding: local and remote forwarding. They are also called outgoing and incoming tunnels, respectively. Local port forwarding forwards traffic coming to a local port to a specified remote port.

For example, if you issue the command

```
ssh2 -L local_port:remote:remote_port user@remote
```

all traffic which comes to port `local_port` on the local host will be forwarded to port `remote_port` on the remote host.

Remote port forwarding does the opposite: it forwards traffic coming to a remote port to a specified local port.

For example, if you issue the command

```
ssh2 -R remote_port:local:local_port user@remote
```

all traffic which comes to port `remote_port` on the remote host will be forwarded to port `local_port` on the local host.

If you have three hosts, `client`, `sshdserver` and `appserver`, and you forward the traffic coming to `client`'s port `x` to `appserver`'s port `y` but you connect to `sshdserver` only, the connection between `client` and `sshdserver` is secure. See Figure 2.2 (Forwarding to a third host). The command you use would look like the following:

```
ssh2 -L x:appserver:y user@sshdserver
```

2.4.3 Agent Forwarding

See section 3.5 (Using Authentication Agent).

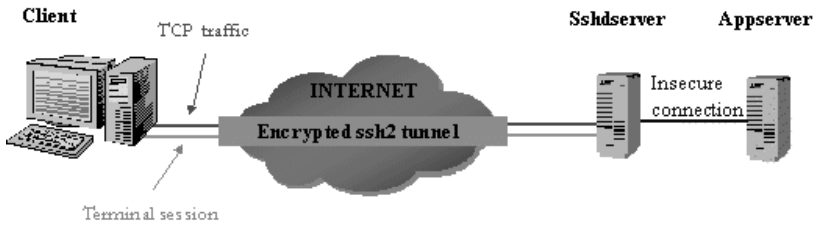

Figure 2.2: Forwarding to a third host.

2.5 Configuring SSH2 for SSH1 Compatibility

The SSH2 and SSH1 protocols are not compatible with each other. This inconvenience is necessary, since the SSH2 protocol includes remarkable security and performance enhancements that would not have been possible if protocol-level compatibility with SSH1 had been retained.

However, the current implementations of SSH2 and SSH1 are designed so that they can both be run on the same machine. This makes the transition from the old but well-established SSH1 protocol to the more secure and more flexible SSH2 protocol much easier. The SSH2 server daemon includes a fallback function that automatically invokes the SSH1 server when required.

To set up both SSH1 and SSH2 servers on the same Unix system, you should do the following:

1. Install the latest available version of SSH1. (As of this printing, the latest version is `ssh-1.2.30`.) The SSH1 compatibility fallback requires version 1.2.26 or later.
2. Install SSH2.
3. If you previously had SSH1 installed, please make sure that the old `sshd` is no longer run at boot. Only `sshd2` should be run. If you have the SSH1

version of `sshd` running, you should kill the master daemon. You can find its process id in `/var/run/sshd.pid`.

4. Make sure that `/usr/local/sbin/sshd2` is run automatically at boot. On most systems, you should add running it into `/etc/rc.local` or under `/etc/rc.d`.

   - When you run `sshd2`, the SSH1 daemon should not be running. When using SSH2 with SSH1 compatibility, you should only run `sshd2`. It will then automatically start SSH1 as needed.

5. If you don't want to reboot, you should now manually run `/usr/local/sbin/sshd2`.

2.6 Configuring SSH Secure Shell for TCP Wrappers Support

To enable usage of TCP Wrappers with SSH Secure Shell, do the following as root:

1. If SSH Secure Shell was previously installed from binaries, you may want to uninstall it before continuing.

2. Compile the source code:

   ```
   ./configure --with-libwrap
   make
   make install
   ```

   Note: If `configure` does not find `libwrap.a`, do the following:

   - Locate `libwrap.a`
   - Run `configure` again:

```
make distclean
./configure --with-libwrap=/path_to/libwrap.a
```

3. Create or edit the `/etc/hosts.allow` and `/etc/hosts.deny` files.

   When a user tries to connect to SSH Secure Shell server, the TCP wrapper daemon (`tcpd`) reads the `/etc/hosts.allow` file for a rule that matches the client's hostname or IP. If `/etc/hosts.allow` doesn't contain a rule allowing access, `tcpd` reads `/etc/hosts.deny` for a rule that would deny access. If neither file contains an accept or deny rule, access is granted by default.

   The syntax for the `/etc/hosts.allow` and `/etc/hosts.deny` files is as follows:

   ```
   daemon : client_hostname_or_IP
   ```

   The typical setup is to deny access to everyone in the `/etc/hosts.deny` (This example shows both SSH1 and SSH2):

   ```
   sshd1: ALL
   sshd2: ALL
   sshdfwd-X11 : ALL
   ```

   or simply

   ```
   ALL: ALL
   ```

   And then allow access only to trusted clients in the `/etc/hosts.allow`:

   ```
   sshd1 : trusted_client_IP_or_hostname
   sshd2 : .ssh.com foo.bar.fi
   sshdfwd-X11 : .ssh.com foo.bar.fi
   ```

   Based on the `/etc/hosts.allow` file above, users coming from any host in the *ssh.com* domain or from the host *foo.bar.fi* are allowed to get in.

Troubleshooting

1. Make sure that you are not having any network problems.
2. Make sure that SSH Secure Shell server is running:

   ```
   kill -0 ‘cat /var/run/sshd2_22.pid‘
   ```

   or

   ```
   kill -0 ‘cat /etc/ssh2/sshd2_22.pid‘
   ```

   If you get a message ”No such process.”, restart the `sshd2` daemon.
3. Check your `/etc/hosts.allow` and `/etc/hosts.deny` files.
   - Ensure that the client’s IP address or host name is correct.
   - If you are using a host name, you must supply the fully qualified domain name.
4. If you changed something in the `sshd2_config` file, you need to `HUP` the `sshd2` daemon.
5. Run `tcpdchk` and `tcpdmatch`. These programs are used to analyze and report problems with your TCP Wrappers setup. Please see the man pages for more information on these commands.

Chapter 3

Using Secure Shell

This chapter provides information on how to use the Secure Shell software suite after it has been successfully installed and set up.

3.1 Using the Secure Shell Server Daemon (sshd2)

The server daemon program for Secure Shell is called `sshd2`.

`Sshd2` is normally started at boot time from `/etc/rc.local` or its equivalent. It forks a new daemon for each incoming connection. The forked daemons handle key exchange, encryption, authentication, command execution, and data exchange.

The Secure Shell daemon is normally run as root. If it is not run as root, it can only log in as the user it is running as, and password authentication may not work if the system uses shadow passwords. An alternative host key file must also be used.

3.1.1 Manually Starting the Secure Shell Server Daemon

To manually start the Secure Shell daemon, type the command `sshd`. (**Note:** If the installation was successfully completed, `sshd` is a symbolic link to `sshd2`. If you also have SSH1 installed, the SSH2 installation process modifies the existing link. If SSH1 compatibility is desired, `sshd2` can be configured to execute `sshd1` when the client only supports SSH1.)

Sshd2 can be configured using command-line options or a configuration file. Command-line options override values specified in the configuration file.

3.1.2 Automatically Starting the Secure Shell Server Daemon at Boot Time

If you have installed from RPM packages on RedHat or on SuSE, `sshd2` is already starting at boot time. The same is true if you have installed from `depot` in HP-UX.

In the following, two different ways of starting Secure Shell Daemon at boot time are introduced. If neither of these work in your system, refer to your operating system documentation on how to start services at boot time.

Starting from `/etc/rc.d/rc.local`

In order to start `sshd2` automatically at boot time on System V based operating systems, there should be symbolic links to its startup script in `/etc/rc.d/rc?.d`, where ? is the runlevel. You can either add these links manually or use `chkconfig`. The startup script `sshd2` should be in the `/etc/rc.d/init.d` directory.

Note: `Chkconfig` is only available on RedHat. In SuSE, add the symbolic links manually.

If you want to use `chkconfig`, check that the first lines in `sshd2` are similar to the following ones:

```
#!/bin/sh
#
# Author: Sami Lehtinen <sjl@ssh.com>
#
# sshd2    This shell script takes care of starting
#          and stopping sshd2.
#
# chkconfig: 345 34 70
# description: Secure Shell daemon
#
```

This means that `sshd` will be started in runlevels 3, 4 and 5, and that its starting priority is 34 and its killing priority is 70. You can choose the runlevels and priorities as you want as long as `sshd` is started after the network is up.

After adding the links manually or giving the command

```
chkconfig --add sshd2
```

you should have links `/etc/rc.d/rc?.d`, similar to

```
lrwxrwxrwx 1 root  root  14 Aug 16 10:07
                           S34sshd -> ../init.d/sshd
lrwxrwxrwx 1 root  root  14 Aug 16 10:07
                           K70sshd -> ../init.d/sshd
```

Starting from `/etc/rc.local`

On BSD based operating systems, you have to add a similar line to the following to the `rc.local` file in `/etc` directory:

```
echo "Starting sshd2..."; /usr/local/sbin/sshd2
```

After this, the Secure Shell daemon will start automatically at boot time.

3.1.3 Operation of the Server Daemon

When `sshd2` is started, it begins to listen on a port for a socket. The default port is port 22, now a well-known port for Secure Shell. This can be changed to suit any custom environments, e.g. if you want to run `sshd2` from a non-privileged account; however, make sure that no other process is using the port you are planning to use. The Secure Shell daemon can also be started from the Internet daemon `inetd`. For the purpose of this text, it is assumed that `sshd2` is not invoked through `inetd` but started on its own.

When the daemon is listening for a socket, it waits until a client initiates a socket connection. Once connected, the daemon forks a child process, which in turn initiates key exchange with the client. The child process handles the actual connection with the client, including authentication, supported cipher negotiation, encrypted data transfer, and termination of the connection. After the connection has been terminated, the child process terminates as well. The parent process remains listening for other connections until explicitly stopped.

Login Process

When a user successfully logs in, `sshd2` does the following:

1. Changes to run with normal user privileges.
2. Sets up basic environment.
3. Reads `/etc/environment` if it exists.
4. Changes to the user's home directory.

5. Runs the user's shell or specified command.

3.1.4 Resetting and Stopping the Server Daemon

When the Secure Shell daemon is started, its process identifier (PID) is stored in `/var/run/sshd2_22.pid` or, if the directory `/var/run` does not exist, in `/etc/ssh2/sshd2_22.pid`. This makes it easy to kill the appropriate daemon:

```
kill ‘cat /var/run/sshd2_22.pid‘
```

or send signals to it:

```
kill -SIGNAL ‘cat /var/run/sshd2_22.pid‘
```

The Secure Shell daemon handles signals like `inetd`: you can send it a `SIGHUP` signal to make it reread its configuration file. The daemon can be stopped by sending the `SIGKILL` signal.

3.1.5 Configuration File and Command-Line Options

`Sshd2` reads configuration data from `/etc/ssh2/sshd2_config` (or the file specified with `-f` on the command line). The file contains keyword-value pairs, one per line. Lines starting with a number sign # as well as empty lines are interpreted as comments.

For detailed information about the options available in the configuration file and on the command line, please refer to the *sshd2_config*(5) and to the *sshd2*(8) manual pages.

3.2 Using the Secure Shell Client (ssh2)

The basic Secure Shell client program is called `ssh2`.

`Ssh2` can be used either to initiate an interactive session, resembling `rlogin`, or to execute a command in a way similar to the `rsh` command.

The Secure Shell client connects to the server on port 22, which is a well-known port for Secure Shell.

3.2.1 Starting the Secure Shell Client

`Ssh2` has a very simple syntax:

```
ssh2 [options] hostname [command]
```

The most common usage is to establish an interactive session to a remote host. This can be done simply by typing `ssh hostname.domain`. A real-world example could be `ssh somehost.ssh.com`. As with `rsh` and `rlogin`, the user ID to be used can be specified with the `-l` option.

Note: As shown in the above example, in the normal case, you do not have to type `ssh2`. The installation process creates a symbolic link, `ssh`, that points to the actual `ssh2` executable. If you also have SSH1 installed, you will need to type `ssh1` to run the SSH1 client.

The `ssh2` command line options are documented in detail on the *ssh2*(1) manual page.

3.2.2 Configuration File and Command-Line Options

`Ssh2` reads configuration data from `/etc/ssh2/ssh2_config` and from

`$HOME/.ssh2/ssh2_config` (or the file specified with `-F` on the command line). The file contains keyword-value pairs, one per line. Lines starting with a number sign # as well as empty lines are interpreted as comments. For detailed information about the options available in the configuration file and on the command line, please refer to the *ssh2_config*(5) and to the *ssh2*(5) manual pages.

3.3 Using Secure Copy (scp2)

Scp2 is a program for copying files over the network securely. It uses `ssh2` for data transfer, and uses the same authentication and provides the same security as `ssh2`.

`Scp2` uses a special file transfer protocol for the data exchange between the client and server. This is not to be confused with `FTP`.

The basic syntax for `scp2` is like this:

```
scp user@source:/directory/file
                user@destination:/directory/file
```

Note: As shown in the above example, in the normal case, you do not have to type `scp2`. The installation process creates a symbolic link, `scp`, that points to the actual `scp2` executable. If you also have SSH1 installed, you will need to type `scp1` to run the SSH1 client.

`Scp2` can be used to copy files in either direction; that is, from the local system to the remote system or vice versa. Local paths can be specified without the `user@system:` prefix. Relative paths can also be used; they are interpreted relative to the user's home directory.

The `scp2` command line options are documented in detail on the *scp2*(1) manual page.

3.4 Using Secure File Transfer (sftp2)

Sftp2 is a FTP-like client that works in a similar fashion to `scp2`. Just like `scp2`, `sftp2` runs with normal user privileges and uses `ssh2` for transport. Even though it functions like `ftp`, `sftp2` does not use the FTP daemon or the FTP client for its connections. The `sftp2` client can be used to connect to any host that is running the Secure Shell server daemon (`sshd2`).

The basic syntax for `sftp2` is like this:

```
sftp [options] hostname
```

Note: As shown in the above example, in the normal case, you do not have to type `sftp2`. The installation process creates a symbolic link, `sftp`, that points to the actual `sftp2` executable. `sftp` was not included in SSH1.

Actual usage of `sftp2` is similar to the traditional `ftp` program.

The `sftp2` command line options are documented in detail on the *sftp2*(1) manual page.

3.5 Using Authentication Agent (ssh-agent2, ssh-add2)

Ssh-agent2 is a program to hold private keys for authentication. With *Ssh-add2*, you can add identities to the authentication agent. When you use the authentication agent, it will automatically be used for public key authentication. This way, you only have to type the passphrase of your private key once to the agent. Authentication data does not have to be stored on any other machine than the local machine, and authentication passphrases or private keys never go over the network.

Start `ssh-agent2` with the command

```
eval ‘ssh-agent2‘
```

or with the command

```
exec ssh-agent $SHELL
```

After that, you can add identities like this:

```
% ssh-add2 id_dsa_1024_a
Adding identity: id_dsa_1024_a
Need passphrase for id_dsa_1024_a (1024-bit dsa,
      user@localhost, Tue Aug 01 2000 19:41:42).
Enter passphrase:
```

When you connect to a remote host and use public key authentication, you will get straight in.

If you want the connection to the agent to be forwarded over `ssh` remote logins, you should have this line in your `/etc/ssh2/sshd2_config` file:

```
AllowAgentForwarding            yes
```

The `ssh-agent2` and `ssh-add2` command line options are documented in detail on the *ssh-agent2*(1) and *ssh-add2*(1) manual pages.

3.6 Using Chroot Manager (ssh-chrootmgr)

Ssh-chrootmgr is a helper application to be used in instances where you would like to restrict users to their own home directory when they use *ssh2* and *sftp2*. Note

that this works only for static builds, because they do not use any shared libraries. Also, this functionality is not available directly in the binaries, and does not work on Solaris.

1. First, compile the source.

   ```
   ./configure --enable-static
   make
   make install
   ```

2. Run `ssh-chrootmgr` with root privileges and specify the appropriate user names on the command line.

   ```
   ssh-chrootmgr user1 user2 user3
   ```

   If you want, you can run `ssh-chrootmgr` with the `-v` option to get more information, or with the `-q` option to suppress any output.

3. Edit the following line in the configuration file `/etc/ssh2/sshd2_config`:

   ```
   ChRootUsers     user1,user2,user3
   ```

   If all the users are in the same group, edit the following instead:

   ```
   ChRootGroups    group1,group2,group3
   ```

4. Edit the `/etc/passwd` file so that the user's shell is `/bin/ssh-dummy-shell`.

5. Try to connect with `sftp`, for example as `user1`, and verify that the environment is chrooted.

The `ssh-chrootmgr` command line options are documented in detail on the *ssh-chrootmgr*(1) manual page.

If you want to establish a chrooted environment manually without using `ssh-chrootmgr`, do the following after compiling static binaries:

1. Create a `bin` directory under the user's home directory
2. Copy `ssh-dummy-shell.static` and `sftp-server2.static` from the `/usr/local/bin` directory to the `$HOME/bin` directory
3. Create the following symbolic links:

   ```
   ln -s sftp-server2.static sftp-server
   ln -s ssh-dummy-shell.static ssh-dummy-shell
   ```

4. As root, edit the `/etc/passwd` file so that the user's shell is `/bin/ssh-dummy-shell`.

3.7 Using Public Key Manager (ssh-pubkeymgr)

`Ssh-pubkeymgr` creates the user files needed to use public key authentication with `ssh2.` After all the required files have been created, it provides an interface that can upload your user public key to a remote host using `scp2.`

In the following usage example, it is assumed that user Et has not yet generated any keys. Et is currently logged on host Earth and wants to use public key authentication between the hosts Earth and Home. The user name is Et in both hosts, Earth and Home.

1. `Ssh-pubkeymgr` is started by giving the command

   ```
   ssh-pubkeymgr
   ```

2. `Ssh-pubkeymgr` runs `ssh-keygen2` and prompts Et for a passphrase:

```
Checking for existing user public keys..
Couldn't find your DSA keypair.. I'll generate
    you a new set..
Running ssh-keygen2... don't forget to give it
    a passphrase!
Generating 1024-bit dsa key pair
   4 .oOo.oOo.ooO
Key generated.
1024-bit dsa, Et@Earth, Fri Aug 18 2000
    15:48:38 +0300
Passphrase :
Again      :
Private key saved to /home/Et/.ssh2/id_dsa_1024_a
Public key saved to
                /home/Et/.ssh2/id_dsa_1024_a.pub
Creating your identity file..
Creating your authorization file..
...
```

3. Next, `ssh-pubkeymgr` asks if any hosts need to be added to the authorization file. In order to use public key authentication when connecting from Home to Earth, the answer must be yes.

```
Do you want to add any hosts to your
     authorization file? (Default: yes)
```

4. After this, the system asks for the required information:

```
Type in their hostname, press return after
each one. Press return on a blank line to
finish.

Add which user?
```

```
Et
Add which host?
Home
You added Et at Home as a trusted login.
Press return to continue or Ctrl-D to exit.
```

5. Next, the user public key can be uploaded to a remote host:

```
Do you want to upload Et@Earth key to a
     remote host? (Default: yes)
Upload to which host?
Home
Which user account?
Et
Where is the  Et's home directory?
(e.g. /home/anne, /u/ahc, etc.)
/home/Et
Now running scp2 to connect to ssh2-test3..
Most likely you'll have to type a password :)
Et@Home's password:
Et-Earth.pub | 738B | 0.7 kB/s | TOC: 00:00:01
                                | 100%

Press return to upload to more hosts or Ctrl-D
to exit.
```

Everything is now set up on host Earth. Now, user Et has to connect to host Home with `ssh2` and run the command `ssh-pubkeymgr` on host Home. After this, user Et can use public key authentication.

If you are not prompted for a passphrase after setting up the public key authentication, check that you have all the keys listed in the authorization file in your `$HOME/.ssh2` directory.

Chapter 4

Troubleshooting

This chapter contains information on some common Secure Shell problems and the suggested actions.

Some more information related to Secure Shell troubleshooting can be found in the text files `README.SSH2` and `FAQ.SSH2` in the source distribution package.

4.1 Secure Shell Daemon Problems

1. When running `sshd2` from `inetd`, it fails with a *Packet too long* error.

   Make sure you run `sshd2` as `sshd2 -i` when starting it via `inetd` (without any debug parameters). Also, don't compile in tcp_wrapper support in this case (`--with-libwrap=no`). In almost every case where this error is encountered, it is because `sshd2's` stdin and stdout (given by `inetd`) are the stream that `sshd2` handles, and if any debug messages etc. are put to that stream, the protocol gets messed up.

2. SSH terminal connections work, but `sftp` and `scp` connections fail.

 © 2000 SSH Communications Security

Make sure that `sftp` is along the `PATH`. Depending on what `SHELL` you are using, the startup script for non-interactive logins is different. You need to define the `PATH` variable in this file, because the shell invoking `sftp-server` is not interactive. For example, the correct place for environment variable settings in case of `zsh` is `.zshenv`.

If you don't want to edit the shell startup script, you can also use `sftp server`'s absolute path in the `/etc/ssh2/sshd2_config` file:

```
subsystem-sftp      /usr/local/bin/sftp-server
```

(The `/usr/local/bin/` is the default directory.)

Remember to reset the `sshd2` process (send it a `HUP` signal) after editing the configuration file.

4.2 Authentication Problems

1. When connecting to a host, where I know I have an account, `ssh2` says *Disconnected; authentication error (No further authentication methods available.)* (for ssh-2.0.13 server), or doesn't let me in, even when I type the correct password (for newer servers).

   The server is probably trying to check that the hostname has a valid DNS record. This is not the case with most hosts connected by dial-up lines etc. In older versions of `ssh2`, the default `/etc/sshd2_config` file contained the statement `RequireReverseMapping yes`, when, in fact, it should default to *no*. Ask your system administrator to change this, and see if the situation improves. If you still have problems, consult your system administrator about the situation.

Index

 © 2000 SSH Communications Security