NAME

rshd − remote shell server

SYNOPSIS

shell stream tcp nowait root /usr/sbin/in.rshd in.rshd
tcpd shell /usr/sbin/in.rshd

DESCRIPTION

Rshd is the server for the **rcmd**(3) routine and, consequently, for the **rsh**(1) program. The server provides remote execution facilities with authentication based on privileged port numbers from trusted hosts.

Rshd listens for service requests at the port indicated in the ‘‘cmd’’ service specification; see **services**(5). When a service request is received the following protocol is initiated:

1) The server checks the client’s source port. If the port is not in the range 0-1023, the server aborts the connection.

2) The server reads characters from the socket up to a null (‘\0’) byte. The resultant string is interpreted as an ASCII number, base 10.

3) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client’s machine. The source port of this second connection is also in the range 0-1023.

4) The server checks the client’s source address and requests the corresponding host name (see **gethostbyaddr**(3N), **hosts**(5) and **named**(8)). If the hostname cannot be determined, the dot-notation representation of the host address is used.

5) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as the user identity on the **client**’s machine.

6) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as a user identity to use on the **server**’s machine.

7) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system’s argument list.

8) **Rshd** then validates the user according to the following steps. The local (server-end) user name is looked up in the password file and a **chdir** is performed to the user’s home directory. If either the lookup or **chdir** fail, the connection is terminated. If the user is not the super-user, (user id 0), the file **/etc/hosts.equiv** is consulted for a list of hosts considered ‘‘equivalent’’. If the client’s host name is present in this file, the authentication is considered successful. If the lookup fails, or the user is the super-user, then the file **.rhosts** in the home directory of the remote user is checked for the machine name and identity of the user on the client’s machine. If this lookup fails, the connection is terminated.

9) A null byte is returned on the initial socket and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rshd*.

DIAGNOSTICS

Except for the last one listed below, all diagnostic messages are returned on the initial socket, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 9 above upon successful completion of all the steps prior to the execution of the login shell).

‘‘locuser too long’’
The name of the user on the client’s machine is longer than 16 characters.

‘‘remuser too long’’
The name of the user on the remote machine is longer than 16 characters.

‘‘command too long ’’
The command line passed exceeds the size of the argument list (as configured into the system).

‘‘Login incorrect.’’
No password file entry for the user name existed.

‘‘No remote directory.’’
The **chdir** command to the home directory failed.

‘‘Permission denied.’’
The authentication procedure described above failed.

‘‘Can’t make pipe.’’
The pipe needed for the **stderr**, wasn’t created.

‘‘Try again.’’
A **fork** by the server failed.

‘‘<shellname>: ...’’
The user’s login shell could not be started. This message is returned on the connection associated with the **stderr**, and is not preceded by a flag byte.

SEE ALSO

rsh(1), **rcmd**(3).

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an ‘‘open’’ environment.

A facility to allow all data exchanges to be encrypted should be present.

A more extensible protocol should be used.