OpenSSH from the Administrator's Perspective

# Out of Sight

OpenSSH has become the standard tool for providing encrypted remote access. But you will need background knowledge if you intend to implement the security features that OpenSSH offers. BY ANDREW JONES

You still occasionally come across them – computer networks where protocols such as rsh, rlogin, telnet, FTP and POP3 will transmit passwords in clear text across the LAN and into the Web. It is a well-known fact that any host in the path between the client and the server can view these passwords, and packet sniffers such as Sniffit or Dsniff can make it child's play to do so.

Networks that rely on these traditional services are low-hanging fruit for crackers or script kiddies, and the services we just mentioned are the top targets for exploitation. Having said that, there is no real reason to expose a network to this risk. SSH provides a viable alternative with far more functionality than rlogin, rcp, or telnet could offer.

In addition to providing secure authentification for hosts and users, SSH offers encrypted data transfers and recognizes attempts at manipulation. The term SSH refers both to the cryptographic protocol and to its implementations. In contrast to IPSec, where encryption occurs at IP level, SSH provides encryption within the application itself.

## Installing SSH

Most current Linux distributions include OpenSSH packages. If you are a Debian user, you can type "dpkg -L ssh" to discover which SSH files are currently installed. The equivalent command for any RPM based systems would be "rpm -ql openssh".

The post-installation script creates the server keypairs (private and public keys) for both versions of the protocol and stores them in the "ssh_host_key" and "ssh_host_key.pub" files for SSH1. SSH2 can utilize RSA and DSA keys, storing RSA keys in "ssh_host_rsa_key" and "ssh_host_rsa_key.pub" and DSA keys in "ssh_host_dsa_key" and the matching "ssh_host_ dsa_key.pub".

Only the root user should possess read/write access to the server's private keys (without the ".pub" suffix). The server uses a total of six keyfiles to authenticate against various clients. If a key file is missing, a client that requires a specific key type will not be able to connect. But missing server keys can be created later, if needed, using the "ssh-keygen" command.

The configuration file found at "/etc/ssh/ssh_config" contains system defaults for the client program "ssh"; although it tends to be more or less empty under normal circumstances. Users can configure their clients via "~/.ssh/config" (on the command line).

## Configuring the Server

The SSH server, "sshd", normally runs as an independent daemon, however, it can be launched via inetd. The daemon method will provide better performance, as "sshd" needs to calculate a server key for SSH1 when launched. However, the inetd variant does provide a practical fallback solution. If the daemon crashes, an admin user will still be able to log on remotely and solve the issue. Of course, the second server will need to listen on a different port ("sshd -i -p *port*"). You can also use the config file to change the port number (Listing 1, line 8).

For multi-homed hosts the admin user can specify the address on which "sshd" will listen. The daemon will bind to any and all available addresses by default. You can change the default behaviour using the "ListenAddress *IP*" syntax (line 11); multiple occurrences of this option are permissible.

"sshd" normally uses syslog to store log output in "/var/log/auth.log" at the "LogLevel INFO" priority level. Additional log output is useful for troubleshooting: Use "VERBOSE" to

THE AUTHOR

*Andrew Jones is a contractor to the Linux Information Systems AG http://www.linux-ag.com in Berlin. He has been using Open Source Software for many years.*

*Andrew spends most of his scarce leisure resources looking into Linux and related Open Source projects.*

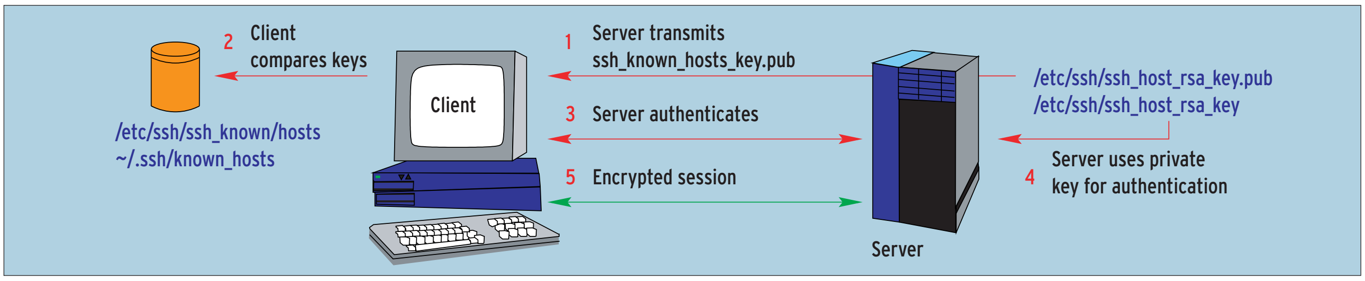

2 Client compares keys
/etc/ssh/ssh_known/hosts
~/.ssh/known_hosts
Client
1 Server transmits ssh_known_hosts_key.pub
3 Server authenticates
5 Encrypted session
Server
/etc/ssh/ssh_host_rsa_key.pub
/etc/ssh/ssh_host_rsa_key
4 Server uses private key for authentication

**Figure 1: The server transmits its public key to the client (1), the client compares it to the expected key (2) and authenticates the server (3, 4) using its private key. Any ensuing data will be encrypted before transmission (5); this includes any passwords that may be used**

choose the level immediately above "INFO" or failing that "DEBUG" to locate the cause for failed connections server side (lines 31 and 32).

When configuring the server you can specify which protocol version is announced during handshake. SSH2 has been default since OpenSSH 2.9, followed by SSH 1: "Protocol 2,1" (line 9). To permit SSH2 only, you will need to change this line to "Protocol 2".

## Authentification Procedure

Both protocol versions provide SSH with various methods of authenticating users. The most sophisticated method is the public key approach, which you enable using "PubkeyAuthentication yes" (line 35). In this case users will store their public keys in their home directories and authenticate using the corresponding private key. Server defaults and the OpenSSH packages provided by many distributions also permit password based authentification schemes, which are enabled using "PasswordAuthentication yes" (line 50). Very few admins change this default which leads to many users working with "ssh" without ever thinking about SSH's best features.

If you require strict logon security, you can specify "PasswordAuthentication no". SSH encrypts any passwords that cross the wire, but their inherent weaknesses still apply (too short, too easy to guess, rarely changed, and so on). There have been some attempts to guess passwords indirectly by performing timing analysis on the encrypted data, thus removing the need to decrypt the data (SSH Keystroke Timing Attack). Password based login can only be disabled entirely by additionally stipulating "PAMAuthenticationViaKbdInt no" (compare to line 58).

Insecure rhost authentication is disabled by default for the server (lines 38 through 41). The trusted host authentication means that the "/etc/hosts.equiv" and "~/.rhosts" files include those hosts considered trustworthy enough to log on without authenticating – the server trusts the client computer and the authentication process that has occurred on the client.

Using rhosts, the server only checks the IP address and port number of the client. If the port number is below 1024, the client process on Unix hosts must be root equivalent. This is intended to prevent simple user programs from spoofing their identity to the server. However, the whole model works on the assumption that the IP address is genuine – an assumption that cannot be safely made in today's networks.

## Reciprocal Trust Relationships between Hosts

SSH provides a far superior variant on the host based authentication scheme. The client program is required to authenticate with the host key of the client computer. The client will need root privileges to access this key (the set UID bit needs to be set). In this case SSH also trusts the hosts listed in "~/.shosts" and "/etc/shosts.equiv"; however, this setting is also disabled by default (lines 42 through 45).

In most cases you will want to avoid users logging on as root. If multiple users work with the root account it is extremely difficult to determine which administrator has just logged on. The "PermitRootLogin no" (line 31) permits logging on by normal users only. If you happen to be a user with administrative privileges, you can use "su" to step up to this level temporarily.

OpenSSH handles not only authentification but also the other steps involved in logging on to the system (launching a

## SSH History

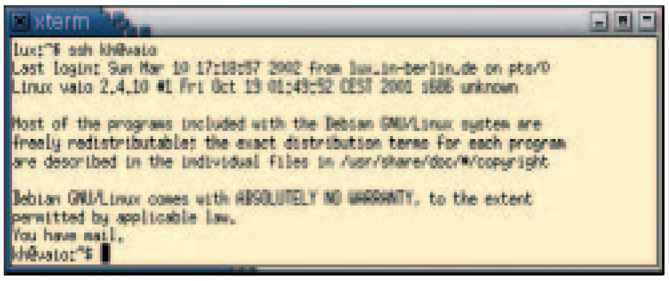
The designer of the protocol and the author of the first implementation was Tatu Ylönen, who went on to found SSH Communications Security Ltd. He released SSH 1.0 for Unix in June 1995. Ylönens software was freely available up to version 1.2.12, but licensing became increasingly restrictive. Two variants of version 1 of this protocol were developed and released as versions 1.3 and 1.5. SSH.com stopped maintaining and developing the commercial implementation of the protocol version 1 in May 2001.

**Offshoots – OpenSSH**

The OpenSSH SSH implementation was originally based on Ylönens SSH 1.2.12 sources, which were provided without any restrictions on their use, however, it uses the free SSL implementation, OpenSSL, as its cryptobase. The OpenBSD Group is responsible for the OpenSSH project, although a whole group of independent developers are now involved in it. The basic version of OpenSSH (as of this writing currently at 3.1), that runs on OpenBSD systems only, has been ported to a variety of platforms. A modified version, referred to as portable version 3.1p1, was implemented for this purpose.

As of version 2.1.0 (dated May 2000) OpenSSH can handle SSH version 2 in addition to version 1. In contrast to commercial SSH both versions are available in a single server binary. If an older client attempts to connect to the server, the server switches to compatability mode as defined in the IETF "SSH Transport Layer Protocol" draft. The server indicates its compatibility mode capability by means of a handshake string, "SSH-1.99". You can "telnet host 22" to view the string, or simply type "ssh -v host" ("Remote protocol version 1.99" is displayed in this case).

Version 2 of the protocol is documented in various IETF drafts; the architecture is described in "draft-ietf-secsh-architecture-09.txt". All current drafts are available from the IETF website [3].

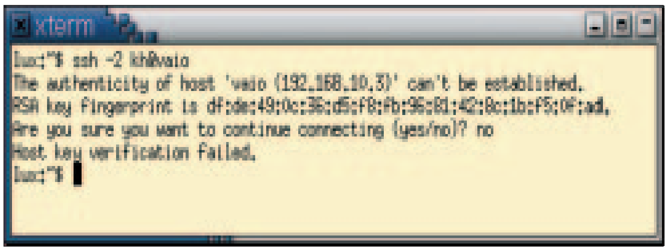


```
lux:~$ ssh kh@vaio
Last login: Sun Mar 10 17:18:57 2002 from lux.in-berlin.de on pts/0
Linux vaio 2.4.10 #1 Fri Oct 19 01:49:52 CEST 2001 i686 unknown

Most of the programs included with the Debian GNU/Linux system are
freely redistributable; the exact distribution terms for each program
are described in the individual files in /usr/share/doc/*/copyright

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have mail.
kh@vaio:~$
```

**Figure 2: The user "kh" logs on to the host "vaio", coming from "lux" via "ssh". The usual system messages are displayed**



```
lux:~$ ssh -2 kh@vaio
The authenticity of host 'vaio (192.168.10.3)' can't be established.
RSA key fingerprint is df:de:49:0c:36:d5:f8:fb:96:81:42:8c:1b:f5:0f:ad.
Are you sure you want to continue connecting (yes/no)? no
Host key verification failed.
lux:~$
```

**Figure 3: When you log on to an unknown host SSH prompts you to confirm that you want to trust the server key. "kh" fails to confirm which leads to the connection being terminated**

session including any required entries in logfiles, launching shells and so on). If you enable the "UseLogin yes" option, OpenSSH will access the system logon program to do so. This makes sense in some environments – where "login" applies restrictions of which "sshd" is not aware. However, this option also has a few security issues. CERT has reported security leakholes on two occasions [6].

## Working with Keypairs

Public key authentication has several advantages in comparison with a simple password logon scheme – although it does mean some additional setup tasks for the user. Administrators may also need to explain one or two facts to their users, however, you would expect most users to be able to adjust. Three steps are required before logging on remotely with an RSA or DSA key:

- The user must generate a keypair (public and private keys).
- The public key must be copied to the "~/.ssh/authorized_keys" file on the remote host.
- The private key must be available on the local host.

The "ssh-keygen" is used to create keypairs. If you launch this tool without any arguments, it will create an SSH1 compatible RSA keypair and store it in "~/.ssh/". The files created by this syntax are called "identity" (private key) and "identity.pub" (public key). As already mentioned, these files apply to SSH 1 only, but OpenSSH also uses the version 2 protocol, which is more secure.

DSA keys were introduced into SSH 2. To create a DSA keypair use the "ssh-keygen -t dsa" command to create the "~/.ssh/id_dsa" (private key, version 2) and "~/.ssh/id_dsa.pub" (public key, version 2) files. For RSA keys for SSH 2, type "ssh-keygen -t rsa". In this case the file names will include the "rsa" string.

Any files containing private keys should only be readable/writeable to their users. OpenSSH checks these privileges during key evaluation and refuses the connection if the privileges are too loose.

## Encrypting Keys

Access privileges for the key files provide effective protection against inquisitive users on the system, however, this is no defence against a user with root access. If the home directory is on an NFS server, the key will even be transferred across the wire in plain text from the NFS server to the local workstation. And this would defeat any advantages gained from key based authentication – but there is a solution to this issue.

Keys can be passphrase protected on creation. "ssh-keygen" uses the passphrase to encrypt the private key assuring that it is protected from any snoopers – not even root can decrypt this data without the passphrase. The passphrase can be modified later using "ssh-keygen -p -f ~/.ssh/id_dsa".

The public key must reside in "~/.ssh/authorized_keys" on the target host. This allows the server to verify that the public key really does belong to the user attempting to log on. In the case of OpenSSH servers prior to version 3.0 a file called "~/.ssh/authorized_keys2" with the same content is also required for SSH-2 keys.

As the name implies the public does not need to be kept a secret. So the admin of the target system can safely store the key in the home directory, after ascertaining that the key really is the genuine article. Just sending an email with the key as an attachment is fairly risky, however, you could use GPG to sign the mail. Alternatively the admin user could compare the fingerprint "ssh-keygen -l -f *keyfile*" of the received key with the original fingerprint. A short phonecall would be sufficient to deal with this issue.

## Behind the Barricades

It should now be possible to log on to the target system:

```
ssh -v user@hostcomputer
```

The "-v" option makes OpenSSH output debugging information, which can be

### SSH and Security

SSH has also (and unfortunately) had its share of (in)security incidents. Version 1 of the protocol is susceptible to man in the middle attacks, as it relies on cryptographically weak CRC 32 encoding to ensure packet integrity. Specially crafted data packets that include the correct CRC enable an attacker to inject data into an encrypted session without SSH noticing it.

The "SSH CRC32 attack detection" facility was designed to detect injection attacks. However, this code was found to contain a buffer overflow that allowed remote attackers to gain root on several older versions. CERT reports that these older versions are still being systematically sought out and exploited on a large scale [4].

Version 2 completely removed the CRC vulnerability. The new protocol relies on a cryptographically robust MAC algorithm (Message Authentication Code) – to be more precise it uses an RFC 2104 HMAC (Keyed Hashing for Message Authentication).

OpenSSH documents security and bugfixes on the project website [5]. A recent incident showed how important it is to install the latest stable releases of programs relevant to system security: OpenSSH 3.0.2 is susceptible to an off by one error, which can allow an authenticated user to achieve root privileges (refer to "InSecurity News" in this issue).

extremely useful if any issues occur. The SSH program running locally then prompts the user for the passphrase for the private key belonging to the target account. Assuming that the correct passphrase is entered, SSH will then authenticate the user on the target host, and the user will be placed in the Shell environment he expects (see Figure 2). The client can also specify the protocol type (SSH 2 or SSH 1) using the "-2" and "-1" flags:

```
kh@lux:~$ ssh -2 kh@vaio
```

The login does not need to be the same locally as on the remote host. Admins will generally prefer to work as a normal user locally, but need to be root on a remote host. No problem for SSH:

```
kh@lux:~$ ssh root@vaio
```

You can also store your own public key in "~/.ssh/authorized_keys", in the home directory for root on the target system. In this case you will need to set "PermitRootLogin yes" in "sshd_config" (line 31 in Listing 1).

Keypairs are not only available for users, but also for hosts (see Figure 1). This allows the client to verify that it is really connected to the required server. To do so, during connection setup the remote SSH daemon transmits its public key to the client and authenticates using its own private key.

## The "known_hosts" Security Database

The client stores the host key in the text file "~/.ssh/known_hosts". The SSH 2 drafts specify that SSH clients must request confirmation by the user, in case of unknown servers, that the user really does want to connect to the target. If the user cancels, the connection is terminated (see Figure 3).

Users should avoid typing "yes" without considering their options at this point – after all this prompt is one of

### Listing 1: Server Configuration "sshd_config"

```
01 #       $OpenBSD:
   sshd_config,v 1.42 2001/09/20
   20:57:51 mouring Exp $
02
03 # This sshd was compiled with
   PATH=/usr/bin:/bin:/usr/sbin:
   /sbin
04
05 # This is the sshd server
   system-wide configuration
   file. See sshd(8)
06 # for more information.
07
08 Port 22
09 #Protocol 2,1
10 #ListenAddress 0.0.0.0
11 #ListenAddress ::
12
13 # HostKey for protocol
   version 1
14 HostKey /etc/ssh_host_key
15 # HostKeys for protocol
   version 2
16 HostKey /etc/ssh_host_rsa_key
17 HostKey /etc/ssh_host_dsa_key
18
19 # Lifetime and size of
   ephemeral version 1 server
   key
20 KeyRegenerationInterval 3600
21 ServerKeyBits 768
22
23 # Logging
24 SyslogFacility AUTH
25 LogLevel INFO
26 #obsoletes QuietMode and
   FascistLogging
27
28 # Authentication:
29
30 LoginGraceTime 600
31 PermitRootLogin yes
32 StrictModes yes
33
34 RSAAuthentication yes
35 PubkeyAuthentication yes
36 #AuthorizedKeysFile
   %h/.ssh/authorized_keys
37
38 # rhosts authentication should
   not be used
39 RhostsAuthentication no
40 # Don't read the user's
   ~/.rhosts and ~/.shosts files
41 IgnoreRhosts yes
42 # For this to work you will
   also need host keys in
   /etc/ssh_known_hosts
43 RhostsRSAAuthentication no
44 # similar for prot. version 2
45 HostbasedAuthentication no
46 # Uncomment if you don't trust
   ~/.ssh/known_hosts for
   RhostsRSAAuthentication
47 #IgnoreUserKnownHosts yes
48
49 # To disable tunneled clear
   text passwords, change to no
   here!
50 PasswordAuthentication yes
51 PermitEmptyPasswords no
52
53 # Uncomment to disable s/key
   passwords
54
   #ChallengeResponseAuthenticat
   ion no
55
56 # Uncomment to enable PAM
   keyboard-interactive
   authentication
57 # Warning: enabling this may
   bypass the setting of
   'PasswordAuthentication'
58 #PAMAuthenticationViaKbdInt
   yes
59
60 # To change Kerberos options
61 #KerberosAuthentication no
62 #KerberosOrLocalPasswd yes
63 #AFSTokenPassing no
64 #KerberosTicketCleanup no
65
66 # Kerberos TGT Passing does
   only work with the AFS
   kaserver
67 #KerberosTgtPassing yes
68
69 X11Forwarding no
70 X11DisplayOffset 10
71 PrintMotd yes
72 #PrintLastLog no
73 KeepAlive yes
74 #UseLogin no
75
76 #MaxStartups 10:30:60
77 #Banner /etc/issue.net
78 #ReverseMappingCheck yes
79
80 Subsystem  sftp
   /usr/libexec/sftp-server
```

major security features of SSH. The fingerprint can easily be used to verify the key by calling the admin on the target host. The admin can display the fingerprint of the original host key by typing "ssh-keygen -l -f *keyfile*". Users should only place keys in their "known_hosts" files, if these fingerprints do match (Figure 4).

Verifying the server host key provides protection against "man in the middle" attacks, where the attacker will manipulate DNS or ARP, or spoof the IP address of the genuine server so to impersonate that server.

At the same time, the attacker connects to the real server and relays the data without the user noticing any difference. Cryptography can protect your users against attacks of this kind, but only if they play the game. If the client has no data on the server, it cannot authenticate the server.

If the SSH client already knows the genuine server's public key – that is, the key is stored in "known_hosts", the client can automatically detect an attack. The attacker will not know the original secret key and thus be unable to successfully use the public key of the required target, instead attackers would be forced to transmit their own keys. On comparing the key with the entry in its key ring the client would notice a discrepancy, warn the user (Figure 5) and cancel the connection.

However, the warning can be harmless – if the server key has really changed. This occurs when an admin user generates a new key after a disk crash where a backup is not available, after a hardware replacement or simply reinstalling SSH without saving the old key. The warning will be issued to the client until the user removes the old server key from "known_hosts". The next time the user connects she will again be prompted to confirm the identity of the server (Figure 4).

## Key Management

An admin may prefer not to bother her users with this procedure and maintain a global "/etc/ssh/ssh_known_hosts" file. If a host key changes, the admin user can modify the entry in the list of public keys.

Key authentication schemes like the one described may make connections safer, but you still need to input a passphrase instead of a password. The "ssh-agent" and "ssh-add" programs make short work of this onerous task. The SSH agent is a kind of cache agent that provides access to decrypted private keys.

It runs as a daemon and is available only to the user that launched it. The daemon communicates over Unix Domain Sockets; using the environment variables "SSH_AUTH_SOCK" and "SSH_AGENT_PID" to let its child processes know which socket it will use.

The X11 init scripts in many distributions launch the SSH agent as the parent process of the X11 session making it available to every X11 terminal. You can verify this by typing the following command:

```
kh@lux:~$ set | grep SSH
SSH_AGENT_PID=2097
SSH_AUTH_SOCK=/tmp/⮯
ssh-XX70h6xH/agent.2062
```

If the SSH agent is not running, and is only required in a single Shell, you can use the following syntax:

```
kh@lux:~$ ssh-agent $SHELL
```

The SSH agent launches the "$SHELL" subshell with the required environment variables and then retires to the background. Now you can use "ssh-add" to save any number of decrypted private keys in the cache:

```
kh@lux:~$ ssh-add ~/.ssh/id_dsa
Enter passphrase for ⮯
/home/kh/.ssh/id_dsa:
Identity added: ⮯
/home/kh/.ssh/id_dsa ⮯
(/home/kh/.ssh/id_dsa)
kh@lux:~$
```

The keys are now available to the SSH client without having to repeat the passphrase. The following syntax displays the keys currently being cached by the SSH agent:

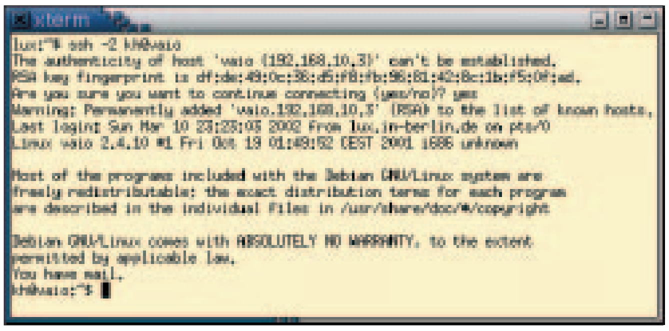
```
kh@lux:~$ ssh-add -l
1024 87:db:4c:0a:6a:c5:56:6b:⮯
74:6f:1c:8e:65:0a:ce:b2 ⮯
/home/kh/.ssh/id_dsa (DSA)
kh@lux:~$
```

Typing "ssh-add -D" deletes the whole key cache. To remove individual keys, you can simply type the command "sh-add -d ~/.ssh/id_dsa".

## Agent Forwarding

If you log on to various hosts in succession, you might like to look into the "ForwardAgent" option. This means you can avoid storing a public/private key combination at each step of the way and launching additional agent processes. A single SSH agent on a trusted host suffices; any hosts you connect to via this host refer back to the "ssh-agent" at the top of the tree.

There are three ways of enabling agent forwarding: globally using the

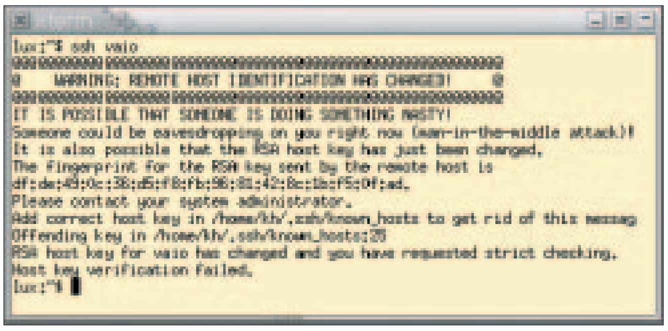


```
lux:~$ ssh -2 kh@vaio
The authenticity of host 'vaio (192.168.10.3)' can't be established.
RSA key fingerprint is df:de:49:0c:38:d5:f8:fb:98:81:42:8c:1b:f5:0f:ad.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'vaio.192.168.10.3' (RSA) to the list of known hosts.
Last login: Sun Mar 10 23:28:05 2002 from lux.in-berlin.de on pts/0
Linux vaio 2.4.10 #1 Fri Oct 19 01:49:52 CEST 2001 i686 unknown

Most of the programs included with the Debian GNU/Linux system are
freely redistributable; the exact distribution terms for each program
are described in the individual files in /usr/share/doc/*/copyright

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have mail.
kh@vaio:~$
```

**Figure 4: If the user is sure that the host key really does belong to the desired target host, she can place the key in the list of trusted hosts**



```
lux:~$ ssh vaio
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
df:de:49:0c:38:d5:f8:fb:98:81:42:8c:1b:f5:0f:ad.
Please contact your system administrator.
Add correct host key in /home/kh/.ssh/known_hosts to get rid of this messag
Offending key in /home/kh/.ssh/known_hosts:28
RSA host key for vaio has changed and you have requested strict checking.
Host key verification failed.
lux:~$
```

**Figure 5: If the SSH client determines that the public key of a server has changed, it assumes a man in the middle attack and issues a warning**

## Checking Your Configuration

A thorough review of the default configuration provided by your distribution is particularly relevant for security programs. If you determine any discrepancies between your configuration and Listing 1, you should ensure that they were made deliberately. In case of doubt, check "man sshd". You need to restart the daemon to activate any modifications made to "sshd_conf". You can use standard tools to check whether the daemon is running:

```
ps ax | grep sshd
```

Or alternatively:

```
netstat -tpan
```

By the way, when you restart the master "sshd" the forked SSH connections are maintained – no need to worry about users being logged out. To find out which process is the master process, simply view "/var/run/sshd.pid". Of course, if the daemon fails to restart, you will no longer be able to log on remotely via SSH.

There is an unexpected trap for computers secured independently of SSH using TCP-Wrappers ("/etc/hosts.allow" and "/etc/hosts.deny"). "sshd" evaluates your TCP-Wrappers settings, even if the daemon was launched independently of inetd. "sshd" makes direct use of "libwrap.a".

On the practical side, you can supply most of the options set in "sshd.conf" as arguments on launching the program. This allows you to test the effect of various options. Provided you launch the test daemon on a different port than usual, you can even perform tests without interfering with your production server. You simply set your client to access the new port: "ssh -p port".

"ForwardAgent yes" entry in the "/etc/ssh/ssh_config", for individual users in "~/.ssh/config" or via the "-A" option of the "ssh" command.

However, this approach does have some negative implications. Root may be able to perform a core dump to view the decrypted keys – however, if you do not trust the root account you may prefer not to save any secrets on that machine. Even if you decide against using the SSH agent, root could use a trojanized SSH client or a TTY sniffer to access the secret data when a user is entering her secret passphrase.

## Secure File Transfer with "scp" and "sftp"

SSH can be used for more than just remote logins. One example of SSH's flexibility is its ability to copy files across an encrypted connection. The "scp" and "sftp" programs, which are part of the OpenSSH suite, are provided for this purpose. "scp" uses the same syntax as the less secure "rcp". Copying a local file to a target host, type "scp *localfile user@host.remote:targetfile*". To copy in the other direction – that is, to copy a remote file to a local host, simply type "scp *user@host.remote:remotefile local⮐ target*". Just like the interactive "ssh" tool, the command allows you to specify a variety of options, such as the protocol versions, the verbosity level, user names and levels of compression.

The "sftp" program fulfills the same task as "scp", although its usage is similar to a command-line "ftp" client. You will need to enable the server subsystem "sftp-server" in "sshd_config" (last line in Listing 1). Most users should feel at home using "sftp" interactively:

```
lux:/tmp$ sftp root@vaio
Connecting to vaio...
sftp> pwd
Remote working directory: /root
sftp>
```

Gftp[8] even provides a friendly GUI for FTP and SFTP.

## Backup via SSH

One feature that will appeal to administrators is the ability to perform backups across the wire via SSH. "scp" or "sftp" are not required for this task as you can pipe SSH using the shell:

```
tar czvf - /the/directory | ⮐
ssh user@host "cat ⮐
>/tmp/foo.tar.gz"
```

The receiving end can also write the data directly to a tape drive:

```
tar cvf - /the/directory | ⮐
ssh user@host dd of=/dev/tape
```

You should be aware that your tape drive's performance may be seriously affected. This is caused by "dd" and the tape drive both having to wait for data. If the datastream is interrupted, the tape drive has to stop and backtrack before it can carry on writing. A small tool can help solve this issue:

```
tar cvf - /the/directory | ⮐
buffer | ssh -c blowfish ⮐
root@vaio buffer -o /dev/tape
```

Buffer[9] spawns two separate processes that independently read data from the network and write to the tape drive, providing caching for enhanced performance.

In our example we also set the OpenSSH option "-c blowfish" to enable the extremely quick but secure Blowfish encryption algorithm. Thus, OpenSSH can deal with requirements for security and speed, which are often viewed as contradictory. ■

## INFO

[1] OpenSSH project website: *http://www.openssh.com*

[2] SSH newsgroup: *news:comp.security.ssh*

[3] Current drafts on SecSH: *http://www.ietf.org/ids.by.wg/secsh.html*

[4] CERT Incident Note on SSH exploits: *http://www.cert.org/incident_notes/⮐ IN-2001-12.html*

[5] Security history of OpenSSH: *http://www.openssh.com/security.html*

[6] "UseLogin" vulnerabilities: *http://www.kb.cert.org/vuls/id/157447*, *http://www.kb.cert.org/vuls/id/40327*

[7] Daniel J. Barrett and Richard E. Silverman, "SSH: The Secure Shell", O'Reilly 2001, *http://www.snailbook.com/*

[8] Gftp, GUI for SFTP: *http://gftp.seul.org/*

[9] Buffer: *http://packages.debian.org/⮐ testing/utils/buffer.html*

[10] SSH FAQ: *http://www.employees.org/⮐ ~satch/ssh/faq/ssh-faq.html*

[11] Beginner-friendly series on OpenSSH: *http://www.mandrakeuser.org/docs/⮐ secure/sssh.html*