

OpenSSH Lab

OpenSSH is a free, open source implementation of the SSH (Secure *SH*ell) protocols. It replaces telnet, ftp, rlogin, rsh, and rcp with secure, encrypted network connectivity tools. OpenSSH supports versions 1.3, 1.5, and 2 of the SSH protocol. Since OpenSSH version 2.9, the default protocol is version 2, which uses RSA keys as the default.

OpenSSH server configuration:

1. Install the openssh-server and openssh rpm package included in the Red Hat Linux 7.3. Please note that the openssh-server package depends on the openssh package. OpenSSH packages also require the OpenSSL package (openssl) which installs several important cryptographic libraries that help OpenSSH provide encrypted communications

2. OpenSSH daemon, sshd, uses the configuration file /etc/ssh/sshd_config

The default config file is sufficient. For customization, refer to sshd man page for config options.

3. Edit/view the sshd_config file

```
# vi /etc/ssh/sshd_config
```

#Port 22	- Specifies the port number that sshd listens on
#Protocol 2,1	- Specifies the protocol versions sshd supports
#ListenAddress 0.0.0.0	- Specifies the local addresses sshd should listen on

#HostKey /etc/ssh/ssh_host_rsa_key	- Specifies a file containing a private host key used by SSH
------------------------------------	--

#SyslogFacility AUTH	- Gives the facility code that is used when sshd logs messages
#LogLevel INFO	- Gives the verbosity level that is used when sshd logs messages

#LoginGraceTime 600	- time limit for a user to log in
#PermitRootLogin yes	- Specifies whether root can login using ssh
#StrictModes yes	- Specifies whether sshd should check file modes and ownership of the user's files and home directory before accepting login
#PubkeyAuthentication yes	- Specifies whether public key authentication is allowed
#PasswordAuthentication yes	- Specifies whether password authentication is allowed
#PermitEmptyPasswords no	- When password authentication is allowed, it specifies whether the server allows login to accounts with empty password strings
#MaxStartups 10	- Specifies the maximum number of concurrent unauthenticated connections to the sshd daemon
#KeepAlive yes	- Specifies whether the system should send TCP keepalive messages to the other side
#VerifyReverseMapping no	- Specifies whether sshd should try to verify the remote host name
Subsystem sftp	/usr/libexec/openssh/sftp-server

Managing the sshd service

To start the service: `# /sbin/service sshd start`
To stop the service: `# /sbin/service sshd stop`
To restart the service: `# /sbin/service sshd restart`

To automatically run the service: `# chkconfig sshd on`

OpenSSH Client configuration:

To connect to an OpenSSH server from a client machine, you must have the openssh-clients and openssh packages installed on the client machine.

System wide ssh client configuration: defaults for all system wide users

`/etc/ssh_config`

The configuration values can be changed in per-user configuration files or on the command line:

`~/.ssh`

Using the ssh command with password authentication:

1. Make sure you have the following enabled in `/etc/sshd_config` file on the ssh server

`PasswordAuthentication yes` - Specifies whether password authentication is allowed

2. To log in to a host named server.com, type the following:

`$ ssh ritesh@server.com`

The first time you ssh to a remote machine, you will see a message similar to the following:

The authenticity of host 'server.com (202.52.255.22)' can't be established.
RSA key fingerprint is 0f:41:6c:52:31:59:43:0f:dd:49:5f:3d:47:9d:b5:9e.
Are you sure you want to continue connecting (yes/no)? yes

Type **yes** to continue.

This will add the server to your list of known hosts as seen in the following message:

Warning: Permanently added 'server.com,202.52.255.22' (RSA) to the list of known hosts.
ritesh@server.com's password:
Last login: Thu Jan 16 19:34:13 2003 from mc-gw.mos.com.np
[ritesh@server.com ritesh]\$

3. known_hosts file in .ssh folder contains the remote servers' host keys:

```
$ more known_hosts
server.com,202.52.255.22 ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAIEA5lwKqHiqSeXjrX3dCpS1gXZo9GqJ0hkk+clf+WmABYbEH
6lGMy2CeARtaR6QLpqB1SaGEFsn84dqA6kWLYfn4FuDc8KTyABLVEMOm6NnLZkHPKr3
Cb0RgivDYSYHlwgWuDi7XBvmoC44WA2EbM7eBy5h1kHrXZ5yPXq3rxl0=
```

Using the ssh command with public key authentication:

1. Make sure you have the following enabled in /etc/sshd_config file on the ssh server.

PubkeyAuthentication yes - specifies whether public key authentication is allowed

2. Key pair generation:

ssh-keygen - authentication key generation, management and conversion

To generate a RSA key pair to work with version 2 of the protocol:

```
$ ssh-keygen -t rsa
```

Generating public/private rsa key pair.

Enter file in which to save the key (/home/ritesh/.ssh/id_rsa):

Enter a passphrase different from your account password and confirm it by entering it again

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /home/ritesh/.ssh/id_rsa.

Your public key has been saved in /home/ritesh/.ssh/id_rsa.pub.

The key fingerprint is:

e5:a1:ac:ce:8d:16:3a:b9:3a:e4:e9:06:9c:90:b6:da ritesh@myhost.com

The public key is written to ~/.ssh/id_rsa.pub.

The private key is written to ~/.ssh/id_rsa

Never distribute your private key to anyone.

Change the permissions of your .ssh directory using the command `chmod 755 ~/.ssh`

```
$ ll .ssh/
total 8
-rw----- 1 ritesh ritesh 951 Jan 16 19:37 id_rsa
-rw-r--r-- 1 ritesh ritesh 236 Jan 16 19:37 id_rsa.pub
```

Copy the contents of ~/.ssh/id_rsa.pub to ~/.ssh/authorized_keys on the machine to which you want to connect. If the file ~/.ssh/authorized_keys does not exist, you can copy the file ~/.ssh/id_rsa.pub to the file ~/.ssh/authorized_keys on the remote SSH server.

Securely copy your public key file to the remote ssh server:

```
$ scp ~/.ssh/id_rsa.pub ritesh@server.com:
```

Login to the remote ssh server using password authentication, one more time:

```
$ ssh ritesh@server.com
```

Copy the content of the public key file to authorized_keys file on the remote host:

```
$ cat id_rsa.pub >> ~/.ssh/authorized_keys
```

Now, login to the remote ssh server via public key authentication:

```
$ ssh ritesh@server.com
```

Enter passphrase for key '/home/ritesh/.ssh/id_rsa':

Last login: Thu Jan 16 19:40:11 2003 from myhost.com

```
[ritesh@server.com ritesh]$
```

To run a command on the remote host with ssh and exit:

```
$ ssh penguin.example.net ls /usr/share/doc
```

Using the scp command

The scp command is used to transfer files between machines over a secure, encrypted connection

To transfer the local file shadowman to /home/username/shadowman on penguin.example.net:

```
$ scp shadowman username@penguin.example.net:/home/username
```

To transfer a remote file to the local system:

```
scp username@tohostname:/remotefile /newlocalfile
```

To transfer the contents of the directory /downloads to an existing directory called uploads on the remote machine penguin.example.net:

```
scp /downloads/* username@penguin.example.net:/uploads/
```

Using the sftp command

The sftp command is used to open a secure, interactive FTP session via an encrypted channel.

```
sftp username@hostname.com
```

Once authenticated, you can use a set of commands similar to using FTP

Port forwarding

With SSH you can secure otherwise insecure TCP/IP protocols via port forwarding. When using this technique, the SSH server becomes an encrypted conduit to the SSH client.

Port forwarding works by mapping a local port on the client to a remote port on the server. SSH allows you to map any port from the server to any port on the client; the port numbers do not need to match for it to work.

To create a TCP/IP port forwarding channel which listens for connections on the localhost, use the following command:

```
ssh -L local-port:remote-hostname:remote-port user@remote-hostname
```

Note: Setting up port forwarding to listen on ports below 1024 requires root access.

So if you want to check your email on a server called mail.domain.com using POP through an encrypted SSH connection to the POP server, you can use the following command:

```
ssh -L 1100:mail.domain.com:110 mail.domain.com
```

Once the port forwarding channel is in place between the two machines, you can direct your POP mail client to use port 1100 on localhost to check for new mail. Any requests sent to port 1100 on your system will be directed securely to the mail.domain.com server.

If mail.domain.com is not running an SSH server daemon, but you can log in via SSH to a machine on the same network, you can still use SSH to secure the part of the POP connection. However, a slightly different command is needed:

```
ssh -L 1100:mail.domain.com:110 ssh-server.domain.com
```

In this example, you are forwarding your POP request from port 1100 on your machine through the SSH connection on port 22 to ssh-server.domain.com. Then, ssh-server.domain.com connects to port 110 on mail.domain.com to allow you to check for new mail. Using this technique, only the connection between your system and ssh-server.domain.com is secure. Connection between ssh-server.domain.com and mail.domain.com is insecure.

Port forwarding can also be used to get information securely through network firewalls. If the firewall is configured to allow SSH traffic via its standard port (22) but block access through other ports, a connection between two hosts using the blocked ports is still possible by redirecting their communication over an established SSH connection to the firewall.

Note: Using port forwarding to forward connections in this manner allows any user on the client system to connect to the service to which you are forwarding connections. If the client system becomes compromised, the attacker will also have access to forwarded services.

X11 Forwarding

Opening an X11 session over an established SSH connection is as easy as running an X program on the local machine. When an X program is run from the secure shell prompt, the SSH client and server create a new secure channel, and the X program data is sent over that channel to your client machine transparently.

To enable X11 forwarding:

1. Make sure that the SSH software was compiled with X forwarding support. However, if X security extensions are wanted, it is necessary to compile from source. When compiling, make sure not to run `./configure` with any X disabling options.
2. Ensure that `xauth` is in the path of the user running `./configure`. Also, make sure that you have the following line in your `/etc/ssh/sshd_config` file:

```
X11Forwarding      yes
```

3. X11 forwarding also needs to be enabled in the client by setting the following line in the `ssh_config` file:

```
ForwardX11         yes
```

4. 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.

OpenSSH commands:

- `ssh` - The basic rlogin/rsh-like client program.
- `sshd` - The daemon that permits you to login.
- `ssh-agent` - An authentication agent that can store private keys.
- `ssh-add` - Tool which adds keys to in the above agent.
- `sftp` - FTP-like program that works over SSH1 and SSH2 protocol.
- `scp` - File copy program that acts like `rcp(1)`.
- `ssh-keygen` - Key generation tool.
- `sftp-server` - SFTP server subsystem (started automatically by `sshd`).
- `ssh-keyscan` - Utility for gathering public host keys from a number of hosts.
- `ssh-keysign` - Helper program for hostbased authentication.