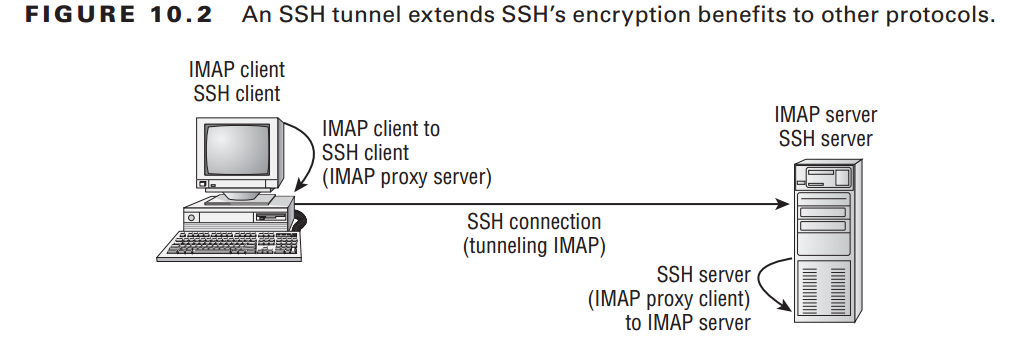
**Setting Up SSH Port Tunnels**  
SSH has the ability to extend its encryption capabilities to other protocols, but doing  
so requires extra configuration. The way this is done is known as *tunneling*.

Chapter 6 described a special type of SSH tunneling involving X, but the process can work for  
other protocols.  
Figure 10.2 illustrates the basic idea behind an SSH tunnel. The server computer runs  
two server programs: a server for the tunneled protocol (Figure 10.2 uses the Internet Mail  
Access Protocol, IMAP, as an example) and an SSH server. The client computer also runs  
two clients: one for the tunneled protocol and one for SSH. The SSH client also listens for  
connections for the tunneled protocol; it’s effectively both a client and a server. When the  
SSH client receives a connection from the tunneled protocol’s client, the result is that the  
tunneled protocol’s connection is encrypted using SSH, tunneled to the SSH server, and  
then directed to the target server. Thus, data passes over the network in encrypted form,  
even if the target protocol doesn’t support encryption!



Of course, all of this requires special configuration. The default configuration on the  
server enables tunneling, but to be sure, check the /etc/ssh/sshd\_config file on the server  
for the following option:  
AllowTcpForwarding no  
If this line is present, change no to yes. If it’s not present or if it’s already set to yes, you  
shouldn’t need to change your SSH server configuration.  
On the client side, you must establish a special SSH connection to the server computer.  
You do this with the normal ssh client program, but you must pass it several parameters.  
An example will help illustrate this use of ssh:

$ **sudo ssh -N -f -L 142:mail.luna.edu:143 benf@mail.luna.edu**  
$  
-N: Do not execute a remote command. This is useful for just forwarding ports (protocol version 2 only).

-f Requests ssh to go to background just before command execution. This is useful if ssh is going to ask for passwords or passphrases, but the user wants it in the background. This implies -n. The recommended way to start X11 programs at a remote site is with something like ssh -f host xterm.

If the ExitOnForwardFailure configuration option is set to “yes”, then a client started with -f will wait for all remote port forwards to be success fully established before placing itself in the background.

-L: -L [bind\_address:]port:host:hostport # Specifies that the given port on the local (client) host is to be forwarded to the given host and port on the remote side. This works by allocating a socket to listen to port on the local side, optionally bound to the specified bind\_address. Whenever a connection is made to this port, the connection is forwarded over the secure channel, and a connection is made to host port hostport from the remote machine. Port forwardings can also be specified in the configuration file. IPv6 addresses can be specified by enclosing the address in square brackets. Only the superuser can forward privileged ports. By default, the local port is bound in accordance with the GatewayPorts setting. However, an explicit bind\_address may be used to bind the connection to a specific address. The bind\_address of “localhost” indicates that the listening port be bound for local use only, while an empty address or ‘\*’ indicates that the port should be available from all interfaces.

The -N and -f options tell ssh not to execute a remote command and to execute in the  
background *after* asking for a password, respectively. These options are necessary to create  
a tunnel.  
The -L option specifies the local port on which to listen, the remote computer to which  
to connect, and the port on the remote computer to which to connect. This example listens  
on the local port 142 and connects to port 143 on mail.luna.edu.  
The final parameter (benf@mail.luna.edu in this example) is the remote username and  
computer to which the tunnel goes. Note that this computer need not be the same as the  
target system specified via -L.

If you want SSH on the client system to listen to a privileged port (that is,  
one numbered below 1024), you must execute the ssh program as root,  
as shown in the preceding example. If listening to a nonprivileged port is  
acceptable, the ssh client can be run as a normal user.

With the tunnel established, you can use the client program to connect to the local port  
specified by the first number in the -L parameter (port 142 in the preceding example).  
For instance, this example is intended to forward IMAP traffic, so you’d configure a mail  
reader on the client to retrieve IMAP email from port 142 on localhost. When the email  
reader does this, SSH kicks in and forwards traffic to the SSH server, which then passes the  
data on to the SSH server computer’s local port 143, which is presumably running the real  
IMAP server.  
All of this is hidden from the email reader program. As far as the reader program is  
concerned, it is retrieving email from a local IMAP server.