Background

FTP, short for File Transfer Protocol, is widely used for transferring files between machines. However it is well known as being insecure. There are two primary concerns when using FTP. First, with FTP both your password and your data travel over the network in clear text, meaning it is easily readable by those that have the resources to observe your network traffic. Second, a server that directly accepts FTP traffic opens itself up to attack. For example, after observing your network traffic and stealing your password, an intruder may be able to use FTP to read the private files on your server or install dangerous new files.

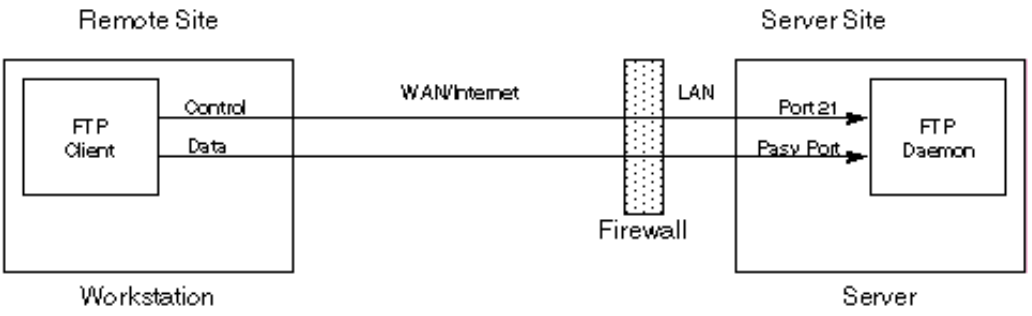
SSH, which stands for Secure SHell, is a program designed to provide secure access to a server. It encrypts authentication (passwords) and data before sending it over the open network. It is also designed to be very difficult to attack. It can be configured to provide secure data channels, called tunnels, that allow you to securely run insecure services such as FTP.

Most implementations of SSH also provide a secure version of FTP called SFTP. With these, you can use a SFTP client, such as WinSCP. However, some applications, such as web authoring software, provide built-in FTP capabilities that are incompatible with SFTP. It is in these situations where it becomes desirable to use SSH to secure FTP traffic.

Unlike most other services, such as those used to support e-mail (POP, IMAP, SMTP), FTP uses two distinct channels, which makes it more difficult to secure with SSH. However, in some situations it is possible to fully protect an FTP link. In this case you protect both the control channel, which carries passwords and control, and the data channel. To be able to use this approach, you must be able to reconfigure the FTP server and you must use passive-mode FTP transfers when using SSH. Other approaches exist, (as documented in the book by Barrett and Silverman), but they are less secure.

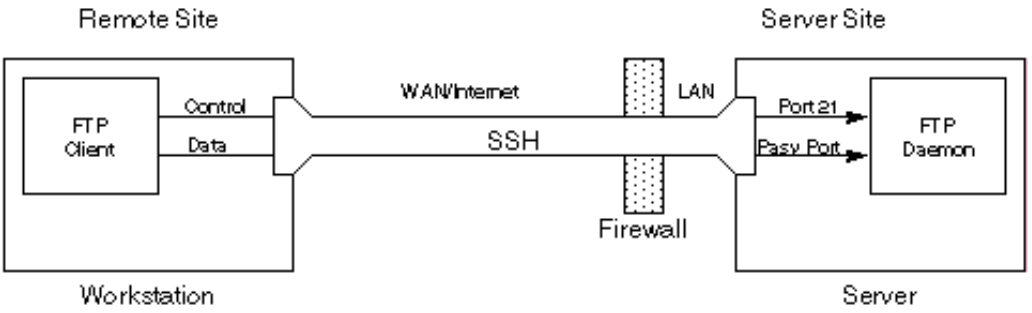
Approach

FTP provides two different transfer modes; active and passive (see slacksite.com for information on these two modes). It is not possible to protect an active FTP connection with SSH, but it is possible to protect passive connections by constraining the ports that FTP uses and forwarding all of these potential ports using SSH. Normally, passive mode works like this



In this case, the control channel is established first, and then through it the server passes to the client the port number to use for the data channel. However, this requires that the control and all possible data (PASV) ports be opened on the server's firewall, which increases the server's exposure to possible attack. And of course, neither channel is encrypted.

Instead, we first establish a secure link between the remote workstation and the server using SSH. Tunnels are passed through this link to carry the FTP traffic. Since the FTP data channel will connect to one port from a range of possible ports, SSH tunnels must be provided for the entire range of potential ports. Of course, this is only practical if there are a small number of ports in the allowed range, meaning that the approach itself is not practical if the server must be able to service a large number of simultaneous users.



The benefit of this approach is only the SSH port must be open on the firewall, and not the FTP ports. And of course, both the control and data channels are fully encrypted.

Example

The server must have both an FTP and SSH daemon present and running. This example assumes the use of VSFTP and OpenSSH. If you use something else you might need to modify these instructions. Assume that the server is internally named *popeye* but is accessible from the internet as *www.spinach.com*. The FTP server (/etc/vsftp.conf) is configured

- to support passive transfers (pasv_enable=yes),
- to constrain the ports used for the data channel to fall into a small range, such as 2534-2536 (pasv_min_port=2534, pasv_max_port=2536),
- and to direct the client to use 127.0.0.1 as the address that it uses when sending data (pasv_address=127.0.0.1).

Your workstation must have both an FTP client and an SSH client. There are many different FTP clients available, and any of them should work as long as they support passive transfer mode. This example assumes that the workstation is running Windows and that the SSH client is PuTTY.

You must configure your SSH client to forward all of the potential pasv data ports, 2534-2536, to your server (L2534 *popeye*:2534, ..., L2536 *popeye*:2536) where *popeye* is the local host name of your server. You must also configure SSH to forward port 21, the one associated with the control channel (L21 *popeye*:21). I'm told it is important that you use the true local host name (*popeye*) rather than *localhost* when establishing these connections (though I don't know why).

Now, from the remote workstation use your SSH client to create a link between the workstation and *www.spinach.com*. This will require that the firewall be configured to forward SSH traffic to *popeye*.

Finally, to initiate an FTP connection to your server once SSH is configured and running, direct your FTP

client to connect to localhost (127.0.0.1) using passive mode.

When you do so, the following things happen in sequence. First, your FTP client connects to *localhost*:21. In this case, localhost refers to your workstation, so FTP connects to port 21 on your workstation. SSH then forwards this connection to the server, where it connects to the FTP daemon to form the control channel. The FTP client requests a passive mode connection, so the server responds by telling the client to use 127.0.0.1:253*x*, where *x* is 4, 5, or 6. The address 127.0.0.1 is the address of the workstation, so FTP directs its data to port 253*x* on the workstation, which SSH forwards on to the server, and the connection is established.

To make this work, the FTP server hard codes the address used for passive transfers to 127.0.0.1, the localhost loopback address. As such, it prevents FTP connections to your server that might occur on the server's local network. However, these connections can be established using active mode instead.

Adapting the Example

In this example, the port numbers (2534-2536) are arbitrary. You can use whatever range of port numbers is available on your server. You should avoid using ports with numbers smaller that 1024.

In this example 3 pasv ports were reserved and so at most three users can be accomodated at a time. You are free to use as many ports as you like, but all will have to be configured on the ssh client, which is generally must be done manually for each port. This will be tedious if you reserve many ports.

The names *popeye* and *www.spinach.com* are made-up and should be adjusted to fit your situation. However, the name *localhost* and the loopback address 127.0.0.1 are generic and should not be modified.

Additional Resources

SSH is a very flexible and very powerful tool. It is designed to be a secure remote shell, but can also be used to provide secure remote access for mail (POP, IMAP, SMTP), file access, synchronization, and versioning (SFTP, RSync, CVS), windowing (X windows, VNC), etc. You can learn about it in depth with the following books. Clicking on the images on the right takes you to Amazon, where you can learn more about them. For more information on configuring PuTTY, visit the Nurdletech SSH page.

Any questions or comments on these notes can be directed to theNurds@nurdletech.com.