IBM Systems MAGAZINE

Issue Date: Open Systems edition
October | November 2006

Stronger Security With OpenSSH

Jaqui Lynch
jaqui.lynch@mainline.com

Offering provides strong authentication for closing security holes related to IP, routing and DNS spoofing

In today's on demand world, businesses are becoming more dependant on online system access. Since this kind of business requires 24-7 diligence, administrators must be able to securely obtain access to systems from remote or local locations. One of the ways to provide this kind of login access is by using OpenSSH.

What is SSH?

Secure shell (SSH) was written to provide more secure alternatives to insecure protocols such as rlogin, RCP, FTP and Telnet. It provides authentication, encryption and data integrity across the Internet. In previous articles, I've explained general security and the importance of correctly setting up logging and TCP wrappers, but for this article it's assumed that this has been done.

Users of Telnet, FTP, rlogin and other similar protocols transmit information such as passwords in clear text (i.e., unencrypted). This makes it possible for people to sniff those passwords out on the network. On the other hand, SSH encrypts all authentication traffic, helping ensure that user names and passwords don't travel in the clear. SSH provides several authentication options, including the use of UNIX* passwords or the use of a public/private key pair for authentication. Additionally, SSH interfaces with TCP wrappers for logging and access control and has its own built-in access control that's fairly granular. The suite includes several features, including the capability to do secure backups, tunneling and X11 forwarding, all within a secure environment. SSH also comes with a secure FTP server - SFTP - which should be used to replace Telnet, rlogin, RCP and FTP (see "Examples" sidebar).

The OpenSSH suite is a free version of SSH tools that includes several programs that can be used to help ensure secure network traffic: these include sshd, SSH, SCP, ssh-keygen, ssh-agent, SFTP and several others that I'll explore in further detail later in the article.

Why Should You Bother With SSH?

SSH provides strong authentication that closes security holes related to IP, routing and DNS spoofing. IP spoofing is where a remote host sends out packets that pretend to come from another, trusted host. SSH even protects against a spoofer on the local network. The spoofer pretends to be your router to the outside. DNS spoofing is where an attacker forges name-server records and redirects server requests.

Additionally, SSH provides improved privacy using encryption. This provides protection for passwords in particular and stops the interception of clear-text passwords and other data by others. This protects from the manipulation of that data by the people in control of those intercepting hosts. X11 sessions can also be secured and forwarded, thus avoiding attacks based on listening to X authentication data and a spoofed connection to the X11 server. SSH also provides for setting up tunnels to redirect other network ports in an encrypted, secure fashion.

Effectively, SSH never trusts the network, which means that, if somebody hostile has taken over the network, they can only force SSH to disconnect; they can't decrypt or play back the traffic or hijack the connection.

Other Things You Can do With SSH

Some of the other useful things you can do with SSH include:

- X11 forwarding that allows the encryption of network X windows traffic so that the data and command streams can't be modified in-flight.
- Port forwarding allows the forwarding of TCP/IP connections to a remote system over an encrypted channel. This can also be done using SSL tunnels, but there are many applications that don't support AAL encryption. These applications - such as POP or SNMP - can instead be tunneled through secure SSH channels. This can also be used to tunnel through the firewall rather than allowing other less secure ports to be opened.
- Backup using tar via an SSH tunnel.
- Add SSH to the rdist/rsync configs and tunnel them.
- Run PPP over an SSH tunnel.

- Support is also provided for a number of other tools and techniques including Socks support, AFS/Kerberos support and PGP key support.
- OpenSSH compresses data before encryption using zlib. This can improve overall performance.
- OpenSSH uses the OpenSSL cryptographic library.

SSH doesn't, however, protect you against yourself. If you set encryption to "none," you're not protected, just as you're not protected if you set up SSH without requiring passwords. SSH also doesn't work with UDP based protocols, such as NIS or NFS.

Programs and Tools

Some of the more valuable programs and tools associated with OpenSSH include:

- ssh - This is the client utility (basic rlogin or rsh-type client program) that's used to log into a remote system and to perform commands on that system. It replaces the rsh, RCP and rlogin utilities, which don't provide a secure connection. The user must prove his identity to the remote system using one of several authentication methods, depending on the protocol version that's used.

  The ssh command can also be integrated into the rdist and rsync configurations so they can use ssh tunneling. It also provides secure encrypted communications between two systems over the Internet or an untrusted network. X11 connections and arbitrary TCP/IP ports can also be forwarded over the secure channel.

  Additionally, ssh is used as a client to establish an SSH connection to the Hardware Management Console (HMC) with the IBM* POWER5* technology-based IBM System i* and System p* platforms. The SSH connection with the HMC provides administrators with a command-line interface.

- sshd - The ssh daemon (sshd) is the daemon (server) program for ssh that listens on port 22 for connections from clients and permits users to request a login. It's normally started at boot time and spawns a new daemon process for each incoming connection.
- ssh_config client configuration file - There's a system-wide configuration file for ssh client settings. These settings are used for every SSH, SFTP and SCP client request. The system-wide configuration file - with the name ssh_config - is stored in the /usr/local/etc directory unless you specify a different default.
- sshd_config daemon configuration file - Customizing the sshd configuration file is optional and only required when the default values don't meet your requirements. This file is used to customize the sshd dameon and how it reacts to requests. By default, it's found in /usr/local/etc/sshd_config and it has all configuration directives commented out. I normally modify this file to at least allow for the use of login banners and to add SFTP rather than FTP as follows:

  ```
  Banner /etc/motd
  # override default of no subsystems
  Subsystem sftp /usr/local/libexec/sftp-server
  ```

  You then need to stop and start the sshd daemon from a local terminal, not an SSH connection, although a kill -HUP may work.

- ssh-agent - ssh-agent is an authentication agent that's used to store private keys for use with public-key authentication. It allows a user to load a private key into memory to avoid retyping the pass phrase each time an SSH connection is started.
- ssh-add (tool that adds keys to the ssh-agent) - ssh-add tries to load private keys from private-key files in a user's ~/.ssh directory. The file names that ssh-add looks for are defined in the ssh_config file. This is the method for making those keys available to ssh-agent.
- SFTP - SFTP is a secure replacement for FTP. The generic SFTP client can only transfer data in binary format. However, there are some Windows*-based clients that can transfer the data in other formats. Sftp doesn't provide the enhanced functions that are available in the the default FTP utility, but it can be very useful.
- scp - scp is a secure file-copy program that can be used instead of RCP, ftp, SFTP and other network file-copy programs to securely copy data. It's the OpenSSH version of RCP.
- ssh-keygen - ssh-keygen is used to generate and manage both public and private keys. Since SSH allows users to authenticate using these public and private keys as an alternative to using their OS sign-on password, it's important to protect those keys. One should note that public keys can be freely distributed, while private keys should be protected.

How to Get OpenSSH

The current version (as of Aug. 3, 2006) of OpenSSH is 4.3p2 and can be downloaded from the OpenSSH Web site

(www.openssh.org/portable.html) - toward the bottom of the page is the list of download servers where you can access the source.

Alternatively, IBM currently ships (as of February 2006) the binaries on the AIX 5L* Expansion Pack and Web Download Pack. This is currently at 3.4p1. I personally prefer to download the latest source and compile it myself so that I know exactly which options have been enabled in the binaries.

Additionally, you will need to download and install zlib and openssl. I'm assuming here that logging and TCP wrappers are set up - while these aren't prerequisites, they should be part of the overall security plan.

How to Install OpenSSH

To start an OpenSSH installation, you'll need some additional pieces.

First, you need to get the bffs for gcc, freeware-bison and freeware-flex. You need to add two groups: sshd and freeware and two user IDs with the same names. You can obtain flex and bison at:

ftp://aixpdslib.seas.ucla.edu/pub/flex/RISC/5.3/exec/flex.2.5.4.tar.Z

ftp://aixpdslib.seas.ucla.edu/pub/bison/RISC/5.3/exec/bison.2.1.tar.Z

Other binaries are available for 5.2 and earlier versions at the same sites.

Compiling the code requires a C compiler, and gcc works very well if you don't have the IBM C compiler. The gcc binary can be obtained at aixpdslib.seas.ucla.edu/packages/gcc.html.

You will also need a working version of gzip and of zlib. gzip should already be on your system and the source for zlib on AIX can be obtained at aixpdslib.seas.ucla.edu/packages/zlib.html. (Note: Don't use an old version of zlib, as versions previous to v1.2.3 have security holes or other issues.)

For flex, bison, zlib and gcc, I normally just use the binaries from UCLA as it's very difficult to compile gcc without a compiler.

Once these are installed, you're ready to install openssl and then openssh. Assuming that the tarfiles have been unzipped into /usr/local/src/tarfiles, the following will install openssl and then openssh:

- OPENSSL
  1. cd /usr/local/src
  2. tar -xvf tarfiles/openssl-0.9.8a.tar
  3. cd openssl*
  4. ./Configure aix-gcc -prefix=/usr/local -openssldir=/usr/local/openssl
  5. make
  6. make test
  7. make install
- OPENSSH
  1. mkdir /var/empty
  2. chown root.sys /var/empty
  3. chmod 755 /var/empty
  4. cd /usr/local/src
  5. tar -xvf tarfiles/openssh-4.2p1.tar
  6. cd openssh*
  7. ./configure -with-tcp-wrappers - with-ssl-dirr=/usr/local/openssl
  8. make
  9. make install
  10. /usr/local/sbin/sshd
  11. Test ssh access
  12. Add /usr/local/sbin/sshd to /etc/rc.local or wherever you put things you want to start at boot time

Many other options can be used on the configure statements, depending on authentication protocols you use (LDAP, PAM, etc.) and what you're trying to accomplish. There's excellent documentation on the OpenSSH site (www.openssh.org) on all of these.

Using SSH

Using SSH is very straightforward. If you're on a Linux* or UNIX system and have installed ssh, then all of the commands (ssh, SFTP, scp, etc.) are available for you to use, and full man commands are installed into /usr/local/man.

You may have to update your MANPATH setting to access them but they're also available in the online documentation at openssh.org. If you're on a Windows* system, you'll need to download clients or purchase a Windows SSH suite. A popular graphical ssh client is the PuTTY client. You can download it freely from the Internet (www.chiark.greenend.org.uk/~sgtatham/putty/download.html).

Better Security

It takes very little time to install openSSH, and the benefits are immense. However, one of the biggest issues is the resistance to using open-source software. If a system isn't running TCP wrappers and SSH - and is instead using generic logging and Telnet - the system is exposed, and attempts to access it are most likely not being logged.

I highly recommend replacing insecure network services such as Telnet, FTP, RSH and rlogin with the more secure versions in the OpenSSH suite. You will have better security and better logging by doing so.

Examples

Tunneling Telnet and FTP

On ssh server.com

```
ssh -R 1234:localhost:23 -l jaqui ssh.client.com
```

This maps port 1234 (note >1024) on ssh.client.com to the servers port 23 (telnet) and starts an encrypted session

Now from client.com

```
telnet localhost 1234
```

You're now connecting via a secure tunnel back to the server.

```
ssh -L 1234:ftphost:21 ssh.host.com
```

Now from client - ftp localhost 1234

X11 Forwarding

`./configure -with-x` (as well as the other chosen parameters)

```
sshd_config
      X11Forwarding  yes
sshd2_config
      ForwardX11   yes

hosts.allow
      sshdfwd-x11: ip addrs: options
```

- J.L.

Jaqui Lynch, *an architect and systems engineer, has been responsible for a wide variety of projects and OSs across multiple vendor platforms, including mainframes, UNIX systems, midrange systems and personal workstations. Jaqui can be reached at jaqui.lynch@mainline.com.*