

Configuring OpenSSH for passwordless login on the Interix subsystem (SFU or SUA)

Tech Note #4 in the "Interop Components in Windows" series

By Rodney Ruddock (Interop Systems)

The biggest single reason people come to the SUA Community's ["Tool Warehouse"](#) is to obtain a secure shell solution that they can run on Windows. It is de facto on Unix and Linux systems today that a secure shell solution be available. Fortunately, OpenSSH is available for SFU and SUA.

Why Install OpenSSH?

It is highly recognized by system administrators and most users that establishing a secure connection between two computers for data transfer, monitoring and just doing work is critical. The number of 'bad' people and script kiddies trying to break into computer systems at work and at home has gone beyond being just annoying. Perhaps not as bad as the volume level of e-mail spam we are all deluged with every day, but certainly more serious.

In the good old days when someone tried to connect to your computer there was a good chance you actually knew the person. And if you didn't, you knew someone who did. One way or another everyone knew everyone else. But over the years the number of computers connected to (what is now called) the Internet continued to grow beyond the point of casual security still being effective security. Even tools that had a modicum of security such as the Berkeley 'R' commands (rlogin, rsh, rcp) and telnet began to be exposed for their security weaknesses. Plaintext passwords being sent over the Internet was fine when bad people were not using packet sniffers. Files listing the trust relationship between machines were secure enough back then until nasty folks started spoofing DNS records.

The solution came along eventually in the form of Secure Shell. Originally Secure Shell was started as an open source project and then morphed into a commercial enterprise. Several years ago the gang at OpenBSD revived the open source of Secure Shell to produce OpenSSH. OpenSSH has been maintained, updated and expanded over these years to where it is the most installed version of Secure Shell on Unix and Linux systems.

Secure Shell, including OpenSSH, allows for a person to connect between two machines with a secure link so that passwords and other data can be sent between the machines without the worry that some packet sniffer is going to easily discover your password. With the secure communications link, data can be transferred in either direction without it being read by someone who shouldn't be reading it.

At the same time OpenSSH provides the ease of use that the Berkeley R commands did. For example, you can log in to multiple machines in a chain and each machine accessed with the special key combination of "~^Z" (tilde-control-Z). The **ssh** command matches well to **rlogin** and **telnet** use, **scp** matches the old **rcp**, and **sftp** gives a secure version of **ftp**. This is a very nice toolset for users and system administrators.

Using 'SSH' is Simple

It is very easy for a user wanting to connect from one machine to another machine using Secure Shell. While there are several vendors of Secure Shell today (OpenSSH counting as a vendor even though it is open source) the level of interoperability among them is very high for operating systems, protocol versions and brands.

For users there are command line and GUI-based Secure Shell clients. Many go by the name "ssh" or have "ssh" in their name. A few go by different names, but all identify themselves in their brief explanation that they are an "ssh client."

A user simply starts **ssh** and indicates the Internet name or address of the machine to be connected with. Optionally a user name can be specified for the remote machine if it is to be different than the user name on the client machine. When a connection is successfully established a password is requested. For **scp** and **sftp** the method of connecting two machines is virtually identical.

Both the command line and the GUI-based clients allow the user to interact with the remote machine, where the secure shell server is, in a text-based manner.

Installing a Secure Shell Server

There are more Secure Shell clients available than Secure Shell servers. The servers are the programs that allow remote clients to connect to a machine. While the client may be on a Windows system the server can be on a Unix system (or visa-versa) without any problem. That a client was obtained for free (closed or open source) doesn't really matter to the server (closed or open source).

On Windows systems the easiest way to get a Secure Shell server is to install the pre-build binary package of OpenSSH for Interix, Microsoft's Unix-style subsystem. This requires that you have the Interix subsystem installed already. This is accomplished by installing Windows Services for Unix (SFU 3.5) for Windows 2000, Windows Server 2003 (pre-R2) and Windows XP; or installing the Subsystem for Unix-based Applications (SUA) which is a component of Windows Server 2003 R2, Vista Ultimate or Enterprise or Windows Server 2008 Beta.

With Interix installed, a visit to the /Tools Warehouse will get you started. First you need to install the bootstrap installer. This will allow for the correct installation of OpenSSH and any other software that it depends on (such as the OpenSSL libraries). Future updates and patches are smoothly and easily handled via the installer. The collection of installer tools is known as "**pkg**" with the key tool being **pkg_update**. You will need to install everything as the Administrator account or as a member of the Administrators group.

With the installer tools ready you can install OpenSSH with the command:

```
% pkg_update -L openssh
```

The **pkg_update** command does the work of getting and correctly placing the **openssh** package. Version numbers are found and used behind the scenes so you don't need to worry about them. The "-L" option indicates that the "openssh" package (all in lower-case letters) and any other packages needed will be retrieved from the /Tools warehouse automatically over the Internet.

After it has installed you may or may not need to reboot the system; it depends on the user account you use to do the installation. A message at the end of the installation will tell you.

The server (named **sshd**, the 'd' for daemon) will now be running and ready to accept connections. If you are running Microsoft's Windows Firewall the installation will have requested that the Secure Shell port be opened for connections.

Best Practices After Installation for Users

With the server installed there are some actions that the system administrator should do to provide the best environment for secure shell users. This is needed because correct user setup may not happen. In particular, a user's account on Windows is rarely assigned a home directory in the user database. [The user database can be Active Directory when the user is part of a domain. The user database may also be the local machine as managed

through the Computer Management panel. Information lookups can happen against both of these user databases.]

Assigning a home directory for each user is critically important for security. This allows for public keys and machine names for each user to be kept unshared with other users. There's an old Russian saying that once more than one person knows a secret it is a secret no more. So there can be no sharing of secure shell data between user accounts.

A common question is: "But isn't \$HOME already my home directory?" No. "HOME" is an environment variable that can be easily spoofed. For those who do not understand the word spoofed this means that the environment variable can very easily be made to indicate a different location on the disk. In turn this can mean that another user (who did the spoofing) can collect data about other users.

When the user's home directory is set in the user database, such as Active Directory, then it is a secure and reliable location. When set in the user database the HOME environment variable will match. The easiest way to check if a home directory has been correctly set is to use the finger command. The output will show if a home directory has been set in the user database. Here's an example:

```
% finger -l harry
```

The '-l' (dash small ell) asks for the long listing of information for the user "harry". The output will look something like this:

```
Login: harry
Name: harry
Directory: /dev/fs/C/users/harry
Shell: /bin/csh
Never logged in.
No Mail.
No Plan.
```

You can see the home directory is "/dev/fs/C/users/harry" above. If the directory is listed as just "/" (a forward slash) then a home directory has not been set in the user database.

With a secure home directory a special subdirectory named ".ssh" will hold special files that contain information for hosts you have authorized connections with or perhaps even information about allowing a login without a password being given by the user logging in.

To set a user's home directory from the command line you must be running as either Administrator or as a member of the Administrators group. The home directory may be set with either the Windows GUI tool or a command line. The fastest is with the Windows command line tool net, as run from a CMD.EXE:

```
net user harry /HOMEDIR:C:\users\harry
```

which will set the home directory for the local machine. To set the home directory for a user in the domain the "/DOMAIN" option must be added:

```
net user harry /HOMEDIR:\\SERVER\c_drive\users\harry /DOMAIN
```

Note that in this second example a UNC path has been given for the home directory. This is a good practice so that the user has the same home directory on the client machines of the domain. Also it means that the user does not need to mount a “letter drive” to access their home directory. This can avoid access conflicts while maintaining high access security.

The default login shell that all users are assigned to use is `/bin/sh`. This can be changed to be another shell if the user prefers such as `csh` or `bash`. While **sh**, **ksh**, **csh** and **tcsh** are installed with the base Interix installation you will have to install **bash**, **zsh** and others yourself from the [SUA Community](#) Tool Warehouse.

To change a user’s default login shell you will need have an account with enough permission. You will typically need to be the Administrator or a member of the Administrators group. Most users do not have enough authority to change the user database even for their own account. The change is most easily done with the Interix command **chsh**. For example,

```
% chsh -u harry /bin/tcsh
```

Remember that the only valid choices are those that are listed in the file `/etc/shells`.

Passwordless Login With Public Key Exchange

Logging into another machine without a password is extremely convenient. This is in part what made the Berkeley R commands so popular and useful. But no password is also very dangerous because someone could then easily pretend to be another user. If that other user is a system administrator and the person using that account is a bad guy, then the situation can be quite dire. This could lead to lost or changed data, company secrets exposed to competitors or your machine being used to attack other machines or even to host unwanted material that could get you in serious trouble with the law.

With Secure Shell a passwordless login can happen, but it is done with an exchange of secure keys taking the place of the password. Authorization happens, but without the need to verify the authorization with the user. The secure keys are called “public keys.”

The logistics of it are that a special key pair is generated on the secure server. One of the keys is kept with the server while the other key is assigned to the client. These keys are exchanged in a special way when the client and server communicate to determine that each is who they claim to be.

On Interix the setup is very close to the same as on other Unix systems. We’ll start with the basic setup, which is common, and then cover the extended case with more detail.

First the user must generate a key pair on the client machine. If you are using a GUI-based client you’ll have to poke around in the menus for “key generation.” With OpenSSH installed you can do the generation from a shell command line as follows:

```
% ssh-keygen -t rsa
```

The option and its argument indicate that the generated key pair is to be "rsa." The possible key types are "rsa1" (for protocol version 1), "rsa" (protocol version 2) and "dsa" (protocol version 2). The "rsa" type is the most widely used.

Running **ssh-keygen** will create a file named "id_rsa.pub" in the "~/.ssh" directory (where "~" means your home directory as listed in the user database). You want to copy this file to the secure shell server machine as the file "authorized_keys" in the "~/.ssh" directory. Appropriately, you can do this copying using **sftp** or **scp** to keep this data transfer secure.

For most Unix systems, that's it. The next time a connection to this server is made from the client that **ssh-keygen** was run on, no password will be asked for and the login will succeed.

There are always a couple of caveats. The major one is that the "~/.ssh" directory and the files within must have correct ownership and permissions that are not promiscuous. If ownership or permissions are too loose then the files are not secure, meaning they cannot be trusted and, therefore, OpenSSH won't use them. This applies on Interix as it does for all Unix systems.

A specific Interix caveat is "where is the home directory located?"

If the home directory is located on a disk local to the machine that **sshd** is running on, then no additional tweaking is needed for passwordless login with key exchange.

If the home directory is located on a disk remote to the machine that **sshd** is running on then an additional step is needed.

Why the additional step? For security the **sshd** process (the server) is running with a security token that by default only allows for access to local resources (including disks). This helps contain problems if one of the machines in a domain gets taken over at the "root level" by a nasty person. This means the generated key that you copied to the ~/.ssh directory cannot be accessed by the server which means that a key exchange, or passwordless, login cannot happen. A regular password based login still works of course.

So what is the additional step? The user will run the utility **regpwd** on the server to register their password. This allows access to the network disk that has the files in ~/.ssh so the keys can be used to verify a passwordless login. The password is stored securely on the machine. Access to this stored password is limited. If the user changes their password then **regpwd** must be run again.

You can determine that a home directory is on a network disk by the path. If the path begins as "/net/..." then it is network based. In a domain where a user can login on several machines it is preferred to have their home directory common for all machines in that domain. It provides common and consistent access to data. If you are setting a home directory for a user in the Win32-world you will specify it with a UNC path. This will be translated into the more Unix-style path of "/net/...".

More information on each of the commands and utilities written about above can be found within their respective manual pages, e.g. "**man ssh**". OpenSSH works nearly identical on Interix as on other systems.

Feel free to post a question in the [SUA Community](#) forums.