SSH CONFIGURATION HOWTO

Following a discussion of the importance of **ssh** and the public key authentication method, a process for configuring the capability to establish secure connections between your workstation and remote computers without using passwords is described.

Table of Contents

Introduction: What is SSH?

Until about ten years ago, there was no secure way to log into a remote computer and to do work on that machine while using the keyboard and monitor on your own workstation. The method that was normally used combined the Internet **telnet** service with a Unix application running on the user's machine, enabling the user to connect to a shell running on a remote machine and interact with it as if it were local. To establish a session with the remote machine, the user would enter her username and password.

Authentication by username and password was acceptable practice during the early days of inter-networking, when most connections were within college campuses or between academic laboratories. There was really very little security, however. There was no encryption, either of the username and password nor of message content. Anyone with the ability to tap a communications circuit could eavesdrop or, worse yet, steal a password.

An even less secure practice was the use of another Internet service called **rsh**, or the remote shell, which was very similar to **telnet**, but which could be set up to establish a connection to a shell running on a remote system *without using a password*. In this mode, the application depended completely on trust; there was no authentication and no encryption. A companion to **rsh** was **rcp**, the remote copy command, which copied files between systems with a similar lack of security.

Well before the Internet was opened up to the world in the 1990s, these practices were obsolete due to their lack of security. With the advent of the World Wide Web there was an explosion of the number of machines connected to the Internet. There was a need to administer machines remotely, but this required secure connections.

A number of projects to develop secure replacements for **rsh** and **rcp** were mounted. The effort that succeeded and was rapidly adopted was that of the Finnish developer Tatu Yloenen. He named his products **ssh** and **scp**, the secure shell and secure copy, respectively.

These products were never open source, but they were offered free of charge to the Linux and OpenBSD communities for a number of years. Because of the importance of this software, this arrangement was not satisfactory for the long term. The OpenSSH project was launched to create open source replacements for the proprietary ones. The OpenSSH versions of **ssh** and **scp** have been distributed with OpenBSD and Linux for several years now.

With **ssh**, you can log into a remote machine with a high degree of security. All of the communication between your workstation and the remote system is encrypted, including a password if it is needed. It is also possible to establish sessions with remote

machines without providing a password, in a way that is functionally identical to **rsh**, except that this can be accomplished with nearly complete security. This mode of operation and the process for setting it up will be described in the sections that follow..

The **ssh** command can also be used to set up an *encrypted tunnel* between your workstation and an application running on a remote machine. This feature is especially useful when running an X-Windows application on the remote host. The following services are provided:

- All communications are encrypted.
- The X-Windows messages, which can be voluminous, can optionally be compressed. This is especially useful over a slow connection.
- The user does not have to set up the DISPLAY variable (a requirement for X-Windows). This is done automatically.

Normally you do not have to do anything to benefit from encrypted tunneling of X-Windows sessions. You simply establish a connection with **ssh** and start up an X-Windows application on the remote host. That application will use the X-Server running in your workstation to display its output.

Public Key Authentication

When two parties wish to exchange secret messages using traditional encryption methods (also known as *symmetric* encryption), they must first exchange a secret key. Distributing keys reliably has always been a problem. With *public key* encryption, it is not necessary for the parties to share secret keys. Instead, each party has a pair of keys: a private key and a public key. Each party gives the other his public key. Each party keeps his private key secret, never revealing it to anyone else. If a third party discovers one of the public keys, it is not a problem. Public keys can be posted to the world with no loss of security.

In order to send a message, the author uses the recipient's public key to encrypt it. Once encrypted, the only key that can decrypt it is the recipient's private key, and only the recipient knows that key. This is the method used by **ssh** for public key authentication.

When you call upon the **ssh** on your machine to connect you to another machine, your **ssh** sends a connection request to the **ssh** on the remote machine. The remote **ssh** then generates a random number, encrypts it with your *public key* and sends it back. Your **ssh** then decrypts it using your *private key* to reveal the original number. Your **ssh** sends the number back to the remote **ssh**, proving that you are who you claim to be, since only you possess the necessary key.

Public Key Authentication and the Gnome Desktop

There is a very convenient way that users of the Gnome desktop can set up **ssh** so that they can connect to remote machines without having to supply a username and password but with a high level of security. In order for this to work, **ssh** depends on two related applications: **ssh-agent** and **ssh-add**.

When you log into the Gnome desktop, the **ssh-agent** authentication agent is started automatically. It continues to run for as long as you are logged into the desktop. The mission of **ssh-agent** is to use your private key to decrypt authentication messages sent to your **ssh** client by the remote **ssh** daemon. **ssh-agent** is available to provide this service to any **ssh** started from your desktop.

In order to send requests to **ssh-agent** and to receive responses, **ssh** must have a communications channel. When **ssh-agent** starts up, it creates a *unix domain socket* for this

purpose and writes the path-name of the socket to standard output. The parent process, which is the Gnome desktop, receives this path-name and uses it as the value of the `SSH_AUTH_SOCK` environment variable. The Gnome desktop is an ancestor to any process started from the desktop, including **ssh**. Processes inherit all of the environment variables of their parents, hence `SSH_AUTH_SOCK` is part of the environment of every **ssh** client that you run. By querying the value of this variable, **ssh** knows which socket to use to communicate with **ssh-agent**.

ssh-agent does not know your pass phrase, hence it cannot decrypt your private key. This is where **ssh-add** comes in. After starting **ssh-agent**, the Gnome desktop runs **ssh-add** to ask you for your pass phrase, which it then uses to decrypt your private key. It passes the decrypted private key via the Unix domain socket to **ssh-agent**, which caches it for future use. It is possible to run **ssh-add** several times, to add additional private keys to the cache.

Configuring Password-Free Remote Connection

This section describes the process for setting up the Gnome desktop for password-free connection to remote machines.

Create a Pair of Keys

Open up a shell window. Determine whether or not the `.ssh` directory exists in your home directory:

```
cd
ls -al
```

If there is no .ssh directory, create it as follows, and set the permission mode bits:

```
mkdir .ssh
chmod 755 .ssh
```

Next, you need to generate your two keys, using the **ssh-keygen** command:

```
ssh-keygen -t dsa
```

The *-t dsa* option specifies that you prefer to use the dsa encryption algorithm (rsa is the other possibility).

You will be asked to supply a *pass phrase.* This is very important! The pass phrase is used to encrypt your *private key* when it is stored on the disk of your computer. If it is not encrypted, anyone who can break into your computer can steal your private key and then enter without challenge any of the machines you have set up for password-free connection.

Be very careful to select an excellent pass phrase It is like the passwords that you use to log in to your workstation and other machines, but it is even more important because it will provide access to multiple machines. It should have the following properties:

- Be at least nine characters long.
- Be neither a word nor a given name in any language.
- Contain several digits or special characters

Note also that the pass phrase is case sensitive.

If you look in your `.ssh` directory after running **ssh-keygen**, you will find that two files have been added: `id_dsa`, which is your *encrypted* private key and `id_dsa.pub`, which is your public key.

Transfer Your Public Key to the Remote Machine

Next, you need to copy your public key to the remote host that you want access to. You can use **scp** to accomplish this. In the following example, we assume that the remote machine is named *whitechuck*:

```
scp ~/.ssh/id_dsa.pub whitechuck:
```

You will be asked to supply your username and password for the remote host, because public key authentication is not yet configured.

Log in to the remote host using your username and password. Then use **ls -al** to determine whether or not `.ssh` exists in your home directory on that machine. If not, create the directory and set the permission mode bits:

```
mkdir .ssh
chmod 755 .ssh
```

Concatenate your *public key* onto the end of the list of public keys in the file `authorized_keys` and insure that the mode bits are set properly (**ssh** is very fussy about mode bits):

```
cat id_dsa.pub >> .ssh/authorized_keys
chmod 644 .ssh/authorized_keys
```

Configure Gnome

The final step is performed on your workstation. You need to configure the Gnome desktop to call **ssh-add** when X-Windows starts up, in order to ask you for your pass phrase. Here is the menu sequence:

```
Main => Extras => Preferences => Sessions => left-click
```

This will open the Sessions window. Press the **Startup Programs** button followed by the **Add** button. Then set the following variables:

```
Startup Command: /usr/bin/ssh-add
Priority: 70
```

Close the window by pressing the **OK** button.

Verify the results

Log out of the Gnome desktop on your workstation, and then restart it. You should be asked for your pass phrase. If not, check to make sure that you did the Gnome configuration' correctly.

Open up a shell window on your workstation and connect to the remote machine with **ssh**:

```
ssh whitechuck
```

If everything has been correctly configured, the connection should go through without you having to supply a pass phrase or password.

If you are asked for a password, something is wrong. Here are some things that you can check:

- Are the permission mode bits on the `.ssh` directory in your home directory of your workstation and on the `.ssh` directory in your home directory on the remote machine set to 775?
- On the remote machine, are the permission modes on the file `~/.ssh/authorized_keys` set to 644?
- Does a copy of the file `~/.ssh/id_dsa.pub` in your workstation appear at the end of the file `~/.ssh/authorized_keys` on the remote machine?