OpenSSH Quick Reference

Author: Jialong He
Jialong_he@bigfoot.com
http://www.bigfoot.com/~jialong_he

What is OpenSSH and where to get it

OpenSSH is a protocol suite of network connectivity tools that replace **telnet**, **ftp, rsh, and rcp**. It encrypts all traffic (including passwords) to effectively eliminate eavesdropping, connection hijacking, and other network-level attacks. OpenSSH comes with most Linux distributions.

Use command "**ssh -V**" to check the SSH version installed. The latest version can be found from: **www.openssh.org**

Server Configuration

sshd is the OpenSSH server (daemon). It is controlled by a configuration file **sshd_config** which normally resides in **/etc/ssh** directory. You can specify command-line options to override their configuration file equivalents. Here are some useful options. For the complete list of keywords, see sshd_config (5) manual page.

Keyword	Description	Default
AllowGroups	Allow only specified groups to connect. May use '*' and '?'.	*
AllowUsers	Allow only specified users to connect. May use '*' and '?'.	*
DenyGroups	Groups NOT allowed connecting.	none
DenyUsers	Users NOT allowed connecting.	none
AllowTcpForwarding	TCP forwarding allowed.	yes
GatewayPorts	Allow other computers to connect to the forwarding port.	no
HostbasedAuthentication	Allow host based authentication (use .shosts or /etc/shosts.equiv)	no
IgnoreRhosts	Ignore per user .rhosts and .shosts in hostbased authentication.	yes
IgnoreUserKnownHosts	Ignore $HOME/.ssh/known_hosts, use only */etc/ssh/ssh_known_hosts*	no
PasswordAuthentication	Password authentication allowed	yes
PermitEmptyPasswords	Allow blank password	no
PublicKeyAuthentication	Public key authentication allowed	yes
AuthorizedKeysFile	Public key file name. Default: **$HOME/.ssh/authorized_keys**	see left
ListenAddress	IP address to accept connection	0.0.0.0
Port	Listening port	22
LogLevel	sshd verbosity level	info
PermitRootLogin	Allow root login	yes
PrintLastLog	Print last login date	yes
PrintMotd	Print /etc/motd file	yes
Protocol	SSH protocol	2, 1
StrictModes	check files ownership and perm.	yes
SyslogFacility	Syslog facility code	AUTH
TCPKeepAlive	Send TCP keepalive to client	yes
UseDNS	lookup client DNSname	yes
Compression	Compress network traffic	yes
X11Forwading	Permit X11 forwarding	no

Client Configuration

ssh (**sftp, scp**) are OpenSSH commands to replace **telnet, ftp, rcp**. The properties of these program are controlled by (1) **command line** options, (2) per user configuration file **$HOME/.ssh/config** and (3) system wide configuration file **/etc/ssh/ssh_config**.

Usage Example:
ssh user@remotehost # connect to remote host as user
scp myfile user@remotehost:/home/user # remote copy "myfile"

Here are useful keywords in **ssh_config**. For the complete list of keywords, see ssh_config (5) manual page.

Keyword	Description	Default
HostName	Default host to connect	none
User	Default user name	none
PreferredAuthentications	Preferred authentication methods hostbased, publickey, password	see left
HostbasedAuthentication	Try hostbased authentication	no
PubkeyAuthentication	Try Public key authentication	yes
PasswordAuthentication	Try password authentication	yes
LocalForward	Specify TCP port forwarding in LPORT RHOST:RPORT	none
RemoteForward	Remote forward port RPORT LHOST:LPORT	none
GatewayPorts	Allow hosts other than this host to connect to the forwarding port	no
ForwardX11	Forward X11 connection	no
Compression	Compress network traffic	no
CompressionLevel	If use compress, compress level	6
Port	Default remote port	22
Protocol	SSH protocol	2, 1
StrictHostKeyChecking	Allow connect to a host which is not in $HOME/.ssh/known_hosts or /etc/ssh/ssh_known_hosts	ask
LogLevel	Verbosity level	info
NumberOfPasswordPrompts	Allow the number of password tries	3
TCPKeepAlive	Send TCP keepalive to other end	yes
VerifyHostKeyDNS	Verify the remote key using DNS	no
CheckHostIP	Check the host IP address in the known_hosts file	yes

Public Key Authentication

Public key authentication is a preferred method. It is more secure than password authentication because no password travels through the network, but you have to do some setup before you can use public key authentication. Public key authentication is configured for *individual user*.

(1) Modify SSH server's configuration file (**sshd_config**) to enable public key authentication: (**PublicKeyAuthentication yes**). Also modify client's configuration file (**ssh_config**) to use public key authentication (**PubkeyAuthentication yes**). Normally, these are default settings.

(2) Generate a key pair for this user
ssh-keygen –t rsa –f $HOME/.ssh/id_rsa

It will prompt you a **passphrase** to encrypt private key. Two files "id_rsa" and "id_rsa.pub" will be generated.

(3) Transfer user's public key (**id_rsa.pub**) to SSH server and append its contents to:
$HOME/.ssh/authorized_keys or **$HOME/.ssh/authorized_keys2.**

You may also restrict from which computers allowed to use public key authentication. For example, in authorized_key file, you put "from" before the public key.
from="Goat.domain.com" AAAAB3NzaC1yc2EAAA ….

(4) Now you can log on to remote system with
ssh *my_sshserver*

It will prompt you *passphrase* to decrypt the private key. If you did not give a passphrase in the step 2, you will be connected with asking password.

(5) If you do give a passphrase to protect private key, but don't want to type this passphrase every time, it is possible to use *ssh agent* command:

eval `ssh-agent`
ssh-add ~/.ssh/id_isa

This will prompt you passphrase once. As long as the current terminal is open, you can connect to the SSH server without typing passphrase. Note, this is only valid for the *current terminal*, you still need to type passphrase in other terminal.

In order to run scripts without typing password, the easiest way is to use a blank passphrase in step 2. Unlike password, passphrase never travels through the network. It is used for protecting local private key.

Host-based Authentication

Hosted based authentication can be useful to run batch files or scripts on remote computers. It is very tricky to configure host based authentication. Even if you follow the instructions exactly, you might still get a password prompt. In this case, double check file permissions (.shosts) and computer names (must use FQDN). Restart computer (in order to have **sshd** read configuration file).

Server Side

(1) Modify **/etc/ssh/sshd_config** to enable host based authentication:

```
HostbasedAuthentication    yes
IgnoreRhosts               no
IgnoreUserKnownHosts       no     # optional
RhostsAuthentication       yes    # optional, not recommended
```

Let **SSH** daemon to re-read configuration file by either reboot the computer or send "**kill –HUP /var/run/sshd.pid**". On Redhat Linux, you can restart SSH daemon using: **service sshd restart**

(2) Copy client's public key to the SSH server. Client's public key usually stored in **/etc/ssh/ssh_host_rsa_key.pub** on client computer.

If client also has OpenSSH server running, you can fetch its public key by:
ssh-keyscan –t rsa *client_FQDN* **> /etc/ssh/ssh_known_hosts2**

If per user known hosts is enabled (**IgnoreUserKnownHosts no**), you connect to the client's SSH daemon from the server, the client's host key will be saved in: **$HOME/.ssh/known_hosts**

Note: You MUST use FQDN of client computer to get its public key. Following files are used to store client's public key on the server.

System wide:	Per user:
/etc/ssh/ssh_known_hosts	$HOME/.ssh/known_hosts
/etc/ssh/ssh_known_hosts2	$HOME/.ssh/known_hosts2

(3) Add client's FQDN in $HOME/.shosts. Please note the permissions for this file must be owned by the user and NOT writable by group/others.

If (**RhostsAuthentication yes**), you can also use **/etc/hosts.equiv**, but this is NOT recommended. Besides, it has NO effect for root login.

Client Side

(1) Enable host based authentication in SSH client configuration file:
/etc/ssh/ssh_config
HostbasedAuthentication yes

(2) You should have RSA host key pair (normally in /etc/ssh)
ssh_host_rsa_key
ssh_host_rsa_key.pub

If not, generate key pair with:
ssh-keygen –t rsa –f /etc/ssh/ssh_host_rsa_key –N ""

TCP Port Forwarding

OpenSSH can forward TCP traffic through SSH connection and secure TCP applications such as POP3, IMAP or HTTP by direct clear text TCP traffic through SSH (tunneling). Port forwarding can also redirect some TCP traffics through firewall.

In order to use port forwarding, you must first establish SSH connection and the connection must stay on as long as forwarding needed. In other words, you have to logon on to SSH server. There are two kinds of port forwarding: local and remote forwarding

Local Forwarding

In local forwarding, application servers (e.g., mail server) are on the same computer as the SSH server. For example, suppose we have a server named "*horse*" and it has web and SSH servers running. On another computer named "*goat*", using following command forwards traffic to an arbitrarily chose port (here 12345) on "goat" to port 80 on "horse",

ssh –g –L 12345:horse:80 horse

If you point a web browser to http://goat:12345, it will show the contents of http://horse. Here "-g" means that other hosts can access this forwarding port (here 12345). Similarly, you can forward other TCP traffic (e.g., POP3 110, IMAP 143) through SSH tunnel.

Remote Forwarding

If your application server is on the same machine as SSH client (i.e., you run SSH client on the application server), you should use remote forwarding. For example, we have a server named "*horse*" and client named "*goat*". On "*horse*", you run

ssh –R 12345:horst:80 goat

You can point your web browser to http://goat:12345, it will show the content as if you accessed http://horse. This time, you can only access port "12345" on "goat" (no Gateway port).