Configuring OpenSSH on z/OS Hands-on Lab

SHARE in San Francisco
Session: 12745 on February 5, 2013

C.T. Ware
IBM Poughkeepsie, NY
ctware@us.ibm.com

Trademarks and Disclaimers

See http://www.ibm.com/legal/copytrade.shtml for a list of IBM trademarks.
The following are trademarks or registered trademarks of other companies:

- UNIX is a registered trademark of The Open Group in the United States and other countries
- CERT® is a registered trademark and service mark of Carnegie Mellon University.
- ssh® is a registered trademark of SSH Communications Security Corp
- X Window System is a trademark of X Consortium, Inc

All other products may be trademarks or registered trademarks of their respective companies

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices are subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

Lab Overview

Objectives of this lab

At the end of this lab, you will be able to do the following:
- Configure and start the OpenSSH server (sshd).
- Log into an OpenSSH server (sshd) using the OpenSSH client (ssh) with public-key user authentication.
- Run both sshd and ssh in debug mode.
- Work with other sshd and ssh options.

This lab is written for IBM Ported Tools for z/OS: OpenSSH V1R2. As a result, some of the lab steps and exercises don't work or need to be modified for use with V1R1. The "**New in V1R2**" phrase will be used to identify such lab steps and exercises. Refer to the "IBM Ported Tools for z/OS: OpenSSH User's Guide" for more information on the new release (V1R2) and for differences between the new and old release (V1R1).

Note: The ssh –V option can be used to verify the OpenSSH release in use. V1R2 will output "`OpenSSH_5.0p1, OpenSSL 0.9.8k 25 Mar 2009`" and V1R1 (with recent PTFs applied) will output "`OpenSSH_3.8.1p1, OpenSSL 0.9.7d 17 Mar 2004`".

How is this lab different than the real world?

1. You do not have system administrator privileges. So...
    - When you start the OpenSSH server (sshd) it will be as a regular user on an unprivileged port.
    - Some of the system-wide configuration files will be kept in the home directory of your SHARE user ID.
2. The local and remote hosts are the same system.

The following example information box will help identify when this lab is different than the real world.

> **Information for the real world…**
>
> - This is the information for the real world…

Typing suggestions to get through this lab

If you prefer to copy/paste commands, a text file in your $HOME/sshlab directory contains the commands for this lab. You can view it using "more" or "cat". For example:

```
prompt=> cat $HOME/sshlab/labcommands
```

> **Note:** This file is personalized for your SHARE user ID. You may want to login twice (opening 2 PuTTY sessions) if using this file so you can copy the lab commands from one session while doing the lab exercises in another session. By default, the right mouse button can be used to copy highlighted text in a PuTTY session and the left mouse button can be used to paste that text.

Another alternative is to use the /bin/tcsh shell to take advantage of the up and down arrows for retrieving previous commands. However, this lab is designed for use with the /bin/sh shell. It is recommended that you don't use the /bin/tcsh shell unless you are familiar with the /bin/tcsh versus /bin/sh shell differences (e.g. syntax for command substitution, environment variable specification, etc.).

Options for editing configuration files

By default, this lab edits configuration files by echoing data into them. If you prefer to edit the configuration files manually, you can use vi or oedit. However, note that the OpenSSH client (ssh) cannot be run from a 3270 PCOMM session. So if you prefer to use oedit, then you can log in twice: once with PuTTY and once with PCOMM. You can then do your editing from the PCOMM session, and everything else from the PuTTY session. See the Appendix for instructions on logging in through TSO (3270 interface).

Other important lab information

This SHARE lab is self-paced and self-contained. However, you need to go in order; don't do the lab exercises out-of-order. Also be sure to **substitute your SHARE user ID** when the lab shows **shar___** or **SHAR___** from now on (e.g. sharb04 where 'b' is the lab letter and '04' is your lab number).

Your SHARE user ID went through some basic configuration in preparation for this lab. The configuration consisted of miscellaneous file (e.g. $HOME/sshlab/labcommands), directory (e.g. $HOME/sshlab and $HOME/.ssh) and authority setup. If you would like more information on this configuration or on the general setup for this lab, please contact the lab presenter.

This document shows the /bin/sh shell prompt like the following. Note that that actual text of your shell prompt will vary.

```
prompt=>
```

This document shows the commands to be run after the shell/command prompt like the following:

```
prompt=> cat $HOME/sshlab/labcommands
```

Some of the commands run are very long and may get truncated on the display like the following:

```
prompt=> abcommands  <
```

If this occurs, do the following to increase the number of columns displayed (try using 120):

```
prompt=> stty columns [number of columns]
```

This should already be configured for you, but if while in your PuTTY session, you press the backspace key and control characters are displayed (i.e. it doesn't actually delete the previous character), for example:

```
prompt=> ^H ^H
```

Do the following to map your backspace key:

```
prompt=> stty erase [press Backspace key]
```

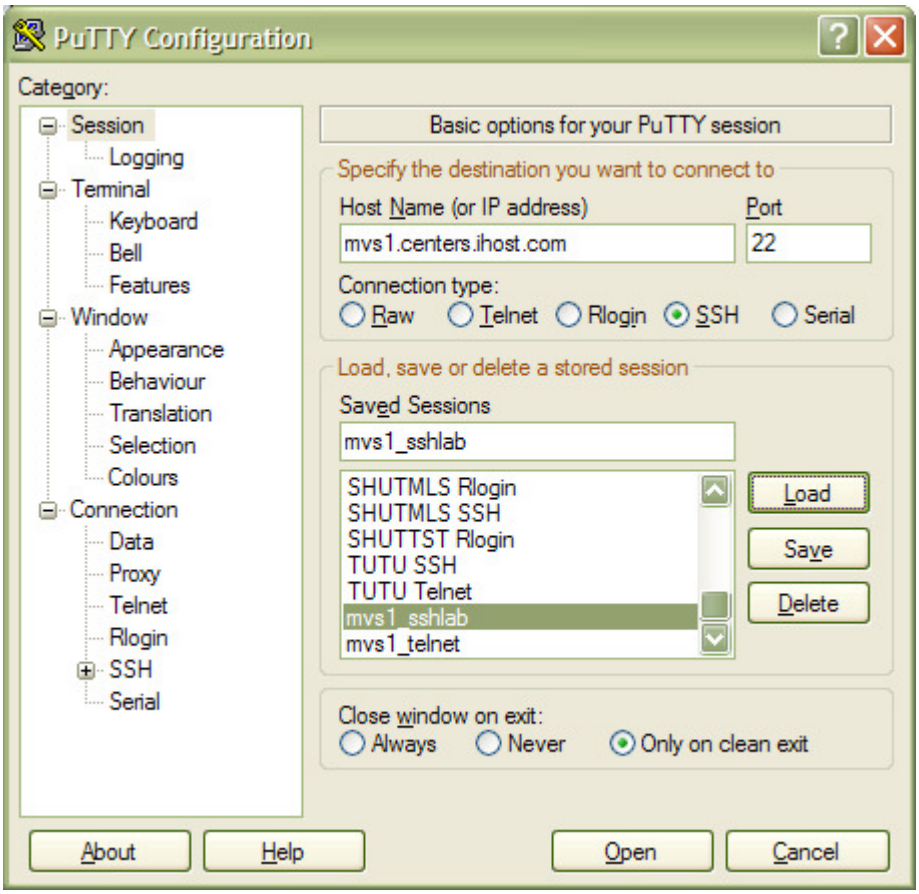
In the real world, this command can be added to a shell profile (/etc/profile or user-specific $HOME/.profile), or your PuTTY configuration can be updated to pass the proper backspace key

Lab Exercises

Where do I start?

☐ Start a PuTTY session. Double-click on PuTTY.

The following configuration window will appear:



☐ Load the "mvs1_sshlab" saved session. If the session does not exist, enter the following fields:

```
Host Name:                  mvs1.centers.ihost.com
Port:                       22
Connection Type:            SSH
```

Note: If you don't have PuTTY on your workstation, refer to the Appendix on where to download it. You can also telnet into `mvs1.centers.ihost.com` using port `623`. However, do not use OMVS to access the z/OS UNIX shell environment for this lab, because the OpenSSH client (ssh) cannot be run under OMVS (i.e. 3270 PCOMM session) due to password visibility issues.

☐ Click “Open”.

Note: If you are using SSH to connect to an sshd daemon/server for the first time, you may see a message looking something like the following:

```
The server's host key is not cached in the registry. You have no
guarantee that the server is the computer you think it is.  The
server's key fingerprint is:
ssh-rsa 1024 7b:e5:6f:a7:f4:f9:81:62:5c:e3:1f:bf:8b:57:6c:5a
If you trust this host, hit Yes to add the key to PuTTY's cache and
carry on connecting.  If you want to carry on connecting just once,
without adding the key to the cache, hit No.  If you do not trust
this host, hit Cancel to abandon the connection.
```

This is a feature of the SSH protocol. It is designed to protect you against a network attack known as spoofing: *secretly redirecting your connection to a different computer, so that you send your password to the wrong system. Using this technique, an attacker would be able to learn the password that guards your login account, and could then log in as if they were you and use the account for their own purposes.*

For now, hit yes to add the key.

> **Information for the real world…**
>
> You should verify public keys obtained from an sshd daemon either visually, or via the key's fingerprint to ensure that you aren’t being spoofed.

☐ Enter your SHARE user ID and password. Remember to **substitute your SHARE user ID** when the lab shows **shar___** or **SHAR___** from now on (e.g. sharb04 where 'b' is the lab letter and '04' is your lab number).

- User ID: **shar___**
- Password: **[The lab presenter will provide the password.]**

How do I configure the OpenSSH server (sshd)?

- ☐ Setup for server authentication when keys are stored in key rings. **New in V1R2**, OpenSSH keys can now be stored in key rings or z/OS UNIX files. In V1R1, OpenSSH keys could only be stored in z/OS UNIX files. For more information on this new V1R2 support, refer to the “Steps for setting up server authentication when keys are stored in key rings” section in the "IBM Ported Tools for z/OS: OpenSSH User's Guide".

  - ☐ Create the server's key ring (1), generate the server's public and private key pair (2), connect the key pair to the key ring (3) and secure the key ring (4, 5 and 6).

Information for the real world… Skip these steps in this SHARE lab.

The following commands for the server’s key ring setup were completed for you by the digital certificate administrator. In the real world, this may not be the same user that configures OpenSSH. Also, your SHARE user ID doesn’t have the necessary authority to do this setup.

```
1. RACDCERT ADDRING(SSHDring) ID(SHAR___)
2. RACDCERT GENCERT ID(SHAR___) SUBJECTSDN(CN('shar___-sshd-
   rsa-cn')) SIZE(2048) WITHLABEL('shar___-sshd-rsa')
3. RACDCERT CONNECT(ID(SHAR___) LABEL('shar___-sshd-rsa')
   RING(SSHDring) USAGE(PERSONAL)) ID(SHAR___)
4. RDEFINE RDATALIB SHAR___.SSHDring.LST UACC(NONE)
5. PERMIT SHAR___.SSHDring.LST CLASS(RDATALIB) ID(SHAR___)
   ACCESS(READ)
6. SETROPTS RACLIST(RDATALIB) REFRESH
```

☐ Verify that you have access to the server's key ring. This step serves as a sanity test of the setup performed by the digital certificate administrator. If set up properly, the ssh-keygen –e command will display the public key in RFC 4716 SSH Public Key File Format. Enter the following command on one line.

```
prompt=> _ZOS_SSH_KEY_RING_LABEL="SHAR___/SSHDring shar___-sshd-rsa"
ssh-keygen -e
```

Information for the real world…

- To allow remote users to ssh into this local host, you need to "publish" your public host keys. To do this, you would send the local host's public keys to the remote host and then configure the remote host's system-wide resources accordingly. However, in this SHARE lab, you don't have access to the system-wide resources (e.g. the /etc/ssh/ssh_known_hosts file), so you will instead configure this later, at a user-level.
- To allow users on this local host to ssh to remote hosts, you would get the public host key files from the remote hosts, and then configure the local host's system-wide resources accordingly. However, for this SHARE lab, this is the same as the above bullet (since the local and remote hosts are the same system).
- You need to create the SSHD privilege separation user. However, for this SHARE lab, this was already done for you.

☐ Create and modify the sshd daemon configuration files (sshd_config and zos_sshd_config).

Information for the real world… Skip these steps in this SHARE lab.

- The sshd_config and zos_sshd_config files are normally stored in /etc/ssh/.
- As part of this setup…
  - You would normally create the system-wide OpenSSH client (ssh) configuration file (/etc/ssh/ssh_config). However, for purpose of this lab, you will instead (later) set up a user-specific ssh configuration file.
  - You would also copy other files (see below) that OpenSSH needs to run. However, this was already done for you, since your SHARE user ID doesn't have write access to the /etc/ssh directory.

```
prompt=> cp -p /samples/moduli /etc/ssh/moduli
prompt=> cp -p /samples/ssh_prng_cmds /etc/ssh/ssh_prng_cmds
```

- ☐ Copy the sample zos_sshd_config file and set the proper permissions.

```
prompt=> cp /samples/zos_sshd_config $HOME/sshlab/zos_sshd_config
prompt=> chmod 600 $HOME/sshlab/zos_sshd_config
```

- ☐ Change the zos_sshd_config file to use the server key ring that was setup earlier. Enter the following command on one line.

```
prompt=> echo "HostKeyRingLabel \"SHAR___/SSHDring shar___-sshd-rsa\""
>> $HOME/sshlab/zos_sshd_config
```

- ☐ Copy the sample sshd_config file and set the proper permissions.

```
prompt=> cp /samples/sshd_config $HOME/sshlab/sshd_config
prompt=> chmod 600 $HOME/sshlab/sshd_config
```

- ☐ Change the sshd_config file to use your assigned Port number. Use 7000 + the last 2 digits of your SHARE user ID. For example, user sharb30 should use port 7030.

```
prompt=> echo "Port 70xx" >> $HOME/sshlab/sshd_config
```

- ☐ Change the sshd_config file to update the location of the sshd process ID (PID) file. The sshd PID file holds the PID of your sshd process. Enter the following command on one line.

```
prompt=> echo "PidFile $HOME/sshlab/sshd.pid" >>
$HOME/sshlab/sshd_config
```

- ☐ Verify that your sshd daemon configuration file modifications were made by issuing the following commands. Your modifications to the sshd_config and zos_sshd_config files should be listed.

```
prompt=> tail -1 $HOME/sshlab/zos_sshd_config
prompt=> tail -2 $HOME/sshlab/sshd_config
```

☐ Start the sshd daemon using your sshd_config and zos_sshd_config configuration files. The sshd daemon should fork itself into the background and return you to the command prompt once it's started.

```
prompt=> export _ZOS_SSHD_CONFIG=$HOME/sshlab/zos_sshd_config
prompt=> /usr/sbin/sshd -f $HOME/sshlab/sshd_config
```

☐ Verify that your sshd daemon is running by issuing the following ps command. Your sshd daemon process should have a parent PID (PPID) of 1 and a TTY of "?".

```
prompt=> ps -f -p $(cat $HOME/sshlab/sshd.pid)
     UID        PID       PPID  C    STIME TTY       TIME CMD
 SHAR___   67109011          1  - 12:27:50 ?         0:00
/usr/sbin/sshd -f /sharelab/shar___/sshlab/sshd_config
```

☐ Verify that your sshd daemon is running on your port. Substitute "xx" below with the last 2 digits of your SHARE user ID.

```
prompt=> netstat -P 70xx
MVS TCP/IP NETSTAT CS V1R11       TCPIP Name: TCPIP           14:35:28
User Id  Conn     Local Socket           Foreign Socket         State
-------  ----     ------------           --------------         -----
SHAR___n 00002302 0.0.0.0..70xx          0.0.0.0..0             Listen
```

How do I configure a user to use the OpenSSH client (ssh)?

☐ Complete the setup for server authentication when keys are stored in key rings. Recall that in this SHARE lab, you don't have access to the system-wide resources and that the local and remote hosts are the same. As a result, this needs to be configured at a user-level and on the same system. Also remember that **New in V1R2**, OpenSSH keys can now be stored in key rings or z/OS UNIX files.

☐ Create your known hosts key ring (1), connect the server key pair to the key ring (2) and secure the key ring (3, 4 and 5).

Information for the real world… Skip these steps in this SHARE lab.

The following commands for the known hosts key ring setup were completed for you by the digital certificate administrator. In the real world, this may not be the same user that configures OpenSSH. Also, your SHARE user ID doesn't have the necessary authority to do this setup.

```
1. RACDCERT ADDRING(SSHKnownHostsRing) ID(SHAR___)
2. RACDCERT CONNECT(ID(SHAR___) LABEL('shar___-sshd-rsa')
   RING(SSHKnownHostsRing) USAGE(PERSONAL)) ID(SHAR___)
3. RDEFINE RDATALIB SHAR___.SSHKnownHostsRing.LST UACC(NONE)
4. PERMIT SHAR___.SSHKnownHostsRing.LST CLASS(RDATALIB)
   ID(SHAR___) ACCESS(READ)
5. SETROPTS RACLIST(RDATALIB) REFRESH
```

☐ Verify that you have access to your known hosts key ring. This step serves as a sanity test of the setup performed by the digital certificate administrator. If set up properly, the ssh-keygen –e command will display the public key in RFC 4716 SSH Public Key File Format. Enter the following command on one line.

```
prompt=> _ZOS_SSH_KEY_RING_LABEL="SHAR___/SSHKnownHostsRing
shar___-sshd-rsa" ssh-keygen -e
```

☐ Create the known hosts file to use the known hosts key ring that was setup earlier. The OpenSSH client uses this file when verifying the authenticity of the server that is being logged into. Be sure to enter the first command on one line.

```
prompt=> echo "localhost zos-key-ring-label=\"SHAR___/SSHKnownHostsRing
shar___-sshd-rsa\"" > $HOME/.ssh/known_hosts
prompt=> chmod 600 $HOME/.ssh/known_hosts
```

- ☐ Setup your SHARE user ID to use public key user authentication when keys are stored in key rings. **New in V1R2**, OpenSSH keys can now be stored in key rings or z/OS UNIX files. In V1R1, OpenSSH keys could only be stored in z/OS UNIX files. For more information on this new V1R2 support, refer to the "Steps for setting up user authentication when keys are stored in key rings" section in the "IBM Ported Tools for z/OS: OpenSSH User's Guide".

  - ☐ Create your user key ring (1), generate your user public and private key pair (2), connect the key pair to the key ring (3) and secure the key ring (4, 5 and 6).

Information for the real world… Skip these steps in this SHARE lab.

The following commands for the user key ring setup were completed for you by the digital certificate administrator. In the real world, this may not be the same user that configures OpenSSH. Also, your SHARE user ID doesn't have the necessary authority to do this setup.

```
1. RACDCERT ADDRING(SSHring) ID(SHAR___)
2. RACDCERT GENCERT SUBJECTSDN(CN('shar___-ssh-dsa-cn'))
   SIZE(1024) DSA WITHLABEL('shar___-ssh-dsa') ID(SHAR___)
3. RACDCERT CONNECT(ID(SHAR___) LABEL('shar___-ssh-dsa')
   RING(SSHring) USAGE(PERSONAL)) ID(SHAR___)
4. RDEFINE RDATALIB SHAR___.SSHring.LST UACC(NONE)
5. PERMIT SHAR___.SSHring.LST CLASS(RDATALIB) ID(SHAR___)
   ACCESS(READ)
6. SETROPTS RACLIST(RDATALIB) REFRESH
```

  - ☐ Verify that you have access to your user key ring. This step serves as a sanity test of the setup performed by the digital certificate administrator. If set up properly, the ssh-keygen –e command will display the public key in RFC 4716 SSH Public Key File Format. Enter the following command on one line.

```
prompt=> _ZOS_SSH_KEY_RING_LABEL="SHAR___/SSHring shar___-ssh-dsa"
ssh-keygen -e
```

☐ Create your authorized keys key ring (1), connect your user key pair to the key ring (2) and secure the key ring (3, 4 and 5).

Information for the real world… Skip these steps in this SHARE lab.

The following commands for the authorized keys key ring setup were completed for you by the digital certificate administrator. In the real world, this may not be the same user that configures OpenSSH. Also, your SHARE user ID doesn't have the necessary authority to do this setup.

```
1. RACDCERT ADDRING(SSHAuthKeysRing) ID(SHAR___)
2. RACDCERT CONNECT(ID(SHAR___) LABEL('shar___-ssh-dsa')
   RING(SSHAuthKeysRing) USAGE(PERSONAL)) ID(SHAR___)
3. RDEFINE RDATALIB SHAR___.SSHAuthKeysRing.LST UACC(NONE)
4. PERMIT SHAR___.SSHAuthKeysRing.LST CLASS(RDATALIB)
   ID(SHAR___) ACCESS(READ)
5. SETROPTS RACLIST(RDATALIB) REFRESH
```

☐ Verify that you have access to your authorized keys key ring. This step serves as a sanity test of the setup performed by the digital certificate administrator. If set up properly, the ssh-keygen –e command will display the public key in RFC 4716 SSH Public Key File Format. Enter the following command on one line.

```
prompt=> _ZOS_SSH_KEY_RING_LABEL="SHAR___/SSHAuthKeysRing
shar___-ssh-dsa" ssh-keygen -e
```

Information for the real world…

If locally you are USER1 on LOCALHOST, and you want to log into REMOTEHOST as USER2, then you would normally export USER1's public key for use by USER2's authorized keys key ring on the REMOTEHOST. However for this SHARE lab, since both the local and remote users and the local and remote hosts are the same, there is no need to export the public key.

☐ Create and modify the ssh configuration files ($HOME/.ssh/config and $HOME/.ssh/zos_user_ssh_config) and the authorized keys file ($HOME/.ssh/authorized_keys).

☐ Copy the sample zos_user_ssh_config file and set the proper permissions.

```
prompt=> cp /samples/zos_user_ssh_config $HOME/.ssh/zos_user_ssh_config
prompt=> chmod 600 $HOME/.ssh/zos_user_ssh_config
```

- ☐ Change the zos_user_ssh_config file to use the user key ring that was setup earlier. Enter the following command on one line.

```
prompt=> echo "IdentityKeyRingLabel \"SHAR___/SSHring
shar___-ssh-dsa\"" >> $HOME/.ssh/zos_user_ssh_config
```

- ☐ Create the authorized keys file to use the authorized keys key ring that was setup earlier. The OpenSSH server uses this file when verifying the authenticity of the user that is logging in during public key authentication. Be sure to enter the first command on one line.

```
prompt=> echo "zos-key-ring-label=\"SHAR___/SSHAuthKeysRing
shar___-ssh-dsa\"" > $HOME/.ssh/authorized_keys
prompt=> chmod 600 $HOME/.ssh/authorized_keys
```

- ☐ Copy the sample ssh_config file and set the proper permissions.

```
prompt=> cp /samples/ssh_config $HOME/.ssh/config
prompt=> chmod 600 $HOME/.ssh/config
```

- ☐ Change the ssh_config file to use your sshd daemon's port number. Substitute "xx" below with the last 2 digits of your SHARE user ID.

```
prompt=> echo "Port 70xx" >> $HOME/.ssh/config
```

- ☐ Log in using ssh. Your should be allowed to login without entering a password. If you see "`shar___@localhost's password:`" then public key user authentication isn't configured properly, so ssh continued onto password authentication. If this occurs, review that you've correctly completed all the above steps.

```
prompt=> ssh shar___@localhost
```

- ☐ Once logged in (you will see a command prompt), echo the SSH_CONNECTION environment variable. It shows the **client** and **server** ends of the SSH connection, respectively. Both the client and server IP addresses should match that of the SHARE system and the server port number (70xx) should match your sshd daemon's port.

```
prompt=> echo $SSH_CONNECTION
127.0.0.1 1055 127.0.0.1 70xx
```

- ☐ Exit from this ssh session.

```
prompt=> exit
```

How do I run ssh in debug mode?

☐ Run ssh with the –vvv option (i.e. debug mode), to see its progress. You will see pages of verbose output. Running ssh in debug mode can be useful when debugging problems.

Note: This ssh command will perform remote command execution of the id command. Therefore, you will see the output from the id command within the verbose ssh output. After the id command completes, ssh will automatically log you out.

```
prompt=> ssh -vvv shar___@localhost id
```

How do I run sshd in debug mode?

For this lab, you can view sshd daemon log information by running sshd in debug mode.

> **Information for the real world…**
>
> Rather than running sshd in debug mode, you would likely use the z/OS UNIX syslog to gather sshd log information. This can be done by using the sshd_config LogLevel and SyslogFacility keywords.

☐ Kill your current sshd daemon, so you can reuse that port.

```
prompt=> kill -TERM $(cat $HOME/sshlab/sshd.pid)
```

☐ Open another PuTTY session, so you have one for the ssh client, and one for the sshd daemon.

☐ From your original PuTTY session, start the sshd daemon as follows. Notice that it doesn't fork itself into the background, and all output is written to standard error (stderr). Enter the following command on one line.

```
prompt=> /usr/sbin/sshd -De -oLogLevel=DEBUG3 -f
$HOME/sshlab/sshd_config
```

> **Information for the real world…**
>
> Logging with a debug level (e.g. DEBUG3), violates the privacy of users and is not recommended. It should only be used to debug problems.

☐ After you see the output "`Server listening on ...`", ssh to your sshd daemon from the other PuTTY session. In both PuTTY sessions you will see verbose output.

Note: You can ignore the sshd daemon output "`Attempt to write login records by non-root user (aborting)`" since it is a result of your SHARE user ID not having system administrator privileges

```
prompt=> ssh -vvv shar___@localhost
```

☐ Exit from your ssh session. We will leave the sshd daemon running for subsequent exercises. When all lab exercises are complete, you can use `Ctrl-C` to end the sshd daemon.

```
prompt=> exit
```

How do I restrict OpenSSH client access?

You can use the sshd_config DenyUsers keyword to have your SHARE user ID denied OpenSSH client access to your sshd daemon. Also **New in V1R2**, you can use the sshd_config Match and ForceCommand keywords to further limit OpenSSH client access. These OpenSSH client restrictions apply to ssh, scp and sftp.

> **Information for the real world…**
>
> The sshd_config ChrootDirectory keyword can also be used to restrict OpenSSH client access. However, your SHARE user ID doesn't have the appropriate privileges to use the keyword. For more information on restricting OpenSSH client access refer to the "Limiting file system name space for sftp users" section in the "IBM Ported Tools for z/OS: OpenSSH User's Guide".

☐ Edit your sshd_config file to contain a DenyUsers entry. Substitute "___" below with the last 3 characters of your SHARE user ID. First, go to your lab directory.

```
prompt=> cd $HOME/sshlab
```

☐ Back up your current sshd_config file.

```
prompt=> cp sshd_config sshd_config.bak
```

☐ Modify your sshd_config file. Your SHARE user ID must be in all uppercase letters (e.g. SHARB04). This is important because the user ID must be in the same alphabetical case as is stored in the user database.

```
prompt=> echo "DenyUsers SHAR___" >> sshd_config
```

☐ Send a SIGHUP signal to the sshd daemon process, so it re-reads the sshd_config file.

```
prompt=> kill -HUP $(cat $HOME/sshlab/sshd.pid)
```

☐ After you see the sshd daemon output "`Server listening on ...`" in your other PuTTY session, attempt to ssh to your sshd daemon. You will be prompted for your password. Enter it. It should be refused. In the sshd daemon window, you will see (you may have to scroll back to see the message) "`FOTS2306 User SHAR___ from localhost not allowed because listed in DenyUsers`". Continue entering your password when prompted until the ssh session is ended.

```
prompt=> ssh shar___@localhost
```

☐ Restore your sshd_config file.

```
prompt=> cp sshd_config.bak sshd_config
```

☐ Edit your sshd_config file to contain a Match and ForceCommand entry. Substitute "___" below with the last 3 characters of your SHARE user ID. As indicated earlier, your SHARE user ID must be in all uppercase letters (e.g. SHARB04).

The sshd_config Match keyword *introduces a conditional block. If all of the criteria on the Match line are satisfied, the keywords on the following lines override those set in the global section of the config file, until either another Match line or the end of the file*. This keyword allows you to customize your sshd daemon configuration files based on various match criteria such as users or groups.

```
prompt=> echo "Match User SHAR___" >> sshd_config
prompt=> echo "  ForceCommand internal-sftp" >> sshd_config
```

☐ Send a SIGHUP signal to the sshd daemon process, so it re-reads the sshd_config file.

```
prompt=> kill -HUP $(cat $HOME/sshlab/sshd.pid)
```

☐ After you see the sshd daemon output "`Server listening on ...`" in your other PuTTY session, attempt to ssh to your sshd daemon. Your connection will appear hung. This is because you requested an ssh session but the sshd daemon forced you to use an sftp session. Type `Ctrl-C` to exit the ssh session. Attempting to scp to your sshd daemon will yield the same result.

```
prompt=> ssh shar___@localhost ls
```

☐ Attempt to sftp to your sshd daemon. Your connection should now succeed. Type `quit` at the sftp prompt to exit the sftp session.

```
prompt=> sftp shar___@localhost
sftp> quit
```

☐ Undo your changes for the next exercise.

```
prompt=> cp sshd_config.bak sshd_config
prompt=> kill -HUP $(cat $HOME/sshlab/sshd.pid)
prompt=> cd
```

How do I share an OpenSSH client connection?

For this lab, you will use the **New in V1R2** OpenSSH client connection sharing feature. Connection sharing can improve the overall performance of OpenSSH client connections by avoiding the overhead of the setup and authentication steps. This OpenSSH client feature applies to ssh, scp and sftp.

☐ Start the master process to enable ssh connection sharing. This ssh connection **does** go through the setup and authentication steps. We use the "&" /bin/sh shell syntax to keep the master process running in the background.

```
prompt=> ssh -Nn -oControlMaster=yes -oControlPath=$HOME/sshlab/control
shar___@localhost &
```

☐ Check that the master process is active. This command should indicate that the master process is running.

```
prompt=> ssh -O check -oControlPath=$HOME/sshlab/control
shar___@localhost
```

☐ Access the master process via the ssh connection sharing socket (which is specified using the ssh_config ControlPath keyword). This ssh connection **does not** go through the setup and authentication steps because it uses the master process which already completed the steps.

```
prompt=> ssh -oControlPath=$HOME/sshlab/control shar___@localhost id
```

☐ The following commands help illustrate that the master process was used above. The first command doesn't access the master process and authenticates using password authentication. The second command accesses the master process and is thus able to bypass the password authentication seen on the first command. In addition, the second command should connect faster than the first command.

```
prompt=> ssh -oPubkeyAuthentication=no shar___@localhost id
prompt=> ssh -oPubkeyAuthentication=no
-oControlPath=$HOME/sshlab/control shar___@localhost id
```

☐ Exit the master process.

```
prompt=> ssh -O exit -oControlPath=$HOME/sshlab/control
shar___@localhost
```

Use OpenSSH for secure file transfers

OpenSSH implements the Secure Shell (SSH) protocol for doing secure file transfers. The SSH protocol and OpenSSH are built on open standards and are supported on most platforms. This makes OpenSSH a popular secure file transfer option.

- ☐ OpenSSH secure file transfers can be done interactively using the sftp or scp commands. However, users wanting to automate their secure file transfers often create shell scripts that can be run in batch. For example, the first command in this step displays a shell script that does a secure file transfer. The second command runs the shell script. Run the following commands to complete the secure file transfer.

```
prompt=> cat $HOME/sshlab/labsftp.sh
prompt=> $HOME/sshlab/labsftp.sh SHAR___
```

- ☐ Many z/OS users create JCL scripts in order to run their secure file transfers in batch. For example, the first command below displays a JCL script that runs the shell script from the previous step. The second command submits the batch job. Run the following commands to complete the secure file transfer. (If you wish to verify the success of this, first remove the 'myxfer' file from the lab directory, then when complete the 'myxfer' file will have been recreated).

```
prompt=> cat $HOME/sshlab/labsftp.jcl
prompt=> cat $HOME/sshlab/labsftp.jcl | submit
```

- ☐ **New in V1R2** with the PTF for APAR OA37278, OpenSSH can use ICSF to implement certain ciphers and MAC algorithms in order to take advantage of CP Assist for Cryptographic Function (CPACF) hardware support when applicable. This support can improve the performance and minimize CPU time of SSH session on z/OS, since the ciphers (encryption and decryption) and MAC algorithms (hashing) represent a significant portion of the processing done during an SSH session. This benefit applies to secure file transfers as well. Complete the setup for this support on the client side. Note that similar setup can be done on the server side as well.

```
prompt=> echo "CiphersSource any" >> $HOME/.ssh/zos_user_ssh_config
prompt=> echo "MACsSource any" >> $HOME/.ssh/zos_user_ssh_config
prompt=> echo "Ciphers aes128-cbc,3des-cbc,aes192-cbc,aes256-
cbc,aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,blowfish-
cbc,cast128-cbc,arcfour" >> $HOME/.ssh/config
prompt=> echo "MACs hmac-sha1,hmac-sha1-96,hmac-md5,umac-
64@openssh.com,hmac-ripemd160,hmac-md5-96" >> $HOME/.ssh/config
```

Information for the real world…

Your SHARE user ID doesn't have the necessary authority to complete some of the setup to use ICSF ciphers and MAC algorithms. As a result, some of the setup has been done for you. For more information on the setup, refer to the "Setting up OpenSSH to use ICSF ciphers and MAC algorithms" section in the "IBM Ported Tools for z/OS: OpenSSH User's Guide".

☐ To determine the cipher and MAC algorithm source used by OpenSSH, start ssh in debug mode. If OpenSSH is set up to use ICSF, the debug mode also provides ICSF Query Algorithm (CSFIQA) debug statements to help determine how ICSF is implementing the ciphers and MAC algorithms. Run the following command to verify the ICSF ciphers and MAC algorithms setup. You should see debug statements as detailed below that show ICSF is implementing the cipher (i.e. aes128-cbc, AES) and MAC (i.e. hmac-sha1, SHA-1) using CPACF (i.e. CPU):

```
prompt=> ssh -vv shar___@localhost id
...
debug2: AES      256      SECURE   CPU
...
debug2: SHA-1    160      NA       CPU
...
debug1: mac_setup_by_id: hmac-sha1 from source ICSF
...
debug1: cipher_init: aes128-cbc from source ICSF
...
```

☐ Complete the secure file transfer using ICSF ciphers and MAC algorithms support.

```
prompt=> $HOME/sshlab/labsftp.sh SHAR___
```

Check the man pages and try different options

If you have extra time and want to try a few things, here are some other options that you may want to try. Reminder, if you still have an sshd daemon running in debug mode, you can use `Ctrl-C` to end it.

☐ Issue the man command for more information on the ssh and sshd options and the ssh_config and sshd_config keywords.

```
prompt=> man ssh
prompt=> man sshd
prompt=> man ssh_config
prompt=> man sshd_config
```

ssh options to consider trying:
-c cipher → Try a different cipher for encryption.
-m mac → Try a different MAC for message authentication.
-V → Display the OpenSSH version information.

sshd options to consider trying:
-t → Test the validity of the sshd_config and zos_sshd_config files and the host keys.

ssh_config keywords to consider trying:
Ciphers → Try a different cipher for encryption.
LogLevel → Try a different logging level.
MACs → Try a different MAC for message authentication.
NumberOfPasswordPrompts → Limit the number of password prompts.
StrictHostKeyChecking → Try different values after removing your known hosts file.

sshd_config keywords to consider trying:
Banner → Have sshd display the contents of a banner file before authentication.
Ciphers → Try a different cipher for encryption.
MACs → Try a different MAC for message authentication.
MaxStartups → Limit the number of concurrent un-authenticated connections to sshd.
StrictModes → Have sshd skip checking file modes and ownership during authentication.

This is the end of the lab.
Hope you had fun!

Appendix

Shell command-line editing quick reference

Issue the following to enable vi command-line editing: `set -o vi`
Note: This has already been enabled by default on the SHARE system.

To leave **insertion** mode and enter **command** mode (so the characters you type are understood as commands), press the Escape key [ESC]. Do this before using the commands below. While in **command** mode, [ESC] will return you to **insertion** mode.

To do the following (after [ESC]):	Type this command:
Recall previous command line	k
Move cursor left	h
Move cursor right	l (this is a lowercase L)
Insert characters after cursor	i
Append characters after cursor	a
Replace characters	R[type your text][ESC]
Replace 1 character	r[type your character]
Delete 1 character	x
Execute command line (while in command mode)	Enter (when line is displayed)
Discard command line	^C (Ctrl-C)
Complete filename	\

Documentation

IBM Publications for IBM Ported Tools for z/OS:
http://www.ibm.com/systems/z/os/zos/features/unix/ported/

OpenSSH Home Page:
http://www.openssh.org/

IETF Secure Shell (secsh) RFCs:
http://tools.ietf.org/wg/secsh/

SSH The Secure Shell, The Definitive Guide.
Barret, Silverman & Byrnes. 2005 O'Reilly & Associates, Inc.

PuTTY download

http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

How to logon to TSO (3270 interface)

Start the emulator

1. Double click on **SHARE System** icon. This starts a PCOMM 3270 session using mvs1.centers.ihost.com. **Note:** The Enter key is the right Ctrl key
2. You can skip this step for now, but for real use, you may want to configure session parameters to use:
   Screen Size: 43x80
   Host Code Page: 1047 United States

Logon to TSO/E

1. When prompted for Userid/Password/Application, enter TSO in the Application field and press the Enter key.
2. User ID: **shar___**
3. Password: **[The lab presenter will provide the password.]**
4. ISPF will be started
5. From ISPF, enter option 6
6. Enter: `omvs esc('@')`
   This starts a login shell with an escape character of '@'. The escape character is used to simulate the Ctrl key. The default is the cent sign, which would need to be configured in the emulator. You can also configure the emulator so that popular Ctrl keys (e.g. Ctrl-C, Ctrl-Z) generate the appropriate OMVS escape sequence. With the above command, to interrupt a running command, you enter @c on the command line.

For example, if you see:

```
[press Ctrl-D]
```

You will instead:

```
[press @D Enter]
```

The Enter key is required because a 3270 session is a line mode terminal.