SSHDaemon (Win32) User's Guide

Table of Contents

Introduction

SSHDaemon is a Windows-based SSH server written in JAVA. Based upon the proven SSHTools J2SSH library, the server provides an extensible solution for rapid development of SSH-enabled applications. The aims of the project are to provide a free-to-use SSH server, based upon a platform-independent code to speed and simplify development of security-related open source software destined for deployment across today's multi-platform networks. Initially based upon Microsoft Windows, given the popularity bestowed upon this platform in the corporate sector - SSHDaemon is constantly under development and being ported to other major platforms, with a Linux based server currently under development.

Installation

The following instructions should be suitable for Windows NT/2000/XP where a compatible Java Virtual Machine (JVM) is available. The latest version of the server can be obtained from our SourceForge project page [http://sourceforge.net/projects/sshtools] or from the SSHTools web site [???].

System Requirements

SSHDaemon has the following system requirements:

- Sun Java[tm] runtime 1.4.0 or above (1.4.1 or above highly recommended).

Before installation please verify the location of the Java Runtime Environment to ensure that either the environment variable JAVA_HOME exists or that the path to java.exe is included within your PATH environment variable.

Extracting the Distribution

Extract the distribution to your chosen location. By default the zip or tar.gz package will expand into a directory named *sshdaemon-VERSION*. This directory will be referred to from this point as $SSHD_HOME.

Generating the Server's Host Key(s)

Open up a command console and change directory to $SSHD_HOME/bin. To use the default configuration, issue the following command to create a DSA key called *server_host_key* in the configuration directory. The server host key MUST have an empty passphrase, be sure to press return when asked and type 'yes' to confirm.

```
> ssh-keygen ..\conf\server_host_key
      ****Sshtools.com SSH Key Pair Generator**** Generating 1024 bit ssh-dss
      key pair ................................................... Creating
      Public Key file ..\conf\server_host_key.pub Generating Private Key file
      ..\conf\server_host_key Enter passphrase: Confirm passphrase: You
      supplied an empty passphrase, are you sure? [Yes|No]: yes
```

You can optionally add a second RSA key or change the location/name of the the DSA key, this requires a manual configuration of the *server.xml* configuration file. To generate additional keys use the ssh-keygen.bat script, either passing the required command line parameters or using the -g option to invoke the key generation GUI.

For more information on how to configure the xml file, refer to the Configuring server.xml section.

```
> ssh-keygen.bat Usage: SshKeyGenerator
      [options] filename Options: -b bits Number of bits in the key to create.
      -e Convert OpenSSH to IETF SECSH key file. -i Convert IETF SECSH to
      OpenSSH key file. -t type The type of key to create. -p Change the
      passphrase of the private key file. -g Use GUI to create key
```

Checking System Dependencies

The server requires a system DLL file called MSCVCP60.DLL. This is distributed with many applications and your server may already have it installed. The file has been provided in case your server is a clean installation and does not contain it. MSVCP60.DLL will be located in your system directory, typically on a Windows 2000 machine this would be c:\winnt\system32. Verify that the file exists, if it does not copy the file from the *$SSHD_HOME\jni\daemon\win32\lib* directory into your system path.

For example, use the following command from $SSHD_HOME:

```
>copy jni\daemon\win32\lib\msvcp60.dll
      c:\winnt\system32
```

Installing as a Windows Service (Recommended)

As of release 0.0.6, SSHDaemon may be installed and run as a Windows service. This is the recommended method and most secure way of using the server. When running as a service, the process is executed within the LocalSystem account. This requires no additional security policy configuration since the LocalSystem account holds all the necessary user rights assignments by default. The use of this account also allows the server to continue to operate when no user is logged into the system.

After extracting the distribution and configuring the server host keys, if java.exe is within your PATH then no further configuration is necersary. However, if not then you need to edit the *sshd-service.conf* file located in *$SSHD_HOME\conf*. Locate the *wrapper.java.command* property and set the value to your java.exe location.

```
wrapper.java.command=c:\j2sdk1.4.0\bin\java.exe
```

Run the *Install-SSHDaemon-Service.bat* file which is located in the *$SSHD_HOME/bin* directory. This installs and starts the "SSHDaemon" service.

Your installation is now complete and you should now be able to connect using a standard SSH client. You can uninstall the service using the *Uninstall-SSHDaemon-Service.bat* file.

The ability to install SSHDaemon as a service under Microsoft Windows may be credited to the Java Service Wrapper project, created by Tanuki Software Organization. [???]

Installing as a Standalone Process

If you do not wish to install the server as a Windows service, you may install it as a standalone process that the user can start when required. After extracting the distribution package, some additional configuration is required to Windows user-access rights before starting the server. In order for the server to provide shell access when running outside of the LocalSystem account, the user running the server process requires a number of user rights that can be assigned within the Local Security Policy editor.

These user rights assignments DO NOT need to be set if you are running SSHDaemon as a Windows service, as described in the previous section.

The following user rights are required for assignment:

- *Act as part of the operating system* - This policy allows a process to authenticate as any user, and therefore gain access to the same resources as any user.
- *Replace a process level token* - Determines which user accounts can initiate a process to replace the default token associated with a launched subprocess.
- *Create a token object* - Allows the server to create a token which can be used to get access to any local resources when the process uses NTCreateToken() or other token-creation API's. This is required for logging in a user using Public-key authentication.

Reboot the computer after assigning these rights either directly to the user, or to the User's group.

Once the computer has restarted and the user logged in, you can start the server by running the *$SSHD_HOME/bin/sshd-start.bat* script.

Similary, to stop the server run the *$SSHD_HOME/bin/sshd-stop.bat* script.

Server Configuration

The server has several useful configuration files which allow the user to modify the operation of the server and determine an access policy. This section details a number of common tasks in addition to a breakdown of the configuration items available in each file.

Configuring a User Account for Public-key Access

When a user logs into the server using public-key authentication, the server looks for an authorization file named *authorization.xml*. This file is located in the path governed by the <UserConfigDirectory> element of the *server.xml* configuration file.

```
<UserConfigDirectory>%D\.ssh2</UserConfigDirectory>
```

The path specified in UserConfigDirectory may also contain a number of wildcards for ease of administration. The following wildcards are permitted:

%D - The current user's home directory

%U - The current user's name

Taking the example above - if the current user's home directory is *"c:\users\john"*, the server will look in the *"c:\users\john\.ssh2"* directory. The default file should be called authorization.xml but you can also configure this by amending the following property of server.xml.

```
<AuthorizationFile>authorization.xml</AuthorizationFile>
```

Once you have located the correct directory, create an authorization.xml tempate as follows:

```
<?xml version="1.0"
      encoding="UTF-8" ?> <!-- Sshtools User Authorization
      File --> <AuthorizedKeys> <!-- Enter authorized public
      key elements here --> </AuthorizedKeys>
```

Next, generate a key pair using the *ssh-keygen* script that came with the distribution using the following command:

```
ssh-keygen.bat mykey
```

This will generate two files - the private key *mykey*, and the public keyfile *mykey.pub*. Transfer the public key file to the user's configuration directory where the authorization.xml file is located. Finally, add the following element to the authorization.xml file between the <AuthorizedKeys> elements:

```
<Key>mykey.pub</Key>
```

You should now be able to connect from an SSHTools client using the private key.

Configuring Server.xml

The following options are available for configuration within the server.xml configuration file.

<ServerHostKey>

The <ServerHostKey> element installs a private key for server authentication.

```
<ServerHostKey
        PrivateKeyFile="server_host_key"/>
```

The PrivateKeyFile attribute must point to a valid private key file that IS NOT passphrase protected that has been created using ssh-keygen.bat script.

<Subsystem>

The <Subsystem> element installs an SSH subsystem for use.

```
<Subsystem Name="sftp"
        Type="class"
        Provider="com.sshtools.j2ssh.sftp.SftpSubsystemServer"/>
```

The Type element defines the type of subsystem provider. This can either be 'class' in which case the Provider attribute must point to a valid Java subsystem implementation. If the type is 'exe' the Provider attribute must point to a windows executable that implements an SSH subsystem.

<AuthenticationBanner>

Specifes the message to display to each client before authentication. The value must point to a text file, if an absolute path is not provided, the file is assumed to be located in the $SSHD_HOME\conf directory.

```
<AuthenticationBanner>secuity.txt</AuthenticationBanner>
```

<MaxConnections>

Specifies the maximum number of connections that are allowed at any one time.

```
<MaxConnections>10</MaxConnections>
```

<ListenAddress>

Configures the server to listen on a specific address. The default setting of 0.0.0.0 listens on all addresses available.

```
<ListenAddress>0.0.0.0</ListenAddress>
```

<Port>

Configures the server to listen on a specific port. The default setting is the standard SSH port 22.

```
<Port>22</Port>
```

<CommandPort>

The server listens on a port for commands, for example, to stop the server a message is sent through the local loopback address to instruct the server to close. This setting configures the port that is used for this opertion.

```
<CommandPort>10001</CommandPort>
```

<TerminalProvider>

Specifies the executable that is executed when the remote user requests a shell.

```
<TerminalProvider>c:\winnt\system32\cmd.exe</TerminalProvider>
```

<AllowedAuthentication>

Each authentication method that will be available to the remote users should be specified as an <AlloedAuthentication>.

```
<AllowedAuthentication>password</AllowedAuthentication>
```

The available authentications are

- password
- publickey
- keyboard-interactive

<RequiredAuthentication>

Specifies which authentications that are included within the <AllowedAuthentication> list should be required by the user.

```
<RequiredAuthentication>password</RequiredAuthentication>
```

<AuthorizationFile>

Specifies the name of the public key authorization file.

```
<AuthorizationFile>authorization.xml</AuthorizationFile>
```

<UserConfigDirectory>

Specifies the location where the users' authorization file will be located.

```
<UserConfigDirectory>%D\.ssh2</UserConfigDirectory>
```

The following tokens are valid:

- %D - Replaced by the user's home directory

- %U - Replaced by the user's name

Configuring windows.xml

The *windows.xml* file configures the standard J2SSH SSH server class for the Windows Platform. The following elements MUST NOT be changed as these settings are crucial to the opertion of the Windows Server.

- *<NativeProcessProvider> determines the platforms process provider*:

  This should always be set to com.sshtools.daemon.windows.WindowsProcess

- *<NativeAuthenticationProvider> determines the platforms authentication provider*:

  This should always be set to com.sshtools.daemon.windows.WindowsAuthentication

The remaining elements are for advanced server configuration.

- *<NativeFileSystemProvider> determines the platforms file system*:

  This is defaulted to com.sshtools.j2ssh.sftp.VirtualFileSystem which provides abstracted file access. File system access is provided through mounts set within this configuration file. For further details see the following SFTP Configuration section.

- <NativeSetting name="AuthenticateOnDomain" value="."/>

  This setting instructs the Windows authentication provider to authenticate ona particular domain. The default setting of "." authenticates using the local account database. To authenticate on a different domain, change the value to the name of the domain.

SFTP Configuration

Secure file access is provided by the Virtual File System (VFS). This section decribes the VFS operations.

Configuring Mounts

All file paths used by the SFTP clients must adhere to the SFTP protocol specification which uses the "/" character as the root of the system. There are no means to access Windows drives using the SFTP protocol, so the VFS provides configurable mounts that enable you to map a mount to either a drive or other system path.

Add the following element to the windows.xml configuration file to configure a mount:

```
<VFSMount
        mount="/sshtools" path="C:\sshtools"
        permissions="rw"/>
```

When accessing through an SFTP client, use the following command to change to the C:\sshtools directory:

```
sftp> cd
        /sshtools
```

Permissions

The permissions attribute specifies the default attributes for the mount. The VFS is pure JAVA and therefore cannot determine the exact file permissions. You set the default permissions using this attribute and if you require different permissions per user, then add a nested VFSPermission element as follows:

```
<VFSMount
        mount="/sshtools" path="C:\sshtools"
        permissions="r"> <VFSPermission
        name="administrator" permissions="rwx"/>
        <VFSPermission name="lee" permissions="rw"/>
        </VFSMount>
```

In future versions we will be providing a Windows File System implementation that will be able to access the windows permissions directly.

Root Mount

You can also configure the root mount, that is the path mapped to "/". The VFS will use this mount as a last resort should non of the configured mounts be correct for the path. Configure the root mount using the following element:

```
<VFSRoot
        path="C:\" permissions="rw"/>
```

User's Home Directory

The VFS provides a mount to the users home directory. The mount is always prefixed by /home/ followed by the user's name.