CHAPTER 3

Configuring the Secure Shell

Configuration is the the technical implementation of the local security policy. When setting the policy, management decided the level of protection needed for machines and data. Now you must implement their decisions when configuring Secure Shell. Secure Shell has a variety of options, some of which may not be appropriate to your local situation. Configure according to your policy. Again, if you do not have a security policy, it is important to establish one.

In configuring Secure Shell, keep in mind two principles:

- Defense-in-depth

  Let no single point of configuration or defense be the only gatekeeper for security.

- Plan on failure

  Secure Shell can, and should, be configured at multiple points (build-time, server configuration, and client configuration). No single misconfiguration should completely break the system security.

Example client and server configurations can be found in “Scripts and Configuration Files” on page 159. Consult the appendixes on server and client configuration options for information on individual options. Also refer to vendor documentation because the appendixes are not all encompassing. OpenSSH exists in a particularly fluid state with new options occasionally appearing.

Configuration Details

In order of precedence, Secure Shell configuration occurs at the following places: the software build-time, the server command-line options, the server configuration file (`sshd_config`), the client command-line options, the user client configuration file

(`~/.ssh/config`), and the global client configuration file (`ssh_config`). Build-time configuration is the strongest. It cannot be changed without rebuilding the software. This makes it inconvenient if a change is needed.

The server configuration involves the following: how the `sshd`(1M) daemon will present itself on the network, what protocols and authentication methods are acceptable, and how the user environment is constructed. The client configuration involves the following: determining which server to transact with which protocol, verifying the server identity, determining the user identity presentation, and choosing the ease-of-use features. Policy details are implemented on the server side. The client cannot override or provide a feature that the server does not offer.

The available features can be enabled or disabled by either command-line options or the applicable configuration file. Command-line options apply to a particular instantiation of either the server or client. Configuration file options are persistent until the file is altered and a new instantiation started. The most reliable configuration method uses the configuration file. This gives a repeatable, reproducible invocation. Changes can also be tracked by using source control. For information on command-line options, consult the vendor documentation.

Mechanics of Configuration Files

When OpenSSH is built, `sshd_config` and `ssh_config` are placed at the location specified by `sysconfdir` . Usual locations are `/etc`, `/usr/local/etc`, `/etc/ssh`, or `/etc/openssh`. The Solaris Secure Shell software stores the two files at `/etc/ssh`. These files should be owned by user `root` and group `sys`. The file permission mode should be either 644 or 444.

Configuration files contain two types of entries: comments and keyword-value pairs. Comments are blank lines and lines beginning with the hash mark (#). Keyword-value pairs consist of an identifier (keyword), a space, and the value associated with the identifier. Keywords are case insensitive, where as values are case sensitive.

Traditionally, the first letter of each word in a keyword is capitalized for readability. Some values are lists that are either comma delimited or space delimited, depending on the keyword. Consider keeping configuration files under source control to track revisions. The source control tags can be hidden by the comment character (the hash mark).

```
# Example config file - two comments and one
# keyword-value pair
Port 22
```

Recommendations

During configuration, you will need to make trade-offs between security, ease-of-use, and legacy compatibility. A wide variety of options covering network and protocol support, authentication, and user environment, obscure the individual option's impact on the whole. This section includes some configuration recommendations and discusses the consequences of their usage.

Note – Only the Solaris Secure Shell software and OpenSSH versions that are current at the time of this writing are used. Not all of the options are covered. Consult the vendor documentation for information on the other options and on the options presented here.

Server Recommendations

Server configuration specifies how the daemon presents itself on the network, what protocols are offered, and what authentication methods are allowed. Specific recommendations are given for each topic. Recommendations specific to a particular Secure Shell implementation have also been noted.

Protocol Support

Two major versions of the Secure Shell protocol exist. Protocol 1 has been deprecated because of vulnerabilities, such as packet insertion and password-length determination. Whenever possible, use Protocol 2. Unfortunately, many legacy clients support only Protocol 1. If this protocol must be enabled, consult the Legacy Support recommendations later in this chapter. Consider migrating to clients that support Protocol 2 as soon as reasonably possible.

Network Access

By default, the `sshd`(1M) daemon listens on all network interfaces on its bound ports. For workstations or other systems on which accessibility is desired for all interfaces, this behavior is not a problem. For architectures such as the Service

Delivery Network, in which management traffic is limited to a particular interface, this behavior is a problem. Limit network access with the `ListenAddress` keyword. Access is limited by a particular IP address, not by a network interface.

```
# Listen only to the management network.
ListenAddress 192.168.0.10
```

To further narrow down what the daemon will listen to, use either a host-based firewall, such as the SunScreen™ software, or TCP Wrappers.

For information about traffic-limited architectures, consult the Sun BluePrints OnLine article “Building Secure N-Tier Environments” (October 2000).

Keep-Alives

Occasionally, connections are temporarily suspended when a route is downed, a machine crashes, a connection is hijacked, or a man-in-the-middle attack is attempted. TCP keep-alives should be sent to detect any of these cases. If TCP keep-alives fail, the server will disconnect the connection and return allocated resources. Regular disconnects can aggravate users on faulty networks.

```
KeepAlive yes
```

Data Compression

Optionally, compression can be used on the encrypted data streams. This use results in bandwidth savings for compressible data, such as interactive logins or log files, at the expense of more CPU resources. For uncompressible data such as encrypted or compressed files, the extra CPU time is wasted and decreases performance. For a single Secure Shell session, these losses are inconsequential. For a file server, the extra load could impact performance. In this case, turn compression off to prevent misconfigured clients from driving up the system load.

```
# Transferring ASCII data such as interactive logins or log files
Compression yes
```

```
# Transferring random data such as compressed or encrypted files
# Prevents performance issues and reduces CPU load
Compression no
```

Privilege Separation

Privilege separation is an OpenSSH-only feature. The `sshd`(1M) daemon is split into two parts: a privileged process to deal with authentication and process creation and an unprivileged process to deal with incoming network connections. After successful authentication, the privileged process spawns a new process with the privileges of the authenticated user. The goal is to prevent compromise from an error in the network facing process. Unfortunately, privilege separation is not really compatible with pluggable authentication modules or SunSHIELD Basic Security Module (BSM) auditing. Some OpenSSH features are also disabled. If privilege separation is desired, consult the vendor documentation.

Note – The compilation options presented in Chapter 2 disable privilege separation.

```
# OpenSSH only
UsePrivilegeSeparation no
```

Login Grace Time

The default login grace time is the time a connection is allowed to exist before being successfully authenticated. The default of 600 seconds for the Solaris Secure Shell software and 120 seconds for later OpenSSH versions is too long. Reduce the time to 60 seconds.

```
LoginGraceTime 60
```

Password and Public Key Authentication

Passwords are not always appropriate. A stronger means may be required, such as public-key cryptographic two-factor authentication. Secure Shell refers to the key pair as an identity. See “Managing Keys and Identities” on page 71 for more details. When passwords are deemed sufficient, do not allow empty passwords. They are too easy to guess.

```
PasswordAuthentication yes
PermitEmptyPasswords no
PubKeyAuthentication yes
DSAAuthentication yes
```

Superuser (root) Logins

Neither the Solaris Secure Shell software nor OpenSSH honors the values set in the /etc/default/login file. To prevent network superuser (root) logins, they must be explicitly denied. The Solaris Secure Shell software and OpenSSH default to yes. However, the default sshd_config(4) file in the Solaris Secure Shell software disables this feature. This forces a system administrator to log in as an unprivileged user, then change users (su) to the superuser. Enable superuser logins only if the system has no user accounts and the appropriate host protection is in place.

```
PermitRootLogin no
```

Banners, Mail, and Message-of-the-Day

Some sites require that a banner be displayed after a user connects to a system, but before logging in. You can accomplished this with the Banner keyword. Set Banner to /etc/issue so that only one banner file exists for the entire system.

```
Banner /etc/issue
```

In the Solaris OE, the interactive login shell is expected to display the message-of-the-day (MOTD) and to check for new mail. With some versions of OpenSSH, this feature causes the MOTD display and mail check to be done twice. Set these keywords to no to eliminate the duplication.

```
CheckMail no
PrintMotd no
```

Connection and X11 Forwarding

Secure Shell can tunnel TCP and X protocol connections through encrypted connections established between the client and server. Tunneling the traffic is referred to as forwarding. The forwarding occurs at the application level and is not completely transparent to the applications being forwarded. The applications need some configuration to use the tunnel.

Note – Data is protected only while it is in the tunnel between the client and server. After that, it is normal network traffic in the clear.

Tunneled traffic bypasses firewalls and intrusion detection systems. Allowing connection (TCP port) forwarding allows remote users safer access to email or the corporate web server. X forwarding allows system administrators to run GUI applications remotely, such as the Solaris™ Management Console software. This may not be functionality you want your users setting up. You can inconvenience users by turning off the functionality, but as long as they have shell access, they can run their own forwarders. Use role-based access control to explicitly limit what you want your users to do in this case.

If port forwarding is enabled, disable `GatewayPorts` and notify your users. `GatewayPorts` allows machines, other than the client, to access the forwarded ports in the tunnel. This access effectively bypasses any firewall usage. Again, users could run their own private forwarders on their client machines to defeat the server restrictions. Consider placing an intrusion detection sensor on the private network side of a Secure Shell bastion host to detect problem traffic.

```
AllowTCPForwarding yes
GatewayPorts no
X11DisplayOffset 10
X11Forwarding yes
XAuthLocation /usr/X/bin/xauth
```

User Access Control Lists

User access control lists (ACLs) can be specified in the server configuration file. No other part of the Solaris OE honors this ACL. You can either specifically allow or deny individual users or groups. The default is to allow access to anyone with a valid account. You can use ACLs to limit access to particular users in NIS environments, without resorting to custom pluggable authentication modules. Use only one of the following four ACL keywords in the server configuration file: `AllowGroups`, `AllowUsers`, `DenyGroups`, or `DenyUsers`.

```
# Allow only the sysadmin staff
AllowGroups staff
```

```
# Prevent unauthorized users.
DenyUsers cheng atkinson
```

User File Permissions

If a user has left their home directory or `.ssh` files world writable either by accident or by trickery, an intruder could insert identities allowing password-free access or alter the `known_hosts` file to allow man-in-the-middle attacks. With `StrictModes` enabled, the `sshd`(1M) daemon will not allow a login. But, users can be easily confused because they will not know why they cannot log in. No different error message is presented to them.

```
StrictModes yes
```

If you decide to disable `StrictModes`, consider eliminating public-key-based authentication to prevent user account compromise. The consequence is the elimination of password-free logins for users or automated jobs.

`UseLogin` Keyword

For OpenSSH only, `UseLogin` specifies that the OpenSSH `sshd`(1M) call `login`(1) instead of performing the initial login tasks itself for interactive sessions. `login`(1) must be called for BSM auditing. Unless `UseLogin` is set to `yes`, `cron`(1M) will partially break if SunSHIELD BSM auditing is enabled. See “Auditing” on page 81 for details on the consequences of `UseLogin` and on getting BSM auditing to work successfully. `UseLogin` will not work if `UsePrivilegeSeparation` is enabled. Enabling `UseLogin` disables X11 and port forwarding.

```
UseLogin no
```

Legacy Support

If legacy clients must be supported, strengthen the default configuration as much as possible. Default to Protocol 2 for the clients that support it. Disable all of the `rhosts`-style authentication methods. Increase the server key size and decrease the ephemeral key regeneration interval to minimize offline factoring attacks against the keys.

```
# Enable protocol 1 but default to protocol 2.
Protocol 2,1
# Legacy support options - protocol 1 only
HostKey /etc/ssh/ssh_host_key
IgnoreRhosts yes
IgnoreUserKnownHosts yes
KeyRegenerationInterval 1800
RhostsAuthentication no
RhostsRSAAuthentication no
RSAAuthentication yes
ServerKeyBits 1024
```

Client Recommendations

Client configuration specifies host option assignment, data compression, keep-alives, protocol support, and identity management. Specific recommendations are given for each topic.

Host Option Assignment

Configuration options can be assigned to a specific host or to all hosts by using the `Host` keyword. The value is matched to what the user types on the command line, not to the actual host name of the server. An asterisk (*) is used to set global default options. Options assigned to a specific host have precedence over the global defaults.

```
# Only for a specific host
Host legacy
Protocol 1

# For all hosts
Host *
Protocol 2
```

Data Compression

Data compression can be used on the encrypted data stream to save bandwidth. Set to `off` by default, you should enable it for interactive sessions or for transferring easily compressible data. The compression cost is asymmetric in that compressing the data is more computationally expensive than decompression. Client-side CPU cycles are generally cheaper than server-side CPU cycles. Avoid attempting to compress already compressed or encrypted data to avoid needlessly raising the CPU load on the server.

```
# For interactive sessions, low bandwidth links, or easily
# compressable files
Compression yes
```

Keep-Alives

Enable TCP keep-alives to detect downed connections. See “Keep-Alives” on page 44 for the server recommendations.

```
KeepAlive yes
```

Protocol Support

Always use Protocol 2 when possible. See “Protocol Support” on page 43 for the server recommendation.

```
Protocol 2
```

`rlogin` and `rsh`

The `rlogin` and `rsh` protocols should not be used. Prevent the client from attempting to execute `rsh` if a Secure Shell connection is refused.

```
FallBackToRsh no
UseRsh no
```

Server Identity

Verify server identity both by its host key and IP address. For higher levels of identity assurance, set `StrictHostKeyChecking` to `yes` and distribute host keys out-of-band. This is impractical when users frequently encounter new hosts. Set `StrictHostKeyChecking` to `ask`, and train the users to verify the offered host key with the stored host key on the server. See “Managing Keys and Identities” on page 71 for more information.

```
CheckHostIP yes
# only access one host
StrictHostKeyChecking yes
```

```
CheckHostIP yes
# access a variety of hosts
StrictHostKeyChecking ask
```

User Identity

User identities are stronger and provide more flexibility than does password authentication. When user identities are combined with agents, password-free logins can safely be obtained if the server permits it. See “Integrating Secure Shell” on page 59 and “Managing Keys and Identities” on page 71 for details.

```
DSAAuthentication yes
PubkeyAuthentication yes
```