Managing Secure Shell Access in Oracle® Solaris 11.4

Part No: E61025
June 2019

Managing Secure Shell Access in Oracle Solaris 11.4

Part No: E61025

Access to Oracle Support

Contents

Using This Documentation

Describes how to administer and use the Secure Shell feature for secure remote access.

- **Overview** – Describes how to administer and use the OpenSSH implementation of Secure Shell on Oracle Solaris systems for secure remote access.
- **Audience** – System administrators who must implement security on the enterprise.
- **Required knowledge** – Familiarity with security concepts and terminology.

Product Documentation Library

Documentation and resources for this product and related products are available at `http://www.oracle.com/pls/topic/lookup?ctx=E37838-01`.

Feedback

Provide feedback about this documentation at `http://www.oracle.com/goto/docfeedback`.

♦ ♦ ♦ C H A P T E R 1

Using Secure Shell

The Secure Shell feature of Oracle Solaris provides secure access to a remote host over an unsecured network. The shell provides commands for remote login, remote window display, and remote file transfer. This chapter covers the following topics:

- "What's New in Secure Shell in Oracle Solaris 11.4" on page 9
- "About Secure Shell" on page 10
- "Secure Shell and Remote Public Keys" on page 13
- "Secure Shell and FIPS 140-2" on page 14
- "Configuring Secure Shell" on page 19
- "Using Secure Shell" on page 26

What's New in Secure Shell in Oracle Solaris 11.4

For existing customers, this section highlights the key changes in this release.

OpenSSH is the sole implementation of Secure Shell in Oracle Solaris 11.4. The `sunssh` implementation of Secure Shell is not supported. The Oracle Solaris implementation of OpenSSH remains compatible with the OpenSSH project.

- For information about the OpenSSH project, see OpenSSH Project (`http://www.openssh.com/`).
- For a summary of differences between the Oracle Solaris implementation of OpenSSH and the OpenSSH project, see "Oracle Solaris Modifications to OpenSSH" on page 10.

Note - Use the `man` command to view the Oracle Solaris versions of OpenSSH man pages. The OpenSSH project man pages might not reflect Oracle Solaris modifications.

- For a summary of differences between SunSSH and OpenSSH, see "Transition From SunSSH to OpenSSH" on page 11.

The OpenSSH server (`sshd`) can retrieve user account information, including `ssh` public keys, from an external directory,such as OpenLDAP. For more information, see "Secure Shell and Remote Public Keys" on page 13.

About Secure Shell

Secure Shell is the active remote access protocol on a newly installed Oracle Solaris system.

In Secure Shell, authentication is provided by the use of passwords, public keys, or both. All network traffic is encrypted. Thus, Secure Shell prevents a would-be intruder from being able to read an intercepted communication. Secure Shell also prevents an adversary from spoofing the system.

Secure Shell can also be used as an on-demand virtual private network (VPN). A VPN can forward X Window system traffic or can connect individual port numbers between the local systems and remote systems over an encrypted network link.

OpenSSH Implementation of Secure Shell

In this Oracle Solaris release, `openssh` is the only available implementation of Secure Shell. OpenSSH in Oracle Solaris is built on the latest version of the OpenSSH project, plus additions that are particular to the Oracle Solaris environment. The `/network/ssh:default` SMF service runs the OpenSSH implementation of Secure Shell.

Oracle Solaris Modifications to OpenSSH

To see which version of the OpenSSH project is the basis for OpenSSH on your system, run the `pkg info openssh` command. Oracle Solaris engineers have modified the man pages to include Oracle Solaris changes, so use the `man` command to view Secure Shell man pages in Oracle Solaris. Do not use the man pages from the OpenSSH project.

Oracle Solaris differences from the OpenSSH project include the following:

- Secure Shell protocol version 1 (v1) is not supported on the client.

  If you have network entities that support only v1, such as old network routers, you can no longer connect to such devices. However, Oracle Solaris legacy releases, such as Oracle Solaris 10 and Oracle Solaris 11.3 can use SunSSH to access systems that use v1.

- Oracle Solaris removed unsafe algorithms from OpenSSH. To view which ciphers, MACs, and key exchange algorithms are supported, list them with the following commands:

  ```
  $ ssh -Q cipher
  $ ssh -Q mac
  $ ssh -Q kex
  ```

  The `sshd_config`(5) and `ssh_config`(5) man pages list the supported algorithms.

  If you are using or connecting with systems that use deprecated or removed algorithms, see Using OpenSSH with Legacy SSH Implementations.
- Oracle Solaris stores delegated credentials in a default credential cache such as `/tmp/krb5cc_101`. Credentials in a default credential cache can be used for accessing NFS file systems protected by Kerberos. To configure this setting, see "Per-Session GSS-API Credentials" on page 18.

  In contrast, the OpenSSH project stores delegated credentials in a non-default credential cache such as `/tmp/krb5cc_101_WO4082`.

  For more information, see "Trusted Delegated Services in Kerberos" in *Managing Kerberos in Oracle Solaris 11.4*. Also, use the `man` command to view the `sshd_config`(5) and `ssh_config`(5) man pages.
- Oracle Solaris provides an LDAP schema and helper application to enable `ssh` to get a user's public key from an LDAP keystore. For more information, see "Secure Shell and Remote Public Keys" on page 13.

Transition From SunSSH to OpenSSH

For users familiar with SunSSH, note the following differences from OpenSSH.

- OpenSSH contains keywords that SunSSH did not.

  ```
  ## sshd_config file
  ...
  KexAlgorithms
  ```
- Unlike SunSSH, OpenSSH is not compiled with TCP wrappers. To configure OpenSSH to handle the function of TCP wrappers, see "Replacing TCP Wrappers With `sshd_config` Entries" on page 12.
- Unlike SunSSH, OpenSSH supports only the `PATH` variable from a user's login shell. OpenSSH does not use the values from the `/etc/default/login` file, even when the `sshd_config` file does not set the equivalent keywords.
- OpenSSH does not use X.509 public key certificates. To use public key certificates for Secure Shell authentication, see the `CERTIFICATES` section in the `ssh-keygen(1)` man page.

- The default value of the `UseDNS` configuration variable is `No` in OpenSSH.

  Therefore, replace host names with IP addresses in OpenSSH configuration files and commands. Some examples from configuration files include the following:

  ```
  ## sshd_config file
  Match Address 192.0.2.10
  AllowUsers jdoe@192.0.2.10
  AllowGroups staff@192.0.2.10
  DenyUsers jsmith@192.0.2.10

  ## /etc/ssh/shosts.equiv or /.shosts file
  192.0.2.10

  ## ~/.ssh/authorized_keys file
  from="192.0.2.10" ssh-rsa AAAAB3...Q== jdoe@192.0.2.11
  ```

Replacing TCP Wrappers With `sshd_config` Entries

OpenSSH does not support TCP wrappers. You can configure a firewall to replace the `/etc/hosts.allow` and `/etc/hosts.deny` files. For a `sshd_config` file solution, see Example 1, "Using a `Match` Block to Allow Logins From a Subnet Only," on page 12.

Note - The `openssh` implementation of Secure Shell continues to use TCP connections. Only the TCP wrapper function, `libwrap`, is no longer supported.

EXAMPLE 1 Using a `Match` Block to Allow Logins From a Subnet Only

In this example, the SunSSH administrator had configured TCP wrappers to allow logins only from the `192.0.2.0/16` subnet.

```
jdoe-uk:~$ cat /etc/hosts.allow
    sshd : 192.0.
jdoe-uk:~$ cat /etc/hosts.deny
    ALL : ALL
```

In OpenSSH, the following entry in the `sshd_config` file sets an equivalent restriction:

```
Match Address *,!192.0.2.0/16
            MaxAuthTries 0
```

Secure Shell and Remote Public Keys

The Secure Shell server (`sshd`) can retrieve user account information, including `ssh` public keys, from an external directory, such as OpenLDAP. Access is granted if the `ssh` client successfully authenticates against any one of the user's public keys that are stored the LDAP directory.

In Oracle Solaris, the helper application `/usr/lib/ssh/ssh-pubkey-ldap` retrieves the key. The administrator specifies the application in the `/etc/ssh/sshd_config` file and also configures the `sshPublicKey` schema in LDAP to recognize the request.

Secure Shell Keywords for Using Remote Public Keys

The `ssh-pubkey-ldap` application enables the OpenSSH server to query an OpenLDAP server for user account information, such as an `ssh` public key. The addition of `sshPublicKey` schema to the LDAP configuration file, and the addition of two keywords to the `sshd_config` file connect OpenSSH with the LDAP server.

- See "OpenLDAP Configuration for Secure Shell Use of Public Keys" on page 14 for the LDAP schema that enable the connection on the LDAP server.
- The two keywords that enable the connection on the OpenSSH server are:
  - `AuthorizedKeysCommand` – Specifies the absolute path of the application, `/usr/lib/ssh/ssh-pubkey-ldap`.

    The only `AuthorizedKeysCommand` keyword token that `ssh-pubkey-ldap` accepts is `%u`, a username. If the caller does not specify a user, the application uses the username of the target `ssh` client user.
  - `AuthorizedKeysCommandUser` – Specifies `daemon` as the caller of the application. For more information about the `daemon` account, see "Special System Accounts" in *Securing Systems and Attached Devices in Oracle Solaris 11.4*.

`ssh-pubkey-ldap` interacts with the name service configuration as a client of an already existing external LDAP directory server. The application uses standard Oracle Solaris LDAP commands to determine whether to use the `ldap` or `ldaps` protocol and which LDAP server to contact to retrieve the user's `ssh` public key.

This application retrieves the public key in the same format as the keys are stored in public keys files.

Note - By default, no `AuthorizedKeysCommand` is run.

OpenLDAP Configuration for Secure Shell Use of Public Keys

For OpenLDAP and Secure Shell to interact, the following conditions must be in place:

- The LDAP server must have the OpenSSH server as a client.
- The OpenLDAP server's schema must include an OpenSSH public key attribute named `sshPublicKey` and an object for a user's LDAP public key.

For an example extended schema configuration and how to connect OpenLDAP with Secure Shell, see "How to Set Up Secure Shell User Authentication From Public Keys Stored in LDAP" on page 23.

Secure Shell and FIPS 140-2

The `openssh` implementation of Secure Shell is a consumer of the OpenSSL FIPS 140-2 module. If your system must comply with FIPS 140-2 requirements, you need to configure and use the FIPS 140-2 options.

OpenSSH in Oracle Solaris sets FIPS 140-2 mode dynamically. If the OpenSSL library that OpenSSH links with is FIPS 140-2 capable, OpenSSH runs in FIPS 140-2 mode. For information, see "Example of Running in FIPS 140-2 Mode on an Oracle Solaris 11.4 System" in *Using a FIPS 140-2 Enabled System in Oracle Solaris 11.4*. In a change from the SunSSH implementation, the administrator is not required to explicitly enable FIPS 140-2 mode, so the `UseFips140` configuration option does not exist.

To confirm that OpenSSH is running in FIPS 140-2 mode, check for a `FIPS` suffix in the OpenSSH version string. This information is indicated in debug messages, as shown in this example:

```
jdoe:~$ ssh -vvv localhost date 2>&1 | grep -i fips
 OpenSSH_7.7p1, OpenSSL 1.0.2r-fips Mar 2019
 debug3: Running in FIPS mode.
 debug1: /etc/ssh/ssh_config line 33: Deprecated option "usefips140"
 debug1: Local version string SSH-2.0-OpenSSH_7.7 FIPS
 debug1: Remote protocol version 2.0, remote software version OpenSSH_7.7 FIPS
 debug1: match: OpenSSH_7.7 FIPS pat OpenSSH* compat 0x04000000
 debug3: Temporarily unsetting FIPS mode to compute MD5 for GSS-API key exchange method
 names
 debug3: Running in FIPS mode.
```

FIPS 140-2 Approved Algorithms for OpenSSH

When running in FIPS 140-2 mode, only FIPS 140-2 approved ciphers, message authentication codes (MACs), and digests are allowed.

You can use the following commands to list all supported ciphers and MACs:

```
$ ssh -Q cipher
$ ssh -Q mac
```

Ciphers, MACs and digests that are not FIPS 140-2 approved are disabled in FIPS 140-2 mode. Attempts to use non-approved algorithms fail, as shown in this example:

```
jdoe:~$ ssh -c arcfour somehost
 Unknown cipher type 'arcfour'
```

To list OpenSSL ciphers, use the following command:

```
$ openssl ciphers -v
```

FIPS 140-2 Approved Key Formats and Sizes for OpenSSH

Administrators must use key formats and key sizes that are approved for FIPS 140-2.

The following FIPS 140-2 approved key types and key sizes are supported in OpenSSH:

- `ed25519`
- For `ssh-rsa`, the following key sizes are supported:

  ```
  1024
  2048
  3072
  ```

Use of a weak key can result in a failure in FIPS 140-2 mode as shown in the following example:

```
root@S12-91:~# ssh -i rsa512 localhost
```

```
 sign_and_send_pubkey: signing failed: error:04066078:rsa
routines:RSA_EAY_PRIVATE_ENCRYPT:key size too small
```

For more information about FIPS 140-2, review the following:

- *Using a FIPS 140-2 Enabled System in Oracle Solaris 11.4*
- `ssh`(1)
- `sshd`(8)
- `sshd_config`(5) and `ssh_config`(5) man pages, viewed with the `man` command

Sharing `.ssh/config` Files Between Multiple Oracle Solaris Releases

If your home directory is on a network, you can share the `~/.ssh/config` file among multiple systems, even if those systems are running different Oracle Solaris releases or different Secure Shell implementations. However, Secure Shell implementations might not recognize all the configuration options from different Secure Shell implementations. In some cases, Secure Shell implementations might not recognize configuration options from different versions of the same Secure Shell implementation.

When the Secure Shell configuration options cannot be recognized by the different systems on the network, you can modify the `ssh_config` files to ignore unrecognized options, thus enabling use of the shared `~/.ssh/config` file among multiple systems.

`Ignore` Keywords in Secure Shell

Two keywords, `IgnoreUnknown` and `IgnoreIfUnknown`, can be used to ignore Secure Shell configuration keywords that are unrecognized among multiple systems. The `IgnoreUnknown` keyword is available in OpenSSH and the `IgnoreIfUnknown` keyword is available in legacy SunSSH releases.

Both `IgnoreUnknown` and `IgnoreIfUnknown` specify a comma-separated list of `ssh_config` keywords, which, if unknown to the `ssh` program, are ignored by Secure Shell. However, the OpenSSH `IgnoreUnknown` keyword applies only to unknown keywords that follow it in the configuration file, while `IgnoreIfUnknown` applies to the entire configuration file.

The following table identifies the Secure Shell implementations in each Oracle Solaris release and the `Ignore` keywords that are available in each implementation.

TABLE 1 Ignore Keywords in Secure Shell

Release	Secure Shell Implementation	Supported Ignore Keyword
Oracle Solaris 11.4	OpenSSH	IgnoreUnknown
Oracle Solaris 11.3	SunSSH	IgnoreIfUnknown and IgnoreUnknown
Oracle Solaris 11.3	OpenSSH	IgnoreUnknown
Oracle Solaris 11 releases prior to Oracle Solaris 11.3	SunSSH	IgnoreIfUnknown
Oracle Solaris 10 Update 11	SunSSH	IgnoreIfUnknown

The following releases do not support the Ignore keywords and cannot be included as part of a shared Secure Shell configuration over a network.

- Oracle Solaris 9
- Oracle Solaris 10 prior to Update 11
- OpenSSH 6.2 and older OpenSSH versions

Ignoring Secure Shell Keywords to Enable Interoperability

If your ~/.ssh/config file is on a network whose systems run different implementations of Secure Shell, you can enable the Secure Shell configuration keywords to work by adding the IgnoreUnknown and IgnoreIfUnknown keywords to the file.

Note - All systems must be able to use at least one of the Ignore keywords as listed in Table 1, "Ignore Keywords in Secure Shell," on page 17.

EXAMPLE 2 Sharing a Secure Shell Configuration Across Releases That Support Different Keywords

This example shows how to use the HostBasedKeyTypes keyword that was introduced in OpenSSH 6.8. The users are on a network where some systems are running a release of Secure Shell that does not support this keyword.

Add the following entries to the ssh_config file:

```
---
IgnoreUnknown HostBasedKeyTypes,IgnoreIfUnknown
IgnoreIfUnknown HostBasedKeyTypes,IgnoreUnknown
```

```
HostBasedKeyTypes ssh-rsa-cert-v01@openssh.com, ssh-rsa
---
```

You add both the `Ignore` keywords to enable all Secure Shell implementations to communicate. For more information, use the `man` to view the `ssh_config`(5) man page.

GSS-API Authentication in Secure Shell

To use GSS-API authentication in Secure Shell, the server must have GSS-API acceptor credentials and the client must have GSS-API initiator credentials. Support is available for `mech_krb5`.

For `mech_krb5`, the server has GSS-API acceptor credentials when the host principal that corresponds to the server has a valid entry in `/etc/krb5/krb5.keytab`.

The client has initiator credentials for `mech_krb5` if one of the following has been done:

- The `kinit` command has been run.
- The `pam_krb5` module is used in the `pam.conf` file.

See also "Per-Session GSS-API Credentials" on page 18. For more information about mechanisms, see the `mech`(5) man page.

Per-Session GSS-API Credentials

Oracle Solaris supports per-session GSS-API credentials. The `sshd_config` file contains the option, `GSSAPIUseDefaultCCache`. By default, this option is set to `yes`.

The default behavior is the following:

- Delegated credentials are stored in the default credential cache
- `KRB5CCNAME` is not set
- The credentials are not deleted when the session ends

When you change this value to `no`, you create per-session credentials:

- Delegated credentials are stored in a per-session credential cache in `/tmp/krb5cc_uid_`*XXXXXX*

  The file path is created using a random pattern.
- `KRB5CCNAME` points to the cache file path.

- If the `GSSAPICleanupCredentials` variable is set to the default value of `yes`, the credential cache is destroyed upon logout. This variable is in the `sshd_config` file.

For more information, use the `man` command to view the `sshd_config`(5) man page.

Security Considerations in Secure Shell

Consider the following security issues when configuring Secure Shell:

- Protect the `/etc/ssh/ssh_known_hosts` file.

  Each host that needs to communicate securely with another host must have the server's public key stored in the local host's `/etc/ssh/ssh_known_hosts` file. Although a script could be used to update the `/etc/ssh/ssh_known_hosts` files, such a practice is heavily discouraged because a script opens a major security vulnerability.

  The `/etc/ssh/ssh_known_hosts` file should be distributed only by a secure mechanism as follows:

  - Over a secure connection, such as Secure Shell, IPsec, or Kerberized `ftp` from a known and trusted system
  - At system install time

  To avoid the possibility of an intruder gaining access by inserting bogus public keys into a `known_hosts` file, you should use a known and trusted source of the `ssh_known_hosts` file. The `ssh_known_hosts` file can be distributed during installation. Later, scripts that use the `scp` command can be used to copy the latest version.

  Use the `man` command to view the `ssh_config`(5) man page.
- Use the `DisableBanner` keyword with caution.

  Do not disable the default banner message without careful planning in accordance with your company's security policies. For example, you might disable the banner for repeated logins within an internal network, while maintaining the default banner for connections from outside the company, if that is consistent with your company's security policies.

Configuring Secure Shell

Secure Shell is configured at installation. The following tasks are provided for convenience or to highlight security issues.

- "How to Set Up Host-Based Authentication for Secure Shell" on page 20
- "How to Create an Isolated Directory for `sftp` Files" on page 25

Procedures that regular users can perform are in "Using Secure Shell" on page 26.

▼ How to Set Up Host-Based Authentication for Secure Shell

The following procedure sets up a public key system where the client's public key is used for authentication on the Secure Shell server. The user must also create a public/private key pair.

In the procedure, the terms *client* and *local host* refer to the system where a user types the `ssh` command. The terms *server* and *remote host* refer to the system that the client is trying to reach.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.4*.

1. **On the client, enable host-based authentication.**

   In the client configuration file, `/etc/ssh/ssh_config`, add the following entry:

   ```
   HostbasedAuthentication yes
   ```

   For the syntax of the file, use the `man` command to view the `ssh_config`(5) man page.

2. **On the Secure Shell server, enable host-based authentication.**

   In the server configuration file, `/etc/ssh/sshd_config`, add the same entry:

   ```
   HostbasedAuthentication yes
   ```

3. **On the server, either you or the user should configure a file that enables the client to be recognized as a trusted host.**

   For more information, see the `FILES` section of the `sshd(8)` man page.

   - **If you are doing the configuration, add the client as an entry to the server's `/etc/ssh/shosts.equiv` file.**

     client-host

   - **If your users are doing the configuration, they should add an entry for the client to their `~/.shosts` file on the server.**

     client-host

4. **On the server, ensure that the `sshd` daemon can access the list of trusted hosts.**

Set `IgnoreRhosts` to `no` in the `/etc/ssh/sshd_config` file.

```
## sshd_config
IgnoreRhosts no
```

5. **Ensure that users of Secure Shell at your site have accounts on both hosts.**

6. **Put the client's public key on the server by using one of the following methods:**

   - **Modify the `sshd_config` file on the server, then instruct your users to add the client's public host keys to their `~/.ssh/known_hosts` file.**

     ```
     ## sshd_config
     IgnoreUserKnownHosts no
     ```

     For user instructions, see "How to Generate a Public/Private Key Pair for Use With Secure Shell" on page 27.

   - **Copy the client's public key to the server.**

     The host keys are stored in the `/etc/ssh` directory. The keys are typically generated by the `sshd` daemon on first boot.

     a. **Add the key to the `/etc/ssh/ssh_known_hosts` file on the server.**

        On the client, type the following command on one line with no backslash.

        ```
        # cat /etc/ssh/ssh_host_rsa_key.pub | ssh RemoteSys \
        'cat >> /etc/ssh/ssh_known_hosts && echo "Host key copied"'
        ```

Note - If host keys are missing from the server, using Secure Shell generates an error message similar to the following:

```
Client and server could not agree on a key exchange algorithm:
client "diffie-hellman-group-exchange-sha256,diffie-hellman-group-
exchange-sha1,diffie-hellman-group14-sha1,diffie-hellman-group1-sha1",
server "gss-group1-sha1-toWM5Slw5Ew8Mqkay+al2g==". Make sure host keys
are present and accessible by the server process. See sshd_config(5)
description of "HostKey" option.
```

   b. **When you are prompted, supply your login password.**

      When the file is copied, the message "Host key copied" is displayed.

   c. **Prepend *RemoteHost* to the copied entry in the `ssh_known_hosts` file.**

Each line in the `/etc/ssh/ssh_known_hosts` file consists of fields that are separated by spaces:

hostnames algorithm-name publickey comment

Place *RemoteHost* in the *hostnames* field.

```
## /etc/ssh/ssh_known_hosts File
RemoteHost  <copied entry>
```

Example 3 Setting Up Host-based Authentication

In the following example, each host is configured as a server and as a client. A user on either host can initiate an `ssh` connection to the other host. The following configuration makes each host a server and a client:

- On each host, the Secure Shell configuration files contain the following entries:

```
## /etc/ssh/ssh_config
HostbasedAuthentication yes
#
## /etc/ssh/sshd_config
HostbasedAuthentication yes
IgnoreRhosts no
```

- On each host, the `shosts.equiv` file contains an entry for the other host:

```
## /etc/ssh/shosts.equiv on system2
system1

## /etc/ssh/shosts.equiv on system1
system2
```

- The public key for each host is in the `/etc/ssh/ssh_known_hosts` file on the other host:

```
## /etc/ssh/ssh_known_hosts on system2
... system1

## /etc/ssh/ssh_known_hosts on system1
... system2
```

- Users have an account on both hosts. For example, the following information would appear for user Jane Doe:

```
## /etc/passwd on system1
jdoe:x:3111:10:J Doe:/home/jdoe:/bin/sh

## /etc/passwd on system2
jdoe:x:3111:10:J Doe:/home/jdoe:/bin/sh
```

▼ How to Set Up Secure Shell User Authentication From Public Keys Stored in LDAP

The following procedure enables the user's public key to be retrieved from the LDAP server. The user must have a public/private key pair.

Before You Begin You have configured LDAP as the system name service for user accounts. For information, see the `ldapclient`(8) man page.

You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.4*.

1. **Modify the Secure Shell server with two new keywords.**

   Add the `AuthorizedKeysCommand` and `AuthorizedKeysCommandUser` entries to the `/etc/ssh/sshd_config` configuration file. For example:

   ```
   #
    # This is the sshd server system-wide configuration file.  See
    # sshd_config(5) for more information.
    #
    ...
    AuthorizedKeysCommand /usr/lib/ssh/ssh-pubkey-ldap
    AuthorizedKeysCommandUser daemon
   ```

   Note - In configurations where read access to the LDAP server is tightly controlled, you might need to specify `root` rather than `daemon` as the value of `AuthorizedKeysCommandUser`.

   If a key supplied by the `AuthorizedKeysCommand` keyword does not successfully authenticate the user, then public key authentication continues by using the usual `AuthorizedKeysFile` setting or the default keys files. For more information, see the `ssh-pubkey-ldap`(8) man page.

2. **Modify the LDAP server to accept user queries from the Secure Shell server.**

   a. **Make sure that your LDAP configuration file includes entries that enable connections from Secure Shell.**

      Your `/etc/openldap/ldap.conf` file must point to the appropriate LDAP servers. Also, your OpenSSH server must be a client of those LDAP systems. For example:

      ```
      #
      # LDAP Defaults
      #
      ...
      ```

```
BASE dc=example,dc=com
URI ldap://ldap.example.com
```

b. **Add the following `sshPublicKey` entries to the `/etc/openldap/slapd.conf` file.**

```
------------------------------------- # octetString SYNTAX
 attributetype (
  1.3.6.1.4.1.24552.500.1.1.1.13 NAME 'sshPublicKey' DESC
  'MANDATORY: OpenSSH Public key' EQUALITY
  octetStringMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.40 )

 # printableString SYNTAX yes|no
 objectclass (
  1.3.6.1.4.1.24552.500.1.1.2.0
  NAME 'ldapPublicKey'
  SUP top
  AUXILIARY
  DESC 'MANDATORY: OpenSSH LPK objectclass'
  MUST uid
  MAY sshPublicKey )
```

`sshPublicKey` is the name of the default `ssh` public key attribute name for the LDAP schema.

Tip - If your OpenLDAP server already has an LDAP public key object and an attribute name is defined, then create an attribute alias from its name to `sshPublicKey`.

---

c. **Restart `slapd` to load the changes.**

```
$ svcadm restart slapd
```

3. **Add users' public keys to their LDAP attributes.**

For example, add `jdoe`'s Secure Shell public key to the LDAP server:

a. **Create an LDIF file called `jdoe_publickey.ldif` with the following content:**

```
dn: cn=jdoe,ou=People,dc=oracle,dc=com
changeType: modify
add: objectClass
objectClass: ldapPublicKey
-
add: sshPublicKey
ssh-rsa AAAAB3NKey jdoe@example.com
```

b. **Add the LDIF file to the LDAP server configuration.**

Use the LDAP protocol that connects you to your OpenLDAP server, in this instance, `ldap://`:

```
$ ldapmodify -a -x -D "cn=admin,dc=example,dc=com" -w <password> -H \
           ldap:// -f jdoe_publickey.ldif
```

4. **(Optional) Test that `jdoe` can successfully authenticate to Secure Shell.**

   a. **On the user's system, ensure that the public key for LDAP is not present in the user's `authorized_keys` file.**

   ```
   # mv /home/jdoe/.ssh/authorized_keys /home/jdoe/.ssh/dontuse_authorized_keys
   ```

   b. **Test `jdoe`'s Secure Shell connection.**

   ```
   jdoe$ ssh
   ```

▼ How to Create an Isolated Directory for `sftp` Files

This procedure configures an `sftponly` directory that you create specifically for `sftp` transfers. Users cannot see any files or directories outside this directory.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.4*.

1. **On the Secure Shell server, create the isolated directory as a `chroot` environment.**

```
# groupadd sftp
# useradd -m -G sftp -s /bin/false sftponly
# chown root:root /export/home/sftponly
# mkdir /export/home/sftponly/WWW
# chown sftponly:staff /export/home/sftponly/WWW
```

In this configuration, `/export/home/sftonly` is the `chroot` directory that only the `root` account has access to. Users have write permission to the `sftponly/WWW` subdirectory.

2. **Still on the server, configure a `Match` block for the `sftp` group.**

In the `/etc/ssh/sshd_config` file, locate the `sftp subsystem` entry and modify the file as follows:

```
# pfedit /etc/ssh/sshd_config
...
```

```
## Match Group for Subsystem
## At end of file, to follow all global options
Match Group sftp
ChrootDirectory %h
ForceCommand internal-sftp
AllowTcpForwarding no
```

You can use the following variables to specify the `chroot` path:

- `%h` – Specifies the home directory.
- `%u` – Specifies the username of the authenticated user.
- `%%` – Escapes the `%` sign.

3. **On the client, verify that the configuration works correctly.**

   The files in your `chroot` environment might be different.

```
root@client:~# ssh sftponly@server
This service allows sftp connections only.
Connection to server closed.       No shell access, sftp is enforced.
root@client:~# sftp sftponly@server
sftp> pwd      sftp access granted
Remote working directory: /      chroot directory looks like root directory
sftp> ls
WWW             local.cshrc     local.login     local.profile
sftp> get local.cshrc
Fetching /local.cshrc to local.cshrc
/local.cshrc    100%  166     0.2KB/s   00:00     user can read contents
sftp> put /etc/motd
Uploading /etc/motd to /motd
Couldn't get handle: Permission denied     user cannot write to / directory
sftp> cd WWW
sftp> put /etc/motd
Uploading /etc/motd to /WWW/motd
/etc/motd     100%  118     0.1KB/s   00:00     user can write to WWW directory
sftp> ls -l
-rw-r--r--    1 101  10    118 Jul 20 09:07 motd     successful transfer
sftp>
```

Using Secure Shell

The following user tasks are provided for convenience or to highlight security issues.

- "How to Generate a Public/Private Key Pair for Use With Secure Shell" on page 27

- “How to Remotely Administer ZFS With Secure Shell” on page 29
- “How to Set Up Default Secure Shell Connections to Hosts Outside a Firewall” on page 31

▼ How to Generate a Public/Private Key Pair for Use With Secure Shell

Users must generate a public/private key pair when their site implements host-based authentication or user public-key authentication. For additional options, see the `ssh-keygen(1)` man page.

Before You Begin Ask your system administrator whether host-based authentication is configured.

1. **Start the key generation program.**

   ```
   mySystem$ ssh-keygen -t rsa
   Generating public/private rsa key pair.
   ...
   ```

   where `-t` is the type of algorithm, either `rsa`, `dsa`, or `rsa1`.

2. **Specify the path to the file that will hold the key.**

   By default, the file name `id_rsa`, which represents an RSA v2 key, appears in parentheses. You can select this file by pressing the Return key or provide an alternative file name.

   ```
   Enter file in which to save the key (/home/username/.ssh/id_rsa):     <Press Return>
   ```

   The file name of the public key is created automatically by appending the string `.pub` to the name of the private key file.

3. **Type a passphrase for using your key.**

   This passphrase is used for encrypting your private key. A null entry is *strongly discouraged*. Note that the passphrase is not displayed when you type it in.

   ```
   Enter passphrase (empty for no passphrase): passphrase
   ```

4. **Retype the passphrase to confirm it.**

   ```
   Enter same passphrase again: passphrase
   Your identification has been saved in /home/username/.ssh/id_rsa
   Your public key has been saved in /home/username/.ssh/id_rsa.pub
   The key fingerprint is:
   ```

```
0e:fb:3d:57:71:73:bf:58:b8:eb:f3:a3:aa:df:e0:d1 username@mySystem
```

5. **Check that the path to the key file is correct.**

```
$ ls ~/.ssh
id_rsa
id_rsa.pub
```

At this point, you have created a public/private key pair.

6. **Log in to the remote host.**

Choose one of the following login steps based on your site's authentication method.

- **For host-based authentication, copy the local host's public key to the remote host.**

  a. **Type the following command on one line with no backslash.**

  ```
  $ cat /etc/ssh/ssh_host_rsa_key.pub | ssh RemoteSys \
  'cat >> ~/.ssh/known_hosts && echo "Host key copied"'
  ```

  b. **When you are prompted, supply your login password.**

  ```
  Enter password: password
  Host key copied
  $
  ```

  You can now log in to the remote host.

- **For user authentication with public keys, populate your `authorized_keys` file on the remote host.**

  a. **Copy your public key to the remote host.**

  Type the following command on one line with no backslash.

  ```
  mySystem$ cat ~/.ssh/id_rsa.pub | ssh myRemoteSys \
  'cat >> ~/.ssh/authorized_keys && echo "Key copied"'
  ```

  b. **When you are prompted, supply your login password.**

  ```
  Enter password: password
  Key copied
  mySystem$
  ```

7. **(Optional) Avoid future prompting for passphrases.**

See the ssh-agent(1) and ssh-add(1) man pages.

▼ How to Remotely Administer ZFS With Secure Shell

By default, the root role cannot log in remotely with Secure Shell. Historically, root has used Secure Shell for important tasks, such as sending ZFS pool data to storage on a remote system. In this procedure, the root role creates a user who can act as a remote ZFS administrator.

Before You Begin You must assume the root role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.4*.

1. **Create the user on both systems.**

   For example, create the zfsroot user and provide a password.

   ```
   source # useradd -c "Remote ZFS Administrator" -u 1201 -d /home/zfsroot zfsroot
   source # passwd zfsroot
   New Password: password
   Re-enter new password: password
   passwd: password successfully changed for zfsroot
   #

   dest # useradd -c "Remote ZFS Administrator" -u 1201 -d /home/zfsroot zfsroot
   dest # passwd zfsroot
   ...
   ```

   The zfsroot user must be identically defined on both systems.

2. **On both systems, assign the ZFS File Management rights profile to zfsroot.**

   ```
   source # usermod -P +'ZFS File System Management' -S files zfsroot
   dest # usermod -P +'ZFS File System Management' -S files zfsroot
   ```

3. **Verify that zfsroot on the destination system is assigned the rights profile.**

   ```
   dest # profiles zfsroot
   zfsroot:
   ZFS File System Management
   Basic Solaris User
   All
   ```

4. **Create the user's key pair for Secure Shell authentication.**

The key pair is created on the source system. Then, the public key is copied to the `zfsroot` user on the destination system.

a. **Generate the key pair and put it in the file `id_migrate`.**

```
# ssh-keygen -t rsa -P "" -f ~/id_migrate
Generating public/private rsa key pair.
Your identification has been saved in /root/id_migrate.
Your public key has been saved in /root/id_migrate.pub.
The key fingerprint is:
SHA256:BLNj0v9...izsQ cpltester@Local
The key's randomart image is:
+---[RSA 2048]----+
|      o       .=B|
...
```

b. **Send the public part of the key pair to the destination system.**

```
# scp ~/id_migrate.pub zfsroot@dest:
The authenticity of host 'dest (192.0.2.126)' can't be established.
RSA key fingerprint is 44:37:ab:4e:b7:2f:2f:b8:5f:98:9d:e9:ed:6d:46:80.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'dest,192.0.2.126' (RSA) to the list of known hosts.
Password:
id_migrate.pub 100% |*****************************| 399 00:00
```

5. **On the destination system, move the public part of the key pair to the private `/home/zfsroot/.ssh` directory.**

```
root@dest # su - zfsroot
Oracle Corporation      SunOS 5.11      11.1    May 2012
zfsroot@dest $ mkdir -m 700 .ssh
zfsroot@dest $ cat id_migrate.pub >> .ssh/authorized_keys
```

6. **Verify that the configuration works.**

```
root@source# ssh -l zfsroot -i ~/id_migrate dest \
pfexec /usr/sbin/zfs snapshot zones@test
root@source# ssh -l zfsroot -i ~/id_migrate dest \
pfexec /usr/sbin/zfs destroy zones@test
```

7. **(Optional) Verify that you can create a snapshot and replicate the data.**

```
root@source# zfs snapshot -r rpool/zones@migrate-all
root@source#  zfs send -rc rpool/zones@migrate-all | \
ssh -l zfsroot -i ~/id_migrate dest pfexec /usr/sbin/zfs recv -F zones
```

8. **(Optional) Remove the ability to use the `zfsroot` account for ZFS administration.**

```
root@dest# usermod -P -'ZFS File System Management' zfsroot
root@dest#  su - zfsroot
zfsroot@dest#  cp .ssh/authorized_keys .ssh/authorized_keys.bak
zfsroot@dest#  grep -v root@source .ssh/authorized_keys.bak> .ssh/authorized_keys
```

▼ How to Set Up Default Secure Shell Connections to Hosts Outside a Firewall

You can use Secure Shell to make a connection from a host inside a firewall to a host outside the firewall. This task is done by specifying a proxy command for `ssh` either in a configuration file or as an option on the command line. For the command-line option, see Example 4, “Connecting to Hosts Outside a Firewall From the Secure Shell Command Line,” on page 32.

You can customize your `ssh` interactions through your own personal configuration file, `~/.ssh/config`, or you can use the settings in the administrative configuration file, `/etc/ssh/ssh_config`.

The files can be customized with two types of proxy commands. One proxy command is for HTTP connections. The other proxy command is for SOCKS5 connections. For more information, use the `man` command to view the `ssh_config`(5) man page.

1. **Specify the proxy commands and hosts in a configuration file.**

   Use the following syntax to add as many lines as you need:

```
[Host outside-host]
ProxyCommand proxy-command [-h proxy-server] \
[-p proxy-port] outside-host|%h outside-port|%p
```

`Host` *outside-host*

Limits the proxy command specification to occasions when a remote host name is specified on the command line. If you use a wildcard for *outside-host*, you apply the proxy command specification to a set of hosts.

proxy-command

Specifies the proxy command.

The command can be either of the following:

- `/usr/lib/ssh/ssh-http-proxy-connect` for HTTP connections

- `/usr/lib/ssh/ssh-socks5-proxy-connect` for SOCKS5 connections

`-h` *proxy-server* and `-p` *proxy-port*

These options specify a proxy server and a proxy port, respectively. If present, the proxies override any environment variables that specify proxy servers and proxy ports, such as `HTTPPROXY`, `HTTPPROXYPORT`, `SOCKS5_PORT`, `SOCKS5_SERVER`, and `http_proxy`. The `http_proxy` variable specifies a URL. If the options are not used, then the relevant environment variables must be set. For more information, see the `ssh-socks5-proxy-connect(1)` and `ssh-http-proxy-connect(1)` man pages.

outside-host

Designates a specific host to connect to. Use the `%h` substitution argument to specify the host on the command line.

outside-port

Designates a specific port to connect to. Use the `%p` substitution argument to specify the port on the command line. By specifying `%h` and `%p` without using the `Host` *outside-host* option, the proxy command is applied to the host argument whenever the `ssh` command is invoked.

2. **Run Secure Shell, specifying the outside host.**

For example:

```
mySystem$ ssh myOutsideHost
```

This command looks for a proxy command specification for `myOutsideHost` in your personal configuration file. If the specification is not found, then the command looks in the system-wide configuration file, `/etc/ssh/ssh_config`. The proxy command is substituted for the `ssh` command.

Example 4 Connecting to Hosts Outside a Firewall From the Secure Shell Command Line

"How to Set Up Default Secure Shell Connections to Hosts Outside a Firewall" on page 31 explains how to specify a proxy command in a configuration file. In this example, a proxy command is specified on the `ssh` command line.

```
$ ssh -o'Proxycommand=/usr/lib/ssh/ssh-http-proxy-connect \
-h myProxyServer -p 8080 myOutsideHost 22' myOutsideHost
```

The `-o` option to the `ssh` command provides a command-line method of specifying a proxy command. This example command does the following:

- Substitutes the HTTP proxy command for `ssh`

- Uses port `8080` and `myProxyServer` as the proxy server
- Connects to port `22` on `myOutsideHost`

Index