Oracle Linux

Connecting to Remote Systems With OpenSSH

F22963-11
July 2023

Oracle Linux Connecting to Remote Systems With OpenSSH,

F22963-11

Contents

5 Using OpenSSH Client Utilities

Preface

Oracle Linux: Connecting to Remote Systems With OpenSSH describes how to configure the OpenSSH feature and use it to connect to remote systems.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

For information about the accessibility of the Oracle Help Center, see the Oracle Accessibility Conformance Report at https://www.oracle.com/corporate/accessibility/templates/t2-11535.html.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry

standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1 About OpenSSH

OpenSSH secures communications between networked systems.

> **Note:**
>
> This document includes content that was tested against Oracle Linux 8 and Oracle Linux 9, but generally applies to most Oracle Linux releases, and might also apply to other distributions.

OpenSSH is suite of network connectivity tools that provides secure communications between systems. The tools include:

- **`scp`** - Secure file copying. (Deprecated in Oracle Linux 9)
- **`sftp`** - Secure File Transfer Protocol (FTP).
- **`ssh`** - Secure shell to log on to or run a command on a remote system.
- **`sshd`** - Daemon that supports the OpenSSH services.
- **`ssh-keygen`** - Creates RSA authentication keys.

> **Note:**
>
> The Digital Signature Algorithm (DSA) is considered deprecated. As such, authentication mechanisms that depend on DSA keys don't work in the default configuration for Oracle Linux 8 or later. Note also that OpenSSH clients don't accept DSA host keys, even at the LEGACY system-wide cryptographic policy level on Oracle Linux 8 or later.

Unlike utilities such as `rcp`, `ftp`, `telnet`, `rsh`, and `rlogin`, OpenSSH tools encrypt all network packets between the client and server, including password authentication.

To use the OpenSSH tools, a user must have an account on both the client and server systems. You do not need to configure these accounts identically on each system. OpenSSH supports the SSH version 2 (SSH2) protocol. You can use any SSH2 client to access an OpenSSH server and equally you can use the OpenSSH client to access any SSH2 server.

OpenSSH also provides a secure way of using graphical applications over a network by using X11 forwarding. You can also use port forwarding as another way to secure otherwise insecure TCP/IP protocols.

2
Configuring OpenSSH Server

To set up the SSH server, install the `openssh` and `openssh-server` packages and enable the `sshd` service. Then, you can edit settings within the configuration files found in the `/etc/ssh` directory.

Installing OpenSSH Server and Enabling sshd

A default Oracle Linux installation includes the `openssh` and `openssh-server` packages, but the `sshd` service isn't enabled by default.

1. If the packages aren't installed, run the following command:

   ```
   sudo dnf install openssh openssh-server
   ```

2. Start the `sshd` service and configure it to start following a system reboot:

   ```
   sudo systemctl start sshd
   sudo systemctl enable sshd
   ```

You can set `sshd` configuration options for features such as Kerberos authentication, X11 forwarding, and port forwarding in the `/etc/ssh/sshd_config` file. For more information, see the `sshd(8)` and `sshd_config(5)` manual pages.

Modifying OpenSSH Server Configuration Files

To configure specific OpenSSH settings, edit the global configuration files in the `/etc/ssh` directory. These files include:

- **`moduli`**

  Contains key-exchange information that is used to set up a secure connection.

- **`ssh_config`**

  Contains default client configuration settings that can be overridden by the settings in a user's `~/.ssh/config` file.

- **`ssh_host_rsa_key`**

  Contains the RSA private key for SSH2.

- **`ssh_host_rsa_key.pub`**

  Contains the RSA public key for SSH2.

- **`sshd_config`**

  Contains configuration settings for the `sshd` service.

You can configure other files in the `/etc/ssh` directory. For details, see the `sshd(8)` manual page.

For Oracle Linux 8 or later, files saved in the `/etc/ssh/sshd_config.d` directory override any settings defined in the `/etc/ssh/sshd_config` configuration file.

For more information, see the `ssh_config(5)`, `sshd(8)`, and `sshd_config(5)` manual pages.

Restricting Access to SSH Connections

The Secure Shell (SSH) provides protected, encrypted communications with other systems. Because SSH is an entry point into the system, disable SSH if it isn't required. Optionally, you can edit the `/etc/ssh/sshd_config` file to restrict its use.

> **Important:**
>
> After applying changes to the configuration file, you must restart the `sshd` service for the changes to take effect.

Restrict Root Access

Set `PermitRootLogin` to `no` to prohibit `root` from logging in with SSH. Then, elevate a user's privileges after logging in.

```
PermitRootLogin no
```

Restrict Specific Users

You can restrict remote access to certain users and groups by specifying the `AllowUsers`, `AllowGroups`, `DenyUsers`, and `DenyGroups` settings, for example:

```
DenyUsers carol dan
AllowUsers alice bob
```

For more information about configuring users and groups, see Oracle Linux 8: Setting Up System Users and Authentication or Oracle Linux 9: Setting Up System Users and Authentication.

Set a Timeout Period

The `ClientAliveInterval` and `ClientAliveCountMax` settings cause the SSH client to time out automatically after a period of inactivity, for example:

```
# Disconnect client after 300 seconds of inactivity
ClientAliveCountMax 0
ClientAliveInterval 300
```

Disable Password Authentication

The `PasswordAuthentication` and `PubkeyAuthentication` settings define the method of authentication the SSH client implements for users: either with a password or with an SSH public key. By default, OpenSSH uses passwords for authentication. However, if you have configured key based authentication, which is more secure, you can optionally disable that functionality:

```
PasswordAuthentication no
PubkeyAuthentication yes
```

For more information, see the `sshd_config(5)` manual page.

Configuring the OpenSSH Server For User Access

User specific configuration on the server side of a connection is in the `$HOME/.ssh` directory and contains the following files:

- **`authorized_keys`**

  Contains the authorized public keys for a user. The server uses the signed public key in this file to authenticate a client.

- **`environment`**

  Contains definitions of environment variables. This file is optional.

- **`rc`**

  Contains commands that `ssh` runs when a user logs in, before the user's shell or command runs. This file is optional.

For more information, see the `ssh(1)` and `ssh_config(5)` manual pages.

Related Topics

- Validating Configuration Permissions

Restricting SSH Key Access to Specific Commands

You can add user specific configurations on the server side of a connection by editing the `$HOME/.ssh/authorized_key` file. In addition to listing SSH keys with which a user can authenticate, you can optionally impose further restrictions on what that user can do with each of those keys.

For example, with the `command` option, you can specify a single command to configure all connections made with one key, after which the command immediately ends.

```
command=command ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEA6OabJhWABsZ4F3mcjEPT3sxnXx1OoUcvuCiM6fg5s...
```

By using the `command` option, security conscious users can restrict system accesses available to a particular key that might be used for a scripted action and which might not be passphrase protected.

You can also ensure that the key is only accepted if the inbound connection originates from the internal network by using the `from` option to set an authorized range of IPv4 addresses. For example, to prevent any IP addresses from outside the 192.0.2.0/24 range from connecting with an SSH key, you would append the following line to the `$HOME/.ssh/authorized_key` file with the correct `key` value:

```
from=192.0.2.0/24 ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEA6OabJhWABsZ4F3mcjEPT3sxnXx1OoUcvuCiM6fg5s...
```

For more information, see the `sshd(8)` manual pages.

Good Practice Recommendations for Configuring OpenSSH Server

We recommend the following guidelines to secure OpenSSH configuration against the most common remote exploits:

- Disable remote root user logins over SSH.
- After you have correctly configured key based authentication, Disable SSH password authentication.
- Consider setting a non standard SSH port for Internet-facing systems.

For more information, see Restricting Access to SSH Connections.

3
Configuring the OpenSSH Client

To set up OpenSSH on the client, you need the `openssh` and `openssh-clients` packages.

Installing the OpenSSH Client Packages

A default Oracle Linux installation includes both the `openssh` and `openssh-clients` packages. If the packages aren't installed, run the following command:

```
sudo dnf install openssh openssh-clients
```

Configuring OpenSSH Client Configuration Files

The `$HOME/.ssh` directory on the client system contains the OpenSSH client configuration files for a particular user as follows:

- **`id_rsa`** and **`id_rsa.pub`**

  Contains a user's SSH2 RSA private and public keys. SSH2 RSA is most commonly used key-pair type. `id_rsa` and `id_rsa.pub` are the conventional names for these files, but no restrictions exist on the file name to use. You can store multiple key pairs in this directory to use across different connections.

  > **Caution:**
  >
  > The private key file can be readable and writable by the user but must not be accessible to other users.

- **`known_hosts`**

  Contains the public host keys that OpenSSH has obtained from SSH servers. OpenSSH adds an entry for each new server to which a user connects.

- **`config`**

  Contains client configuration settings.

  > **Caution:**
  >
  > A `config` file can be readable and writable by the user but must not be accessible to other users.

For more information, see the `ssh(1)` and `ssh-keygen(1)` manual pages.

Validating Configuration Permissions

OpenSSH applies strict permissions to the `$HOME/.ssh` directory and files stored in this directory. If the permissions in the directories on either side of the connection are wrong, OpenSSH prevents the connection and errors out with a `Permission Denied` message.

Access to contents `$HOME/.ssh` directory must be limited to the individual user. An exception to this rule is the `authorized_keys` file, which contains public keys that can be readable to other users.

1. Set the directory and file permissions as follows. Some of these files may not be present on the system where you're running these commands:

```
chmod 700 $HOME/.ssh                  # The user .ssh directory
chmod 600 $HOME/.ssh/id_rsa           # A user's private key
chmod 644 $HOME/.ssh/id_rsa.pub       # A user's public key
chmod 600 $HOME/.ssh/config           # Customized configuration entries for
the ssh client
chmod 644 $HOME/.ssh/authorized_keys  # A user's authorized public key
entries to allow login
chmod 600 $HOME/.ssh/known_hosts      # A user's known hosts entries for
system fingerprints
chown -R $USER:$USER $HOME/.ssh       # Recursively set ownership of
all .ssh files
```

2. Verify that file permissions are correct.

```
ls -al .ssh

drwx------+ 2 user group     5 Jun 12   08:33 .
drwxr--r--+ 3 user group     9 Jun 12   08:32 ..
-rw-r--r--+ 1 user group   397 Jun 12   08:33 authorized_keys
-rw-------. 1 user group  2283 Nov 22 13:22 config
-rw-------. 1 user group   963 Aug 22 09:27 id_rsa
-rw-r--r--. 1 user group   221 Aug 22 09:27 id_rsa.pub
-rw-------. 1 user group 85531 Nov  9 10:01 known_hosts
```

4

Working with SSH Key Pairs

SSH can use key pairs for authentication. Key-based authentication is more secure than password authentication because it helps to avoid brute-force attacks if you disable password authentication in the server configuration.

How SSH Key Pairs Work

To use key authentication, you must first have a key pair: a public key and a corresponding private key. You can either use an existing key pair or generate a new one. Typically, you only generate an SSH key pair one time and only change the key pair if it might have been compromised or when using a key to access systems with different encryption standards. Not all key pairs are compatible with OpenSSH and you might need to convert keys as required. For example, keys generated using the PuTTY ssh client software aren't directly compatible with OpenSSH and might need to be converted before use. See the client software documentation if you're unsure about key format.

After you have obtained a key pair, copy the public key to any server to which you want to connect. Then to connect to the server, provide the matching private key. You can store the private key safely on a single client that you use to access the servers. For security, avoid copying the private key to multiple locations.

When generating key pairs, you can either configure them to have a password or not. Key pairs that don't have passphrases, can help with scripted automation as they can access remote systems instantly so you won't need to enter the passphrase each time you connect. However, using a key without a passphrase can be poor security practice. Instead, you can use SSH Agent to remember a key passphrase for the entire login session.

Consider using SSH Agent Forwarding to connect from trusted system to trusted system, or use the ProxyJump command option where you might need to connect to another system through an untrusted or heavily shared bastion host.

Generating Key Pairs Using the ssh-keygen Command

Use the `ssh-keygen` command to generate a public and private authentication key pair. Authentication keys enable you to connect to a remote system without needing to supply a password each time that you connect. Each user must generate their own pair of keys.

Running ssh-keygen

To create a public and private SSH2 RSA key pair:

```
ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key (/home/guest/.ssh/id_rsa): <Enter>
Created directory '/home/guest/.ssh'.
Enter passphrase (empty for no passphrase): password
Enter same passphrase again: password
Your identification has been saved in /home/guest/.ssh/id_rsa.
```

```
Your public key has been saved in /home/guest/.ssh/id_rsa.pub.
The key fingerprint is:
5e:d2:66:f4:2c:c5:cc:07:92:97:c9:30:0b:11:90:59 guest@host01
The key's randomart image is:
+--[ RSA 2048]----+
|      .=Eo++.o   |
|      o  ..B=.   |
|          o.= .  |
|         o + .   |
|        S * o    |
|       . = .     |
|        .        |
|       .         |
|                 |
+-----------------+
```

To create an SSH key pair by using an algorithm other than the default RSA algorithm, use the `-t` option. Possible values that you can specify include the following: `dsa`, `ecdsa`, `ed25519`, and `rsa`.

For security, in case an attacker gains access to the private key, you can specify a passphrase to encrypt the private key. If you encrypt the private key, you must enter this passphrase each time that you use the key. If you don't specify a passphrase, you arent prompted for a passphrase.

For more information, see the `ssh-keygen(1)` manual page.

Location of key files

`ssh-keygen` generates a private key file and a public key file in `~/.ssh` (unless you specify an different directory for the private key file):

```
ls -l ~/.ssh

total 8
-rw-------. 1 guest guest 1743 Apr 13 12:07 id_rsa
-rw-r--r--. 1 guest guest  397 Apr 13 12:07 id_rsa.pub
```

Enabling Remote System Access Without Requiring a Password

You can create a key pair that doesn't require a passphrase, which is useful for scripted environments where a tool might need SSH access to a remote system but shouldn't prompt for a passphrase.

For general use, and as a better practice, set a passphrase on the private key and then to use the SSH Agent to remember key passphrases for the entire login session. See Using the SSH Key Agent to Remember Passphrases for more information.

However, using the SSH Agent isn't always practical and for some services that are loaded at boot time you might need to create a key that doesn't use a passphrase.

To use OpenSSH utilities to access a remote system without supplying a password each time that you connect:

1. Use `ssh-keygen` to generate a public and private key pair, for example:

   ```
   ssh-keygen
   ```

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa): <Enter>
Created directory '/home/user/.ssh'.
Enter passphrase (empty for no passphrase): <Enter>
Enter same passphrase again: <Enter>
...
```

Press `Enter` each time the prompt to enter a passphrase appears.

2. Copy the public key to the remote server. See Copy the public key to the remote server.
3. If the user names are different on the client and the server systems, create a `~/.ssh/config` file entry for this connection. See Setting SSH Client Configuration Options For a Host.
4. Validate that permissions for the `$HOME/.ssh` configuration files are correct on both the server and client side. See Validating Configuration Permissions for more information.
5. To access the remote system without supplying a password, use `ssh` to log into the remote system to verify that the `~/.ssh/authorized_keys` file contains only the keys for the systems from which you expect to connect, for example:

   ```
   ssh remote_user@host
   ```

If your key file is named in a nonstandard way, you can specify which key file to use by using the `-I` option when you connect:

```
ssh -I ~/.ssh/my_private_key remote_user@host
```

For more information, see the `ssh-copy-id(1)`, `ssh-keygen(1)`, and `ssh_config(5)` manual pages.

Copying Public Keys to Remote Servers

Add the public key to the remote server file at `$HOME/.ssh/authorized_keys`. Various approaches are available for setting up the contents of this file. You can run `ssh-copy-id` or manually configure the file.

Run ssh-copy-id

For systems with password authentication enabled, you can copy the public key from the client system to the remote server using the `ssh-copy-id` command. The tool also sets the permissions of `$HOME/.ssh` and `$HOME/.ssh/authorized_keys` appropriately.

1. Use the `ssh-copy-id` command to append the public key in the local `~/.ssh/id_rsa.pub` file to the `~/.ssh/authorized_keys` file on the remote system, for example:

   ```
   ssh-copy-id remote_user@host
   ```

2. When prompted, enter the password for the remote system.

For more information, see the `ssh-copy-id(1)` manual page.

Manually Setting the authorized_keys File

If you don't have access to the `ssh-copy-id` command or are unable to access the system remotely with a password, you must populate the `$HOME/.ssh/authorized_keys` file manually.

1. Copy the contents of the public key file, typically `$HOME/.ssh/id_rsa.pub`, on the client system and append the contents to `$HOME/.ssh/authorized_keys` on the server system.
2. Ensure that the permissions of `$HOME/.ssh` and `$HOME/.ssh/authorized_keys` are set correctly on the server system.
3. On the remote system, output the `~/.ssh/authorized_keys` file:

   ```
   cat .ssh/authorized_keys
   ```

4. Note whether the key entry is included in the output. For example, an entry might appear as follows:

   ```
   ssh-rsa
   AAAAB3NzaC1yc2EAAAABIwAAAQEA6OabJhWABsZ4F3mcjEPT3sxnXx1OoUcvuCiM6fg5s/ER
   ... FF488hBOk2ebpo38fHPPK1/rsOEKX9Kp9QWH+IfASI8q09xQ== local_user@local_host
   ```

Related Topics

- Validating Configuration Permissions

Centralizing Storage of Authorized Keys

If you have many systems for which you need to provision access for users, consider options to centralize the storage of the `$HOME/.ssh/authorized_keys` file so that revoking a public key for a user with a compromised key pair.

A common approach would be to configure the SSH server to use the System Security Services Daemon to access keys stored in a central location such as an LDAP or Identity Management (IPA) service. To configure user authentication against these services, see Oracle Linux 8: Setting Up System Users and Authentication or Oracle Linux 9: Setting Up System Users and Authentication.

OpenSSH provides a tool to use SSSD to maintain and automatically update a separate cache of public keys when authenticating users. The `sss_ssh_authorizedkeys` command is responsible for retrieving a user's public key from the user entries in an Identity Management (IPA) domain. After the key is retrieved, the key is stored in the `$HOME/.ssh/sss_authorized_keys`, in the standard authorized keys format.

To configure the SSH server to use SSSD to retrieve public keys for users, edit `/etc/ssh/sshd_config` and veerify that the following entries are present:

```
AuthorizedKeysCommand /usr/bin/sss_ssh_authorizedkeys
AuthorizedKeysCommandUser nobody
```

If you have edited the server configuration, you must restart the service:

```
sudo systemctl restart sshd
```

SSD must already be configured and running and the keys must be stored appropriately so that SSH can use the service.

See the `sss_ssh_authorizedkeys(1)` manual page for more information.

Working With known_hosts

Whenever you connect to a remote host, the SSH server on the remote host provides a public key. You can use this key to validate that you're connecting to the same host in the future to prevent Man-In-The-Middle (MITM) attacks. On the server side, this public key is stored as part of the `HostKey` pair. On the client system, the `known_hosts` database stores the public key for the host in the file `$HOME/.ssh/known_hosts`.

RSA key fingerprint

When you connect to a remote system and the `known_hosts` database doesn't contain a key, the client prompts you to accept the fingerprint for the key. For example:

```
The authenticity of host 'server1.example.com (198.51.100.172)' can't be established.
RSA key fingerprint is SHA256:M/Qa7GZf45KPhXsGgQkCpA5dH8BeY5c6nO87chXBWbk.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

If you accept the fingerprint, the `$HOME/.ssh/known_hosts` file stores the public key on the client system and you're no longer prompted every time you connect.

Listing the key fingerprints

You can list the fingerprints for keys stored in the known_hosts database by running:

```
ssh-keygen -l -f $HOME/.ssh/known_hosts
```

StrictHostKeyChecking

If your OpenSSH client has the `StrictHostKeyChecking` option set by default and the public key returned by the server changes, you're unable to connect to the remote server and a warning is displayed:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@       WARNING: POSSIBLE DNS SPOOFING DETECTED!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The RSA host key for server1.example.com has changed,
and the key for the corresponding IP address 198.51.100.172
is unchanged. This could either mean that
DNS SPOOFING is happening or the IP address for the host
and its host key have changed at the same time.
Offending key for IP in /home/user/.ssh/known_hosts:20
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
SHA256:qMBpuawaY58v7LrcpY2BwtbXHOwP/LXLV8FVZk7tDxY.
Please contact your system administrator.
Add correct host key in /home/user/.ssh/known_hosts to get rid of this message.
Offending RSA key in /home/user/.ssh/known_hosts:125
RSA host key for server1.example.com has changed and you have requested strict
checking.
Host key verification failed.
```

Host keys don't change, so when you see this warning, you might not be connecting to the same system that you have connected to before. However, the key can be different for a legitimate reason, such as if the remote system is reinstalled, the OpenSSH Server keys are

regenerated, or the domain name entry or IP address is reassigned to a new system. In such cases, you might want to remove any existing record of the system in the `known_hosts` database.

Removing an existing key

If you're certain that you can trust a new key provided by a remote server, you can remove an existing key from the `known_hosts` database by running:

```
ssh-keygen -R server1.example.com
```

Disabling StrictHostKeyChecking

In test environments, where servers are constantly reinstalled or replaced, you might want to disable `StrictHostKeyChecking` for particular hosts. You can disable host key checking when you connect to a remote system as follows:

```
ssh -o StrictHostKeyChecking=no user@server1.example.com
```

If you have to constantly disable strict host checking, consider adding this option to a host entry in the client configuration. See Setting SSH Client Configuration Options For a Host for more information.

Strict host key checking is enabled by default to prevent Man-In-The-Middle (MITM) attacks, so disabling that functionality isn't considered good security practice and isn't recommended on production systems.

Related Topics

- Modifying OpenSSH Server Configuration Files

Good Practice Recommendations for Working with SSH Key Pairs

Follow these guidelines so that you can manage and use SSH key pairs to connect to remote hosts securely on the network.

- Set a strong passphrase when you generate your SSH key pair.

  For more information, see Generating Key Pairs Using the ssh-keygen Command.
- Verify the SSH key agent to avoid needing to type in the passphrase at every login.

  For more information, see Using the SSH Key Agent to Remember Passphrases.
- Restrict access for any SSH key pair that doesm't have a passphrase and is only used for scripting purposes.

  For more information, see Restricting SSH Key Access to Specific Commands.
- Do'nt share the private key with anyone else. Each member of the team must have their own SSH key pair so that the system administrator can control access to network resources.
- Don't copy the private key, or forward the SSH agent, to any other machine, remote servers, or cloud instances.

  For more information, see Copying Public Keys to Remote Servers.
- Don't store a copy of the private key on a bastion or "jump" host.

For more information, see Using ProxyJump For Access Through a Bastion Host.

5

Using OpenSSH Client Utilities

Use the OpenSSH client utilities to connect to a remote system, copy files between systems, remember passphrases, access through a bastion host, load GUI applications, and port forward.

Connecting to Another System Using the ssh Command

By default, each time you use the OpenSSH utilities to connect to a remote system, you must provide a username and password. When you connect to an OpenSSH server for the first time, the OpenSSH client prompts you to confirm that you're connected to the correct system.

Use the `ssh` command to log into a remote system or to run a command on a remote system:

```
ssh [options] [user@]host [command]
```

where *host* is the name of the remote OpenSSH server to which you want to connect.

For example, you would log in to `host04` by using the same username as is on the local system:

```
ssh host04
```

The remote system prompts you for the password on that system.

To connect as a different user, specify the username and `@` symbol before the remote host name, for example:

```
ssh joe@host04
```

To run a command on the remote system, specify the command as an argument:

```
ssh joe@host04 ls ~/.ssh
```

The `ssh` command logs you in, runs the command, and then closes the connection.

Example of Connecting to a System

The following examples show how you would connect to a remote host, `host04`. You would need to confirm the command action before establishing the connection.

```
ssh host04

The authenticity of host 'host04 (192.0.2.104)' can't be
established.
RSA key fingerprint is 65:ad:38:b2:8a:6c:69:f4:83:dd:3f:8f:ba:b4:85:c7.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'host04,192.0.2.104' (RSA) to the
list of known hosts.
```

When you type `yes` to accept the connection to the server, the client adds the server's public host key to the `$HOME/.ssh/known_hosts` file. When you next connect to the remote server,

the client compares the key in this file to the one that the server supplies. If the keys don't match, you see a warning such as the following:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@       WARNING: POSSIBLE DNS SPOOFING DETECTED!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The RSA host key for host has changed,
and the key for the according IP address IP_address
is unchanged. This could either mean that
DNS SPOOFING is happening or the IP address for the host
and its host key have changed at the same time.
Offending key for IP in /home/user/.ssh/known_hosts:10
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is fingerprint
Please contact your system administrator.
Add correct host key in /home/user/.ssh/known_hosts to get rid of this message.
Offending key in /home/user/.ssh/known_hosts:53
RSA host key for host has changed and you have requested strict checking.
Host key verification failed.
```

Unless the remote server's host key has changed for a known reason, such as an upgrade of either the SSH software or the server, avoid connecting to that machine until you have contacted its administrator about the situation.

For more information, see the `ssh(1)` manual page.

Setting SSH Client Configuration Options For a Host

You can set up Host entries in the `$HOME/.ssh/config` file on a client system. Often usernames don't align on different systems. Sometimes you might choose to use a different key pair to your usual key pair for a particular remote system. To make it easier to connect easily with the correct credentials, you can add a host entry.

Host entries look similar to the following example entry:

```
Host server1
  Hostname server1.example.com
  User remote_user
  IdentityFile ~/.ssh/id_rsa_example
```

With the provided configuration entry, the user can run:

```
ssh server1
```

The SSH client performs a connection to the remote server `server1.example.com` with the username `remote_user` and uses the private key file at `~/.ssh/id_rsa_example`.

This configuration entry lets you connect with the correct credentials each time you want to connect. Without the configuration entry, you would need to enter the following:

```
ssh -I  ~/.ssh/id_rsa_example remote_user@server1.example.com
```

You can use the `$HOME/.ssh/config` file to store other configuration options for any system that you connect to. For example, if you use the ForwardAgent or ProxyJump

options often, consider adding entries for these for each host where you use them. See the `ssh_config(5)` manual page for more information.

Copying Files Between Systems Using the scp and sftp Commands

Using scp

> **Note:**
>
> Secure Copy Protocol (`scp`) is deprecated in Oracle Linux 9. If you are using Oracle Linux 9, use the secure file transfer (SFTP) utility instead. See the *Security* section under the *Deprecated Features* chapter in Oracle Linux 9: Release Notes for Oracle Linux 9.

With the `scp` command, you can copy files or directories between systems. `scp` establishes a connection, copies the files, and then closes the connection.

Upload

To upload a local file to a remote system:

```
scp [options] local_file [user @]host[: remote_file]
```

For example, copy `testfile` to the home directory on `host04`:

```
scp testfile host04
```

Copy `testfile` to the same directory but change its name to `new_testfile`:

```
scp testfile host04:new_testfile
```

Download

To download a file from a remote system to the local system:

```
scp [options] [user@]host[:remote_file] local_file
```

Recursive Option

The `-r` option recursively copies the contents of directories.

For example, copy the directory `remdir` and its contents from the home directory on remote `host04` to the local home directory:

```
scp -r host04:~/remdir ~
```

Using sftp

The `sftp` command is a secure alternative to the `ftp` command that's used to transfer files between systems. Unlike the `scp` command, the `sftp` command enables you to browse the file system on the remote server before copying any files.

To open an FTP connection to a remote system over SSH, use the following command:

```
sftp [options] [user@]host
```

For example, you would open an FTP connect to the system, `host04`, as follows:

```
sftp host04

Connecting to host04...
guest@host04's password: password
sftp>
```

Type `sftp` commands at the `sftp>` prompt.

In the following example, the `put` command is used to upload the file `newfile` from the local system to the remote system, then the `ls` command is used to list it:

```
sftp> put newfile
Uploading newfile to /home/guest/newfile
foo                                           100% 1198     1.2KB/s   00:01
sftp> ls newfile
newfile
```

Type `help` or `?` to display a list of available commands. Type `bye`, `exit`, or `quit` to close the connection and exit the `sftp` interactive session.

For more information, see the `ssh(1)` and `sftp(1)` manual pages.

Using the SSH Key Agent to Remember Passphrases

Use the SSH Key Agent to enter the passphrases for any of the SSH keys a single time throughout the login session. In this manner, you avoid the poor security practice of creating SSH keys without passphrases.

1. After you log in, check that the agent is started:

   ```
   ps -ef|grep -i ssh-agent
   ```

2. Run the `ssh-add` command to add any of the ssh keys to the agent:

   ```
   ssh-add $HOME/.ssh/id_rsa
   ```

   The command prompts you for the key passphrase. The passphrase applies through the entire login session. If you use the key to connect to another system, the prompt for a passphrase no longer appears.

   > **Note:**
   >
   > The error message `Could not open a connection to your authentication agent` could indicate that the agent might not be running. Start it with the `eval $(ssh-agent -s)` command.

3. Repeat the command for each key that you want to add.
4. After adding the keys to the agent, you can open SSH connections to any systems that have the paired public key configured in the `authorized_hosts` file, without being prompted for a passphrase. This behavior applies also to any scripts run as your own user.

Using SSH Agent Forwarding for Access Through a Bastion Host

> **Caution:**
>
> Enable agent forwarding with caution. Users with escalated privileges on the remote host can access the agent through the forwarded SSH session. Although malicious users can't access the keys directly they can hijack the agent session and use the keys in the agent to connect to other systems. If you're connecting to a system that might have untrustworthy users then avoid using agent forwarding.

SSH Agent Forwarding is a powerful tool that can help you keep private keys centralized and safe. Avoid copying private keys to other systems as much as possible. SSH Agent Forwarding lets you connect to a remote system and then use the SSH client on that system to connect to another system by using the same key based authentication but without you needing to copy the private key to the host that you first connected to.

Server-side Configuration

Change the `/etc/ssh/sshd_config` file to configure SSH Agent forwarding. On the server, verify the following parameters:

`AllowAgentForwarding`
Allows SSH Agent forwarding. When omitted, the default is `yes` which enables SSH Agent Forwarding

Client Configuration

To enable this functionality you must use the `ForwardAgent` option when you make a connection to an intermediate system in the chain of hosts that you connect to. You must also have the private key already loaded into the SSH Agent on the primary client host. See Using the SSH Key Agent to Remember Passphrases .

To use SSH Agent Forwarding:

1. Check that the SSH Agent is running and that the SSH key is loaded. Run the following command on the client system to see what keys the agent has loaded:

   ```
   ssh-add -L
   ```

2. Connect to a host using the `ForwardAgent=yes` option:

   ```
   ssh -o ForwardAgent=yes server1.example.com
   ```

3. Use the SSH client on the remote host to connect to another server that has the public key configured in its `authorized_keys`:

   ```
   ssh server2.example.com
   ```

Consider adding the ForwardAgent option to a Host configuration entry in the `$HOME/.ssh/config` file if you use this option often for a particular server. See Setting SSH Client Configuration Options For a Host for more information.

Using ProxyJump For Access Through a Bastion Host

SSH Agent forwarding lets you connect from one server to the next using key-based authentication without copying the private key to each server in the chain. This approach raises some security concerns because users with the appropriate privileges on the remote server could hijack the agent and use it to connect to other systems without requiring authentication.

A more secure approach to jumping from one server to the next is to use the `ProxyJump` option in the OpenSSH client. The `ProxyJump` option functions similarly to an SSH tunnel or port forward in the sense that it proxies all traffic straight through the bastion or jump host. Unlike port forwarding, `ProxyJump` option doesn't require server-side configuration. You only need to have SSH access to the bastion or jump host.

Consider that `internal.example.com` is a host on an internal network that you don't have direct access to. `bastion.example.com` is a host that's connected to the internal network and is also accessible to the client system. To connect to `internal.example.com`, you can use the `ProxyJump` option to connect directly through `bastion.example.com`. For example:

```
ssh -o 'ProxyJump=bastion.example.com' internal.example.com
```

If you constantly connect to `internal.example.com`, you can set the `ProxyJump` option for that host inside the `$HOME/.ssh/config` file. See Setting SSH Client Configuration Options For a Host.

Using X11 Forwarding to Load Remote Graphical Applications

X11 forwarding lets a user start graphical applications installed on a remote Linux system so that they display within the desktop environment of the local system. The remote system doesn't need to have an X11 server or graphical desktop environment running, but the local system must have an X11 compatible service running.

Server-side Configuration

1. Change the `/etc/ssh/sshd_config` file to enable X11 forwarding. On the server, verify the following parameters:

   `X11Forwarding`

   Allows X11 forwarding. When omitted, the default is `no`. To enable X11 forwarding, add an entry that sets the value for this parameter to `yes`.

2. If you edit the configuration file, you must restart the service for the change to take effect:

   ```
   sudo systemctl restart ssh
   ```

3. The remote system must also be able to run X11 applications and authenticate X11 sessions. The `xorg-x11-xauth` package is required for this purpose.

   ```
   dnf install xorg-x11-xauth
   ```

If you have never run a graphical application on the remote server, the first time that you connect to the remote server using X11 forwarding, a warning message is displayed:

```
/usr/bin/xauth: file /home/user/.Xauthority does not exist
```

You can ignore this warning as the .Xauthority file is automatically created.

Client Configuration

1. Use the `-X` option with the SSH client when you connect to a remote server:

   ```
   ssh -X user@server1.example.com
   ```

2. Run a graphical application over the SSH connection by typing the command directly from the SSH terminal. You might opt to run the process in the background so that the terminal remains available to you. For example:

   ```
   gedit &
   ```

Setting Up Port Forwarding Over SSH

SSH port forwarding creates an encrypted SSH tunnel between a client and a server system.

Three types of SSH port forwarding are available:

- Local Port Forwarding - Forwards a port from the client to the SSH server and then to the destination port.
- Dynamic Port Forwarding - Creates a SOCKS proxy server for communications across a range of ports.
- Reverse Port Forwarding - Forwards a port from the server to the client and then to the destination port.

Why Use Port Forwarding?

Port forwarding lets remote servers to access devices within a private local-area network (LAN) and conversely.

You can use port forwarding to access a service that's not exposed to the public network interface. You might set up a local port forward to access a service (such as a database) on a remote server. The database on the server isn't exposed to the public network interface, but you could create a tunnel from a local machine to the internal database server port. You can then connect to localhost and all traffic would get forwarded across the SSH tunnel to the remote database.

You can use reverse port forwarding to give someone outside the local network access to an internal service. For example, you might want to show a fellow developer a web application that you have developed on the local machine. Because the machine doesn't have a public IP, the other developer can't access the application over the internet. However, if you have access to a remote SSH server, you can set up reverse port forwarding to provide the developer access.

Server-side Configuration

Edit the `/etc/ssh/sshd_config` file to configure SSH port forwarding. On the server, at a minimum verify the following parameters:

- **`AllowTCPForwarding`**

Allows TCP port forwarding. When omitted, the default is `yes` which enables single TCP port forwards and SOCKS proxying

- **`AllowStreamLocalForwarding`**

  Allows forwarding of UNIX domain sockets. When omitted, the default is `yes`.

Local Port Forwarding

To create a direct TCP forward tunnel, use the ssh `-L` option:

```
ssh -L [bind_address:]port:destination:destination_port [user@]remote_ssh_server
```

- `bind_address` is optional and assigns a local interface to listen for connections. If omitted, *ssh* only binds on the loopback interfaces. To bind on all interfaces, you can use *"0.0.0.0"* or *"::"*.
- `port` - The local port number. You can use any port number greater than `1024`.
- `destination` - The IP or hostname of the destination machine. If the destination is on the remote server itself, you can use `localhost`.
- `destination_port` - Port on the destination machine.
- `[user@]remote_ssh_server` - The remote SSH user and server IP address.

For example:

```
ssh -L 8080:localhost:8888 user@192.168.1.20
```

This would open an SSH connection to the remote server at 192.168.1.20 and open a tunnel to the localhost port 8888.

Dynamic Port Forwarding

Use dynamic port forwarding to have the SSH client listen on a specified binding port and act as a SOCKS proxy server. You don't need to specify a destination host as all incoming connections on the specified port forward through the tunnel to a dynamic port on the destination machine.

To create a dynamic port forward, use the ssh `-D` option.

```
ssh -D [bind_address:]port [user@]remote_ssh_server
```

Reverse Port Forwarding

A reverse tunnel forwards any connection received on the remote SSH server to the local client network.

To create a reverse port forward, use the ssh `-R` option.

For local port reverse forwarding:

```
ssh -R [bind_address:]port:destination:destination_port [user@]remote_ssh_server
```

For dynamic port reverse forwarding:

```
ssh -R [bind_address:]port [user@]remote_ssh_server
```