15.10.2 Configure Public Key Authentication

Click one of the buttons to take you to that part of the video.

Configure Public Key Authentication 0:00-0:21

In this demonstration, I'm going to show you how to configure public key authentication between two systems--one running the SSHD daemon and the other running the SSH client. This will allow us to authenticate with the SSH client without having to provide a password.

Configure SSHD Daemon 0:22-2:26

In order to do this, we first need to configure the SSH daemon to allow public key authentication. I'm going to run the SSH daemon on this system right here, so I need to switch to my root user account. I'm going to load the '/etc/ssh/sshd_config' file in the vi editor.

At this point, we need to find the pub key parameter. I'm going to search for 'pub key', and there it is right there.

I'm going to press the Insert key. Notice that right now, public key authentication is disabled because we have a # in front of the directive that would turn it on. I'm going to delete that #. That enables public key authentication.

Also be aware that in order for public key authentication to work, this line has to be enabled as well, and it already is by default. This tells us where we can look for authorized public keys. By default, the SSHD daemon is going to look in the user's .ssh hidden directory in their /home directory and look in the authorized_keys file.

This looks good. We'll go ahead and exit out. Save our changes. Before this will work, we do need to restart the SSH daemon running on this system. I'm going to use the 'systemctl' command, and we'll do a 'restart' of the 'sshd' daemon, and then let's do this same command again, 'systemctl'.

We'll lose the status option this time. View the status of the daemon. It looks like everything loaded properly. Before we go any further, I do need to check and see what the IP address is on this system.

It's '10.0.0.228'. With this in mind, let's go over to the system where we are going to run the SSH client and configure public key authentication. Doing this will allow us to authenticate to the SSH daemon running on this system without having to provide the user's password. Authentication will take place using our key files.

Generate a Key Pair 2:27-4:25

We are logged in to this system as the rtracy user, and as the rtracy user, I want to generate an rsa key pair. This is done using the ssh-keygen command.

We run the 'ssh-keygen' and we specify the '-t' option. Specify the type of keys we want to create; we are going to create 'rsa' keys. Press Enter.

It prompts us to specify where we're going to save the private key in the key pair. By default, it's going to save in my .ssh hidden directory with the filename id_rfc. That's just fine. I'll press Enter.

I have to specify a passphrase for this key. This is very important that you don't forget the passphrase that you enter here, because we're going to use it in just a second. We have to reenter it.

Notice up here in the output of the command it tells us that the private key has been saved in id_rsa and the public key, which is going to be the one we're working with, is in id_rsa.pub. At this point, in order to use public key authentication, we have to copy this key file from this system over to the other system where the SSHD daemon is running.

We have to put it in the /home directory of the user who we're going to connect as when we establish an SSH connection between the systems. It's important to note when you copy this key between systems, that it needs to be done securely. Don't attach it as an attachment to an email and send it through the internet, because email is notoriously insecure.

Instead, you could either put it on a USB drive and carry it between the systems, or you could also copy it securely using the scp command. That's what we're going to do here. Enter 'scp' and we want to copy the file in our '.ssh' hidden directory in the rtracy user's /home directory named 'id_rsa.pub'.

Copy Public Key to /home Directory 4:26-8:23

We want to copy that to the ksanders user's /home directory on the system that has the IP address of '10.0.0.228'. Enter. And because it's the first time we made a connection, we have to accept the key fingerprint. Yes, that looks good. We have to provide the password to the ksanders user account, and the file has been copied.

At this point, we need to then copy this file, id_rsa.pub, that we just copied to the /home directory of the ksander's user on the other system. We need to copy that key file to the authorized_keys file that we looked at earlier when we were configuring the SSHD daemon. One way to do this would be to switch over to the other system and do that as the ksanders user. Or because we have ssh running, we can just do it from right here.

Let's open up a new SSH session with the SSH client, '-l ksanders 10.0.0.228'. Notice that I'm accessing the openSUSE Linux system remotely as the ksanders user. Everything that I'm doing at this point is being done on the remote system, not on the local system.

Do an 'ls' command, we should see the file we just copied--id_rsa.pub. One other thing we need to do is run the 'ls' command again and just make sure that a .ssh hidden directory exists for the user. It does. ksanders already has a .ssh hidden directory. If it didn't exist, then we would need to go in and create it using the mkdir command. We don't have to worry about that.

What we need to do now is add this public key file that we just copied up from this system to the authorized_keys file of the ksanders user. One easy way to do that is to simply type 'cat' and then 'id_rsa.pub'. Instead of just displaying it on the screen, which is what cat does by default, we instead want to redirect the output to that very file 'authorized_keys'.

Notice when I type this command that I used two > signs, not one. That's very important because if there are already keys in the authorized_keys file, more than likely we want them there.

If we were just to use one > sign, then this key would overwrite all the other keys in authorized_keys, and that could be a problem. If we use two > signs, it adds this key file to the end of the authorized keys file, instead of overriding everything in the authorized_keys file.

Press Enter. If we want to, we can verify that happened correctly by typing 'cat .ssh/authorized_keys', and there's the public key we just copied into the authorized_keys file. It's associated with the rtracy@fs5.corpnet.com user account. At this point we can enter 'exit'.

For the moment of truth, we're going to test this new configuration. I'm going to establish an SSH connection again to the openSUSE system that has the IP address of 10.0.0.228. I'm going to connect as ksanders again just like I did last time, but this time we should see a very different result.

Notice this time instead of being prompted to provide the password for the ksanders users account, I'm being prompted to enter the passphrase to unlock the private key. Click Unlock. I'm now connected, using the SSH client as the ksanders user to the openSUSE Linux system. It has an IP address of 10.0.0.228. Let's go ahead and close this section by typing 'exit'.

Create Passphrase Shortcut for SSH Client 8:24-9:22

At this point what I want to do is configure this system to remember that passphrase so that I don't have to provide it every time that I make this connection. To do this, I'm going to use the SSH agent. I'm going to type, 'ssh-agent bash' first.

Now I'm going to enter 'ssh-add .ssh/id_rsa'. It's going to ask me to enter the passphrase for the private key. This is the same passphrase we configured when we initially generated the key pair. The identity has been added. When I establish an SSH connection as ksanders to the system with the IP address 10.0.0.228, I should no longer be prompted to specify the passphrase for the key pair. Right in.

Summary 9:23-9:53

That's it for this demonstration. In this demo we talked about how to configure public key authentication for the SSH client and SSH server. We first generated a key pair, then we copied the public key to the user's /home directory on the remote server.

We added that public key to the authorized_key file for the user, and then we were able to authenticate to the system without providing a password. We ended this demonstration by talking about how we can use the SSH-agent and SSH-add commands to make it such that we don't have to enter the passphrase every single time we connect using the SSH client.
